The CiaoPP Program Processor

A Program Analysis, Verification, Debugging, and Optimization Tool
REFERENCE MANUAL
The Ciao Documentation Series
`https://ciao-lang.org/`
Generated/Printed on: 22 July 2023
Technical Report CLIP 1/06 (first version 8/95).
Version 1.5 (2022/3/2, 20:34:30 CET)

The Ciao Development Team
Edited by:
Francisco Bueno
Manuel Hermenegildo
Pedro López
José Francisco Morales
Germán Puebla

The Computational logic, Languages,
Implementation, and Parallelism (CLIP) Lab
`https://www.cliplab.org/`
School of CS, T. U. of Madrid (UPM)
IMDEA Software Institute

Table of Contents

1 Summary

CiaoPP is the abstract interpretation-based preprocessor of the Ciao multi-paradigm program development environment. CiaoPP can perform a number of program debugging, analysis, and source-to-source transformation tasks on (Ciao) Prolog programs. These tasks include:

- Inference of properties of the predicates and literals of the program, including types, modes and other variable instantiation properties, non-failure, determinacy, bounds on computational cost, bounds on sizes of terms in the program, etc.
- Certain kinds of static debugging and verification, finding errors before running the program. This includes checking how programs call system library predicates and also checking the assertions present in the program or in other modules used by the program. Such assertions represent essentially partial specifications of the program.
- Several kinds of source to source program transformations such as program specialization, slicing, partial evaluation of a program, program parallelization (taking granularity control into account), inclusion of run-time tests for assertions which cannot be checked completely at compile-time, etc.
- The abstract model of the program inferred by the analyzers can be used to certify that untrusted mobile code is safe w.r.t. a given policy (this implements the *abstraction-carrying code* approach to mobile code safety).

The information generated by analysis, program properties to be verified, descriptions of the system libraries, directives provided to guide analysis, etc. are all written in the same assertion language, which is in turn also processed by the Ciao system documentation generator, `lpdoc`.

Note: This is the CiaoPP **reference** manual. To begin using CiaoPP, we suggest you start by following one or more of the companion CiaoPP **tutorials** available in the Ciao system documentation.

This documentation corresponds to version 1.5 (2022/3/2, 20:34:30 CET).

2 Introduction

CiaoPP is the abstract interpretation-based preprocessor of the Ciao multi-paradigm program development environment. CiaoPP can perform a number of program debugging, analysis, and source-to-source transformation tasks on (Ciao) Prolog programs. These tasks include:

- Inference of properties of the predicates and literals of the program, including types, modes (⟨undefined⟩ [groundness and sharing], page ⟨undefined⟩), ⟨undefined⟩ [term structure], page ⟨undefined⟩, and other variable instantiation properties, non-failure, determinacy, bounds on computational cost (e.g., ⟨undefined⟩ [number of resolution steps of the computation], page ⟨undefined⟩), bounds on size of terms in the program, etc (see Chapter 7 [Available abstract domains], page 31).
- Certain kinds of static debugging and verification, finding errors before running the program. This includes checking how programs call system library predicates and also checking the assertions present in the program or in other modules used by the program. Such assertions represent essentially partial specifications of the program.
- Several kinds of source to source program transformations such as program specialization, slicing, partial evaluation of a program, program parallelization (taking granularity control into account), inclusion of run-time tests for assertions which cannot be checked completely at compile-time, etc.
- The abstract model of the program inferred by the analyzers can be used to certify that untrusted mobile code is safe w.r.t. a given policy (this implements the *abstraction-carrying code* approach to mobile code safety).

The information generated by analysis, program properties to be verified, descriptions of the system libraries, directives provided to guide analysis, etc. are all written in the same assertion language, which is in turn also processed by the Ciao system documentation generator, `lpdoc`.

CiaoPP is distributed under the GNU general public license.

2.1 How to use this manual

This is a *reference manual*. I also includes a part on the CiaoPP internals. You can use this manual to:

- Look up descriptions for the commands, flags, options, domains, transformations, etc. of CiaoPP. In the html versions of the manual you can use the search facility to this end. In other versions the Predicate/Method Definition Index can help you in locating commands; the Regular Type Definition Index can help in locating the definitions of the types associated to the arguments of commands; the Concept Definition Index may help in locating the part of the manual where a particular feature of CiaoPP is described; and the Global Index includes all of the above plus references to pages where the command, type, or concept is used (not necessarily defined).
- Learn about how to extend CiaoPP, for example by defining new analysis domains.
- Learn about CiaoPP internals.

Since this is a **reference** manual, to begin using CiaoPP, we suggest you start by following one or more of the companion CiaoPP **tutorials** that are available in the Ciao system documentation.

This manual assumes in some places some familiarity with the techniques that implement the different functionalities, such as abstract interpretation, partial evaluation, etc. However, references are included to technical papers that explain in detail such techniques.

2.2 Note

Since we are often merging new functionality into the CiaoPP distribution from more experimental versions, you may find sometimes that some documented functionality in this manual is not available or not working properly. We will be grateful if you report any missing functionality or bugs you find.

2.3 Version/Change Log

Version 1.5 (2022/3/2, 20:34:30 CET)

- ADDED: CiaoPP is included now by default with the `devenv` bundle.
- ADDED: Output options in the `output/2` predicate.
- ADDED: More examples (most of them for verifly).
- ADDED: Documentation for debugging analyses.
- ADDED: `ciaoppcl` can start as an active module (flycheck).
- IMPROVED: Better automatic selection of analysis domains (covering better the properties to be checked; using mode domains before type domains, etc.); `dom_sel` flag to decide whether to automatically select domains for verification.
- IMPROVED: Refactoring and improving `ctchecks` (compile-time assertion checking) code: modular ctchecking working again (i.e., after intermodular fixpoint); report assertions per module; better messages; many bug fixes.
- IMPROVED: Reducing boilerplate code in the implementation of analysis domains using the new `aidomain` package.
- IMPROVED: Simplified menu, unified menu for analysis and ctchecks.
- IMPROVED: Improved precision of (nf,det) analysis domains. Adding (experimental) new `nfdet` combined analysis domain.
- CHANGED: Change default setting for entry point selection from assertions (exported calls by default).
- CHANGED: Disabled output for `ciaoppcl -V`.
- CHANGED: Assertion simplification (eliminating statically proven properties from run-time checks) now enabled by default during assertion checking.
- CHANGED: `trust pred` assertions expanded to `check calls`, `trust success` and `trust comp` assertions.
- CHANGED: Abort `auto_interface` action early if there is a compilation error.
- CHANGED: `pp_info` flag (printing analysis infor at literal level) moved to the naive menu.
- FIXED: Many bug fixes and performance improvements: eliminated some dangling choicepoints; avoid printing some calls assertions twice; wrong use of `denorm_goal_prop/3` changed to `prop_unapply/3`; missing clause to `abs_exec_cond:type_of/4` to handle `deftypes` domain.
- FIXED: Bug in treatment of dynamic predicates.

Version 1.4 (2020/11/18, 6:18:7 CET)

- General changes and new features:
  - LLVM-based frontend improved and moved to its own bundle (see `ciaopp_llvm` bundle).
  - Split `typeslib` as a separate bundle.
  - Allow heads of entry assertions to be non-normalized.
  - Adapted to new hiord.

- [new] Command `ciaopp-dump`, which performs several actions over a ciaopp analysis dump: it subsumes the old `ciaopp-dump-cmp` and `ciaopp-dump-cmp` commands. New options for: showing the contents of a dump file, a code reachability report, static, offline checks of assertions using a dump file, and general statistics for the analysis results.
- [new] Added `-op Suffix` option to ciaopp command-line (useful for fly-check).
- [new] Warning if no entries were found
- [new] Unified messages, now controlled by `ciaopp_log.pl`
- [new] `ciaopp-batch` command to run ciaopp on many files and with time-outs.
- [new] Modular (non-incremental) analysis with types (shortening).
- [new] Bottom-up incremental update of fixpoint (at literal level).
- [new] Modular (non-incremental) analysis with types (shortening).

- Domains:
  - Implemented domains using (a special) trait package.
  - Added `needs/2` operation.
  - Renamed `abstypes_abs` to `auxinfo_asub`.
  - eterms: `member/2`, `=/2` as native properties (for assertions).
  - pdb:
    - Optimize `pdb_compute_lub`: top if any `ASub` is top.
    - Use `fail` literals.
  - sharing: fixed `free/1` handlers
  - shfr: fixes in handlers of `==/2`, `=/2`, `'\=='/2`, `'\='/2`, `free/1` (native version of `var/1`)
- Fixpoints:
  - Split compilation options (`debug`, `trace`, `rtchecks`) in common file `fixpoint_options.pl`
  - Improved tracing of clauses
  - Refactored fixpoints to make them easier to merge
- Fixpoint `dd`:
  - [fix] Reuse a complete with the same head but different vars order
  - [performance] Using get_memo_table because Id+Key is unique
  - [fix] Case in which a literal is not reachable
  - Improved documentation
- Analysis of higher-order code:
  - [new] Flag for automatic generation of entries of meta calls.
  - [fix] `call/N` was not being detected as meta predicate (and flattened when possible).
  - [new] `\+` not treated as meta, flatten calls known at compile time.
- Flags and options menu:
  - [new] Flag to remove useless (unreachable) completes after fixpo.
  - `punit_boundary`: change default value to bundle.
  - Included `incanal` flag, changed `ext_policy` in menu (same as `pp_flag`).
  - Improved options menu (added and simplified dependencies, simplified text, grouped actions, better documentation).

Version 1.3 (2019/11/3, 11:16:23 CEST)

- New functionality:
  - Backported (and improved) incremental analysis (incanal) from CiaoPP 0.8.
  - Integrated incremental and modular analysis (`pp_flag incremental=on`)
  - Integrated modular analysis in the general analysis driver.
  - Added analysis using run-time semantics of assertions to the `dd` fixpoint (`pp_flag old_trusts=off`).
  - Fixes in meta-calls (work in progress).
  - Modularized domains, now they use the aidomain package (work in progress).
  - Added generic, non relational domain with simpler domain operations.
  - New flag value: restrict modular analysis to the modules of a bundle (`pp_flag punit_boundary=bundle`).
- Performance improvements:
  - Added hook to cache library assertions.
  - Added flag to load modules faster by preloading ciao libraries.
  - Fixed many dangling choicepoints across all modules.
  - Fixed performance issues in modular analysis.
  - Fixed bug of metacalls inside a recursion (`dd` fixpoint).
- User interface and documentation:
  - Rearranged manual.
  - Created web interface for demos (see `ciaopp_online` bundle).
  - Added a high level interface for intermental analisis (incanal).
- Benchmarks, internal testing, and debugging:
  - Tests/benchmarks moved to a separate bundle (ciaopp_tests).
  - Added tests for gitlab CI (continuous integration).
  - Some fixes in the davinci interface.
  - Fixed (documentation) assertions. CiaoPP can be run with the package rtchecks.
  - Added analysis_stats.pl to unify statistics collecting.
  - ciaopp-dump-cmp: new command to compare two ciaopp analaysis dumps.
  - ciaopp-dump-fmt: new command that outputs the analysis in dot format.
  - New flag value: Raw analysis printer (`pp_flag output_lang=raw`).
  - Added `pp_flag` to trace fixpoints.
- Other bug fixes, cleanups, refactoring, etc.:
  - Removed `api` (now using `p_unit`).
  - Fixed maintaining `plai_db` for complete operations
  - Fixed classifying mistakenly recursive predicates as non recursive (meta predicates).
  - Fixed issue with reexported predicates in p_unit.
  - Fixed problem with meta_predicates that did not have any meta argument.
  - Added `free/1` to `sharefree_clique` domain.
  - Replaced `runtime_control:statistics/2` by `analysis_stats:pp_statistics/2`.

- Using `p_unit/program_keys.pl` for the keys of ciaopp.

Version 1.2 (2006/1/3, 11:16:23 CEST)
Check point

Version 1.0 (2001/11/7, 19:5:57 CET)
Major code reorganization from CiaoPP 0.8. Moved to repo system. See 0.8 manual for changelog prior to 1.0 (covers 1999-2000).

3 PART I - Using CiaoPP

Author(s): The Ciao Development Team.

This part documents several levels of interaction with CiaoPP. As quick introduction we recommend going through the ⟨undefined⟩ [CiaoPP Quick Tutorial], page ⟨undefined⟩ and ⟨undefined⟩ [Advanced Tutorial on Program Development and Optimization using the CiaoPP Program Processor], page ⟨undefined⟩ sections.

CiaoPP can be used from an intuitive graphical menu interface, based on the `emacs` editor, which allows the interactive selection of configuration options. An equivalent command-line interface, described in Chapter 6 [The CiaoPP command-line interface], page 27, is provided for non-interactive use.

For more advanced uses, CiaoPP can be used directly as a library from a toplevel interface. We provide both a mostly automatic high-level interface, described in Chapter 4 [The CiaoPP high-level interface], page 11, and a low-level interface intended for advanced users, detailed in Chapter 5 [The CiaoPP low-level interface], page 17. Both the graphical and the command-line interfaces are based on the functionality provided by the high and low-level interfaces.

4 The CiaoPP high-level interface

Author(s): CiaoPP development team.

This module defines the high-level interface for CiaoPP, which makes it easy to perform some analysis and transformation tasks, such as checking program assertions (i.e., types, modes, determinacy, non-failure, cost, etc.), and performing optimizations such as specialization and parallelization. The results can be observed both as CiaoPP messages and as a transformed annotated program.

In the `emacs` environment these actions can be performed by clicking on the corresponding button in the toolbar or in the CiaoPP menus. The high-level interface allows calling these actions as predicates from a **CiaoPP top-level** shell:

- `auto_check_assert(File)`: check assertions in File
- `auto_analyze(File)`: analyze File
- `auto_optimize(File)`: optimize File

The actions above can be controlled by a set of **preprocessor flags**. Note that, depending on some of these flags, processing can be of one module or also all the related modules. The default values of the CiaoPP flags can be changed with the following predicates:

- `customize(all)`: select (interactively) the values for the different options (do not perform any action).
- `set_menu_flag/3`: select flag values non-interactively.

The `emacs` environment offers a graphical version of these menus.

These customization menus can be configured to show more or less detailed options, depending on the level of expertise of the user. This can be configured in the menu changing the `Menu Level` flag to `naive` or `expert`. The following predicates provide handy shortcuts to perform customization and preprocessing actions:

- `customize_and_preprocess(File)`: enter a menu to select the preprocessing action and options to be performed on file `File` (or `File` and its related modules), select the different options, and then perform the action.
- `again`: perform again the last actions selected for `customize_and_preprocess/1` on the same file (useful for re-processing after changing a file).

4.1 Usage and interface

- **Library usage:**

  `:- use_module(ciaopp(auto_interface)).`
- **Exports:**
  - *Predicates:*

    `auto_analyze/1`, `auto_optimize/1`, `auto_check_assert/1`, `auto_analyze/2`, `auto_optimize/2`, `auto_check_assert/2`, `customize/0`, `customize/1`, `customize_and_preprocess/0`, `customize_and_preprocess/1`, `customize_but_dont_save/1`, `again/0`.
  - *Multifiles:*

    `hook_menu_flag_values/3`, `hook_menu_check_flag_value/3`, `hook_menu_flag_help/3`, `hook_menu_default_option/3`, `aidomain/1`.

4.2 Documentation on exports

auto_analyze/1: PREDICATE

Usage: `auto_analyze(F)`

Analyze the module `F` with the current analysis options (use `customize(analyze)` to change these options).

auto_optimize/1: PREDICATE

Usage: `auto_optimize(F)`

Optimize file `F` with the current options (use `customize(optimize)` to change these options).

auto_check_assert/1: PREDICATE

Usage: `auto_check_assert(F)`

Check the assertions in file `F`, with the current options, giving errors if assertions are violated (use `customize(check_assertions)` to change these options).

auto_analyze/2: PREDICATE

Usage: `auto_analyze(F,OFile)`

Same as `auto_analyze/1` but the output file will be `OFile`.

auto_optimize/2: PREDICATE

Usage: `auto_optimize(F,OFile)`

Same as `auto_optimize/1` but the output file will be `OFile`.

auto_check_assert/2: PREDICATE

Usage: `auto_check_assert(F,OFile)`

Same as `auto_check_assrt/1` but the output file will be `OFile`.

customize/0: PREDICATE

Usage:

Enter an interactive menu to select the preprocessing action (analysis / assertion checking / transformation / optimization / ...) to be performed by default and the different options (i.e., setting the preprocessor flags).

customize/1: PREDICATE

Usage: `customize(X)`

Customize is used for changing the values of the flags used during preprocessing. These flags are grouped into three main classes of actions: *analyzing*, *checking assertions*, or *optimizing* programs. `X` should be instantiated to one of: `analyze`, `check_assertions`, `optimize`, or `all` (which allows choosing among the previous three).

customize_and_preprocess/0: PREDICATE

Usage:

Select options using `customize/0`, and then call `auto_analyze/1`, `auto_optimize/1`, or `auto_check_assert/1` (as determined by the selected options) on the default file. If no default file is defined, prompt for the name of to be processed, which becomes from now on the default file.

customize_and_preprocess/1: PREDICATE

Usage: `customize_and_preprocess(File)`

Select options using `customize/0`, and then call `auto_analyze/1`, `auto_optimize/1`, or `auto_check_assert/1` (as determined by the selected options) with `File` as argument. `File` is from now on the default file.

customize_but_dont_save/1: PREDICATE

Usage: `customize_but_dont_save(Option)`

Same as customize(`Option`), but menu flags will not be modified.

again/0: PREDICATE

Usage:

Performs the last actions done by `customize_and_preprocess/1`, on the last file previously analyzed, checked, or optimized

get_menu_configs/1: PREDICATE

Usage: `get_menu_configs(X)`

Returns a list of atoms in `X` with the name of stored configurations.

- *The following properties should hold at call time:*
  `X` is a free variable. (`var/1`)
- *The following properties should hold upon exit:*
  `X` is a list of `atm`s. (`list/2`)

save_menu_config/1: PREDICATE

Usage: `save_menu_config(Name)`

Save the current flags configuration under the `Name` key.

- *The following properties should hold at call time:*
  `Name` is an atom. (`atm/1`)

remove_menu_config/1: PREDICATE

Usage: `remove_menu_config(Name)`

Remove the configuration stored with the `Name` key (the same provided in `save_menu_config/1`).

- *The following properties should hold at call time:*
  `Name` is an atom. (`atm/1`)

restore_menu_config/1: PREDICATE

Usage: `restore_menu_config(Name)`

Restore the configuration saved with the `Name` key (the same provided in `save_menu_config/1`).

- *The following properties should hold at call time:*

  `Name` is an atom. (`atm/1`)

show_menu_configs/0: PREDICATE

Usage:

Show all stored configurations.

show_menu_config/1: PREDICATE

Usage: `show_menu_config(C)`

Show specific configuration values pointed by `C` key (the same provided in `save_menu_config/1`).

- *The following properties should hold at call time:*

  `C` is an atom. (`atm/1`)

get_menu_flag/3: (UNDOC_REEXPORT)

Imported from `menu_generator` (see the corresponding documentation for details).

get_menu_flag/3: (UNDOC_REEXPORT)

Imported from `menu_generator` (see the corresponding documentation for details).

set_menu_flag/3: (UNDOC_REEXPORT)

Imported from `menu_generator` (see the corresponding documentation for details).

set_menu_flag/3: (UNDOC_REEXPORT)

Imported from `menu_generator` (see the corresponding documentation for details).

menu_branch/4: (UNDOC_REEXPORT)

Imported from `menu_rt` (see the corresponding documentation for details).

menu_branch/3: (UNDOC_REEXPORT)

Imported from `menu_rt` (see the corresponding documentation for details).

true/2: (UNDOC_REEXPORT)
Imported from `menu_rt` (see the corresponding documentation for details).

true1/1: (UNDOC_REEXPORT)
Imported from `menu_rt` (see the corresponding documentation for details).

functor1/2: (UNDOC_REEXPORT)
Imported from `menu_rt` (see the corresponding documentation for details).

4.3 Documentation on multifiles

hook_menu_flag_values/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

hook_menu_check_flag_value/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

hook_menu_flag_help/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

hook_menu_default_option/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidomain/1: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

4.4 Documentation on imports

This module has the following direct dependencies:

- *Application modules:*
  `frontend_driver`, `analyze_driver`, `transform_driver`, `ciaopp_log`, `p_dump`, `fixpo_ops`, `acc_ops`, `intermod`, `infer_db`, `infer_dom`, `analysis_stats`, `preprocess_flags`, `static_abs_exec_table`, `abs_exec`.
- *System library modules:*
  `datafacts_rt`, `menu_generator`, `menu_rt`, `assrt_lib`, `p_unit`, `p_unit_db`, `lists`, `aggregates`, `system`, `messages`, `prompt`, `pathnames`, `port_reify`, `llists`, `sort`.
- *Internal (engine) modules:*
  `term_basic`, `arithmetic`, `atomic_basic`, `basiccontrol`, `exceptions`, `term_compare`, `term_typing`, `debugger_support`, `basic_props`, `hiord_rt`, `io_basic`.
- *Packages:*
  `prelude`, `initial`, `condcomp`, `assertions`, `assertions/assertions_basic`, `fsyntax`, `dcg`, `datafacts`, `ciaopp(ciaopp_options)`, `isomodes`, `menu`, `argnames`.

5 The CiaoPP low-level interface

Author(s): The Ciao Development Team.

This module includes the low-level predicates for interacting with CiaoPP. In this interaction model the user performs a sequence of commands to obtain a certain result (e.g., load program, analyze, check assertions).

The basic commands are:

- `module(File)`: loads a (main) module into the preprocessor.
- `analyze(A)`: perform analysis `A` (see Chapter 7 [Available abstract domains], page 31) on the loaded module.
- `acheck`: check assertions (using current analysis information).
- `transform(T)`: perform transformation `T` on the loaded module.
- `output`: output a file with the current program state (i.e., the output includes transformations, analysis info, assertion checking, etc. as controlled by the flags set and the actions performed).
- `output(File)`: same as `output/0` but output to `File`.

The analyses and transformations are controlled by preprocessor flags. These flags can be modified or consulted with:

- `current_pp_flag(F, V)`: consult the current value `V` of flag `F`.
- `set_pp_flag(F, V)`: change the value of flag `F` to `V`.
- `pp_flag(F)`: `F` is a preprocessor flag.
- `dump_flags(K)`: dump the set of flags identified by `K` (`all` for all).

Other commands useful when developing or debugging CiaoPP:

- `show_dump(File)`: displays the content of a .dump file.
- `show_types`: display all current types (inferred or read).

5.1 Usage and interface

- **Library usage:**
  `:- use_module(ciaopp(ciaopp)).`

5.2 Documentation on exports

current_pp_flag/2: PREDICATE

Usage: `current_pp_flag(Name,Value)`

Preprocess flag `Name` has the value `Value`.

- *The following properties should hold at call time:*
  `Name` is a valid preprocessor flag. (`pp_flag/1`)
- *The following properties should hold upon exit:*
  `Value` is a valid value for preprocessor flag `Name`. (`valid_flag_value/2`)

current_pp_flag/2: PREDICATE

Usage: `current_pp_flag(Name,Value)`

Preprocess flag `Name` has the value `Value`.

- *The following properties should hold at call time:*
  `Name` is a valid preprocessor flag. (`pp_flag/1`)
- *The following properties should hold upon exit:*
  `Value` is a valid value for preprocessor flag `Name`. (`valid_flag_value/2`)

set_pp_flag/2: PREDICATE

Usage: `set_pp_flag(Name,Value)`

Sets `Value` for preprocessor flag `Name`.

- *The following properties should hold at call time:*
  `Name` is a valid preprocessor flag. (`pp_flag/1`)
  `Value` is a valid value for preprocessor flag `Name`. (`valid_flag_value/2`)

push_pp_flag/2: PREDICATE

Usage: `push_pp_flag(Flag,Value)`

Sets `Value` for preprocessor flag `Flag`, storing the current value to restore it with `pop_pp_flag/1`.

- *The following properties should hold at call time:*
  `Flag` is a valid preprocessor flag. (`pp_flag/1`)
  `Value` is a valid value for preprocessor flag `Flag`. (`valid_flag_value/2`)

pop_pp_flag/1: PREDICATE

Usage: `pop_pp_flag(Flag)`

Restores the value of the preprocessor flag `Flag` previous to the last non-canceled `push_pp_flag/2` on it.

- *The following properties should hold at call time:*
  `Flag` is a valid preprocessor flag. (`pp_flag/1`)

pp_flag/1: PREDICATE

Valid flags:

(This list is outdated – see source code for a complete description)

- for the output:
  - `analysis_info` (`off, on`) Whether to output the results of analysis.
  - `point_info` (`off,on`) Whether to output analysis information for program points within clauses.
  - `collapse_ai_vers` (`off, on`) to output all the versions of call/success patterns inferred by analysis or just one version (summing-up all of them).

- `type_output` (`defined, all`) to output the types inferred for predicates in terms only of types defined by the user or including types inferred anew.
- `simplify_checks` (`on, off`) to output simplified check assertions after verification or not.

- for analysis:
  - `fixpoint` (`plai, dd, di, check_di, check_di2, check_di3, check_di4`) The kind of fixpoint computation used.
  - `multi_success` (`off, on`) Whether to allow success multivariance.
  - `widen` (`off, on`) Whether to perform widening.
  - `intermod` (`off, on, auto`) The policy for inter-modular analysis.
  - `success_policy` (`best, first, all, top, botfirst, botbest, botall, bottom`) The policy for obtaining success information for imported predicates during inter-modular analysis.
  - `entry_policy` (`all, top_level, force, force_assrt`) The policy for obtaining entry call patterns for exported predicates during inter-modular analysis.
  - `process_libraries` (`on, off, no_engine`) Whether to perform the analysis of Ciao system libraries when a modular user program is analyzed.
  - `initial_guess` (`botfirst, botbest, botall, bottom`) The policy for obtaining initial guess when computing the analysis of a predicate from the current module.
  - `use_check_assrt` (`off, on`) Whether to use check assertions for imported predicates as if they were trust assertions.
  - `depth` (a non-negative integer) The maximum depth of abstractions in analyses based on term depth.
  - `type_eval` (`on, off`) Whether to attempt concrete evaluation of types being inferred.
  - `type_precision` (`defined, all`) to use during type analysis only types defined by the user or also types inferred anew.
  - `entry_points_auto` (`none, calls, all`) What kind of assertions to use as entry points: `entry/1` will always be used, `calls/1` and `calls/2` will be used if the flag is **not** set to `none`; and `success/1`, `success/2`, `comp/1` and `comp/2` will be used if the flag is set to `all`.
  - `entry_calls_scope` (`exported, all`) What non-`entry` assertions to use as entry points. If the flag takes the value `exported`, only assertions from exported predicates will be used; if it takes the value `all`, exported and non-exported assertions will be used.
- for partial evaluation:
  - `global_control` (`off, id, inst, hom_emb`) The abstraction function to use to control the creation of new patterns to analyze as a result of unfolding.
  - `comp_rule` (`leftmost, safe_jb, bind_ins_jb, no_sideff_jb, jump_builtin, eval_builtin, local_emb`) The computation rule for the selection of atoms in a goal.
  - `local_control` (`off, orig, inst, det, det_la, depth, first_sol, first_sol_d, all_sol, hom_emb, hom_emb_anc, hom_emb_as, df_hom_emb_as, df_tree_hom_emb, df_hom_emb`) The unfolding rule to use during partial evaluation.
  - `unf_depth` (a non-negative integer) The depth limit for unfolding.
  - `rem_use_cls` (`off, pre, post, both`) Whether to remove useless clauses.

- abs_spec_defs (off, rem, exec, all) Whether to exploit abstract substitutions while obtaining specialized definitions on unfolding.
- filter_nums (off, on) Whether to filter away numbers in partial evaluation.
- exec_unif (off, on) Whether to execute unifications during specialization time or not.
- pres_inf_fail (off, on) Whether infinite failure should be preserved in the specialized program.
- part_concrete (off, mono, multi) The kind of partial concretization to be performed.
- for parallelization and granularity control:
  - granularity_threshold (a non-negative integer) The threshold on computational cost at which parallel execution pays off.

flag_value/1: REGTYPE

Usage: `flag_value(V)`

`V` is a value for a flag.

valid_flag_value/2: PROPERTY

Usage: `valid_flag_value(Name,Value)`

`Value` is a valid value for preprocessor flag `Name`.

– *If the following properties should hold at call time:*

`Name` is a valid preprocessor flag. (`pp_flag/1`)

`Value` is a value for a flag. (`flag_value/1`)

is_library/1: (UNDOC_REEXPORT)

Imported from `frontend_driver` (see the corresponding documentation for details).

get_data/2: (UNDOC_REEXPORT)

Imported from `frontend_driver` (see the corresponding documentation for details).

clean_ciaopp_db/1: (UNDOC_REEXPORT)

Imported from `frontend_driver` (see the corresponding documentation for details).

clean_all_ciaopp_db/0: (UNDOC_REEXPORT)

Imported from `frontend_driver` (see the corresponding documentation for details).

dump/2: (UNDOC_REEXPORT)

Imported from `frontend_driver` (see the corresponding documentation for details).

dump_dir/1: (UNDOC_REEXPORT)
Imported from `frontend_driver` (see the corresponding documentation for details).

add_to_db/1: (UNDOC_REEXPORT)
Imported from `frontend_driver` (see the corresponding documentation for details).

show_dump/1: (UNDOC_REEXPORT)
Imported from `frontend_driver` (see the corresponding documentation for details).

restore/2: (UNDOC_REEXPORT)
Imported from `frontend_driver` (see the corresponding documentation for details).

restore/1: (UNDOC_REEXPORT)
Imported from `frontend_driver` (see the corresponding documentation for details).

dump/1: (UNDOC_REEXPORT)
Imported from `frontend_driver` (see the corresponding documentation for details).

gen_and_load_libcache/0: (UNDOC_REEXPORT)
Imported from `frontend_driver` (see the corresponding documentation for details).

ensure_libcache_loaded/0: (UNDOC_REEXPORT)
Imported from `frontend_driver` (see the corresponding documentation for details).

write_one_type/2: (UNDOC_REEXPORT)
Imported from `frontend_driver` (see the corresponding documentation for details).

output_by_ext/2: (UNDOC_REEXPORT)
Imported from `frontend_driver` (see the corresponding documentation for details).

output_ext/1: (UNDOC_REEXPORT)
Imported from `frontend_driver` (see the corresponding documentation for details).

output/2: (UNDOC_REEXPORT)
Imported from `frontend_driver` (see the corresponding documentation for details).

output/1: PREDICATE

Usage: `output(File)`

Outputs to `File` the current module preprocessing state with the default output options. If unbound, `File` is unified with a default name (encoding the preprocessing steps). Equivalent to `output(File,[])`.

output/0: PREDICATE

Usage:

Outputs the current module preprocessing state with the default output options. Equivalent to `output(File,[])`.

get_output_path/2: (UNDOC_REEXPORT)

Imported from `frontend_driver` (see the corresponding documentation for details).

pop_history/1: (UNDOC_REEXPORT)

Imported from `frontend_driver` (see the corresponding documentation for details).

push_history/1: (UNDOC_REEXPORT)

Imported from `frontend_driver` (see the corresponding documentation for details).

cleanup_history/0: (UNDOC_REEXPORT)

Imported from `frontend_driver` (see the corresponding documentation for details).

is_library/1: (UNDOC_REEXPORT)

Imported from `frontend_driver` (see the corresponding documentation for details).

assrt_used_as_entry/2: (UNDOC_REEXPORT)

Imported from `frontend_driver` (see the corresponding documentation for details).

entry_assertion/3: (UNDOC_REEXPORT)

Imported from `frontend_driver` (see the corresponding documentation for details).

module/1: PREDICATE

Usage 1: `module(FileName)`

Reads the code of `FileName` and its preprocessing unit, and sets it as the current module.

- *The following properties should hold at call time:*
  `FileName` is currently a term which is not a free variable. (`nonvar/1`)
  `FileName` is a source name. (`sourcename/1`)

Usage 2: `module(FileNameList)`

Reads the code of the list of file names `FileNameList` (and their preprocessing units), and sets them as the current modules.

– *The following properties should hold at call time:*

`FileNameList` is currently a term which is not a free variable. (`nonvar/1`)

`FileNameList` is a list of `atm`s. (`list/2`)

language_output_extension/2: (UNDOC_REEXPORT)

Imported from `frontend_driver` (see the corresponding documentation for details).

transform/1: PREDICATE

Usage 1: `transform(Trans)`

Returns on backtracking all available program transformation identifiers.

– *The following properties should hold at call time:*

`Trans` is a free variable. (`var/1`)

– *The following properties should hold upon exit:*

`Trans` is a valid transformation identifier. (`transformation/1`)

Usage 2: `transform(Trans)`

Performs transformation `Trans` on the current module.

– *The following properties should hold at call time:*

`Trans` is currently a term which is not a free variable. (`nonvar/1`)

`Trans` is a valid transformation identifier. (`transformation/1`)

cleanup_for_codegen/0: (UNDOC_REEXPORT)

Imported from `analyze_driver` (see the corresponding documentation for details).

clean_analysis_info0/0: (UNDOC_REEXPORT)

Imported from `analyze_driver` (see the corresponding documentation for details).

acheck/2: (UNDOC_REEXPORT)

Imported from `analyze_driver` (see the corresponding documentation for details).

acheck/0: PREDICATE

Usage:

Checks assertions w.r.t. analysis information, obtains from `domain/1` which analyses were run.

get_assert_count/1: (UNDOC_REEXPORT)

Imported from `analyze_driver` (see the corresponding documentation for details).

acheck_summary/2: (UNDOC_REEXPORT)
Imported from `analyze_driver` (see the corresponding documentation for details).

acheck_summary/1: (UNDOC_REEXPORT)
Imported from `analyze_driver` (see the corresponding documentation for details).

ctcheck_sum/1: (UNDOC_REEXPORT)
Imported from `analyze_driver` (see the corresponding documentation for details).

analyze1/2: (UNDOC_REEXPORT)
Imported from `analyze_driver` (see the corresponding documentation for details).

analyze/1: PREDICATE
Usage: `analyze(Analysis)`

Analyzes the current module with `Analysis`. If the intermod flag is not off, this predicate may call `module/1`.

- *The following properties should hold at call time:*
  `Analysis` is currently a term which is not a free variable. (`nonvar/1`)
  `Analysis` is a valid analysis identifier. (`analysis/1`)
- *The following properties should hold globally:*
  All the calls of the form `analyze(Analysis)` do not fail. (`not_fails/1`)
  A call to `analyze(Analysis)` does not leave new choicepoints. (`no_choicepoints/1`)

last_domain_used/1: (UNDOC_REEXPORT)
Imported from `analyze_driver` (see the corresponding documentation for details).

assert_domain/1: (UNDOC_REEXPORT)
Imported from `analyze_driver` (see the corresponding documentation for details).

menu_branch/4: (UNDOC_REEXPORT)
Imported from `auto_interface` (see the corresponding documentation for details).

menu_branch/3: (UNDOC_REEXPORT)
Imported from `auto_interface` (see the corresponding documentation for details).

true/2: (UNDOC_REEXPORT)
Imported from `auto_interface` (see the corresponding documentation for details).

true1/1: (UNDOC_REEXPORT)
Imported from `auto_interface` (see the corresponding documentation for details).

functor1/2: (UNDOC_REEXPORT)
Imported from `auto_interface` (see the corresponding documentation for details).

customize_and_preprocess/0: (UNDOC_REEXPORT)
Imported from `auto_interface` (see the corresponding documentation for details).

customize/0: (UNDOC_REEXPORT)
Imported from `auto_interface` (see the corresponding documentation for details).

5.3 Documentation on multifiles

typeslib_flag/1: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

typeslib_is_user_type/1: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

typeslib_is_visible/2: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

typeslib_interesting_type/2: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

5.4 Documentation on internals

transformation/1: PROPERTY
The current supported transformations are in Chapter 44 [Available program transformations], page 311.

Usage: `transformation(Transformation)`

`Transformation` is a valid transformation identifier.

analysis/1: PROPERTY
The current supported analyses are in Chapter 7 [Available abstract domains], page 31.

Usage: `analysis(Analysis)`

`Analysis` is a valid analysis identifier.

5.5 Documentation on imports

This module has the following direct dependencies:

- *Application modules:*
  frontend_driver, transform_driver, analyze_driver, auto_interface, preprocess_flags.
- *System library modules:*
  typeslib, p_unit_db, messages, system.
- *Internal (engine) modules:*
  term_basic, arithmetic, atomic_basic, basiccontrol, exceptions, term_compare, term_typing, debugger_support, basic_props, runtime_control.
- *Packages:*
  prelude, initial, condcomp, assertions, assertions/assertions_basic, ciaopp(ciaopp_options).

6 The CiaoPP command-line interface

Author(s): The Ciao Development Team.

This is the top-level and command-line interface to CiaoPP. Please look at `ciaopp` documentation for top-level usage information. The command-line interface allows the use of the system in batch mode, using arguments for setting preprocessor flags and performing actions.

6.1 Command-line options

This interface can be used by means of the following command-line options:

```
Usage 1: (batch mode)
    ciaopp [OPTIONS] Action Filename [FlagsValues]

  Where:
    -o <OutFile>
            after processing Filename, the resulting source
            code is written to OutFile.  If this option is
            omitted, the output is written to a file
            automatically named depending on the actions
            performed.
    -op <Suffix>
            Use Suffix as the optional input code suffix

    --cwd Dir
            Switch to the selected directory
    --timeout Timeout
            Execute with a timeout limit (ms).
            Default is 0 (no timeout).

    Action must be one of the following:
    -Q      runs the interactive (text-based) menu for
            preprocessing Filename.
    -A      analyzes Filename with the default options
            except the flag values set with -f at the
            command line.
    -O      optimizes Filename with the default options
            except the flag values set with -f at the
            command line.
    -V      verifies the assertions of Filename with
            the default options except the flag values set
            with -f at the command line.

    -U <Config>
            processes Filename with the options set in the
            configuration Config.

    FlagsValues is a list of options -fFlagName=FlagValue
    separated by blank spaces, where FlagName is a valid
    CiaoPP flag name.  This list is optional, and does not need
    to include all flags applicable to the action to be performed:
    the flags not included in this list will be assumed to take
```

```
        their default value.  Examples:

        -flocal_control=on
                where local_control is expected to be
                a CiaoPP flag;
        -f local_control=on
                same as above, with additional blank spaces

        Internal flags can also be changed using -pIntFlagName=Value.

    Usage 2: (top-level mode)
        ciaopp -T <toplevel-opts>

        -T     starts a CiaoPP top-level shell (using <toplevel-opts> as
               options for the toplevel).  Any of the predicates described
               in the Section 'CiaoPP User Menu Interface' of the CiaoPP
               Reference Manual can be used in this top-level.

    Usage 3: cache libraries
        ciaopp --gen-lib-cache

        Preloads libraries for faster load in CiaoPP toplevel

    Execution Examples:

      ciaopp -Q myfile.pl
      ciaopp -o myfile_checked.pl -V myfile.pl
      ciaopp -O myfile.pl
      ciaopp -A myfile.pl -ftypes=terms -f modes=pd
      ciaopp -T
```

6.2 Description of the execution examples

- The following command will prompt the user with the options needed to preprocess `myfile.pl`:

  `ciaopp -Q myfile.pl`
- If we want to verify the assertions of `myfile.pl`, and generate the resulting source code that will the new status of the assertions (either `checked`, if CiaoPP has proved that the assertion holds, or `false` if it has falsified the assertion), the command line is as follows:

  `ciaopp -o myfile_checked.pl -V myfile.pl`
- To optimize `myfile.pl`, and write the optimize code in a file named automatically (e.g., `myfile_pd_codegen_af_co.pl`), the following command line must be used:

  `ciaopp -O myfile.pl`
- If the default flag values need to be changed, the `-f` option can be used. For example, in order to analyze `myfile.pl` to change the types analysis domain to `terms` instead of the default one, and the mode-aliasing domain to `pd`, the command line to use should be:

  `ciaopp -A myfile.pl -ftypes=terms -f modes=pd`
- Finally, the following command line can be used to start a top-level CiaoPP shell (this is the default behavior):

```
ciaopp -T
```

7 Available abstract domains

7.0.1 Groundness and sharing

- `gr` tracks groundness in a very simple way (see Chapter 16 [Simple groundness abstract domain], page 103).
- `def` tracks groundness dependencies, which improves the accuracy in inferring groundness.
- `share` tracks sharing among (sets of) variables [MH92], which gives a very accurate groundness inference, plus information on dependencies caused by unification.
- `son` tracks sharing among pairs of variables, plus variables which are linear (see [Son86]).
- `shareson` is a combination of the above two [CMB93], which may improve on the accuracy of any of them alone.
- `shfr` tracks sharing and variables which are free (see [MH91]).
- `shfrson` is a combination of `shfr` and `son`.
- `shfrnv` augments `shfr` with knowledge on variables which are not free nor ground.

7.0.2 Term structure

- `depth` tracks the structure of the terms bound to the program variables during execution, up to a certain depth; the depth is fixed with the `depth` flag.
- `path` tracks sharing among variables which occur within the terms bound to the program variables during execution; the occurrence of run-time variables within terms is tracked up to a certain depth, fixed with the `depth` flag.
- `aeq` tracks the structure of the terms bound to the program variables during execution plus the sharing among the run-time variables occurring in such terms, plus freeness and linearity. The depth of terms being tracked is set with the `depth` flag. Sharing can be selected between set-sharing or pair-sharing.

7.0.3 Types

Type analysis supports different degrees of precision. For example, with the flag `type_precision` with value `defined`, the analysis restricts the types to the finite domain of predefined types, i.e., the types defined by the user or in libraries, without generating new types. Another alternative is to use the normal analysis, i.e., creating new type definitions, but having only predefined types in the output. This is handled through the `type_output` flag.

- `eterms` performs structural widening (see [VB02]).

  Greater precision can be obtained evaluating builtins like `is/2` abstractly: `eterms` includes a variant which allows evaluation of the types, which is governed by the `type_eval` flag.
- `ptypes` uses the topological clash widening operator (see [VHCL95]).
- `svterms` implements the rigid types domain of [JB92].
- `terms` uses shortening as the widening operator (see [GdW94]), in several fashions, which are selected via the `depth` flag; depth 0 meaning the use of restricted shortening [SG94].

7.0.4 Partial evaluation

Partial evaluation is performed during analysis when the `local_control` flag is set to other than `off`. Flag `fixpoint` must be set to `di`. Unfolding will take place while analyzing the program, therefore creating new patterns to analyze. The unfolding rule is governed by flag `local_control` (see `transformation(codegen)`).

For partial evaluation to take place, an analysis domain capable of tracking term structure should be used (e.g., `eterms`, `pd`, etc.). In particular:

- `pd` allows to perform traditional partial evaluation but using instead abstract interpretation with specialized definitions [PAH04].
- `pdb` improves the precision of `pd` by detecting calls which cannot succeed, i.e., either loop or fail.

Note that these two analyses will not infer useful information on the program. They are intended only to enable (classical) partial evaluation.

7.0.5 Constraint domains

- `fr` [Dum94] determines variables which are not constraint to particular values in the constraint store in which they occur, and also keeps track of possible dependencies between program variables.
- `frdef` is a combination of `fr` and `def`, determining at the same time variables which are not constraint to particular values and variables which are constraint to a definite value.
- `lsign` [MS94] infers the signs of variables involved in linear constraints (and the possible number and form of such constraints).
- `difflsign` is a simplified variant of `lsign`.

7.0.6 Properties of the computation

- `det` detects procedures and goals that are deterministic (i.e. that produce at most one solution), or predicates whose clause tests are mutually exclusive (which implies that at most one of their clauses will succeed) even if they are not deterministic (because they call other predicates that can produce more than one solution).
- `nfg` detects procedures that can be guaranteed not to fail (i.e., to produce at least one solution or not to terminate). It is a mono-variant non-failure analysis, in the sense that it infers non-failure information for only a call pattern per predicate [DLGH97].
- `nf` detects procedures *and goals* that can be guaranteed not to fail and is able to infer separate non-failure information for different call patterns [BLGH04].
- `seff` marks predicates as having side-effects or not.

7.0.7 Size of terms

Size analysis yields functions which give bounds on the size of output data of procedures as a function of the size of the input data. The size can be expressed in various measures, e.g., term-size, term-depth, list-length, integer-value, etc.

- `size_ub` infers upper bounds on the size of terms.
- `size_lb` infers lower bounds on the size of terms.
- `size_ualb` infers both upper and lower bounds on the size of terms.
- `size_o` gives (worst case) complexity orders for term size functions (i.e. big O).

7.0.8 Number of resolution steps of the computation

Cost (steps) analysis yields functions which give bounds on the cost (expressed in the number of resolution steps) of procedures as a function of the size of their input data.

- `steps_ub` infers upper bounds on the number of resolution steps. Incorporates a modified version of the CASLOG [DL93] system, so that CiaoPP analyzers are used to supply automatically the information about modes, types, and size measures needed by the CASLOG system.
- `steps_lb` infers lower bounds on the number of resolution steps. Implements the analysis described in [DLGHL97].
- `steps_ualb` infers both upper and lower bounds on the number of resolution steps.
- `steps_o` gives (worst case) complexity orders for cost functions (i.e. big O).

7.0.9 Execution time of the computation

- `time_ap` yields functions which give approximations on the execution time (expressed in milliseconds) of procedures as a function of the size of their input data.

8 aeq (library)

8.1 Usage and interface

- **Library usage:**
  `:- use_module(domain(aeq)).`

8.2 Documentation on multifiles

aidomain/1: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.init_abstract_domain/2: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.amgu/5: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.augment_asub/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.augment_two_asub/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.call_to_entry/10: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.exit_to_prime/8: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.project/6: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.widencall/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.needs/2: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.widen/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.compute_lub/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.compute_clauses_lub/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.compute_clauses_glb/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.fixpoint_covered/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.fixpoint_covered_gfp/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.identical_abstract/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.abs_sort/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.extend/6: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.less_or_equal/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.glb/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.eliminate_equivalent/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.abs_subset/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.call_to_success_fact/10: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.special_builtin/6: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.combined_special_builtin0/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.body_succ_builtin/9: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.split_combined_domain/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.success_builtin/7: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.call_to_success_builtin/7: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.obtain_info/5: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.input_interface/5: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.input_user_interface/6: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.asub_to_native/6: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.concrete/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.unknown_call/5: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.unknown_entry/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.empty_entry/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.part_conc/5: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.multi_part_conc/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.collect_auxinfo_asub/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.rename_auxinfo_asub/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.dom_statistics/2: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.contains_parameters/2: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

8.3 Documentation on imports

This module has the following direct dependencies:

- *Application modules:*
  preprocess_flags, domains, s_eqs, sharing, pool, plai_errors, share_aux.
- *System library modules:*
  datafacts_rt, sets, sort, idlists, lists, llists, keys, messages, terms_check, terms_vars, arrays, bitcodesets, write, streams.
- *Internal (engine) modules:*
  term_basic, arithmetic, atomic_basic, basiccontrol, exceptions, term_compare, term_typing, debugger_support, hiord_rt, io_basic.
- *Packages:*
  prelude, initial, condcomp, datafacts, hiord, aidomain.

9 def (library)

9.1 Usage and interface

- **Library usage:**
  `:- use_module(domain(def)).`
- **Exports:**
  - *Predicates:*
    `call_to_entry/9`, `exit_to_prime/7`, `project/5`, `compute_lub/2`, `compute_lub_el/3`, `glb/3`, `abs_sort/2`, `extend/5`, `call_to_success_fact/9`, `special_builtin/5`, `success_builtin/6`, `input_user_interface/5`, `input_interface/4`, `asub_to_native/5`, `unknown_call/4`, `unknown_entry/3`, `empty_entry/3`, `less_or_equal/2`.
  - *Multifiles:*
    `aidomain/1`, `aidom.init_abstract_domain/2`, `aidom.amgu/5`, `aidom.augment_asub/4`, `aidom.augment_two_asub/4`, `aidom.call_to_entry/10`, `aidom.exit_to_prime/8`, `aidom.project/6`, `aidom.widencall/4`, `aidom.needs/2`, `aidom.widen/4`, `aidom.compute_lub/3`, `aidom.compute_clauses_lub/4`, `aidom.compute_clauses_glb/4`, `aidom.fixpoint_covered/3`, `aidom.fixpoint_covered_gfp/3`, `aidom.identical_abstract/3`, `aidom.abs_sort/3`, `aidom.extend/6`, `aidom.less_or_equal/3`, `aidom.glb/4`, `aidom.eliminate_equivalent/3`, `aidom.abs_subset/3`, `aidom.call_to_success_fact/10`, `aidom.special_builtin/6`, `aidom.combined_special_builtin0/3`, `aidom.body_succ_builtin/9`, `aidom.split_combined_domain/4`, `aidom.success_builtin/7`, `aidom.call_to_success_builtin/7`, `aidom.obtain_info/5`, `aidom.input_interface/5`, `aidom.input_user_interface/6`, `aidom.asub_to_native/6`, `aidom.concrete/4`, `aidom.unknown_call/5`, `aidom.unknown_entry/4`, `aidom.empty_entry/4`, `aidom.part_conc/5`, `aidom.multi_part_conc/4`, `aidom.collect_auxinfo_asub/4`, `aidom.rename_auxinfo_asub/4`, `aidom.dom_statistics/2`, `aidom.contains_parameters/2`.

9.2 Documentation on exports

call_to_entry/9: PREDICATE

No further documentation available for this predicate.

exit_to_prime/7: PREDICATE

No further documentation available for this predicate.

project/5: PREDICATE

No further documentation available for this predicate.

compute_lub/2: PREDICATE

No further documentation available for this predicate.

compute_lub_el/3: PREDICATE
No further documentation available for this predicate.

glb/3: PREDICATE
No further documentation available for this predicate.

abs_sort/2: PREDICATE
No further documentation available for this predicate.

extend/5: PREDICATE
No further documentation available for this predicate.

call_to_success_fact/9: PREDICATE
No further documentation available for this predicate.

special_builtin/5: PREDICATE
No further documentation available for this predicate.

success_builtin/6: PREDICATE
No further documentation available for this predicate.

input_user_interface/5: PREDICATE
No further documentation available for this predicate.

input_interface/4: PREDICATE
No further documentation available for this predicate.

asub_to_native/5: PREDICATE
No further documentation available for this predicate.

unknown_call/4: PREDICATE
No further documentation available for this predicate.

unknown_entry/3: PREDICATE
No further documentation available for this predicate.

empty_entry/3: PREDICATE

No further documentation available for this predicate.

less_or_equal/2: PREDICATE

No further documentation available for this predicate.

9.3 Documentation on multifiles

aidomain/1: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.init_abstract_domain/2: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.amgu/5: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.augment_asub/4: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.augment_two_asub/4: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.call_to_entry/10: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.exit_to_prime/8: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.project/6: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.widencall/4: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.needs/2: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.widen/4: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.compute_lub/3: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.compute_clauses_lub/4: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.compute_clauses_glb/4: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.fixpoint_covered/3: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.fixpoint_covered_gfp/3: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.identical_abstract/3: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.abs_sort/3: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.extend/6: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.less_or_equal/3: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.glb/4: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.eliminate_equivalent/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.abs_subset/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.call_to_success_fact/10: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.special_builtin/6: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.combined_special_builtin0/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.body_succ_builtin/9: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.split_combined_domain/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.success_builtin/7: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.call_to_success_builtin/7: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.obtain_info/5: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.input_interface/5: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.input_user_interface/6: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.asub_to_native/6: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.concrete/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.unknown_call/5: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.unknown_entry/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.empty_entry/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.part_conc/5: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.multi_part_conc/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.collect_auxinfo_asub/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.rename_auxinfo_asub/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.dom_statistics/2: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.contains_parameters/2: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

9.4 Documentation on imports

This module has the following direct dependencies:

- *Application modules:*
  preprocess_flags, domains, deftools.
- *System library modules:*
  datafacts_rt, sets, sort, terms_check, terms_vars, p_unit.
- *Internal (engine) modules:*
  term_basic, arithmetic, atomic_basic, basiccontrol, exceptions, term_compare, term_typing, debugger_support, basic_props.
- *Packages:*
  prelude, initial, condcomp, assertions, assertions/assertions_basic, datafacts, aidomain.

10 deftypes: defined types (based on termsd) (abstract domain)

Author(s): Pawel Pietrzak (original), Ciao Development Team.

This module implements types abstract domain (based on `termsd`) that only includes library types, user-defined types and instances of parametric rules. Like in `termsd`, an abstract sustitution is list of Var:Type elements, where Var is a variable and Type is a pure type term [DZ92].

10.1 Usage and interface

- **Library usage:**
  `:- use_module(domain(deftypes)).`

10.2 Documentation on multifiles

aidomain/1: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.init_abstract_domain/2: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.amgu/5: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.augment_asub/4: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.augment_two_asub/4: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.call_to_entry/10: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.exit_to_prime/8: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.project/6: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.widencall/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.needs/2: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.widen/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.compute_lub/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.compute_clauses_lub/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.compute_clauses_glb/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.fixpoint_covered/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.fixpoint_covered_gfp/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.identical_abstract/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.abs_sort/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.extend/6: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.less_or_equal/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.glb/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.eliminate_equivalent/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.abs_subset/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.call_to_success_fact/10: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.special_builtin/6: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.combined_special_builtin0/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.body_succ_builtin/9: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.split_combined_domain/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.success_builtin/7: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.call_to_success_builtin/7: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.obtain_info/5: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.input_interface/5: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.input_user_interface/6: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.asub_to_native/6: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.concrete/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.unknown_call/5: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.unknown_entry/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.empty_entry/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.part_conc/5: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.multi_part_conc/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.collect_auxinfo_asub/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.rename_auxinfo_asub/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.dom_statistics/2: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.contains_parameters/2: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

10.3 Documentation on imports

This module has the following direct dependencies:

- *Application modules:*
  preprocess_flags, domains, termsd.
- *System library modules:*
  sets, sort, typeslib, terms_check.
- *Internal (engine) modules:*
  term_basic, arithmetic, atomic_basic, basiccontrol, exceptions, term_compare, term_typing, debugger_support, basic_props, hiord_rt.
- *Packages:*
  prelude, initial, condcomp, assertions, assertions/assertions_basic, regtypes, basicmodes, aidomain.

11 depthk (library)

11.1 Usage and interface

- **Library usage:**
  `:- use_module(domain(depthk)).`

11.2 Documentation on multifiles

aidomain/1: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.init_abstract_domain/2: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.amgu/5: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.augment_asub/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.augment_two_asub/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.call_to_entry/10: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.exit_to_prime/8: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.project/6: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.widencall/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.needs/2: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.widen/4: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.compute_lub/3: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.compute_clauses_lub/4: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.compute_clauses_glb/4: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.fixpoint_covered/3: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.fixpoint_covered_gfp/3: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.identical_abstract/3: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.abs_sort/3: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.extend/6: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.less_or_equal/3: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.glb/4: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.eliminate_equivalent/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.abs_subset/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.call_to_success_fact/10: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.special_builtin/6: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.combined_special_builtin0/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.body_succ_builtin/9: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.split_combined_domain/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.success_builtin/7: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.call_to_success_builtin/7: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.obtain_info/5: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.input_interface/5: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.input_user_interface/6: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.asub_to_native/6: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.concrete/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.unknown_call/5: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.unknown_entry/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.empty_entry/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.part_conc/5: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.multi_part_conc/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.collect_auxinfo_asub/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.rename_auxinfo_asub/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.dom_statistics/2: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.contains_parameters/2: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

11.3 Documentation on imports

This module has the following direct dependencies:

- *Application modules:*
  preprocess_flags, domains, s_eqs.
- *System library modules:*
  sets, sort, lists, terms_check, terms_vars, write.
- *Internal (engine) modules:*
  term_basic, arithmetic, atomic_basic, basiccontrol, exceptions, term_compare, term_typing, debugger_support, basic_props, io_basic.
- *Packages:*
  prelude, initial, condcomp, assertions, assertions/assertions_basic, aidomain.

12 eterms: types with lnewiden_el/4 (abstract domain)

Author(s): Claudio Vaucheret, Francisco Bueno, Ciao Development Team.

This module implements the abstract operations of the type domains with widening based on (`lnewiden_el/4`). An abstract sustitution is a list of `Var:(Name,Type)` elements, where `Var` is a variable and `Type` is a pure type term [DZ92].

12.1 Usage and interface

- **Library usage:**
  `:- use_module(domain(eterms)).`
- **Exports:**
  - *Predicates:*
    `get_type/3`, `eterms_init_abstract_domain/1`, `eterms_compute_lub/2`, `eterms_compute_lub_el/3`, `eterms_widencall/3`, `eterms_needs/1`, `eterms_widen/3`, `eterms_call_to_entry/9`, `eterms_exit_to_prime/7`, `replace_names/3`, `normal_asub/2`, `update_names/2`, `generate_subs_exit/4`, `generate_subs/3`, `eterms_project/5`, `eterms_abs_sort/2`, `eterms_extend/5`, `eterms_less_or_equal/2`, `eterms_glb/3`, `eterms_glbnames/3`, `eterms_concrete/3`, `eterms_unknown_entry/3`, `eterms_empty_entry/3`, `eterms_unknown_call/4`, `variables_are_top_type/2`, `eterms_call_to_success_fact/9`, `eterms_special_builtin/5`, `eterms_success_builtin/6`, `keys_same_value/3`, `eterms_arg_call_to_success/9`, `eterms_call_to_success_builtin/6`, `eterms_obtain_info/4`, `eterms_input_user_interface/5`, `eterms_input_interface/4`, `eterms_asub_to_native/5`, `eterms_output_interface/2`, `eterms_collect_auxinfo_asub/3`, `eterms_rename_auxinfo_asub/3`, `eterms_identical_abstract/2`, `getargtypes/6`, `replacetype/4`, `getfunctors/2`, `split_f/4`, `getvalue/2`, `unifytoterm/1`, `abs_eval_arithcomp/4`, `generateSucc0_cond/5`, `abs_eval_arith/3`, `getvalues_comp_cond/3`, `get_list_names_is/2`, `determinate_sel/3`, `replaceintype/5`, `eterms_multi_part_conc/3`, `eterms_part_conc/4`, `obtains_names/4`.
  - *Multifiles:*
    `aidomain/1`, `aidom.init_abstract_domain/2`, `aidom.amgu/5`, `aidom.augment_asub/4`, `aidom.augment_two_asub/4`, `aidom.call_to_entry/10`, `aidom.exit_to_prime/8`, `aidom.project/6`, `aidom.widencall/4`, `aidom.needs/2`, `aidom.widen/4`, `aidom.compute_lub/3`, `aidom.compute_clauses_lub/4`, `aidom.compute_clauses_glb/4`, `aidom.fixpoint_covered/3`, `aidom.fixpoint_covered_gfp/3`, `aidom.identical_abstract/3`, `aidom.abs_sort/3`, `aidom.extend/6`, `aidom.less_or_equal/3`, `aidom.glb/4`, `aidom.eliminate_equivalent/3`, `aidom.abs_subset/3`, `aidom.call_to_success_fact/10`, `aidom.special_builtin/6`, `aidom.combined_special_builtin0/3`, `aidom.body_succ_builtin/9`, `aidom.split_combined_domain/4`, `aidom.success_builtin/7`, `aidom.call_to_success_builtin/7`, `aidom.obtain_info/5`, `aidom.input_interface/5`, `aidom.input_user_interface/6`, `aidom.asub_to_native/6`, `aidom.concrete/4`, `aidom.unknown_call/5`, `aidom.unknown_entry/4`, `aidom.empty_entry/4`, `aidom.part_conc/5`, `aidom.multi_part_conc/4`, `aidom.collect_auxinfo_asub/4`, `aidom.rename_auxinfo_asub/4`, `aidom.dom_statistics/2`, `aidom.contains_parameters/2`.

12.2 Documentation on exports

get_type/3: PREDICATE

No further documentation available for this predicate.

eterms_init_abstract_domain/1: PREDICATE

No further documentation available for this predicate.

eterms_compute_lub/2: PREDICATE

Usage: `eterms_compute_lub(ListASub,Lub)`

It computes the least upper bound of a set of abstract substitutions. For each two abstract substitutions `ASub1` and `ASub2` in `ListASub`, obtaining the lub is foreach X:Type1 in `ASub1` and X:Type2 in `ASub2`, X:TypeUnion is in Lub, and TypeUnion is the deterministic union of Type1 an Type2.

- *The following properties should hold at call time:*
  - `ListASub` is currently a term which is not a free variable. (`nonvar/1`)
  - `Lub` is a free variable. (`var/1`)
  - `ListASub` is a list of `eterms_asub`s. (`list/2`)
- *The following properties should hold upon exit:*
  - `Lub` is an abstract substitution (`eterms_asub/1`)

eterms_compute_lub_el/3: PREDICATE

No further documentation available for this predicate.

eterms_widencall/3: PREDICATE

No further documentation available for this predicate.

eterms_needs/1: PREDICATE

No further documentation available for this predicate.

eterms_widen/3: PREDICATE

Usage: `eterms_widen(Prime0,Prime1,NewPrime)`

Induction step on the abstract substitution of a fixpoint. `Prime0` corresponds to non-recursive and `Prime1` to recursive clauses. `NewPrime` is the result of apply one widening operation to the least upper bound of the formers.

- *The following properties should hold at call time:*
  - `Prime0` is currently a term which is not a free variable. (`nonvar/1`)
  - `Prime1` is currently a term which is not a free variable. (`nonvar/1`)
  - `NewPrime` is a free variable. (`var/1`)
  - `Prime0` is an abstract substitution (`eterms_asub/1`)
  - `Prime1` is an abstract substitution (`eterms_asub/1`)
  - `NewPrime` is any term. (`term/1`)

– *The following properties should hold upon exit:*
`NewPrime` is an abstract substitution (`eterms_asub/1`)

eterms_call_to_entry/9: PREDICATE

Usage: `eterms_call_to_entry(Sv,Sg,Hv,Head,K,Fv,Proj,Entry,ExtraInfo)`

It obtains the abstract substitution `Entry` which results from adding the abstraction of the `Sg` = `Head` to `Proj`, later projecting the resulting substitution onto `Hv`.

This is done as follows:

- If `Sg` and `Head` are identical up to renaming it is just renaming `Proj` and adding the `Fv`
- Otherwise, it will
  - obtain the abstract substitution from unifying `Sg` and `Head` calling "unify_term_and_type_term"
  - add to this abstract substitution the variables in Fv as term type.

The meaning of the variables is

- `Sg` is the subgoal being analysed.
- `Head` is the Head of the clause being analysed.
- `Fv` is a list of free variables in the body of the clause being considered.
- `Proj` is the abstract substitution `Call` projected onto `Sv`.
- `Entry` is the Abstract entry substitution (i.e. the abstract subtitution obtained after the abstract unification of `Sg` and `Head` projected onto `Hv + Fv`).
- `ExtraInfo`is a flag "yes" when no unification is required, i.e., Entry=Proj upto names of vars. and ignoring Fv. It is "no" if unification fails.

– *The following properties should hold at call time:*
`Sv` is currently a term which is not a free variable. (`nonvar/1`)
`Sg` is currently a term which is not a free variable. (`nonvar/1`)
`Hv` is currently a term which is not a free variable. (`nonvar/1`)
`Head` is currently a term which is not a free variable. (`nonvar/1`)
`K` is currently a term which is not a free variable. (`nonvar/1`)
`Fv` is currently a term which is not a free variable. (`nonvar/1`)
`Proj` is currently a term which is not a free variable. (`nonvar/1`)
`Entry` is a free variable. (`var/1`)
`ExtraInfo` is a free variable. (`var/1`)
`Sv` is any term. (`term/1`)
`Sg` is a term which represents a goal, i.e., an atom or a structure. (`cgoal/1`)
`Hv` is a list. (`list/1`)
`Head` is a term which represents a goal, i.e., an atom or a structure. (`cgoal/1`)
`K` is any term. (`term/1`)
`Fv` is a list. (`list/1`)
`Proj` is an abstract substitution (`eterms_asub/1`)
`Entry` is any term. (`term/1`)
`ExtraInfo` is any term. (`term/1`)

– *The following properties should hold upon exit:*
`Entry` is an abstract substitution (`eterms_asub/1`)
`eterms_extrainfo(ExtraInfo)` (`eterms_extrainfo/1`)

eterms_exit_to_prime/7: PREDICATE

Usage: `eterms_exit_to_prime(Sg,Hv,Head,Sv,Exit,ExtraInfo,Prime)`

It computes the prime abstract substitution `Prime`, i.e. the result of going from the abstract substitution over the head variables `Exit`, to the abstract substitution over the variables in the subgoal. It will:

- If `Exit` is '$bottom', `Prime` will be also '$bottom'.
- If `ExtraInfo` = yes (`Head` and `Sg` identical up to renaming) it is just renaming Exit %
- Otherwise, it will:
  - projects `Exit` onto `Hv` obtaining `BPrime`.
  - obtain the abstract substitution from unifying `Sg` and `Head` calling "unify_term_and_type_term"

– *The following properties should hold at call time:*

`Sg` is currently a term which is not a free variable. (`nonvar/1`)
`Hv` is currently a term which is not a free variable. (`nonvar/1`)
`Head` is currently a term which is not a free variable. (`nonvar/1`)
`Sv` is currently a term which is not a free variable. (`nonvar/1`)
`Exit` is currently a term which is not a free variable. (`nonvar/1`)
`ExtraInfo` is a free variable. (`var/1`)
`Prime` is a free variable. (`var/1`)
`Sg` is a list. (`list/1`)
`Hv` is a list. (`list/1`)
`Head` is a term which represents a goal, i.e., an atom or a structure. (`cgoal/1`)
`Sv` is a term which represents a goal, i.e., an atom or a structure. (`cgoal/1`)
`Exit` is an abstract substitution (`eterms_asub/1`)
`ExtraInfo` is any term. (`term/1`)
`Prime` is any term. (`term/1`)

– *The following properties should hold upon exit:*

`eterms_extrainfo(ExtraInfo)` (`eterms_extrainfo/1`)
`Prime` is an abstract substitution (`eterms_asub/1`)

replace_names/3: PREDICATE

No further documentation available for this predicate.

normal_asub/2: PREDICATE

No further documentation available for this predicate.

update_names/2: PREDICATE

No further documentation available for this predicate.

generate_subs_exit/4: PREDICATE

No further documentation available for this predicate.

generate_subs/3: PREDICATE

No further documentation available for this predicate.

eterms_project/5: PREDICATE

Usage: `eterms_project(Sg,Vars,HvFv_u,ASub,Proj)`

`Proj` is the result of eliminating from `ASub` all `X:Value` such that `X` is not in `Vars`.

- *The following properties should hold at call time:*
  - `Sg` is currently a term which is not a free variable. (`nonvar/1`)
  - `Vars` is currently a term which is not a free variable. (`nonvar/1`)
  - `HvFv_u` is currently a term which is not a free variable. (`nonvar/1`)
  - `ASub` is currently a term which is not a free variable. (`nonvar/1`)
  - `Proj` is a free variable. (`var/1`)
  - `Sg` is any term. (`term/1`)
  - `Vars` is a list. (`list/1`)
  - `HvFv_u` is a list. (`list/1`)
  - `ASub` is an abstract substitution (`eterms_asub/1`)
  - `Proj` is any term. (`term/1`)
- *The following properties should hold upon exit:*
  - `Proj` is an abstract substitution (`eterms_asub/1`)

eterms_abs_sort/2: PREDICATE

Usage: `eterms_abs_sort(ASub,ASub_s)`

It sorts the set of `X:Type` in `ASub` ontaining `ASub_s`

- *The following properties should hold at call time:*
  - `ASub` is currently a term which is not a free variable. (`nonvar/1`)
  - `ASub_s` is a free variable. (`var/1`)
  - `ASub` is an abstract substitution (`eterms_asub/1`)
- *The following properties should hold upon exit:*
  - `ASub_s` is an abstract substitution (`eterms_asub/1`)

eterms_extend/5: PREDICATE

Usage: `eterms_extend(Sg,Prime,Sv,Call,Succ)`

If `Prime` = '$bottom', `Succ` = '$bottom' otherwise, `Succ` is computed updating the values of `Call` with those in `Prime`

- *The following properties should hold at call time:*
  - `Sg` is currently a term which is not a free variable. (`nonvar/1`)
  - `Prime` is currently a term which is not a free variable. (`nonvar/1`)
  - `Sv` is currently a term which is not a free variable. (`nonvar/1`)
  - `Call` is currently a term which is not a free variable. (`nonvar/1`)
  - `Succ` is a free variable. (`var/1`)
  - `Sg` is any term. (`term/1`)

Prime is an abstract substitution (eterms_asub/1)
Sv is a list. (list/1)
Call is an abstract substitution (eterms_asub/1)
Succ is any term. (term/1)
- *The following properties should hold upon exit:*
Succ is an abstract substitution (eterms_asub/1)

eterms_less_or_equal/2: PREDICATE

Usage: eterms_less_or_equal(ASub0,ASub1)

Succeeds if ASub1 is more general or equal to ASub0. it's assumed the two abstract substitutions are defined on the same variables

- *The following properties should hold at call time:*
ASub0 is currently a term which is not a free variable. (nonvar/1)
ASub1 is currently a term which is not a free variable. (nonvar/1)
ASub0 is an abstract substitution (eterms_asub/1)
ASub1 is an abstract substitution (eterms_asub/1)

eterms_glb/3: PREDICATE

Usage: eterms_glb(ASub0,ASub1,Glb)

Glb is the great lower bound of ASub0 and ASub1

- *The following properties should hold at call time:*
ASub0 is currently a term which is not a free variable. (nonvar/1)
ASub1 is currently a term which is not a free variable. (nonvar/1)
Glb is a free variable. (var/1)
ASub0 is an abstract substitution (eterms_asub/1)
ASub1 is an abstract substitution (eterms_asub/1)
Glb is any term. (term/1)
- *The following properties should hold upon exit:*
Glb is an abstract substitution (eterms_asub/1)

eterms_glbnames/3: PREDICATE

No further documentation available for this predicate.

eterms_concrete/3: PREDICATE

No further documentation available for this predicate.

eterms_unknown_entry/3: PREDICATE

Usage: eterms_unknown_entry(Sg,Qv,Call)

Gives the "top" value for the variables involved in a literal whose definition is not present, and adds this top value to Call. In this domain the top value is X:term forall X in the set of variables

- *The following properties should hold at call time:*
  `Sg` is currently a term which is not a free variable. (`nonvar/1`)
  `Qv` is currently a term which is not a free variable. (`nonvar/1`)
  `Call` is a free variable. (`var/1`)
  `Sg` is a term which represents a goal, i.e., an atom or a structure. (`cgoal/1`)
  `Qv` is a list. (`list/1`)
  `Call` is any term. (`term/1`)
- *The following properties should hold upon exit:*
  `Call` is an abstract substitution (`eterms_asub/1`)

eterms_empty_entry/3: PREDICATE

Usage: `eterms_empty_entry(Sg,Vars,Entry)`

Gives the "empty" value in this domain for a given set of variables `Vars`, resulting in the abstract substitution `Entry`. I.e., obtains the abstraction of a substitution in which all variables `Vars` are unbound: free and unaliased. In this domain the empty value is giving the variable type to each variable

- *The following properties should hold at call time:*
  `Sg` is currently a term which is not a free variable. (`nonvar/1`)
  `Vars` is currently a term which is not a free variable. (`nonvar/1`)
  `Entry` is a free variable. (`var/1`)
  `Sg` is a term which represents a goal, i.e., an atom or a structure. (`cgoal/1`)
  `Vars` is a list. (`list/1`)
  `Entry` is any term. (`term/1`)
- *The following properties should hold upon exit:*
  `Entry` is an abstract substitution (`eterms_asub/1`)

eterms_unknown_call/4: PREDICATE

Usage: `eterms_unknown_call(Sg,Vars,Call,Succ)`

Gives the “top” value for the variables involved in a literal whose definition is not present, and adds this top value to `Call`

- *The following properties should hold at call time:*
  `Sg` is currently a term which is not a free variable. (`nonvar/1`)
  `Vars` is currently a term which is not a free variable. (`nonvar/1`)
  `Call` is currently a term which is not a free variable. (`nonvar/1`)
  `Succ` is a free variable. (`var/1`)
  `Sg` is a term which represents a goal, i.e., an atom or a structure. (`cgoal/1`)
  `Vars` is a list. (`list/1`)
  `Call` is an abstract substitution (`eterms_asub/1`)
  `Succ` is any term. (`term/1`)
- *The following properties should hold upon exit:*
  `Succ` is an abstract substitution (`eterms_asub/1`)

variables_are_top_type/2: PREDICATE

Usage: `variables_are_top_type(Fv,ASub)`

It assigns the value top_type to the variables in `Fv` and return the abstract substitution `ASub`

− *The following properties should hold at call time:*
  - `Fv` is currently a term which is not a free variable. (`nonvar/1`)
  - `ASub` is a free variable. (`var/1`)
  - `Fv` is a list. (`list/1`)

− *The following properties should hold upon exit:*
  - `ASub` is an abstract substitution (`eterms_asub/1`)

eterms_call_to_success_fact/9: PREDICATE

Usage: `eterms_call_to_success_fact(Sg,Hv,Head,K,Sv,Call,Proj,Prime,Succ)`

Specialized version of call_to_entry + exit_to_prime + extend for facts

− *The following properties should hold at call time:*
  - `Sg` is currently a term which is not a free variable. (`nonvar/1`)
  - `Hv` is currently a term which is not a free variable. (`nonvar/1`)
  - `Head` is currently a term which is not a free variable. (`nonvar/1`)
  - `K` is currently a term which is not a free variable. (`nonvar/1`)
  - `Sv` is currently a term which is not a free variable. (`nonvar/1`)
  - `Call` is currently a term which is not a free variable. (`nonvar/1`)
  - `Proj` is currently a term which is not a free variable. (`nonvar/1`)
  - `Prime` is a free variable. (`var/1`)
  - `Succ` is a free variable. (`var/1`)
  - `Sg` is a term which represents a goal, i.e., an atom or a structure. (`cgoal/1`)
  - `Hv` is a list. (`list/1`)
  - `Head` is a term which represents a goal, i.e., an atom or a structure. (`cgoal/1`)
  - `K` is any term. (`term/1`)
  - `Sv` is a list. (`list/1`)
  - `Call` is an abstract substitution (`eterms_asub/1`)
  - `Proj` is an abstract substitution (`eterms_asub/1`)
  - `Prime` is any term. (`term/1`)
  - `Succ` is any term. (`term/1`)

− *The following properties should hold upon exit:*
  - `Prime` is an abstract substitution (`eterms_asub/1`)
  - `Succ` is an abstract substitution (`eterms_asub/1`)

eterms_special_builtin/5: PREDICATE

Usage: `eterms_special_builtin(SgKey,Sg,Subgoal,Type,Condvars)`

`Type` is a flag indicating what is the abstraction of builtin `SgKey` and to which variables `Condvars` of the goal `Sg` it affects.

- *The following properties should hold at call time:*
  - `SgKey` is currently a term which is not a free variable. (`nonvar/1`)
  - `Sg` is currently a term which is not a free variable. (`nonvar/1`)
  - `Subgoal` is currently a term which is not a free variable. (`nonvar/1`)
  - `Type` is a free variable. (`var/1`)
  - `Condvars` is a free variable. (`var/1`)
  - `SgKey` is an atom. (`atm/1`)
  - `Sg` is a term which represents a goal, i.e., an atom or a structure. (`cgoal/1`)
  - `Subgoal` is a term which represents a goal, i.e., an atom or a structure. (`cgoal/1`)

eterms_success_builtin/6: PREDICATE

Usage: `eterms_success_builtin(Type,Sv_uns,Condvars,HvFv_u,Call,Succ)`

Depending on `Type` it computes the abstraction of a builtin affecting variables `Condvars` and having variables `Sv_uns` with call subs. `Call`.

- *The following properties should hold at call time:*
  - `Type` is currently a term which is not a free variable. (`nonvar/1`)
  - `Sv_uns` is currently a term which is not a free variable. (`nonvar/1`)
  - `Condvars` is currently a term which is not a free variable. (`nonvar/1`)
  - `HvFv_u` is currently a term which is not a free variable. (`nonvar/1`)
  - `Call` is currently a term which is not a free variable. (`nonvar/1`)
  - `Succ` is a free variable. (`var/1`)

keys_same_value/3: PREDICATE

No further documentation available for this predicate.

eterms_arg_call_to_success/9: PREDICATE

No further documentation available for this predicate.

eterms_call_to_success_builtin/6: PREDICATE

Usage: `eterms_call_to_success_builtin(SgKey,Sg,Sv,Call,Proj,Succ)`

Same as above but for each particular builtin.

- *The following properties should hold at call time:*
  - `SgKey` is currently a term which is not a free variable. (`nonvar/1`)
  - `Sg` is currently a term which is not a free variable. (`nonvar/1`)
  - `Sv` is currently a term which is not a free variable. (`nonvar/1`)
  - `Call` is currently a term which is not a free variable. (`nonvar/1`)
  - `Proj` is currently a term which is not a free variable. (`nonvar/1`)
  - `Succ` is a free variable. (`var/1`)
  - `SgKey` is an atom. (`atm/1`)
  - `Sg` is a term which represents a goal, i.e., an atom or a structure. (`cgoal/1`)
  - `Sv` is a list. (`list/1`)

Call is an abstract substitution (eterms_asub/1)
Proj is an abstract substitution (eterms_asub/1)
Succ is any term. (term/1)

- *The following properties should hold upon exit:*

Succ is an abstract substitution (eterms_asub/1)

eterms_obtain_info/4: PREDICATE

No further documentation available for this predicate.

eterms_input_user_interface/5: PREDICATE

Usage: eterms_input_user_interface(InputUser,Qv,ASub,Sg,MaybeCallASub)

Obtains the abstract substitution ASub from user supplied information.

- *The following properties should hold at call time:*

InputUser is currently a term which is not a free variable. (nonvar/1)
Qv is currently a term which is not a free variable. (nonvar/1)
ASub is a free variable. (var/1)
Sg is currently a term which is not a free variable. (nonvar/1)
MaybeCallASub is currently a term which is not a free variable. (nonvar/1)

eterms_input_interface/4: PREDICATE

No further documentation available for this predicate.

eterms_asub_to_native/5: PREDICATE

Usage: eterms_asub_to_native(ASub,Qv,OutFlag,OutputUser,Comps)

Transforms abstract substitution ASub to user friendly format.

- *The following properties should hold at call time:*

ASub is currently a term which is not a free variable. (nonvar/1)
Qv is currently a term which is not a free variable. (nonvar/1)
OutFlag is currently a term which is not a free variable. (nonvar/1)
OutputUser is a free variable. (var/1)
Comps is a free variable. (var/1)
ASub is an abstract substitution (eterms_asub/1)
Qv is a list. (list/1)

eterms_output_interface/2: PREDICATE

Usage: eterms_output_interface(ASub,Output)

Transforms abstract substitution ASub to a more readable but still close to internal format.

- *The following properties should hold at call time:*

ASub is currently a term which is not a free variable. (nonvar/1)
Output is a free variable. (var/1)
ASub is an abstract substitution (eterms_asub/1)

eterms_collect_auxinfo_asub/3: PREDICATE
No further documentation available for this predicate.

eterms_rename_auxinfo_asub/3: PREDICATE
No further documentation available for this predicate.

eterms_identical_abstract/2: PREDICATE
No further documentation available for this predicate.

getargtypes/6: PREDICATE
No further documentation available for this predicate.

replacetype/4: PREDICATE
No further documentation available for this predicate.

getfunctors/2: PREDICATE
No further documentation available for this predicate.

split_f/4: PREDICATE
No further documentation available for this predicate.

getvalue/2: PREDICATE
No further documentation available for this predicate.

unifytoterm/1: PREDICATE
No further documentation available for this predicate.

abs_eval_arithcomp/4: PREDICATE
No further documentation available for this predicate.

generateSucc0_cond/5: PREDICATE
No further documentation available for this predicate.

abs_eval_arith/3: PREDICATE
No further documentation available for this predicate.

getvalues_comp_cond/3: PREDICATE
No further documentation available for this predicate.

get_list_names_is/2: PREDICATE
No further documentation available for this predicate.

determinate_sel/3: PREDICATE
No further documentation available for this predicate.

replaceintype/5: PREDICATE
No further documentation available for this predicate.

eterms_multi_part_conc/3: PREDICATE
No further documentation available for this predicate.

eterms_part_conc/4: PREDICATE
No further documentation available for this predicate.

obtains_names/4: PREDICATE
No further documentation available for this predicate.

12.3 Documentation on multifiles

aidomain/1: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.init_abstract_domain/2: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.amgu/5: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.augment_asub/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.augment_two_asub/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.call_to_entry/10: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.exit_to_prime/8: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.project/6: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.widencall/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.needs/2: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.widen/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.compute_lub/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.compute_clauses_lub/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.compute_clauses_glb/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.fixpoint_covered/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.fixpoint_covered_gfp/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.identical_abstract/3: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.abs_sort/3: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.extend/6: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.less_or_equal/3: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.glb/4: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.eliminate_equivalent/3: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.abs_subset/3: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.call_to_success_fact/10: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.special_builtin/6: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.combined_special_builtin0/3: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.body_succ_builtin/9: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.split_combined_domain/4: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.success_builtin/7: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.call_to_success_builtin/7: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.obtain_info/5: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.input_interface/5: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.input_user_interface/6: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.asub_to_native/6: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.concrete/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.unknown_call/5: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.unknown_entry/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.empty_entry/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.part_conc/5: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.multi_part_conc/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.collect_auxinfo_asub/4: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.rename_auxinfo_asub/4: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.dom_statistics/2: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.contains_parameters/2: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

12.4 Documentation on imports

This module has the following direct dependencies:

- *Application modules:*
  `preprocess_flags`, `domains`, `termsd`, `unfold_builtins`, `apply_assertions_old`.
- *System library modules:*
  `sets`, `sort`, `typeslib`, `hiordlib`, `messages`, `aggregates`, `terms_vars`, `terms_check`, `lists`, `assoc`.
- *Internal (engine) modules:*
  `term_basic`, `arithmetic`, `atomic_basic`, `basiccontrol`, `exceptions`, `term_compare`, `term_typing`, `debugger_support`, `basic_props`, `hiord_rt`.
- *Packages:*
  `prelude`, `initial`, `condcomp`, `assertions`, `assertions/assertions_basic`, `regtypes`, `basicmodes`, `hiord`, `aidomain`.

13 etermsvar: eterms with variables (abstract domain)

Author(s): Claudio Vaucheret, Francisco Bueno, Alejandro Serrano (etermsvar), Ciao Development Team.

This module implements an (experimental) variant of the `eterms` domain with `vr` elements in the lattice.

13.1 Usage and interface

- **Library usage:**
  `:- use_module(domain(etermsvar)).`
- **Exports:**
  - *Predicates:*
    `etermsvar_init_abstract_domain/1`, `etermsvar_compute_lub/2`, `etermsvar_compute_lub_el/3`, `etermsvar_widencall/3`, `etermsvar_needs/1`, `etermsvar_widen/3`, `etermsvar_call_to_entry/9`, `etermsvar_exit_to_prime/7`, `etermsvar_project/5`, `etermsvar_abs_sort/2`, `etermsvar_extend/5`, `etermsvar_less_or_equal/2`, `etermsvar_glb/3`, `etermsvar_concrete/3`, `etermsvar_unknown_entry/3`, `etermsvar_empty_entry/3`, `etermsvar_unknown_call/4`, `etermsvar_call_to_success_fact/9`, `etermsvar_special_builtin/5`, `etermsvar_success_builtin/6`, `etermsvar_arg_call_to_success/9`, `etermsvar_call_to_success_builtin/6`, `etermsvar_obtain_info/4`, `etermsvar_input_user_interface/5`, `etermsvar_input_interface/4`, `etermsvar_asub_to_native/5`, `etermsvar_output_interface/2`, `etermsvar_collect_auxinfo_asub/3`, `etermsvar_rename_auxinfo_asub/3`, `etermsvar_identical_abstract/2`, `etermsvar_multi_part_conc/3`, `etermsvar_part_conc/4`.
  - *Multifiles:*
    `aidomain/1`, `aidom.init_abstract_domain/2`, `aidom.amgu/5`, `aidom.augment_asub/4`, `aidom.augment_two_asub/4`, `aidom.call_to_entry/10`, `aidom.exit_to_prime/8`, `aidom.project/6`, `aidom.widencall/4`, `aidom.needs/2`, `aidom.widen/4`, `aidom.compute_lub/3`, `aidom.compute_clauses_lub/4`, `aidom.compute_clauses_glb/4`, `aidom.fixpoint_covered/3`, `aidom.fixpoint_covered_gfp/3`, `aidom.identical_abstract/3`, `aidom.abs_sort/3`, `aidom.extend/6`, `aidom.less_or_equal/3`, `aidom.glb/4`, `aidom.eliminate_equivalent/3`, `aidom.abs_subset/3`, `aidom.call_to_success_fact/10`, `aidom.special_builtin/6`, `aidom.combined_special_builtin0/3`, `aidom.body_succ_builtin/9`, `aidom.split_combined_domain/4`, `aidom.success_builtin/7`, `aidom.call_to_success_builtin/7`, `aidom.obtain_info/5`, `aidom.input_interface/5`, `aidom.input_user_interface/6`, `aidom.asub_to_native/6`, `aidom.concrete/4`, `aidom.unknown_call/5`, `aidom.unknown_entry/4`, `aidom.empty_entry/4`, `aidom.part_conc/5`, `aidom.multi_part_conc/4`, `aidom.collect_auxinfo_asub/4`, `aidom.rename_auxinfo_asub/4`, `aidom.dom_statistics/2`, `aidom.contains_parameters/2`.

13.2 Documentation on exports

etermsvar_init_abstract_domain/1: PREDICATE

No further documentation available for this predicate.

etermsvar_compute_lub/2: PREDICATE

Usage: `etermsvar_compute_lub(ListASub,Lub)`

– *The following properties should hold at call time:*

`ListASub` is currently a term which is not a free variable. (`nonvar/1`)
`Lub` is a free variable. (`var/1`)
`ListASub` is a list of `absu`s. (`list/2`)
`Lub` is an abstract substitution (`absu/1`)

etermsvar_compute_lub_el/3: PREDICATE

No further documentation available for this predicate.

etermsvar_widencall/3: PREDICATE

No further documentation available for this predicate.

etermsvar_needs/1: PREDICATE

No further documentation available for this predicate.

etermsvar_widen/3: PREDICATE

Usage: `etermsvar_widen(Prime0,Prime1,NewPrime)`

– *The following properties should hold at call time:*

`Prime0` is currently a term which is not a free variable. (`nonvar/1`)
`Prime1` is currently a term which is not a free variable. (`nonvar/1`)
`NewPrime` is a free variable. (`var/1`)
`Prime0` is an abstract substitution (`absu/1`)
`Prime1` is an abstract substitution (`absu/1`)
`NewPrime` is an abstract substitution (`absu/1`)

etermsvar_call_to_entry/9: PREDICATE

Usage: `etermsvar_call_to_entry(Sv,Sg,Hv,Head,K,Fv,Proj,Entry,ExtraInfo)`

– *The following properties should hold at call time:*

`Sv` is currently a term which is not a free variable. (`nonvar/1`)
`Sg` is currently a term which is not a free variable. (`nonvar/1`)
`Hv` is currently a term which is not a free variable. (`nonvar/1`)
`Head` is currently a term which is not a free variable. (`nonvar/1`)
`K` is currently a term which is not a free variable. (`nonvar/1`)
`Fv` is currently a term which is not a free variable. (`nonvar/1`)
`Proj` is currently a term which is not a free variable. (`nonvar/1`)
`Entry` is a free variable. (`var/1`)
`ExtraInfo` is a free variable. (`var/1`)
`Sv` is any term. (`term/1`)

Sg is a term which represents a goal, i.e., an atom or a structure. (cgoal/1)
Hv is a list. (list/1)
Head is a term which represents a goal, i.e., an atom or a structure. (cgoal/1)
K is any term. (term/1)
Fv is a list. (list/1)
Proj is an abstract substitution (absu/1)
Entry is an abstract substitution (absu/1)
extrainfo(ExtraInfo) (extrainfo/1)

etermsvar_exit_to_prime/7: PREDICATE

Usage: `etermsvar_exit_to_prime(Sg,Hv,Head,Sv,Exit,ExtraInfo,Prime)`

– *The following properties should hold at call time:*

Sg is currently a term which is not a free variable. (nonvar/1)
Hv is currently a term which is not a free variable. (nonvar/1)
Head is currently a term which is not a free variable. (nonvar/1)
Sv is currently a term which is not a free variable. (nonvar/1)
Exit is currently a term which is not a free variable. (nonvar/1)
ExtraInfo is a free variable. (var/1)
Prime is a free variable. (var/1)
Sg is a list. (list/1)
Hv is a list. (list/1)
Head is a term which represents a goal, i.e., an atom or a structure. (cgoal/1)
Sv is a term which represents a goal, i.e., an atom or a structure. (cgoal/1)
Exit is an abstract substitution (absu/1)
extrainfo(ExtraInfo) (extrainfo/1)
Prime is an abstract substitution (absu/1)

etermsvar_project/5: PREDICATE

Usage: `etermsvar_project(Sg,Vars,HvFv_u,Asub,Proj)`

– *The following properties should hold at call time:*

Sg is currently a term which is not a free variable. (nonvar/1)
Vars is currently a term which is not a free variable. (nonvar/1)
HvFv_u is currently a term which is not a free variable. (nonvar/1)
Asub is currently a term which is not a free variable. (nonvar/1)
Proj is a free variable. (var/1)
Sg is any term. (term/1)
Vars is a list. (list/1)
HvFv_u is a list. (list/1)
Asub is an abstract substitution (absu/1)
Proj is an abstract substitution (absu/1)

etermsvar_abs_sort/2: PREDICATE

Usage: `etermsvar_abs_sort(Asub,Asub_s)`

– *The following properties should hold at call time:*

`Asub` is currently a term which is not a free variable. (`nonvar/1`)
`Asub_s` is a free variable. (`var/1`)
`Asub` is an abstract substitution (`absu/1`)
`Asub_s` is an abstract substitution (`absu/1`)

etermsvar_extend/5: PREDICATE

Usage: `etermsvar_extend(Sg,Prime,Sv,Call,Succ)`

– *The following properties should hold at call time:*

`Sg` is currently a term which is not a free variable. (`nonvar/1`)
`Prime` is currently a term which is not a free variable. (`nonvar/1`)
`Sv` is currently a term which is not a free variable. (`nonvar/1`)
`Call` is currently a term which is not a free variable. (`nonvar/1`)
`Succ` is a free variable. (`var/1`)
`Sg` is any term. (`term/1`)
`Prime` is an abstract substitution (`absu/1`)
`Sv` is a list. (`list/1`)
`Call` is an abstract substitution (`absu/1`)
`Succ` is an abstract substitution (`absu/1`)

etermsvar_less_or_equal/2: PREDICATE

Usage: `etermsvar_less_or_equal(ASub0,ASub1)`

– *The following properties should hold at call time:*

`ASub0` is currently a term which is not a free variable. (`nonvar/1`)
`ASub1` is currently a term which is not a free variable. (`nonvar/1`)
`ASub0` is an abstract substitution (`absu/1`)
`ASub1` is an abstract substitution (`absu/1`)

etermsvar_glb/3: PREDICATE

Usage: `etermsvar_glb(ASub0,ASub1,Glb)`

– *The following properties should hold at call time:*

`ASub0` is currently a term which is not a free variable. (`nonvar/1`)
`ASub1` is currently a term which is not a free variable. (`nonvar/1`)
`Glb` is a free variable. (`var/1`)
`ASub0` is an abstract substitution (`absu/1`)
`ASub1` is an abstract substitution (`absu/1`)
`Glb` is an abstract substitution (`absu/1`)

etermsvar_concrete/3: PREDICATE

No further documentation available for this predicate.

etermsvar_unknown_entry/3: PREDICATE

Usage: `etermsvar_unknown_entry(Sg,Qv,Call)`

– *The following properties should hold at call time:*

`Sg` is currently a term which is not a free variable. (`nonvar/1`)
`Qv` is currently a term which is not a free variable. (`nonvar/1`)
`Call` is a free variable. (`var/1`)
`Sg` is a term which represents a goal, i.e., an atom or a structure. (`cgoal/1`)
`Qv` is a list. (`list/1`)
`Call` is an abstract substitution (`absu/1`)

etermsvar_empty_entry/3: PREDICATE

Usage: `etermsvar_empty_entry(Sg,Vars,Entry)`

– *The following properties should hold at call time:*

`Sg` is currently a term which is not a free variable. (`nonvar/1`)
`Vars` is currently a term which is not a free variable. (`nonvar/1`)
`Entry` is a free variable. (`var/1`)
`Sg` is a term which represents a goal, i.e., an atom or a structure. (`cgoal/1`)
`Vars` is a list. (`list/1`)
`Entry` is an abstract substitution (`absu/1`)

etermsvar_unknown_call/4: PREDICATE

Usage: `etermsvar_unknown_call(Sg,Vars,Call,Succ)`

– *The following properties should hold at call time:*

`Sg` is currently a term which is not a free variable. (`nonvar/1`)
`Vars` is currently a term which is not a free variable. (`nonvar/1`)
`Call` is currently a term which is not a free variable. (`nonvar/1`)
`Succ` is a free variable. (`var/1`)
`Sg` is a term which represents a goal, i.e., an atom or a structure. (`cgoal/1`)
`Vars` is a list. (`list/1`)
`Call` is an abstract substitution (`absu/1`)
`Succ` is an abstract substitution (`absu/1`)

etermsvar_call_to_success_fact/9: PREDICATE

Usage:

`etermsvar_call_to_success_fact(Sg,Hv,Head,K,Sv,Call,Proj,Prime,Succ)`

- *The following properties should hold at call time:*
  - `Sg` is currently a term which is not a free variable. (`nonvar/1`)
  - `Hv` is currently a term which is not a free variable. (`nonvar/1`)
  - `Head` is currently a term which is not a free variable. (`nonvar/1`)
  - `K` is currently a term which is not a free variable. (`nonvar/1`)
  - `Sv` is currently a term which is not a free variable. (`nonvar/1`)
  - `Call` is currently a term which is not a free variable. (`nonvar/1`)
  - `Proj` is currently a term which is not a free variable. (`nonvar/1`)
  - `Prime` is a free variable. (`var/1`)
  - `Succ` is a free variable. (`var/1`)
  - `Sg` is a term which represents a goal, i.e., an atom or a structure. (`cgoal/1`)
  - `Hv` is a list. (`list/1`)
  - `Head` is a term which represents a goal, i.e., an atom or a structure. (`cgoal/1`)
  - `K` is any term. (`term/1`)
  - `Sv` is a list. (`list/1`)
  - `Call` is an abstract substitution (`absu/1`)
  - `Proj` is an abstract substitution (`absu/1`)
  - `Prime` is an abstract substitution (`absu/1`)
  - `Succ` is an abstract substitution (`absu/1`)

etermsvar_special_builtin/5: PREDICATE

Usage: `etermsvar_special_builtin(SgKey,Sg,Subgoal,Type,Condvars)`

- *The following properties should hold at call time:*
  - `SgKey` is currently a term which is not a free variable. (`nonvar/1`)
  - `Sg` is currently a term which is not a free variable. (`nonvar/1`)
  - `Subgoal` is currently a term which is not a free variable. (`nonvar/1`)
  - `Type` is a free variable. (`var/1`)
  - `Condvars` is a free variable. (`var/1`)

etermsvar_success_builtin/6: PREDICATE

Usage: `etermsvar_success_builtin(Type,Sv_uns,Condvars,HvFv_u,Call,Succ)`

- *The following properties should hold at call time:*
  - `Type` is currently a term which is not a free variable. (`nonvar/1`)
  - `Sv_uns` is currently a term which is not a free variable. (`nonvar/1`)
  - `Condvars` is currently a term which is not a free variable. (`nonvar/1`)
  - `HvFv_u` is currently a term which is not a free variable. (`nonvar/1`)
  - `Call` is currently a term which is not a free variable. (`nonvar/1`)
  - `Succ` is a free variable. (`var/1`)

etermsvar_arg_call_to_success/9: PREDICATE

No further documentation available for this predicate.

etermsvar_call_to_success_builtin/6: PREDICATE

Usage: `etermsvar_call_to_success_builtin(SgKey,Sg,Sv,Call,Proj,Succ)`

- *The following properties should hold at call time:*
  - `SgKey` is currently a term which is not a free variable. (`nonvar/1`)
  - `Sg` is currently a term which is not a free variable. (`nonvar/1`)
  - `Sv` is currently a term which is not a free variable. (`nonvar/1`)
  - `Call` is currently a term which is not a free variable. (`nonvar/1`)
  - `Proj` is currently a term which is not a free variable. (`nonvar/1`)
  - `Succ` is a free variable. (`var/1`)

etermsvar_obtain_info/4: PREDICATE

No further documentation available for this predicate.

etermsvar_input_user_interface/5: PREDICATE

Usage: `etermsvar_input_user_interface(InputUser,Qv,ASub,Sg,MaybeCallASub)`

- *The following properties should hold at call time:*
  - `InputUser` is currently a term which is not a free variable. (`nonvar/1`)
  - `Qv` is currently a term which is not a free variable. (`nonvar/1`)
  - `ASub` is a free variable. (`var/1`)
  - `Sg` is currently a term which is not a free variable. (`nonvar/1`)
  - `MaybeCallASub` is currently a term which is not a free variable. (`nonvar/1`)

etermsvar_input_interface/4: PREDICATE

No further documentation available for this predicate.

etermsvar_asub_to_native/5: PREDICATE

No further documentation available for this predicate.

etermsvar_output_interface/2: PREDICATE

Usage: `etermsvar_output_interface(ASub,Output)`

- *The following properties should hold at call time:*
  - `ASub` is currently a term which is not a free variable. (`nonvar/1`)
  - `Output` is a free variable. (`var/1`)

etermsvar_collect_auxinfo_asub/3: PREDICATE

No further documentation available for this predicate.

etermsvar_rename_auxinfo_asub/3: PREDICATE

No further documentation available for this predicate.

etermsvar_identical_abstract/2: PREDICATE
No further documentation available for this predicate.

etermsvar_multi_part_conc/3: PREDICATE
No further documentation available for this predicate.

etermsvar_part_conc/4: PREDICATE
No further documentation available for this predicate.

13.3 Documentation on multifiles

aidomain/1: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.init_abstract_domain/2: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.amgu/5: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.augment_asub/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.augment_two_asub/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.call_to_entry/10: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.exit_to_prime/8: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.project/6: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.widencall/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.needs/2: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.widen/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.compute_lub/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.compute_clauses_lub/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.compute_clauses_glb/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.fixpoint_covered/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.fixpoint_covered_gfp/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.identical_abstract/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.abs_sort/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.extend/6: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.less_or_equal/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.glb/4: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.eliminate_equivalent/3: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.abs_subset/3: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.call_to_success_fact/10: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.special_builtin/6: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.combined_special_builtin0/3: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.body_succ_builtin/9: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.split_combined_domain/4: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.success_builtin/7: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.call_to_success_builtin/7: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.obtain_info/5: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.input_interface/5: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.input_user_interface/6: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.asub_to_native/6: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.concrete/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.unknown_call/5: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.unknown_entry/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.empty_entry/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.part_conc/5: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.multi_part_conc/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.collect_auxinfo_asub/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.rename_auxinfo_asub/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.dom_statistics/2: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.contains_parameters/2: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

13.4 Documentation on imports

This module has the following direct dependencies:

- *Application modules:*

  `preprocess_flags`, `domains`, `apply_assertions_old`, `eterms`, `termsd`.
- *System library modules:*

  `datafacts_rt`, `sets`, `sort`, `typeslib`, `messages`, `aggregates`, `terms_vars`, `terms_check`, `lists`.
- *Internal (engine) modules:*

  `term_basic`, `arithmetic`, `atomic_basic`, `basiccontrol`, `exceptions`, `term_compare`, `term_typing`, `debugger_support`, `basic_props`, `hiord_rt`.
- *Packages:*

  `prelude`, `initial`, `condcomp`, `assertions`, `assertions/assertions_basic`, `regtypes`, `basicmodes`, `datafacts`, `aidomain`.

14 frdef (library)

14.1 Usage and interface

- **Library usage:**
  `:- use_module(domain(frdef)).`
- **Exports:**
  - *Predicates:*
    `frdef_call_to_entry/9`, `frdef_call_to_success_fact/9`, `frdef_compute_lub/2`, `frdef_exit_to_prime/7`, `frdef_extend/5`, `frdef_identical_abstract/2`, `frdef_input_user_interface/5`, `frdef_input_interface/4`, `frdef_less_or_equal/2`, `frdef_glb/3`, `frdef_asub_to_native/5`, `frdef_project/5`, `frdef_abs_sort/2`, `frdef_special_builtin/5`, `frdef_success_builtin/6`, `frdef_unknown_call/4`, `frdef_unknown_entry/3`, `frdef_empty_entry/3`.
  - *Multifiles:*
    `aidomain/1`, `aidom.init_abstract_domain/2`, `aidom.amgu/5`, `aidom.augment_asub/4`, `aidom.augment_two_asub/4`, `aidom.call_to_entry/10`, `aidom.exit_to_prime/8`, `aidom.project/6`, `aidom.widencall/4`, `aidom.needs/2`, `aidom.widen/4`, `aidom.compute_lub/3`, `aidom.compute_clauses_lub/4`, `aidom.compute_clauses_glb/4`, `aidom.fixpoint_covered/3`, `aidom.fixpoint_covered_gfp/3`, `aidom.identical_abstract/3`, `aidom.abs_sort/3`, `aidom.extend/6`, `aidom.less_or_equal/3`, `aidom.glb/4`, `aidom.eliminate_equivalent/3`, `aidom.abs_subset/3`, `aidom.call_to_success_fact/10`, `aidom.special_builtin/6`, `aidom.combined_special_builtin0/3`, `aidom.body_succ_builtin/9`, `aidom.split_combined_domain/4`, `aidom.success_builtin/7`, `aidom.call_to_success_builtin/7`, `aidom.obtain_info/5`, `aidom.input_interface/5`, `aidom.input_user_interface/6`, `aidom.asub_to_native/6`, `aidom.concrete/4`, `aidom.unknown_call/5`, `aidom.unknown_entry/4`, `aidom.empty_entry/4`, `aidom.part_conc/5`, `aidom.multi_part_conc/4`, `aidom.collect_auxinfo_asub/4`, `aidom.rename_auxinfo_asub/4`, `aidom.dom_statistics/2`, `aidom.contains_parameters/2`.

14.2 Documentation on exports

frdef_call_to_entry/9: PREDICATE

No further documentation available for this predicate.

frdef_call_to_success_fact/9: PREDICATE

No further documentation available for this predicate.

frdef_compute_lub/2: PREDICATE

No further documentation available for this predicate.

frdef_exit_to_prime/7: PREDICATE
No further documentation available for this predicate.

frdef_extend/5: PREDICATE
No further documentation available for this predicate.

frdef_identical_abstract/2: PREDICATE
No further documentation available for this predicate.

frdef_input_user_interface/5: PREDICATE
No further documentation available for this predicate.

frdef_input_interface/4: PREDICATE
No further documentation available for this predicate.

frdef_less_or_equal/2: PREDICATE
No further documentation available for this predicate.

frdef_glb/3: PREDICATE
No further documentation available for this predicate.

frdef_asub_to_native/5: PREDICATE
No further documentation available for this predicate.

frdef_project/5: PREDICATE
No further documentation available for this predicate.

frdef_abs_sort/2: PREDICATE
No further documentation available for this predicate.

frdef_special_builtin/5: PREDICATE
No further documentation available for this predicate.

frdef_success_builtin/6: PREDICATE
No further documentation available for this predicate.

frdef_unknown_call/4: PREDICATE

No further documentation available for this predicate.

frdef_unknown_entry/3: PREDICATE

No further documentation available for this predicate.

frdef_empty_entry/3: PREDICATE

No further documentation available for this predicate.

14.3 Documentation on multifiles

aidomain/1: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.init_abstract_domain/2: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.amgu/5: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.augment_asub/4: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.augment_two_asub/4: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.call_to_entry/10: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.exit_to_prime/8: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.project/6: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.widencall/4: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.needs/2: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.widen/4: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.compute_lub/3: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.compute_clauses_lub/4: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.compute_clauses_glb/4: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.fixpoint_covered/3: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.fixpoint_covered_gfp/3: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.identical_abstract/3: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.abs_sort/3: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.extend/6: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.less_or_equal/3: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.glb/4: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.eliminate_equivalent/3: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.abs_subset/3: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.call_to_success_fact/10: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.special_builtin/6: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.combined_special_builtin0/3: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.body_succ_builtin/9: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.split_combined_domain/4: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.success_builtin/7: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.call_to_success_builtin/7: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.obtain_info/5: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.input_interface/5: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.input_user_interface/6: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.asub_to_native/6: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.concrete/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.unknown_call/5: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.unknown_entry/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.empty_entry/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.part_conc/5: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.multi_part_conc/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.collect_auxinfo_asub/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.rename_auxinfo_asub/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.dom_statistics/2: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.contains_parameters/2: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

14.4 Documentation on imports

This module has the following direct dependencies:

- *Application modules:*
  preprocess_flags, domains, def, fr_sets, fr_shared, fr_top, plai_errors, share_aux.
- *System library modules:*
  sets, sort, lists, terms_vars.
- *Internal (engine) modules:*
  term_basic, arithmetic, atomic_basic, basiccontrol, exceptions, term_compare, term_typing, debugger_support.
- *Packages:*
  prelude, initial, condcomp, aidomain.

15 fr_top (library)

15.1 Usage and interface

- **Library usage:**
  `:- use_module(domain(fr_top)).`
- **Exports:**
  - *Predicates:*
    `fr_call_to_entry/9`, `fr_call_to_success_fact/9`, `fr_compute_lub/2`, `fr_exit_to_prime/7`, `fr_extend/5`, `fr_identical_abstract/2`, `fr_input_user_interface/5`, `fr_input_interface/4`, `fr_less_or_equal/2`, `fr_glb/3`, `fr_output_interface/3`, `fr_project/5`, `fr_abs_sort/2`, `fr_special_builtin/5`, `fr_success_builtin/6`, `fr_asub_to_native/5`, `fr_unknown_call/4`, `fr_unknown_entry/3`, `fr_empty_entry/3`, `get_free_vars/3`.
  - *Multifiles:*
    `aidomain/1`, `aidom.init_abstract_domain/2`, `aidom.amgu/5`, `aidom.augment_asub/4`, `aidom.augment_two_asub/4`, `aidom.call_to_entry/10`, `aidom.exit_to_prime/8`, `aidom.project/6`, `aidom.widencall/4`, `aidom.needs/2`, `aidom.widen/4`, `aidom.compute_lub/3`, `aidom.compute_clauses_lub/4`, `aidom.compute_clauses_glb/4`, `aidom.fixpoint_covered/3`, `aidom.fixpoint_covered_gfp/3`, `aidom.identical_abstract/3`, `aidom.abs_sort/3`, `aidom.extend/6`, `aidom.less_or_equal/3`, `aidom.glb/4`, `aidom.eliminate_equivalent/3`, `aidom.abs_subset/3`, `aidom.call_to_success_fact/10`, `aidom.special_builtin/6`, `aidom.combined_special_`
    `builtin0/3`, `aidom.body_succ_builtin/9`, `aidom.split_combined_domain/4`, `aidom.success_builtin/7`, `aidom.call_to_success_builtin/7`, `aidom.obtain_info/5`, `aidom.input_interface/5`, `aidom.input_user_interface/6`, `aidom.asub_to_native/6`, `aidom.concrete/4`, `aidom.unknown_call/5`, `aidom.unknown_entry/4`, `aidom.empty_entry/4`, `aidom.part_conc/5`, `aidom.multi_part_conc/4`, `aidom.collect_auxinfo_asub/4`, `aidom.rename_auxinfo_asub/4`, `aidom.dom_statistics/2`, `aidom.contains_parameters/2`.

15.2 Documentation on exports

fr_call_to_entry/9: PREDICATE

No further documentation available for this predicate.

fr_call_to_success_fact/9: PREDICATE

No further documentation available for this predicate.

fr_compute_lub/2: PREDICATE

No further documentation available for this predicate.

fr_exit_to_prime/7: PREDICATE
No further documentation available for this predicate.

fr_extend/5: PREDICATE
No further documentation available for this predicate.

fr_identical_abstract/2: PREDICATE
No further documentation available for this predicate.

fr_input_user_interface/5: PREDICATE
No further documentation available for this predicate.

fr_input_interface/4: PREDICATE
No further documentation available for this predicate.

fr_less_or_equal/2: PREDICATE
No further documentation available for this predicate.

fr_glb/3: PREDICATE
No further documentation available for this predicate.

fr_output_interface/3: PREDICATE
No further documentation available for this predicate.

fr_project/5: PREDICATE
No further documentation available for this predicate.

fr_abs_sort/2: PREDICATE
No further documentation available for this predicate.

fr_special_builtin/5: PREDICATE
No further documentation available for this predicate.

fr_success_builtin/6: PREDICATE
No further documentation available for this predicate.

fr_asub_to_native/5: PREDICATE

No further documentation available for this predicate.

fr_unknown_call/4: PREDICATE

No further documentation available for this predicate.

fr_unknown_entry/3: PREDICATE

No further documentation available for this predicate.

fr_empty_entry/3: PREDICATE

No further documentation available for this predicate.

get_free_vars/3: PREDICATE

No further documentation available for this predicate.

15.3 Documentation on multifiles

aidomain/1: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.init_abstract_domain/2: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.amgu/5: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.augment_asub/4: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.augment_two_asub/4: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.call_to_entry/10: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.exit_to_prime/8: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.project/6: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.widencall/4: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.needs/2: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.widen/4: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.compute_lub/3: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.compute_clauses_lub/4: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.compute_clauses_glb/4: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.fixpoint_covered/3: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.fixpoint_covered_gfp/3: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.identical_abstract/3: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.abs_sort/3: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.extend/6: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.less_or_equal/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.glb/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.eliminate_equivalent/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.abs_subset/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.call_to_success_fact/10: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.special_builtin/6: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.combined_special_builtin0/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.body_succ_builtin/9: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.split_combined_domain/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.success_builtin/7: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.call_to_success_builtin/7: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.obtain_info/5: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.input_interface/5: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.input_user_interface/6: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.asub_to_native/6: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.concrete/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.unknown_call/5: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.unknown_entry/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.empty_entry/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.part_conc/5: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.multi_part_conc/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.collect_auxinfo_asub/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.rename_auxinfo_asub/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.dom_statistics/2: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.contains_parameters/2: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

15.4 Documentation on imports

This module has the following direct dependencies:

- *Application modules:*
  `preprocess_flags`, `domains`, `plai_errors`, `fr_sets`, `fr_shared`, `share_aux`.
- *System library modules:*
  `datafacts_rt`, `sets`, `sort`, `p_unit`.
- *Internal (engine) modules:*
  `term_basic`, `arithmetic`, `atomic_basic`, `basiccontrol`, `exceptions`, `term_compare`, `term_typing`, `debugger_support`.
- *Packages:*
  `prelude`, `initial`, `condcomp`, `datafacts`, `aidomain`.

16 Simple groundness abstract domain

Author(s): Claudio Vaucheret.

This module implements the abstract operations of a simple groundness domain for the PLAI framework of abstract interpretation. An abstract substitution is a list of Var/Mode elements, where Var is a variable and Mode is "any", "g" or "ng".

The abstract domain lattice is:

```
             any
            /  \
           /    \
  (ground)   g     ng  (not ground)
           \    /
            \  /
          $bottom
```

16.1 Usage and interface

- **Library usage:**
  `:- use_module(domain(gr)).`

16.2 Documentation on multifiles

aidomain/1: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.init_abstract_domain/2: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.amgu/5: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.augment_asub/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.augment_two_asub/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.call_to_entry/10: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.exit_to_prime/8: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.project/6: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.widencall/4: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.needs/2: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.widen/4: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.compute_lub/3: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.compute_clauses_lub/4: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.compute_clauses_glb/4: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.fixpoint_covered/3: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.fixpoint_covered_gfp/3: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.identical_abstract/3: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.abs_sort/3: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.extend/6: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.less_or_equal/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.glb/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.eliminate_equivalent/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.abs_subset/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.call_to_success_fact/10: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.special_builtin/6: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.combined_special_builtin0/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.body_succ_builtin/9: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.split_combined_domain/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.success_builtin/7: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.call_to_success_builtin/7: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.obtain_info/5: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.input_interface/5: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.input_user_interface/6: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.asub_to_native/6: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.concrete/4: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.unknown_call/5: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.unknown_entry/4: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.empty_entry/4: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.part_conc/5: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.multi_part_conc/4: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.collect_auxinfo_asub/4: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.rename_auxinfo_asub/4: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.dom_statistics/2: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.contains_parameters/2: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

16.3 Documentation on internals

asub/1: PROPERTY

Usage: `asub(A)`

`A` is a well-formed abstract substitution of the gr domain.

asub_elem/1: PROPERTY

Usage: `asub_elem(E)`

`E` is a single substitution

gr_mode/1: REGTYPE

Usage: `gr_mode(M)`

`M` is g (ground), ng (nonground), or any

extrainfo/1: PROPERTY

Usage: `extrainfo(E)`

`E` is a par (asub,binds)

binds/1: PROPERTY

Usage: `binds(B)`

`B` is a list of bindings

binding/1: PROPERTY

Usage: `binding(B)`

`B` is a triple (X,Term,Vars), where X is a variable, Term is a term and Vars is the set of variables in Term

16.4 Documentation on imports

This module has the following direct dependencies:

- *Application modules:*
  preprocess_flags, domains.
- *System library modules:*
  datafacts_rt, native_props, messages, sort, terms_vars, terms_check, sets.
- *Internal (engine) modules:*
  term_basic, arithmetic, atomic_basic, basiccontrol, exceptions, term_compare, term_typing, debugger_support, basic_props, hiord_rt, io_basic.
- *Packages:*
  prelude, initial, condcomp, assertions, assertions/assertions_basic, regtypes, basicmodes, datafacts, nativeprops, aidomain.

17 lsign domain

17.1 Usage and interface

- **Library usage:**
  `:- use_module(domain(lsign)).`
- **Exports:**
  - *Predicates:*
    `lsign_init_abstract_domain/1`, `lsign_call_to_entry/9`, `lsign_call_to_success_fact/9`, `lsign_compute_lub/2`, `lsign_extend/5`, `lsign_eliminate_equivalent/2`, `lsign_exit_to_prime/7`, `lsign_global_info/5`, `lsign_input_user_interface/5`, `lsign_input_interface/4`, `lsign_asub_to_native/5`, `lsign_abs_subset/2`, `lsign_less_or_equal/2`, `lsign_glb/3`, `lsign_project/5`, `lsign_abs_sort/2`, `lsign_special_builtin/5`, `lsign_success_builtin/6`, `lsign_unknown_call/4`, `lsign_unknown_entry/3`, `lsign_empty_entry/3`, `lsign_sum/3`, `lsign_output_interface/2`, `lsign_propagate_fixpoint/8`.
  - *Multifiles:*
    `aidomain/1`, `aidom.init_abstract_domain/2`, `aidom.amgu/5`, `aidom.augment_asub/4`, `aidom.augment_two_asub/4`, `aidom.call_to_entry/10`, `aidom.exit_to_prime/8`, `aidom.project/6`, `aidom.widencall/4`, `aidom.needs/2`, `aidom.widen/4`, `aidom.compute_lub/3`, `aidom.compute_clauses_lub/4`, `aidom.compute_clauses_glb/4`, `aidom.fixpoint_covered/3`, `aidom.fixpoint_covered_gfp/3`, `aidom.identical_abstract/3`, `aidom.abs_sort/3`, `aidom.extend/6`, `aidom.less_or_equal/3`, `aidom.glb/4`, `aidom.eliminate_equivalent/3`, `aidom.abs_subset/3`, `aidom.call_to_success_fact/10`, `aidom.special_builtin/6`, `aidom.combined_special_builtin0/3`, `aidom.body_succ_builtin/9`, `aidom.split_combined_domain/4`, `aidom.success_builtin/7`, `aidom.call_to_success_builtin/7`, `aidom.obtain_info/5`, `aidom.input_interface/5`, `aidom.input_user_interface/6`, `aidom.asub_to_native/6`, `aidom.concrete/4`, `aidom.unknown_call/5`, `aidom.unknown_entry/4`, `aidom.empty_entry/4`, `aidom.part_conc/5`, `aidom.multi_part_conc/4`, `aidom.collect_auxinfo_asub/4`, `aidom.rename_auxinfo_asub/4`, `aidom.dom_statistics/2`, `aidom.contains_parameters/2`.

17.2 Documentation on exports

lsign_init_abstract_domain/1: PREDICATE
No further documentation available for this predicate.

lsign_call_to_entry/9: PREDICATE
No further documentation available for this predicate.

lsign_call_to_success_fact/9: PREDICATE
No further documentation available for this predicate.

lsign_compute_lub/2: PREDICATE
No further documentation available for this predicate.

lsign_extend/5: PREDICATE
No further documentation available for this predicate.

lsign_eliminate_equivalent/2: PREDICATE
No further documentation available for this predicate.

lsign_exit_to_prime/7: PREDICATE
No further documentation available for this predicate.

lsign_global_info/5: PREDICATE
No further documentation available for this predicate.

lsign_input_user_interface/5: PREDICATE
No further documentation available for this predicate.

lsign_input_interface/4: PREDICATE
No further documentation available for this predicate.

lsign_asub_to_native/5: PREDICATE
No further documentation available for this predicate.

lsign_abs_subset/2: PREDICATE
No further documentation available for this predicate.

lsign_less_or_equal/2: PREDICATE
No further documentation available for this predicate.

lsign_glb/3: PREDICATE
No further documentation available for this predicate.

lsign_project/5: PREDICATE
No further documentation available for this predicate.

lsign_abs_sort/2: PREDICATE
No further documentation available for this predicate.

lsign_special_builtin/5: PREDICATE
No further documentation available for this predicate.

lsign_success_builtin/6: PREDICATE
No further documentation available for this predicate.

lsign_unknown_call/4: PREDICATE
No further documentation available for this predicate.

lsign_unknown_entry/3: PREDICATE
No further documentation available for this predicate.

lsign_empty_entry/3: PREDICATE
No further documentation available for this predicate.

lsign_sum/3: PREDICATE
No further documentation available for this predicate.

lsign_output_interface/2: PREDICATE
No further documentation available for this predicate.

lsign_propagate_fixpoint/8: PREDICATE
No further documentation available for this predicate.

17.3 Documentation on multifiles

aidomain/1: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.init_abstract_domain/2: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.amgu/5: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.augment_asub/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.augment_two_asub/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.call_to_entry/10: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.exit_to_prime/8: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.project/6: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.widencall/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.needs/2: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.widen/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.compute_lub/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.compute_clauses_lub/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.compute_clauses_glb/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.fixpoint_covered/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.fixpoint_covered_gfp/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.identical_abstract/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.abs_sort/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.extend/6: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.less_or_equal/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.glb/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.eliminate_equivalent/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.abs_subset/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.call_to_success_fact/10: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.special_builtin/6: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.combined_special_builtin0/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.body_succ_builtin/9: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.split_combined_domain/4: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.success_builtin/7: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.call_to_success_builtin/7: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.obtain_info/5: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.input_interface/5: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.input_user_interface/6: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.asub_to_native/6: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.concrete/4: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.unknown_call/5: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.unknown_entry/4: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.empty_entry/4: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.part_conc/5: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.multi_part_conc/4: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.collect_auxinfo_asub/4: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.rename_auxinfo_asub/4: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.dom_statistics/2: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.contains_parameters/2: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

17.4 Documentation on imports

This module has the following direct dependencies:

- *Application modules:*
  `preprocess_flags`, `domains`, `deftools`, `s_grshfr`, `share_aux`, `plai_errors`.
- *System library modules:*
  `sets`, `sort`, `lists`, `lsets`, `messages`, `terms_check`, `terms_vars`.
- *Internal (engine) modules:*
  `term_basic`, `arithmetic`, `atomic_basic`, `basiccontrol`, `exceptions`, `term_compare`, `term_typing`, `debugger_support`, `basic_props`, `io_basic`.
- *Packages:*
  `prelude`, `initial`, `condcomp`, `assertions`, `assertions/assertions_basic`, `aidomain`.

18 lsigndiff domain

18.1 Usage and interface

- **Library usage:**
  `:- use_module(domain(lsigndiff)).`
- **Exports:**
  - *Predicates:*
    `difflsign_call_to_entry/9`, `difflsign_call_to_success_fact/9`, `difflsign_compute_lub/2`, `difflsign_exit_to_prime/7`, `difflsign_extend/5`, `difflsign_input_user_interface/5`, `difflsign_input_interface/4`, `difflsign_asub_to_native/5`, `difflsign_less_or_equal/2`, `difflsign_glb/3`, `difflsign_project/5`, `difflsign_abs_sort/2`, `difflsign_special_builtin/5`, `difflsign_success_builtin/6`, `difflsign_unknown_call/4`, `difflsign_unknown_entry/3`, `difflsign_empty_entry/3`.
  - *Multifiles:*
    `aidomain/1`, `aidom.init_abstract_domain/2`, `aidom.amgu/5`, `aidom.augment_asub/4`, `aidom.augment_two_asub/4`, `aidom.call_to_entry/10`, `aidom.exit_to_prime/8`, `aidom.project/6`, `aidom.widencall/4`, `aidom.needs/2`, `aidom.widen/4`, `aidom.compute_lub/3`, `aidom.compute_clauses_lub/4`, `aidom.compute_clauses_glb/4`, `aidom.fixpoint_covered/3`, `aidom.fixpoint_covered_gfp/3`, `aidom.identical_abstract/3`, `aidom.abs_sort/3`, `aidom.extend/6`, `aidom.less_or_equal/3`, `aidom.glb/4`, `aidom.eliminate_equivalent/3`, `aidom.abs_subset/3`, `aidom.call_to_success_fact/10`, `aidom.special_builtin/6`, `aidom.combined_special_builtin0/3`, `aidom.body_succ_builtin/9`, `aidom.split_combined_domain/4`, `aidom.success_builtin/7`, `aidom.call_to_success_builtin/7`, `aidom.obtain_info/5`, `aidom.input_interface/5`, `aidom.input_user_interface/6`, `aidom.asub_to_native/6`, `aidom.concrete/4`, `aidom.unknown_call/5`, `aidom.unknown_entry/4`, `aidom.empty_entry/4`, `aidom.part_conc/5`, `aidom.multi_part_conc/4`, `aidom.collect_auxinfo_asub/4`, `aidom.rename_auxinfo_asub/4`, `aidom.dom_statistics/2`, `aidom.contains_parameters/2`.

18.2 Documentation on exports

difflsign_call_to_entry/9: PREDICATE

No further documentation available for this predicate.

difflsign_call_to_success_fact/9: PREDICATE

No further documentation available for this predicate.

difflsign_compute_lub/2: PREDICATE

No further documentation available for this predicate.

difflsign_exit_to_prime/7: PREDICATE
No further documentation available for this predicate.

difflsign_extend/5: PREDICATE
No further documentation available for this predicate.

difflsign_input_user_interface/5: PREDICATE
No further documentation available for this predicate.

difflsign_input_interface/4: PREDICATE
No further documentation available for this predicate.

difflsign_asub_to_native/5: PREDICATE
No further documentation available for this predicate.

difflsign_less_or_equal/2: PREDICATE
No further documentation available for this predicate.

difflsign_glb/3: PREDICATE
No further documentation available for this predicate.

difflsign_project/5: PREDICATE
No further documentation available for this predicate.

difflsign_abs_sort/2: PREDICATE
No further documentation available for this predicate.

difflsign_special_builtin/5: PREDICATE
No further documentation available for this predicate.

difflsign_success_builtin/6: PREDICATE
No further documentation available for this predicate.

difflsign_unknown_call/4: PREDICATE
No further documentation available for this predicate.

difflsign_unknown_entry/3: PREDICATE
No further documentation available for this predicate.

difflsign_empty_entry/3: PREDICATE
No further documentation available for this predicate.

18.3 Documentation on multifiles

aidomain/1: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.init_abstract_domain/2: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.amgu/5: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.augment_asub/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.augment_two_asub/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.call_to_entry/10: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.exit_to_prime/8: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.project/6: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.widencall/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.needs/2: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.widen/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.compute_lub/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.compute_clauses_lub/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.compute_clauses_glb/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.fixpoint_covered/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.fixpoint_covered_gfp/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.identical_abstract/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.abs_sort/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.extend/6: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.less_or_equal/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.glb/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.eliminate_equivalent/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.abs_subset/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.call_to_success_fact/10: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.special_builtin/6: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.combined_special_builtin0/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.body_succ_builtin/9: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.split_combined_domain/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.success_builtin/7: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.call_to_success_builtin/7: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.obtain_info/5: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.input_interface/5: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.input_user_interface/6: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.asub_to_native/6: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.concrete/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.unknown_call/5: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.unknown_entry/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.empty_entry/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.part_conc/5: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.multi_part_conc/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.collect_auxinfo_asub/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.rename_auxinfo_asub/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.dom_statistics/2: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.contains_parameters/2: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

18.4 Documentation on imports

This module has the following direct dependencies:

- *Application modules:*
  preprocess_flags, domains, lsign, s_grshfr, plai_errors.
- *System library modules:*
  sets, sort.
- *Internal (engine) modules:*
  term_basic, arithmetic, atomic_basic, basiccontrol, exceptions, term_compare, term_typing, debugger_support, basic_props.
- *Packages:*
  prelude, initial, condcomp, assertions, assertions/assertions_basic, aidomain.

19 detplai (library)

19.1 Usage and interface

- **Library usage:**
  `:- use_module(domain(nfdet/detplai)).`
- **Exports:**
  - *Predicates:*
    `det_init_abstract_domain/1`, `det_call_to_entry/9`, `det_widen/3`, `det_widencall/3`, `det_compute_clauses_lub/3`, `det_eliminate_equivalent/2`, `det_split_combined_domain/3`, `det_special_builtin/5`, `det_combined_special_builtin0/2`, `det_asub_to_native/5`, `det_dom_statistics/1`, `det_obtain_info/4`, `asub/4`, `det_collect_auxinfo_asub/3`, `det_rename_auxinfo_asub/3`.
  - *Regular Types:*
    `det_asub/1`.
  - *Multifiles:*
    `aidomain/1`, `aidom.init_abstract_domain/2`, `aidom.amgu/5`, `aidom.augment_asub/4`, `aidom.augment_two_asub/4`, `aidom.call_to_entry/10`, `aidom.exit_to_prime/8`, `aidom.project/6`, `aidom.widencall/4`, `aidom.needs/2`, `aidom.widen/4`, `aidom.compute_lub/3`, `aidom.compute_clauses_lub/4`, `aidom.compute_clauses_glb/4`, `aidom.fixpoint_covered/3`, `aidom.fixpoint_covered_gfp/3`, `aidom.identical_abstract/3`, `aidom.abs_sort/3`, `aidom.extend/6`, `aidom.less_or_equal/3`, `aidom.glb/4`, `aidom.eliminate_equivalent/3`, `aidom.abs_subset/3`, `aidom.call_to_success_fact/10`, `aidom.special_builtin/6`, `aidom.combined_special_builtin0/3`, `aidom.body_succ_builtin/9`, `aidom.split_combined_domain/4`, `aidom.success_builtin/7`, `aidom.call_to_success_builtin/7`, `aidom.obtain_info/5`, `aidom.input_interface/5`, `aidom.input_user_interface/6`, `aidom.asub_to_native/6`, `aidom.concrete/4`, `aidom.unknown_call/5`, `aidom.unknown_entry/4`, `aidom.empty_entry/4`, `aidom.part_conc/5`, `aidom.multi_part_conc/4`, `aidom.collect_auxinfo_asub/4`, `aidom.rename_auxinfo_asub/4`, `aidom.dom_statistics/2`, `aidom.contains_parameters/2`.

19.2 Documentation on exports

det_init_abstract_domain/1: PREDICATE
No further documentation available for this predicate.

det_call_to_entry/9: PREDICATE
No further documentation available for this predicate.

det_exit_to_prime/7: PREDICATE
No further documentation available for this predicate.

det_project/5: PREDICATE
No further documentation available for this predicate.

det_extend/5: PREDICATE
No further documentation available for this predicate.

det_widen/3: PREDICATE
No further documentation available for this predicate.

det_widencall/3: PREDICATE
No further documentation available for this predicate.

det_compute_lub/2: PREDICATE
No further documentation available for this predicate.

det_compute_clauses_lub/3: PREDICATE
No further documentation available for this predicate.

det_glb/3: PREDICATE
No further documentation available for this predicate.

det_eliminate_equivalent/2: PREDICATE
No further documentation available for this predicate.

det_less_or_equal/2: PREDICATE
No further documentation available for this predicate.

det_identical_abstract/2: PREDICATE
No further documentation available for this predicate.

det_abs_sort/2: PREDICATE
No further documentation available for this predicate.

det_call_to_success_fact/9: PREDICATE
No further documentation available for this predicate.

det_split_combined_domain/3: PREDICATE
No further documentation available for this predicate.

det_special_builtin/5: PREDICATE
No further documentation available for this predicate.

det_combined_special_builtin0/2: PREDICATE
No further documentation available for this predicate.

det_success_builtin/6: PREDICATE
No further documentation available for this predicate.

det_input_interface/4: PREDICATE
No further documentation available for this predicate.

det_input_user_interface/5: PREDICATE
No further documentation available for this predicate.

det_asub_to_native/5: PREDICATE
No further documentation available for this predicate.

det_unknown_call/4: PREDICATE
No further documentation available for this predicate.

det_unknown_entry/3: PREDICATE
No further documentation available for this predicate.

det_empty_entry/3: PREDICATE
No further documentation available for this predicate.

det_dom_statistics/1: PREDICATE
No further documentation available for this predicate.

det_obtain_info/4: PREDICATE
No further documentation available for this predicate.

det_asub/1: REGTYPE

`det_asub(ASub)`

`ASub` is an abstract substitution term used in det. It contains types, modes and determinism information.

Usage: `det_asub(ASub)`

`ASub` is an abstract substitution term used in det.

asub/4: PREDICATE

No further documentation available for this predicate.

det_collect_auxinfo_asub/3: PREDICATE

No further documentation available for this predicate.

det_rename_auxinfo_asub/3: PREDICATE

No further documentation available for this predicate.

19.3 Documentation on multifiles

aidomain/1: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.init_abstract_domain/2: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.amgu/5: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.augment_asub/4: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.augment_two_asub/4: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.call_to_entry/10: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.exit_to_prime/8: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.project/6: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.widencall/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.needs/2: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.widen/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.compute_lub/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.compute_clauses_lub/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.compute_clauses_glb/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.fixpoint_covered/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.fixpoint_covered_gfp/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.identical_abstract/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.abs_sort/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.extend/6: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.less_or_equal/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.glb/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.eliminate_equivalent/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.abs_subset/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.call_to_success_fact/10: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.special_builtin/6: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.combined_special_builtin0/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.body_succ_builtin/9: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.split_combined_domain/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.success_builtin/7: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.call_to_success_builtin/7: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.obtain_info/5: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.input_interface/5: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.input_user_interface/6: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.asub_to_native/6: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.concrete/4: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.unknown_call/5: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.unknown_entry/4: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.empty_entry/4: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.part_conc/5: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.multi_part_conc/4: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.collect_auxinfo_asub/4: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.rename_auxinfo_asub/4: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.dom_statistics/2: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.contains_parameters/2: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

19.4 Documentation on imports

This module has the following direct dependencies:

- *Application modules:*
  `preprocess_flags`, `domains`, `eterms`, `sharefree`, `detabs`, `infer_dom`.
- *System library modules:*
  `sets`, `sort`, `idlists`, `lists`.
- *Internal (engine) modules:*
  `term_basic`, `arithmetic`, `atomic_basic`, `basiccontrol`, `exceptions`, `term_compare`, `term_typing`, `debugger_support`, `basic_props`, `hiord_rt`.
- *Packages:*
  `prelude`, `initial`, `condcomp`, `assertions`, `assertions/assertions_basic`, `regtypes`, `basicmodes`, `aidomain`.

20 nfplai (library)

20.1 Usage and interface

- **Library usage:**
  `:- use_module(domain(nfdet/nfplai)).`
- **Exports:**
  - *Predicates:*
    `nf_init_abstract_domain/1`, `nf_call_to_entry/9`, `nf_widencall/3`, `nf_compute_clauses_lub/3`, `nf_eliminate_equivalent/2`, `nf_split_combined_domain/3`, `nf_special_builtin/5`, `nf_combined_special_builtin0/2`, `nf_asub_to_native/5`, `nf_dom_statistics/1`, `nf_obtain_info/4`, `asub/4`, `nf_collect_auxinfo_asub/3`, `nf_rename_auxinfo_asub/3`.
  - *Regular Types:*
    `nf_par_asub/2`, `nf_asub/1`, `nf_pred_asub/1`, `nf_pred_asub_sl/1`.
  - *Multifiles:*
    `aidomain/1`, `aidom.init_abstract_domain/2`, `aidom.amgu/5`, `aidom.augment_asub/4`, `aidom.augment_two_asub/4`, `aidom.call_to_entry/10`, `aidom.exit_to_prime/8`, `aidom.project/6`, `aidom.widencall/4`, `aidom.needs/2`, `aidom.widen/4`, `aidom.compute_lub/3`, `aidom.compute_clauses_lub/4`, `aidom.compute_clauses_glb/4`, `aidom.fixpoint_covered/3`, `aidom.fixpoint_covered_gfp/3`, `aidom.identical_abstract/3`, `aidom.abs_sort/3`, `aidom.extend/6`, `aidom.less_or_equal/3`, `aidom.glb/4`, `aidom.eliminate_equivalent/3`, `aidom.abs_subset/3`, `aidom.call_to_success_fact/10`, `aidom.special_builtin/6`, `aidom.combined_special_builtin0/3`, `aidom.body_succ_builtin/9`, `aidom.split_combined_domain/4`, `aidom.success_builtin/7`, `aidom.call_to_success_builtin/7`, `aidom.obtain_info/5`, `aidom.input_interface/5`, `aidom.input_user_interface/6`, `aidom.asub_to_native/6`, `aidom.concrete/4`, `aidom.unknown_call/5`, `aidom.unknown_entry/4`, `aidom.empty_entry/4`, `aidom.part_conc/5`, `aidom.multi_part_conc/4`, `aidom.collect_auxinfo_asub/4`, `aidom.rename_auxinfo_asub/4`, `aidom.dom_statistics/2`, `aidom.contains_parameters/2`.

20.2 Documentation on exports

nf_init_abstract_domain/1: PREDICATE

No further documentation available for this predicate.

nf_call_to_entry/9: PREDICATE

No further documentation available for this predicate.

nf_exit_to_prime/7: PREDICATE

No further documentation available for this predicate.

nf_project/5: PREDICATE

No further documentation available for this predicate.

nf_extend/5: PREDICATE

No further documentation available for this predicate.

nf_widen/3: PREDICATE

No further documentation available for this predicate.

nf_widencall/3: PREDICATE

No further documentation available for this predicate.

nf_compute_lub/2: PREDICATE

No further documentation available for this predicate.

nf_compute_clauses_lub/3: PREDICATE

Usage: `nf_compute_clauses_lub(ASubL,Proj,LubL)`

Performs the covering check that decides if the predicate test encoded in `ASubL` covers the types/modes represented in `Proj`. The predicate test is a conjunction of clause tests (corresponding to those clauses that do not fail so far). The modes/types, taken from `Proj`, corresponds to the parent goal, i.e., the one that unifies with the heads the clauses whose tests are represented in `ASubL`. The covering and nonfailure information derived from the covering check is represented in `LubL`

- *The following properties should hold at call time:*
  - `nf_pred_asub_sl(ASubL)` (`nf_pred_asub_sl/1`)
  - `Proj` is an abstract substitution term used in the combined domain `nf`. It contains types, modes and nonfailure information. (`nf_asub/1`)
  - `LubL` is a free variable. (`var/1`)
- *The following properties should hold upon exit:*
  - `nf_pred_asub_sl(ASubL)` (`nf_pred_asub_sl/1`)
  - `Proj` is an abstract substitution term used in the combined domain `nf`. It contains types, modes and nonfailure information. (`nf_asub/1`)
  - `nf_pred_asub_sl(LubL)` (`nf_pred_asub_sl/1`)

nf_glb/3: PREDICATE

No further documentation available for this predicate.

nf_eliminate_equivalent/2: PREDICATE

No further documentation available for this predicate.

nf_less_or_equal/2: PREDICATE
No further documentation available for this predicate.

nf_identical_abstract/2: PREDICATE
No further documentation available for this predicate.

nf_abs_sort/2: PREDICATE
No further documentation available for this predicate.

nf_call_to_success_fact/9: PREDICATE
No further documentation available for this predicate.

nf_split_combined_domain/3: PREDICATE
No further documentation available for this predicate.

nf_special_builtin/5: PREDICATE
No further documentation available for this predicate.

nf_combined_special_builtin0/2: PREDICATE
No further documentation available for this predicate.

nf_success_builtin/6: PREDICATE
No further documentation available for this predicate.

nf_input_interface/4: PREDICATE
No further documentation available for this predicate.

nf_input_user_interface/5: PREDICATE
No further documentation available for this predicate.

nf_asub_to_native/5: PREDICATE
No further documentation available for this predicate.

nf_unknown_call/4: PREDICATE
No further documentation available for this predicate.

nf_unknown_entry/3: PREDICATE

No further documentation available for this predicate.

nf_empty_entry/3: PREDICATE

No further documentation available for this predicate.

nf_dom_statistics/1: PREDICATE

No further documentation available for this predicate.

nf_obtain_info/4: PREDICATE

No further documentation available for this predicate.

nf_par_asub/2: REGTYPE

`nf_par_asub(NfTyp,ASub)`

`ASub` is an abstract substitution term used in the combined domain `nf`. It contains types, modes and nonfailure information of type `NfTyp`.

Usage: `nf_par_asub(NfTyp,ASub)`

`ASub` is an abstract substitution term used in the combined domain `nf`. It contains types, modes and nonfailure information of type `NfTyp`.

Meta-predicate with arguments: `nf_par_asub(pred(1),?)`.

nf_asub/1: REGTYPE

`nf_asub(ASub)`

`ASub` is an abstract substitution term used in the combined domain `nf`. It contains types, modes and nonfailure information.

Usage: `nf_asub(ASub)`

`ASub` is an abstract substitution term used in the combined domain `nf`. It contains types, modes and nonfailure information.

nf_pred_asub/1: REGTYPE

Usage: `nf_pred_asub(ASub)`

`ASub` is a compact representation for a set of abstract substitutions used in the combined domain `nf`. Each of the substitutions in that set has the same modes and types. However, the nonfailure information gathers together the clause tests of those abstract substitutions as a list, meaning the disjunction of all of them (named a `predicate test`, which is needed for performing the `covering check`).

nf_pred_asub_sl/1: REGTYPE

A regular type, defined as follows:

```
nf_pred_asub_sl([Asub]) :-
    nf_pred_asub(Asub).
```

asub/4: PREDICATE

No further documentation available for this predicate.

nf_collect_auxinfo_asub/3: PREDICATE

No further documentation available for this predicate.

nf_rename_auxinfo_asub/3: PREDICATE

No further documentation available for this predicate.

20.3 Documentation on multifiles

aidomain/1: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.init_abstract_domain/2: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.amgu/5: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.augment_asub/4: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.augment_two_asub/4: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.call_to_entry/10: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.exit_to_prime/8: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.project/6: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.widencall/4: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.needs/2: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.widen/4: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.compute_lub/3: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.compute_clauses_lub/4: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.compute_clauses_glb/4: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.fixpoint_covered/3: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.fixpoint_covered_gfp/3: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.identical_abstract/3: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.abs_sort/3: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.extend/6: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.less_or_equal/3: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.glb/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.eliminate_equivalent/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.abs_subset/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.call_to_success_fact/10: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.special_builtin/6: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.combined_special_builtin0/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.body_succ_builtin/9: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.split_combined_domain/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.success_builtin/7: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.call_to_success_builtin/7: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.obtain_info/5: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.input_interface/5: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.input_user_interface/6: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.asub_to_native/6: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.concrete/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.unknown_call/5: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.unknown_entry/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.empty_entry/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.part_conc/5: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.multi_part_conc/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.collect_auxinfo_asub/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.rename_auxinfo_asub/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.dom_statistics/2: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.contains_parameters/2: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

20.4 Documentation on imports

This module has the following direct dependencies:

- *Application modules:*
  preprocess_flags, domains, eterms, sharefree, nfabs, infer_dom.
- *System library modules:*
  sets, sort, idlists, lists.
- *Internal (engine) modules:*
  term_basic, arithmetic, atomic_basic, basiccontrol, exceptions, term_compare, term_typing, debugger_support, basic_props, hiord_rt.
- *Packages:*
  prelude, initial, condcomp, assertions, assertions/assertions_basic, regtypes, basicmodes, hiord, aidomain.

21 Interval Domain

Author(s): Isabel Garcia-Contreras, Jose F. Morales.

Stability: [**devel**] Currently the subject of active development and/or research. Functionality may be limited and API and/or functionality may change without warning or deprecation period. Not recommended yet for use in production.

A simple *intervals* abstract domain (non relational) derived from `nonrel_base.pl`.

This domain only uses closed intervals as abstractions, over-approximating the necessary builtins.

21.1 Usage and interface

- **Library usage:**
  `:- use_module(domain(nonrel_intervals)).`

21.2 Documentation on multifiles

aidomain/1: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.init_abstract_domain/2: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.amgu/5: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.augment_asub/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.augment_two_asub/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.call_to_entry/10: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.exit_to_prime/8: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.project/6: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.widencall/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.needs/2: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.widen/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.compute_lub/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.compute_clauses_lub/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.compute_clauses_glb/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.fixpoint_covered/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.fixpoint_covered_gfp/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.identical_abstract/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.abs_sort/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.extend/6: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.less_or_equal/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.glb/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.eliminate_equivalent/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.abs_subset/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.call_to_success_fact/10: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.special_builtin/6: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.combined_special_builtin0/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.body_succ_builtin/9: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.split_combined_domain/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.success_builtin/7: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.call_to_success_builtin/7: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.obtain_info/5: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.input_interface/5: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.input_user_interface/6: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.asub_to_native/6: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.concrete/4: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.unknown_call/5: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.unknown_entry/4: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.empty_entry/4: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.part_conc/5: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.multi_part_conc/4: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.collect_auxinfo_asub/4: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.rename_auxinfo_asub/4: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.dom_statistics/2: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.contains_parameters/2: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

21.3 Documentation on internals

nonrel_top/2: PREDICATE

Usage: `nonrel_top(AbsInt,X)`

`X` is the representation of "top" in the abstract domain.

(implement in derived domain)

nonrel_bot/2: PREDICATE

Usage: `nonrel_bot(AbsInt,X)`

`X` is the representation of "bot" in the abstract domain.

(implement in derived domain)

nonrel_var_elem/2: PREDICATE

Usage: `nonrel_var_elem(AbsInt,X)`

`X` is the abstraction of a free variable in the abstract domain.

(implement in derived domain)

nonrel_less_or_equal_elem/3: PREDICATE

Usage: `nonrel_less_or_equal_elem(AbsInt,E1,E2)`

Succeeds if `E1` is smaller than `E2`.

(implement in derived domain)

- *The following properties should hold at call time:*
  `AbsInt` is currently a term which is not a free variable. (`nonvar/1`)
  `E1` is currently a term which is not a free variable. (`nonvar/1`)
  `E2` is currently a term which is not a free variable. (`nonvar/1`)

nonrel_compute_glb_elem/4: PREDICATE

Usage: `nonrel_compute_glb_elem(AbsInt,E1,E2,EG)`

`EG` is the *greatest lower bound* of `E1` and `E2`.

(implement in derived domain)

- *The following properties should hold at call time:*
  - `AbsInt` is currently a term which is not a free variable. (`nonvar/1`)
  - `E1` is currently a term which is not a free variable. (`nonvar/1`)
  - `E2` is currently a term which is not a free variable. (`nonvar/1`)

nonrel_compute_lub_elem/4: PREDICATE

Usage: `nonrel_compute_lub_elem(AbsInt,E1,E2,EL)`

`EL` is the *least upper bound* of `E1` and `E2`.

(implement in derived domain)

- *The following properties should hold at call time:*
  - `AbsInt` is currently a term which is not a free variable. (`nonvar/1`)
  - `E1` is currently a term which is not a free variable. (`nonvar/1`)
  - `E2` is currently a term which is not a free variable. (`nonvar/1`)

nonrel_widen_elem/4: PREDICATE

Usage: `nonrel_widen_elem(AbsInt,E1,E2,EW)`

`EW` is the *widening* of `E1` and `E2`.

(implement in derived domain)

- *The following properties should hold at call time:*
  - `AbsInt` is currently a term which is not a free variable. (`nonvar/1`)
  - `E1` is currently a term which is not a free variable. (`nonvar/1`)
  - `E2` is currently a term which is not a free variable. (`nonvar/1`)

nonrel_amgu/5: PREDICATE

Usage: `nonrel_amgu(AbsInt,Term1,Term2,ASub0,NASub)`

`NASub` is the abstract unification of `Term1` and `Term2`, with `ASub0` an abstract substitution that represents the state of both terms.

(implement in derived domain)

- *The following properties should hold at call time:*
  - `AbsInt` is currently a term which is not a free variable. (`nonvar/1`)
  - `Term1` is currently a term which is not a free variable. (`nonvar/1`)
  - `Term2` is currently a term which is not a free variable. (`nonvar/1`)
  - `ASub0` is currently a term which is not a free variable. (`nonvar/1`)
  - `NASub` is a free variable. (`var/1`)

nonrel_unknown_entry/4: PREDICATE

Usage: `nonrel_unknown_entry(AbsInt,Sg,Vars,Call)`

Gives the "top" value for a given set of variables `Vars`, resulting in abstract constraint `Call`.

- *The following properties should hold at call time:*
  - `AbsInt` is currently a term which is not a free variable. (`nonvar/1`)
  - `Sg` is currently a term which is not a free variable. (`nonvar/1`)
  - `Vars` is currently a term which is not a free variable. (`nonvar/1`)
  - `Call` is a free variable. (`var/1`)
  - `AbsInt` is an atom. (`atm/1`)
  - `Sg` is a term which represents a goal, i.e., an atom or a structure. (`cgoal/1`)
  - `Vars` is a list. (`list/1`)
  - `Call` is any term. (`term/1`)

nonrel_abs_sort/2: PREDICATE

Usage: `nonrel_abs_sort(Asub,Asub_s)`

It sorts the set of `X/Values` in `Asub` obtaining `Asub_s`

- *The following properties should hold at call time:*
  - `Asub` is currently a term which is not a free variable. (`nonvar/1`)
  - `Asub_s` is a free variable. (`var/1`)
  - `Asub` is any term. (`term/1`)
  - `Asub_s` is any term. (`term/1`)

nonrel_project/5: PREDICATE

Usage: `nonrel_project(Sg,Vars,HvFv_u,ASub,Proj)`

`Proj` is the result of eliminating from `ASub` all `X/Value` such that `X` is not in `Vars`

- *The following properties should hold at call time:*
  - `Sg` is currently a term which is not a free variable. (`nonvar/1`)
  - `Vars` is currently a term which is not a free variable. (`nonvar/1`)
  - `HvFv_u` is currently a term which is not a free variable. (`nonvar/1`)
  - `ASub` is currently a term which is not a free variable. (`nonvar/1`)
  - `Proj` is a free variable. (`var/1`)
  - `Sg` is any term. (`term/1`)
  - `Vars` is a list. (`list/1`)
  - `HvFv_u` is a list. (`list/1`)
  - `ASub` is any term. (`term/1`)
  - `Proj` is any term. (`term/1`)

nonrel_call_to_entry/10: PREDICATE

Usage:

`nonrel_call_to_entry(AbsInt,Sv,Sg,Hv,Head,K,Fv,Proj,Entry,ExtraInfo)`

It obtains the abstract substitution `Entry` which results from adding the abstraction of `Sg` = `Head` to `Proj`, later projecting the resulting substitution onto `Hv`.

In the general case, the steps are:

- Add to `Proj` the variables of `Hv` with a "top" value.
- Perform the abstract unification of `Sg` and `Head`.
- Add to the unification the variables of `Fv` with a "top" value.
- Project to the variables in `Hv` and `Fv`.

– *The following properties should hold at call time:*

`AbsInt` is currently a term which is not a free variable. (`nonvar/1`)
`Sv` is currently a term which is not a free variable. (`nonvar/1`)
`Sg` is currently a term which is not a free variable. (`nonvar/1`)
`Hv` is currently a term which is not a free variable. (`nonvar/1`)
`Head` is currently a term which is not a free variable. (`nonvar/1`)
`K` is currently a term which is not a free variable. (`nonvar/1`)
`Fv` is currently a term which is not a free variable. (`nonvar/1`)
`Proj` is currently a term which is not a free variable. (`nonvar/1`)
`Entry` is a free variable. (`var/1`)
`ExtraInfo` is a free variable. (`var/1`)
`AbsInt` is an atom. (`atm/1`)
`Sv` is a list. (`list/1`)
`Sg` is a term which represents a goal, i.e., an atom or a structure. (`cgoal/1`)
`Hv` is a list. (`list/1`)
`Head` is a term which represents a goal, i.e., an atom or a structure. (`cgoal/1`)
`K` is any term. (`term/1`)
`Fv` is a list. (`list/1`)
`Proj` is any term. (`term/1`)
`Entry` is any term. (`term/1`)
`ExtraInfo` is any term. (`term/1`)

nonrel_exit_to_prime/8: PREDICATE

Usage: `nonrel_exit_to_prime(AbsInt,Sg,Hv,Head,Sv,Exit,ExtraInfo,Prime)`

– *The following properties should hold at call time:*

`AbsInt` is currently a term which is not a free variable. (`nonvar/1`)
`Sg` is currently a term which is not a free variable. (`nonvar/1`)
`Hv` is currently a term which is not a free variable. (`nonvar/1`)
`Head` is currently a term which is not a free variable. (`nonvar/1`)
`Sv` is currently a term which is not a free variable. (`nonvar/1`)
`Exit` is currently a term which is not a free variable. (`nonvar/1`)
`ExtraInfo` is a free variable. (`var/1`)
`Prime` is a free variable. (`var/1`)
`AbsInt` is an atom. (`atm/1`)
`Sg` is a list. (`list/1`)
`Hv` is a list. (`list/1`)
`Head` is a term which represents a goal, i.e., an atom or a structure. (`cgoal/1`)
`Sv` is a term which represents a goal, i.e., an atom or a structure. (`cgoal/1`)
`Exit` is any term. (`term/1`)
`ExtraInfo` is any term. (`term/1`)
`Prime` is any term. (`term/1`)

nonrel_compute_lub/3: PREDICATE

Usage: `nonrel_compute_lub(AbsInt,ListASub,Lub)`

`Lub` is the *least upper bound* of the list of abstract substitutions `ListASub`.

- *The following properties should hold at call time:*

  `AbsInt` is currently a term which is not a free variable. (`nonvar/1`)
  `ListASub` is currently a term which is not a free variable. (`nonvar/1`)
  `Lub` is a free variable. (`var/1`)
  `AbsInt` is an atom. (`atm/1`)
  `ListASub` is a list. (`list/1`)
  `Lub` is any term. (`term/1`)

nonrel_extend/6: PREDICATE

Usage: `nonrel_extend(AbsInt,Sg,Prime,Sv,Call,Succ)`

- *The following properties should hold at call time:*

  `AbsInt` is currently a term which is not a free variable. (`nonvar/1`)
  `Sg` is currently a term which is not a free variable. (`nonvar/1`)
  `Prime` is currently a term which is not a free variable. (`nonvar/1`)
  `Call` is currently a term which is not a free variable. (`nonvar/1`)
  `Succ` is a free variable. (`var/1`)
  `AbsInt` is an atom. (`atm/1`)
  `Sg` is any term. (`term/1`)
  `Prime` is any term. (`term/1`)
  `Sv` is a list. (`list/1`)
  `Call` is any term. (`term/1`)
  `Succ` is any term. (`term/1`)

nonrel_call_to_success_fact/10: PREDICATE

Usage:
`nonrel_call_to_success_fact(AbsInt,Sg,Hv,Head,K,Sv,Call,Proj,Prime,Succ)`

Specialized version of `call_to_entry/9` + `exit_to_prime/7` + `extend/4` for facts

- *The following properties should hold at call time:*

  `AbsInt` is currently a term which is not a free variable. (`nonvar/1`)
  `Sg` is currently a term which is not a free variable. (`nonvar/1`)
  `Hv` is currently a term which is not a free variable. (`nonvar/1`)
  `Head` is currently a term which is not a free variable. (`nonvar/1`)
  `K` is currently a term which is not a free variable. (`nonvar/1`)
  `Sv` is currently a term which is not a free variable. (`nonvar/1`)
  `Call` is currently a term which is not a free variable. (`nonvar/1`)
  `Proj` is currently a term which is not a free variable. (`nonvar/1`)
  `Prime` is a free variable. (`var/1`)
  `Succ` is a free variable. (`var/1`)
  `AbsInt` is an atom. (`atm/1`)

`Sg` is a term which represents a goal, i.e., an atom or a structure. (`cgoal/1`)
`Hv` is a list. (`list/1`)
`Head` is a term which represents a goal, i.e., an atom or a structure. (`cgoal/1`)
`K` is a list. (`list/1`)
`Sv` is any term. (`term/1`)
`Call` is any term. (`term/1`)
`Proj` is any term. (`term/1`)
`Prime` is any term. (`term/1`)
`Succ` is any term. (`term/1`)

nonrel_special_builtin/6: PREDICATE

Usage: `nonrel_special_builtin(AbsInt,SgKey,Sg,Subgoal,Type,Condvars)`

This predicate needs to be implemented by the user.

− *The following properties should hold at call time:*
`AbsInt` is currently a term which is not a free variable. (`nonvar/1`)
`SgKey` is currently a term which is not a free variable. (`nonvar/1`)
`Sg` is currently a term which is not a free variable. (`nonvar/1`)
`Subgoal` is currently a term which is not a free variable. (`nonvar/1`)
`Type` is a free variable. (`var/1`)
`Condvars` is a free variable. (`var/1`)
`AbsInt` is an atom. (`atm/1`)
`SgKey` is a predicate name. (`predname/1`)
`Sg` is a term which represents a goal, i.e., an atom or a structure. (`cgoal/1`)
`Subgoal` is a term which represents a goal, i.e., an atom or a structure. (`cgoal/1`)
`Type` is an atom. (`atm/1`)
`Condvars` is any term. (`term/1`)

nonrel_success_builtin/7: PREDICATE

Usage: `nonrel_success_builtin(AbsInt,Sv_uns,Type,Condv,HvFv_u,Call,Succ)`

Obtains the success for some particular builtins.

− *The following properties should hold at call time:*
`AbsInt` is currently a term which is not a free variable. (`nonvar/1`)
`Sv_uns` is currently a term which is not a free variable. (`nonvar/1`)
`Type` is currently a term which is not a free variable. (`nonvar/1`)
`Condv` is currently a term which is not a free variable. (`nonvar/1`)
`HvFv_u` is currently a term which is not a free variable. (`nonvar/1`)
`Call` is currently a term which is not a free variable. (`nonvar/1`)
`Succ` is a free variable. (`var/1`)
`AbsInt` is an atom. (`atm/1`)
`Sv_uns` is any term. (`term/1`)

`Type` is an atom. (`atm/1`)
`Condv` is any term. (`term/1`)
`HvFv_u` is any term. (`term/1`)
`Call` is any term. (`term/1`)
`Succ` is any term. (`term/1`)

nonrel_call_to_success_builtin/7: PREDICATE

Usage: `nonrel_call_to_success_builtin(AbsInt,SgKey,Sg,Sv,Call,Proj,Succ)`

Handles those builtins for which computing `Proj` is easier than `Succ`. Currently only builtins `=/2` and `==/2` are implemented. More builtins can be added by the user.

− *The following properties should hold at call time:*

`AbsInt` is currently a term which is not a free variable. (`nonvar/1`)
`SgKey` is currently a term which is not a free variable. (`nonvar/1`)
`Sg` is currently a term which is not a free variable. (`nonvar/1`)
`Sv` is currently a term which is not a free variable. (`nonvar/1`)
`Call` is currently a term which is not a free variable. (`nonvar/1`)
`Proj` is currently a term which is not a free variable. (`nonvar/1`)
`Succ` is a free variable. (`var/1`)
`AbsInt` is an atom. (`atm/1`)
`SgKey` is a predicate name. (`predname/1`)
`Sg` is a term which represents a goal, i.e., an atom or a structure. (`cgoal/1`)
`Sv` is a list. (`list/1`)
`Call` is any term. (`term/1`)
`Proj` is any term. (`term/1`)
`Succ` is any term. (`term/1`)

nonrel_input_user_interface/6: PREDICATE

Usage:
`nonrel_input_user_interface(AbsInt,InputUser,Qv,ASub,Sg,MaybeCallASub)`

Obtains the abstract substitution from the native properties found in the user supplied info.

− *The following properties should hold at call time:*

`AbsInt` is currently a term which is not a free variable. (`nonvar/1`)
`InputUser` is currently a term which is not a free variable. (`nonvar/1`)
`Qv` is currently a term which is not a free variable. (`nonvar/1`)
`ASub` is currently a term which is not a free variable. (`nonvar/1`)
`Sg` is currently a term which is not a free variable. (`nonvar/1`)
`MaybeCallASub` is currently a term which is not a free variable. (`nonvar/1`)

nonrel_input_interface/5: PREDICATE

Usage: `nonrel_input_interface(AbsInt,Prop,Kind,Struc0,Struc1)`

Adds native property `Prop` to the structure accumulating the properties relevant to this domain.

- *The following properties should hold at call time:*

  `AbsInt` is currently a term which is not a free variable. (`nonvar/1`)
  `Prop` is currently a term which is not a free variable. (`nonvar/1`)
  `Struc0` is currently a term which is not a free variable. (`nonvar/1`)
  `Struc1` is currently a term which is not a free variable. (`nonvar/1`)

nonrel_asub_to_native/6: PREDICATE

Usage: `nonrel_asub_to_native(AbsInt,ASub,Qv,OutFlag,ASub_user,Comps)`

It translates an internal abstract constraint into something friendly for the user.

- *The following properties should hold at call time:*

  `AbsInt` is currently a term which is not a free variable. (`nonvar/1`)
  `ASub` is currently a term which is not a free variable. (`nonvar/1`)
  `Qv` is currently a term which is not a free variable. (`nonvar/1`)
  `OutFlag` is currently a term which is not a free variable. (`nonvar/1`)
  `ASub_user` is a free variable. (`var/1`)
  `Comps` is a free variable. (`var/1`)
  `AbsInt` is an atom. (`atm/1`)
  `ASub` is any term. (`term/1`)
  `Qv` is any term. (`term/1`)
  `OutFlag` is any term. (`term/1`)
  `ASub_user` is any term. (`term/1`)
  `Comps` is any term. (`term/1`)

nonrel_unknown_call/5: PREDICATE

Usage: `nonrel_unknown_call(AbsInt,Sg,Vars,Call,Succ)`

Gives the “top” value for the variables involved in a literal whose definition is not present, and adds this top value to `Call`

- *The following properties should hold at call time:*

  `AbsInt` is currently a term which is not a free variable. (`nonvar/1`)
  `Sg` is currently a term which is not a free variable. (`nonvar/1`)
  `Vars` is currently a term which is not a free variable. (`nonvar/1`)
  `Call` is currently a term which is not a free variable. (`nonvar/1`)
  `Succ` is a free variable. (`var/1`)
  `AbsInt` is an atom. (`atm/1`)
  `Sg` is a term which represents a goal, i.e., an atom or a structure. (`cgoal/1`)
  `Vars` is a list. (`list/1`)
  `Call` is any term. (`term/1`)
  `Succ` is any term. (`term/1`)

nonrel_less_or_equal/3: PREDICATE

Usage: `nonrel_less_or_equal(AbsInt,ASub0,ASub1)`

Succeeds if `ASub1` is more general or equal to `ASub0`. it is assumed the two abstract substitutions are defined on the same variables

- *The following properties should hold at call time:*

  `AbsInt` is currently a term which is not a free variable. (`nonvar/1`)
  `ASub0` is currently a term which is not a free variable. (`nonvar/1`)
  `ASub1` is currently a term which is not a free variable. (`nonvar/1`)
  `AbsInt` is an atom. (`atm/1`)
  `ASub0` is any term. (`term/1`)
  `ASub1` is any term. (`term/1`)

nonrel_glb/4: PREDICATE

Usage: `nonrel_glb(AbsInt,ASub0,ASub1,Glb)`

`Glb` is the great lower bound of `ASub0` and `ASub1`, two substitutions that describe the same set of variables.

- *The following properties should hold at call time:*

  `AbsInt` is currently a term which is not a free variable. (`nonvar/1`)
  `ASub0` is currently a term which is not a free variable. (`nonvar/1`)
  `ASub1` is currently a term which is not a free variable. (`nonvar/1`)
  `Glb` is a free variable. (`var/1`)
  `AbsInt` is an atom. (`atm/1`)
  `ASub0` is a list. (`list/1`)
  `ASub1` is a list. (`list/1`)
  `Glb` is a free variable. (`var/1`)

nonrel_identical_abstract/2: PREDICATE

Usage: `nonrel_identical_abstract(ASub1,ASub2)`

- *The following properties should hold at call time:*

  `ASub1` is currently a term which is not a free variable. (`nonvar/1`)
  `ASub2` is currently a term which is not a free variable. (`nonvar/1`)

nonrel_widen/4: PREDICATE

Usage: `nonrel_widen(AbsInt,Asub1,ASub2,WAsub)`

- *The following properties should hold at call time:*

  `AbsInt` is currently a term which is not a free variable. (`nonvar/1`)
  `Asub1` is currently a term which is not a free variable. (`nonvar/1`)
  `ASub2` is currently a term which is not a free variable. (`nonvar/1`)
  `WAsub` is a free variable. (`var/1`)

nonrel_widencall/4: PREDICATE

Usage: `nonrel_widencall(AbsInt,Asub1,ASub2,WAsub)`

- *The following properties should hold at call time:*
  - `AbsInt` is currently a term which is not a free variable. (`nonvar/1`)
  - `Asub1` is currently a term which is not a free variable. (`nonvar/1`)
  - `ASub2` is currently a term which is not a free variable. (`nonvar/1`)
  - `WAsub` is a free variable. (`var/1`)

21.4 Documentation on imports

This module has the following direct dependencies:

- *Application modules:*
  `preprocess_flags`, `domains`.
- *System library modules:*
  `sets`, `sort`, `terms_check`, `terms_vars`.
- *Internal (engine) modules:*
  `term_basic`, `arithmetic`, `atomic_basic`, `basiccontrol`, `exceptions`, `term_compare`, `term_typing`, `debugger_support`, `basic_props`, `hiord_rt`.
- *Packages:*
  `prelude`, `initial`, `condcomp`, `assertions`, `assertions/assertions_basic`, `regtypes`, `basicmodes`, `aidomain`.

22 PD domain

This abstract domain is the domain with one value, top. PD stands for Partial Deduction.

22.1 Usage and interface

- **Library usage:**
  `:- use_module(domain(pd)).`

22.2 Documentation on multifiles

aidomain/1: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.init_abstract_domain/2: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.amgu/5: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.augment_asub/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.augment_two_asub/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.call_to_entry/10: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.exit_to_prime/8: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.project/6: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.widencall/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.needs/2: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.widen/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.compute_lub/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.compute_clauses_lub/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.compute_clauses_glb/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.fixpoint_covered/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.fixpoint_covered_gfp/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.identical_abstract/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.abs_sort/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.extend/6: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.less_or_equal/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.glb/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.eliminate_equivalent/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.abs_subset/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.call_to_success_fact/10: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.special_builtin/6: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.combined_special_builtin0/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.body_succ_builtin/9: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.split_combined_domain/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.success_builtin/7: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.call_to_success_builtin/7: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.obtain_info/5: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.input_interface/5: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.input_user_interface/6: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.asub_to_native/6: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.concrete/4: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.unknown_call/5: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.unknown_entry/4: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.empty_entry/4: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.part_conc/5: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.multi_part_conc/4: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.collect_auxinfo_asub/4: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.rename_auxinfo_asub/4: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.dom_statistics/2: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.contains_parameters/2: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

22.3 Documentation on imports

This module has the following direct dependencies:

- *Application modules:*
  sharefree, preprocess_flags, domains.
- *System library modules:*
  sets, sort.
- *Internal (engine) modules:*
  term_basic, arithmetic, atomic_basic, basiccontrol, exceptions, term_compare, term_typing, debugger_support, basic_props, hiord_rt.
- *Packages:*
  prelude, initial, condcomp, assertions, assertions/assertions_basic, regtypes, basicmodes, aidomain.

23 PD domain with bottom

This abstract domain is the domain with only two values, top and bottom. This simple improvement over the `pd` domain provides improvements, both in specialization time and quality of the specialized program if abstract specialization is then performed. PDB stands for Partial Deduction + Bottom.

23.1 Usage and interface

- **Library usage:**
  `:- use_module(domain(pdb)).`

23.2 Documentation on multifiles

aidomain/1: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.init_abstract_domain/2: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.amgu/5: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.augment_asub/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.augment_two_asub/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.call_to_entry/10: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.exit_to_prime/8: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.project/6: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.widencall/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.needs/2: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.widen/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.compute_lub/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.compute_clauses_lub/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.compute_clauses_glb/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.fixpoint_covered/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.fixpoint_covered_gfp/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.identical_abstract/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.abs_sort/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.extend/6: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.less_or_equal/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.glb/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.eliminate_equivalent/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.abs_subset/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.call_to_success_fact/10: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.special_builtin/6: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.combined_special_builtin0/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.body_succ_builtin/9: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.split_combined_domain/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.success_builtin/7: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.call_to_success_builtin/7: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.obtain_info/5: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.input_interface/5: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.input_user_interface/6: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.asub_to_native/6: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.concrete/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.unknown_call/5: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.unknown_entry/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.empty_entry/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.part_conc/5: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.multi_part_conc/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.collect_auxinfo_asub/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.rename_auxinfo_asub/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.dom_statistics/2: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.contains_parameters/2: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

23.3 Documentation on imports

This module has the following direct dependencies:

- *Application modules:*
  sharefree, preprocess_flags, domains.
- *System library modules:*
  sets, sort.
- *Internal (engine) modules:*
  term_basic, arithmetic, atomic_basic, basiccontrol, exceptions, term_compare, term_typing, debugger_support, basic_props, hiord_rt.
- *Packages:*
  prelude, initial, condcomp, assertions, assertions/assertions_basic, regtypes, basicmodes, aidomain.

24 polyhedra (library)

24.1 Usage and interface

- **Library usage:**
  `:- use_module(domain(polyhedra)).`
- **Exports:**
  - *Predicates:*
    `polyhedra_initialize/0`, `polyhedra_finalize/0`, `less_or_equal/2`, `input_user_interface/5`.
  - *Multifiles:*
    `aidomain/1`, `aidom.init_abstract_domain/2`, `aidom.amgu/5`, `aidom.augment_asub/4`, `aidom.augment_two_asub/4`, `aidom.call_to_entry/10`, `aidom.exit_to_prime/8`, `aidom.project/6`, `aidom.widencall/4`, `aidom.needs/2`, `aidom.widen/4`, `aidom.compute_lub/3`, `aidom.compute_clauses_lub/4`, `aidom.compute_clauses_glb/4`, `aidom.fixpoint_covered/3`, `aidom.fixpoint_covered_gfp/3`, `aidom.identical_abstract/3`, `aidom.abs_sort/3`, `aidom.extend/6`, `aidom.less_or_equal/3`, `aidom.glb/4`, `aidom.eliminate_equivalent/3`, `aidom.abs_subset/3`, `aidom.call_to_success_fact/10`, `aidom.special_builtin/6`, `aidom.combined_special_builtin0/3`, `aidom.body_succ_builtin/9`, `aidom.split_combined_domain/4`, `aidom.success_builtin/7`, `aidom.call_to_success_builtin/7`, `aidom.obtain_info/5`, `aidom.input_interface/5`, `aidom.input_user_interface/6`, `aidom.asub_to_native/6`, `aidom.concrete/4`, `aidom.unknown_call/5`, `aidom.unknown_entry/4`, `aidom.empty_entry/4`, `aidom.part_conc/5`, `aidom.multi_part_conc/4`, `aidom.collect_auxinfo_asub/4`, `aidom.rename_auxinfo_asub/4`, `aidom.dom_statistics/2`, `aidom.contains_parameters/2`.

24.2 Documentation on exports

polyhedra_initialize/0: PREDICATE
No further documentation available for this predicate.

polyhedra_finalize/0: PREDICATE
No further documentation available for this predicate.

less_or_equal/2: PREDICATE
No further documentation available for this predicate.

input_user_interface/5: PREDICATE
No further documentation available for this predicate.

24.3 Documentation on multifiles

aidomain/1: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.init_abstract_domain/2: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.amgu/5: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.augment_asub/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.augment_two_asub/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.call_to_entry/10: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.exit_to_prime/8: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.project/6: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.widencall/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.needs/2: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.widen/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.compute_lub/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.compute_clauses_lub/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.compute_clauses_glb/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.fixpoint_covered/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.fixpoint_covered_gfp/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.identical_abstract/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.abs_sort/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.extend/6: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.less_or_equal/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.glb/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.eliminate_equivalent/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.abs_subset/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.call_to_success_fact/10: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.special_builtin/6: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.combined_special_builtin0/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.body_succ_builtin/9: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.split_combined_domain/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.success_builtin/7: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.call_to_success_builtin/7: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.obtain_info/5: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.input_interface/5: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.input_user_interface/6: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.asub_to_native/6: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.concrete/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.unknown_call/5: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.unknown_entry/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.empty_entry/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.part_conc/5: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.multi_part_conc/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.collect_auxinfo_asub/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.rename_auxinfo_asub/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.dom_statistics/2: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.contains_parameters/2: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

24.4 Documentation on imports

This module has the following direct dependencies:

- *Application modules:*
  `ppl_interface`, `preprocess_flags`, `domains`.
- *System library modules:*
  `datafacts_rt`, `sets`, `lists`, `streams`, `write`, `sort`, `terms_vars`.
- *Internal (engine) modules:*
  `term_basic`, `arithmetic`, `atomic_basic`, `basiccontrol`, `exceptions`, `term_compare`, `term_typing`, `debugger_support`, `basic_props`, `runtime_control`.
- *Packages:*
  `prelude`, `initial`, `condcomp`, `datafacts`, `aidomain`.

25 ptypes: types with topological class widening (abstract domain)

Author(s): Claudio Vaucheret, Francisco Bueno, Ciao Development Team.

This module implements a types domain (based on `termsd.pl`) using the topological clash widening operator (`hentenwid/3`).

25.1 Usage and interface

- **Library usage:**
  `:- use_module(domain(ptypes)).`

25.2 Documentation on multifiles

aidomain/1: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.init_abstract_domain/2: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.amgu/5: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.augment_asub/4: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.augment_two_asub/4: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.call_to_entry/10: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.exit_to_prime/8: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.project/6: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.widencall/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.needs/2: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.widen/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.compute_lub/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.compute_clauses_lub/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.compute_clauses_glb/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.fixpoint_covered/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.fixpoint_covered_gfp/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.identical_abstract/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.abs_sort/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.extend/6: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.less_or_equal/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.glb/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.eliminate_equivalent/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.abs_subset/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.call_to_success_fact/10: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.special_builtin/6: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.combined_special_builtin0/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.body_succ_builtin/9: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.split_combined_domain/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.success_builtin/7: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.call_to_success_builtin/7: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.obtain_info/5: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.input_interface/5: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.input_user_interface/6: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.asub_to_native/6: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.concrete/4: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.unknown_call/5: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.unknown_entry/4: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.empty_entry/4: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.part_conc/5: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.multi_part_conc/4: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.collect_auxinfo_asub/4: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.rename_auxinfo_asub/4: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.dom_statistics/2: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.contains_parameters/2: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

25.3 Documentation on imports

This module has the following direct dependencies:

- *Application modules:*
  `preprocess_flags`, `termsd`, `domains`.
- *System library modules:*
  `typeslib`, `sets`, `sort`.
- *Internal (engine) modules:*
  `term_basic`, `arithmetic`, `atomic_basic`, `basiccontrol`, `exceptions`, `term_compare`, `term_typing`, `debugger_support`, `basic_props`, `hiord_rt`.
- *Packages:*
  `prelude`, `initial`, `condcomp`, `assertions`, `assertions/assertions_basic`, `regtypes`, `basicmodes`, `aidomain`.

26 sharing+freeness (abstract domain)

Author(s): Maria Garcia de la Banda, Francisco Bueno.

26.1 Usage and interface

- **Library usage:**
  `:- use_module(domain(sharefree)).`
- **Exports:**
  - *Predicates:*
    `needs/1`, `project_freeness/3`, `call_to_entry/9`, `exit_to_prime/7`, `abs_sort/2`, `compute_lub/2`, `compute_lub_el/3`, `compute_lub_sh/3`, `extend/5`, `call_to_success_fact/9`, `call_to_prime_fact/6`, `unknown_entry/3`, `empty_entry/3`, `asub_to_native/5`, `obtain_info/4`, `obtain_info/3`, `special_builtin/5`, `success_builtin/6`, `call_to_success_builtin/6`, `take_coupled/3`, `split_coupled/4`, `all_terms_identical/2`, `update_lambda_sf/5`, `update_lambda_non_free/5`, `partition_sf/4`, `prune/5`, `prune/4`, `covering/3`, `eliminate_non_element/4`, `product/8`, `insert_each/3`, `obtain_freeness/2`, `obtain_prime_var_var/3`, `sh_free_vars_compatible/2`, `make_dependence/5`, `unknown_call/4`, `values_equal/3`, `values_differ/3`, `change_values/4`, `change_values_if_not_g/4`, `change_values_if_f/4`, `member_value_freeness_differ/3`, `project_freeness_n/3`, `propagate_non_freeness/5`, `add_environment_vars/3`.
  - *Multifiles:*
    `aidomain/1`, `aidom.init_abstract_domain/2`, `aidom.amgu/5`, `aidom.augment_asub/4`, `aidom.augment_two_asub/4`, `aidom.call_to_entry/10`, `aidom.exit_to_prime/8`, `aidom.project/6`, `aidom.widencall/4`, `aidom.needs/2`, `aidom.widen/4`, `aidom.compute_lub/3`, `aidom.compute_clauses_lub/4`, `aidom.compute_clauses_glb/4`, `aidom.fixpoint_covered/3`, `aidom.fixpoint_covered_gfp/3`, `aidom.identical_abstract/3`, `aidom.abs_sort/3`, `aidom.extend/6`, `aidom.less_or_equal/3`, `aidom.glb/4`, `aidom.eliminate_equivalent/3`, `aidom.abs_subset/3`, `aidom.call_to_success_fact/10`, `aidom.special_builtin/6`, `aidom.combined_special_`
    `builtin0/3`, `aidom.body_succ_builtin/9`, `aidom.split_combined_domain/4`, `aidom.success_builtin/7`, `aidom.call_to_success_builtin/7`, `aidom.obtain_info/5`, `aidom.input_interface/5`, `aidom.input_user_interface/6`, `aidom.asub_to_native/6`, `aidom.concrete/4`, `aidom.unknown_call/5`, `aidom.unknown_entry/4`, `aidom.empty_entry/4`, `aidom.part_conc/5`, `aidom.multi_part_conc/4`, `aidom.collect_auxinfo_asub/4`, `aidom.rename_auxinfo_asub/4`, `aidom.dom_statistics/2`, `aidom.contains_parameters/2`.

26.2 Documentation on exports

needs/1: PREDICATE

No further documentation available for this predicate.

project/5: PREDICATE

No further documentation available for this predicate.

project_freeness/3: PREDICATE

No further documentation available for this predicate.

call_to_entry/9: PREDICATE

No further documentation available for this predicate.

exit_to_prime/7: PREDICATE

No further documentation available for this predicate.

abs_sort/2: PREDICATE

No further documentation available for this predicate.

compute_lub/2: PREDICATE

No further documentation available for this predicate.

compute_lub_el/3: PREDICATE

No further documentation available for this predicate.

compute_lub_sh/3: PREDICATE

No further documentation available for this predicate.

glb/3: PREDICATE

No further documentation available for this predicate.

extend/5: PREDICATE

No further documentation available for this predicate.

call_to_success_fact/9: PREDICATE

No further documentation available for this predicate.

call_to_prime_fact/6: PREDICATE

No further documentation available for this predicate.

unknown_entry/3: PREDICATE

No further documentation available for this predicate.

empty_entry/3: PREDICATE

Usage: `empty_entry(Sg,Vars,Entry)`

Gives the "empty" value in this domain for a given set of variables `Vars`, resulting in the abstract substitution `Entry`. I.e., obtains the abstraction of a substitution in which all variables `Vars` are unbound: free and unaliased. In Sh is the list of singleton lists of variables and in Fr is X/f forall X in the set of variables

– *The following properties should hold at call time:*

`Sg` is currently a term which is not a free variable. (`nonvar/1`)
`Vars` is currently a term which is not a free variable. (`nonvar/1`)
`Entry` is a free variable. (`var/1`)
`Sg` is a term which represents a goal, i.e., an atom or a structure. (`cgoal/1`)
`Vars` is a list. (`list/1`)
`Entry` is an abstract substitution (`absu/1`)

input_user_interface/5: PREDICATE

No further documentation available for this predicate.

input_interface/4: PREDICATE

No further documentation available for this predicate.

asub_to_native/5: PREDICATE

No further documentation available for this predicate.

obtain_info/4: PREDICATE

No further documentation available for this predicate.

obtain_info/3: PREDICATE

No further documentation available for this predicate.

less_or_equal/2: PREDICATE

No further documentation available for this predicate.

special_builtin/5: PREDICATE

No further documentation available for this predicate.

success_builtin/6: PREDICATE

No further documentation available for this predicate.

call_to_success_builtin/6: PREDICATE

No further documentation available for this predicate.

take_coupled/3: PREDICATE

No further documentation available for this predicate.

split_coupled/4: PREDICATE

No further documentation available for this predicate.

all_terms_identical/2: PREDICATE

No further documentation available for this predicate.

update_lambda_sf/5: PREDICATE

No further documentation available for this predicate.

update_lambda_non_free/5: PREDICATE

No further documentation available for this predicate.

partition_sf/4: PREDICATE

No further documentation available for this predicate.

prune/5: PREDICATE

No further documentation available for this predicate.

prune/4: PREDICATE

No further documentation available for this predicate.

covering/3: PREDICATE

No further documentation available for this predicate.

eliminate_non_element/4: PREDICATE

No further documentation available for this predicate.

product/8: PREDICATE

No further documentation available for this predicate.

insert_each/3: PREDICATE

No further documentation available for this predicate.

obtain_freeness/2: PREDICATE

No further documentation available for this predicate.

obtain_prime_var_var/3: PREDICATE

No further documentation available for this predicate.

sh_free_vars_compatible/2: PREDICATE

No further documentation available for this predicate.

make_dependence/5: PREDICATE

No further documentation available for this predicate.

unknown_call/4: PREDICATE

No further documentation available for this predicate.

values_equal/3: PREDICATE

No further documentation available for this predicate.

values_differ/3: PREDICATE

No further documentation available for this predicate.

change_values/4: PREDICATE

No further documentation available for this predicate.

change_values_if_not_g/4: PREDICATE

No further documentation available for this predicate.

change_values_if_f/4: PREDICATE

No further documentation available for this predicate.

member_value_freeness_differ/3: PREDICATE

No further documentation available for this predicate.

project_freeness_n/3: PREDICATE

No further documentation available for this predicate.

propagate_non_freeness/5: PREDICATE

No further documentation available for this predicate.

add_environment_vars/3: PREDICATE

No further documentation available for this predicate.

26.3 Documentation on multifiles

aidomain/1: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.init_abstract_domain/2: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.amgu/5: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.augment_asub/4: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.augment_two_asub/4: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.call_to_entry/10: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.exit_to_prime/8: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.project/6: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.widencall/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.needs/2: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.widen/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.compute_lub/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.compute_clauses_lub/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.compute_clauses_glb/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.fixpoint_covered/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.fixpoint_covered_gfp/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.identical_abstract/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.abs_sort/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.extend/6: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.less_or_equal/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.glb/4: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.eliminate_equivalent/3: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.abs_subset/3: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.call_to_success_fact/10: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.special_builtin/6: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.combined_special_builtin0/3: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.body_succ_builtin/9: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.split_combined_domain/4: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.success_builtin/7: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.call_to_success_builtin/7: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.obtain_info/5: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.input_interface/5: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.input_user_interface/6: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.asub_to_native/6: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.concrete/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.unknown_call/5: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.unknown_entry/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.empty_entry/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.part_conc/5: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.multi_part_conc/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.collect_auxinfo_asub/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.rename_auxinfo_asub/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.dom_statistics/2: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.contains_parameters/2: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

26.4 Documentation on imports

This module has the following direct dependencies:

- *Application modules:*
  `preprocess_flags`, `domains`, `sharefree_amgu`, `s_grshfr`, `share_aux`, `sharing`, `s_eqs`.
- *System library modules:*
  `datafacts_rt`, `sets`, `sort`, `lists`, `lsets`, `terms_check`, `terms_vars`.
- *Internal (engine) modules:*
  `term_basic`, `arithmetic`, `atomic_basic`, `basiccontrol`, `exceptions`, `term_compare`, `term_typing`, `debugger_support`, `basic_props`, `hiord_rt`, `io_basic`.
- *Packages:*
  `prelude`, `initial`, `condcomp`, `assertions`, `assertions/assertions_basic`, `regtypes`, `isomodes`, `datafacts`, `aidomain`.

27 amgu-based sharing+freeness domain

Author(s): Jorge Navas.

27.1 Usage and interface

- **Library usage:**

  `:- use_module(domain(sharefree_amgu)).`
- **Exports:**
  - *Predicates:*

    `call_to_entry/9`, `exit_to_prime/7`, `amgu/4`, `augment_asub0/3`, `call_to_success_fact/9`, `sharefree_delete_variables/3`, `call_to_prime_fact/6`.
  - *Multifiles:*

    `aidomain/1`, `aidom.init_abstract_domain/2`, `aidom.amgu/5`, `aidom.augment_asub/4`, `aidom.augment_two_asub/4`, `aidom.call_to_entry/10`, `aidom.exit_to_prime/8`, `aidom.project/6`, `aidom.widencall/4`, `aidom.needs/2`, `aidom.widen/4`, `aidom.compute_lub/3`, `aidom.compute_clauses_lub/4`, `aidom.compute_clauses_glb/4`, `aidom.fixpoint_covered/3`, `aidom.fixpoint_covered_gfp/3`, `aidom.identical_abstract/3`, `aidom.abs_sort/3`, `aidom.extend/6`, `aidom.less_or_equal/3`, `aidom.glb/4`, `aidom.eliminate_equivalent/3`, `aidom.abs_subset/3`, `aidom.call_to_success_fact/10`, `aidom.special_builtin/6`, `aidom.combined_special_builtin0/3`, `aidom.body_succ_builtin/9`, `aidom.split_combined_domain/4`, `aidom.success_builtin/7`, `aidom.call_to_success_builtin/7`, `aidom.obtain_info/5`, `aidom.input_interface/5`, `aidom.input_user_interface/6`, `aidom.asub_to_native/6`, `aidom.concrete/4`, `aidom.unknown_call/5`, `aidom.unknown_entry/4`, `aidom.empty_entry/4`, `aidom.part_conc/5`, `aidom.multi_part_conc/4`, `aidom.collect_auxinfo_asub/4`, `aidom.rename_auxinfo_asub/4`, `aidom.dom_statistics/2`, `aidom.contains_parameters/2`.

27.2 Documentation on exports

call_to_entry/9: PREDICATE

No further documentation available for this predicate.

exit_to_prime/7: PREDICATE

No further documentation available for this predicate.

amgu/4: PREDICATE

No further documentation available for this predicate.

augment_asub/3: PREDICATE

No further documentation available for this predicate.

augment_asub0/3: PREDICATE
No further documentation available for this predicate.

call_to_success_fact/9: PREDICATE
No further documentation available for this predicate.

sharefree_delete_variables/3: PREDICATE
No further documentation available for this predicate.

call_to_prime_fact/6: PREDICATE
No further documentation available for this predicate.

special_builtin/5: PREDICATE
No further documentation available for this predicate.

success_builtin/6: PREDICATE
No further documentation available for this predicate.

call_to_success_builtin/6: PREDICATE
No further documentation available for this predicate.

27.3 Documentation on multifiles

aidomain/1: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.init_abstract_domain/2: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.amgu/5: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.augment_asub/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.augment_two_asub/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.call_to_entry/10: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.exit_to_prime/8: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.project/6: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.widencall/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.needs/2: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.widen/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.compute_lub/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.compute_clauses_lub/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.compute_clauses_glb/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.fixpoint_covered/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.fixpoint_covered_gfp/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.identical_abstract/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.abs_sort/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.extend/6: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.less_or_equal/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.glb/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.eliminate_equivalent/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.abs_subset/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.call_to_success_fact/10: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.special_builtin/6: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.combined_special_builtin0/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.body_succ_builtin/9: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.split_combined_domain/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.success_builtin/7: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.call_to_success_builtin/7: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.obtain_info/5: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.input_interface/5: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.input_user_interface/6: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.asub_to_native/6: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.concrete/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.unknown_call/5: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.unknown_entry/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.empty_entry/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.part_conc/5: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.multi_part_conc/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.collect_auxinfo_asub/4: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.rename_auxinfo_asub/4: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.dom_statistics/2: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.contains_parameters/2: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

27.4 Documentation on imports

This module has the following direct dependencies:

- *Application modules:*
  `sharefree`, `preprocess_flags`, `domains`, `share_amgu_sets`, `s_grshfr`, `share_aux`, `sharing`, `sharing_amgu`, `sharefree_amgu_aux`.
- *System library modules:*
  `sets`, `sort`, `terms_vars`, `lists`, `lsets`, `terms_check`.
- *Internal (engine) modules:*
  `term_basic`, `arithmetic`, `atomic_basic`, `basiccontrol`, `exceptions`, `term_compare`, `term_typing`, `debugger_support`, `basic_props`, `hiord_rt`.
- *Packages:*
  `prelude`, `initial`, `condcomp`, `assertions`, `assertions/assertions_basic`, `isomodes`, `aidomain`.

28 CLIQUE-Sharing+Freeness domain

Author(s): Jorge Navas.

28.1 Usage and interface

- **Library usage:**
  `:- use_module(domain(sharefree_clique)).`
- **Exports:**
  - *Predicates:*
    `call_to_entry/9`, `exit_to_prime/7`, `identical_abstract/2`, `call_to_success_fact/9`, `compute_lub_el/3`, `unknown_call/4`, `empty_entry/3`.
  - *Multifiles:*
    `aidomain/1`, `aidom.init_abstract_domain/2`, `aidom.amgu/5`, `aidom.augment_asub/4`, `aidom.augment_two_asub/4`, `aidom.call_to_entry/10`, `aidom.exit_to_prime/8`, `aidom.project/6`, `aidom.widencall/4`, `aidom.needs/2`, `aidom.widen/4`, `aidom.compute_lub/3`, `aidom.compute_clauses_lub/4`, `aidom.compute_clauses_glb/4`, `aidom.fixpoint_covered/3`, `aidom.fixpoint_covered_gfp/3`, `aidom.identical_abstract/3`, `aidom.abs_sort/3`, `aidom.extend/6`, `aidom.less_or_equal/3`, `aidom.glb/4`, `aidom.eliminate_equivalent/3`, `aidom.abs_subset/3`, `aidom.call_to_success_fact/10`, `aidom.special_builtin/6`, `aidom.combined_special_`
    `builtin0/3`, `aidom.body_succ_builtin/9`, `aidom.split_combined_domain/4`, `aidom.success_builtin/7`, `aidom.call_to_success_builtin/7`, `aidom.obtain_info/5`, `aidom.input_interface/5`, `aidom.input_user_interface/6`, `aidom.asub_to_native/6`, `aidom.concrete/4`, `aidom.unknown_call/5`, `aidom.unknown_entry/4`, `aidom.empty_entry/4`, `aidom.part_conc/5`, `aidom.multi_part_conc/4`, `aidom.collect_auxinfo_asub/4`, `aidom.rename_auxinfo_asub/4`, `aidom.dom_statistics/2`, `aidom.contains_parameters/2`.

28.2 Documentation on exports

call_to_entry/9: PREDICATE
No further documentation available for this predicate.

exit_to_prime/7: PREDICATE
No further documentation available for this predicate.

extend/5: PREDICATE
No further documentation available for this predicate.

project/5: PREDICATE
No further documentation available for this predicate.

abs_sort/2: PREDICATE

No further documentation available for this predicate.

identical_abstract/2: PREDICATE

No further documentation available for this predicate.

less_or_equal/2: PREDICATE

No further documentation available for this predicate.

call_to_success_fact/9: PREDICATE

No further documentation available for this predicate.

compute_lub_el/3: PREDICATE

No further documentation available for this predicate.

glb/3: PREDICATE

No further documentation available for this predicate.

input_user_interface/5: PREDICATE

No further documentation available for this predicate.

input_interface/4: PREDICATE

No further documentation available for this predicate.

unknown_call/4: PREDICATE

No further documentation available for this predicate.

empty_entry/3: PREDICATE

No further documentation available for this predicate.

special_builtin/5: PREDICATE

No further documentation available for this predicate.

28.3 Documentation on multifiles

aidomain/1: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.init_abstract_domain/2: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.amgu/5: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.augment_asub/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.augment_two_asub/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.call_to_entry/10: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.exit_to_prime/8: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.project/6: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.widencall/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.needs/2: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.widen/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.compute_lub/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.compute_clauses_lub/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.compute_clauses_glb/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.fixpoint_covered/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.fixpoint_covered_gfp/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.identical_abstract/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.abs_sort/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.extend/6: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.less_or_equal/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.glb/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.eliminate_equivalent/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.abs_subset/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.call_to_success_fact/10: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.special_builtin/6: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.combined_special_builtin0/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.body_succ_builtin/9: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.split_combined_domain/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.success_builtin/7: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.call_to_success_builtin/7: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.obtain_info/5: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.input_interface/5: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.input_user_interface/6: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.asub_to_native/6: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.concrete/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.unknown_call/5: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.unknown_entry/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.empty_entry/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.part_conc/5: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.multi_part_conc/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.collect_auxinfo_asub/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.rename_auxinfo_asub/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.dom_statistics/2: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.contains_parameters/2: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

28.4 Documentation on imports

This module has the following direct dependencies:

- *Application modules:*
  `sharefree`, `preprocess_flags`, `domains`, `share_amgu_sets`, `sharing_clique`, `share_clique_aux`, `s_grshfr`, `sharefree_amgu`, `sharefree_clique_aux`, `sharefree_amgu_aux`, `share_aux`.
- *System library modules:*
  `sets`, `sort`, `lsets`, `terms_check`, `lists`, `terms_vars`.
- *Internal (engine) modules:*
  `term_basic`, `arithmetic`, `atomic_basic`, `basiccontrol`, `exceptions`, `term_compare`, `term_typing`, `debugger_support`, `basic_props`, `hiord_rt`.
- *Packages:*
  `prelude`, `initial`, `condcomp`, `assertions`, `assertions/assertions_basic`, `isomodes`, `aidomain`.

29 CLIQUE-Sharing+Freeness+Def domain

Author(s): Jorge Navas.

29.1 Usage and interface

- **Library usage:**
  `:- use_module(domain(sharefree_clique_def)).`

29.2 Documentation on multifiles

aidomain/1: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.init_abstract_domain/2: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.amgu/5: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.augment_asub/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.augment_two_asub/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.call_to_entry/10: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.exit_to_prime/8: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.project/6: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.widencall/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.needs/2: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.widen/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.compute_lub/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.compute_clauses_lub/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.compute_clauses_glb/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.fixpoint_covered/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.fixpoint_covered_gfp/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.identical_abstract/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.abs_sort/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.extend/6: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.less_or_equal/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.glb/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.eliminate_equivalent/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.abs_subset/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.call_to_success_fact/10: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.special_builtin/6: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.combined_special_builtin0/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.body_succ_builtin/9: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.split_combined_domain/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.success_builtin/7: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.call_to_success_builtin/7: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.obtain_info/5: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.input_interface/5: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.input_user_interface/6: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.asub_to_native/6: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.concrete/4: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.unknown_call/5: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.unknown_entry/4: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.empty_entry/4: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.part_conc/5: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.multi_part_conc/4: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.collect_auxinfo_asub/4: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.rename_auxinfo_asub/4: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.dom_statistics/2: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.contains_parameters/2: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

29.3 Documentation on imports

This module has the following direct dependencies:

- *Application modules:*
  `preprocess_flags`, `domains`, `s_grshfr`, `share_clique_aux`, `share_aux`, `sharefree_clique`, `def`.
- *System library modules:*
  `sets`, `sort`, `messages`.
- *Internal (engine) modules:*
  `term_basic`, `arithmetic`, `atomic_basic`, `basiccontrol`, `exceptions`, `term_compare`, `term_typing`, `debugger_support`, `basic_props`, `hiord_rt`.
- *Packages:*
  `prelude`, `initial`, `condcomp`, `assertions`, `assertions/assertions_basic`, `isomodes`, `aidomain`.

30 sharing+freeness+nonvar (abstract domain)

Author(s): Maria Garcia de la Banda.

30.1 Usage and interface

- **Library usage:**
  `:- use_module(domain(sharefree_non_var)).`

30.2 Documentation on multifiles

aidomain/1: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.init_abstract_domain/2: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.amgu/5: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.augment_asub/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.augment_two_asub/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.call_to_entry/10: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.exit_to_prime/8: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.project/6: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.widencall/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.needs/2: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.widen/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.compute_lub/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.compute_clauses_lub/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.compute_clauses_glb/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.fixpoint_covered/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.fixpoint_covered_gfp/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.identical_abstract/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.abs_sort/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.extend/6: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.less_or_equal/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.glb/4: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.eliminate_equivalent/3: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.abs_subset/3: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.call_to_success_fact/10: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.special_builtin/6: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.combined_special_builtin0/3: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.body_succ_builtin/9: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.split_combined_domain/4: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.success_builtin/7: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.call_to_success_builtin/7: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.obtain_info/5: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.input_interface/5: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.input_user_interface/6: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.asub_to_native/6: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.concrete/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.unknown_call/5: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.unknown_entry/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.empty_entry/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.part_conc/5: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.multi_part_conc/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.collect_auxinfo_asub/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.rename_auxinfo_asub/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.dom_statistics/2: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.contains_parameters/2: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

30.3 Documentation on imports

This module has the following direct dependencies:

- *Application modules:*
  preprocess_flags, domains, sharefree, sharing, s_grshfr, s_eqs, share_aux, plai_errors.
- *System library modules:*
  sets, sort, terms_vars, lists, lsets, terms_check.
- *Internal (engine) modules:*
  term_basic, arithmetic, atomic_basic, basiccontrol, exceptions, term_compare, term_typing, debugger_support, basic_props, hiord_rt.
- *Packages:*
  prelude, initial, condcomp, assertions, assertions/assertions_basic, isomodes, aidomain.

31 shareson (library)

31.1 Usage and interface

- **Library usage:**
  `:- use_module(domain(shareson)).`

31.2 Documentation on multifiles

aidomain/1: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.init_abstract_domain/2: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.amgu/5: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.augment_asub/4: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.augment_two_asub/4: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.call_to_entry/10: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.exit_to_prime/8: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.project/6: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.widencall/4: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.needs/2: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.widen/4: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.compute_lub/3: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.compute_clauses_lub/4: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.compute_clauses_glb/4: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.fixpoint_covered/3: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.fixpoint_covered_gfp/3: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.identical_abstract/3: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.abs_sort/3: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.extend/6: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.less_or_equal/3: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.glb/4: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.eliminate_equivalent/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.abs_subset/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.call_to_success_fact/10: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.special_builtin/6: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.combined_special_builtin0/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.body_succ_builtin/9: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.split_combined_domain/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.success_builtin/7: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.call_to_success_builtin/7: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.obtain_info/5: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.input_interface/5: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.input_user_interface/6: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.asub_to_native/6: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.concrete/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.unknown_call/5: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.unknown_entry/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.empty_entry/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.part_conc/5: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.multi_part_conc/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.collect_auxinfo_asub/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.rename_auxinfo_asub/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.dom_statistics/2: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.contains_parameters/2: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

31.3 Documentation on imports

This module has the following direct dependencies:

- *Application modules:*
  `preprocess_flags`, `domains`, `sharing`, `sondergaard`, `s_grshfr`, `share_aux`, `plai_errors`.
- *System library modules:*
  `sets`, `sort`, `llists`, `lsets`, `terms_vars`.
- *Internal (engine) modules:*
  `term_basic`, `arithmetic`, `atomic_basic`, `basiccontrol`, `exceptions`, `term_compare`, `term_typing`, `debugger_support`.
- *Packages:*
  `prelude`, `initial`, `condcomp`, `aidomain`.

32 sharing (abstract domain)

Author(s): Kalyan Muthukumar, Maria Garcia de la Banda, Francisco Bueno.

32.1 Usage and interface

- **Library usage:**
  `:- use_module(domain(sharing)).`
- **Exports:**
  - *Predicates:*
    `project/5`, `project_share/3`, `call_to_entry/9`, `exit_to_prime/7`, `abs_sort/2`, `compute_lub/2`, `lub/3`, `glb/3`, `extend/5`, `call_to_success_fact/9`, `call_to_prime_fact/6`, `pos/4`, `script_p_star/3`, `script_p/3`, `unknown_entry/3`, `empty_entry/3`, `asub_to_native/5`, `input_user_interface/5`, `input_interface/4`, `unknown_call/4`, `less_or_equal/2`, `special_builtin/5`, `success_builtin/6`, `call_to_success_builtin/6`.
  - *Multifiles:*
    `aidomain/1`, `aidom.init_abstract_domain/2`, `aidom.amgu/5`, `aidom.augment_asub/4`, `aidom.augment_two_asub/4`, `aidom.call_to_entry/10`, `aidom.exit_to_prime/8`, `aidom.project/6`, `aidom.widencall/4`, `aidom.needs/2`, `aidom.widen/4`, `aidom.compute_lub/3`, `aidom.compute_clauses_lub/4`, `aidom.compute_clauses_glb/4`, `aidom.fixpoint_covered/3`, `aidom.fixpoint_covered_gfp/3`, `aidom.identical_abstract/3`, `aidom.abs_sort/3`, `aidom.extend/6`, `aidom.less_or_equal/3`, `aidom.glb/4`, `aidom.eliminate_equivalent/3`, `aidom.abs_subset/3`, `aidom.call_to_success_fact/10`, `aidom.special_builtin/6`, `aidom.combined_special_builtin0/3`, `aidom.body_succ_builtin/9`, `aidom.split_combined_domain/4`, `aidom.success_builtin/7`, `aidom.call_to_success_builtin/7`, `aidom.obtain_info/5`, `aidom.input_interface/5`, `aidom.input_user_interface/6`, `aidom.asub_to_native/6`, `aidom.concrete/4`, `aidom.unknown_call/5`, `aidom.unknown_entry/4`, `aidom.empty_entry/4`, `aidom.part_conc/5`, `aidom.multi_part_conc/4`, `aidom.collect_auxinfo_asub/4`, `aidom.rename_auxinfo_asub/4`, `aidom.dom_statistics/2`, `aidom.contains_parameters/2`.

32.2 Documentation on exports

project/5: PREDICATE

No further documentation available for this predicate.

project_share/3: PREDICATE

No further documentation available for this predicate.

call_to_entry/9: PREDICATE

No further documentation available for this predicate.

exit_to_prime/7: PREDICATE

No further documentation available for this predicate.

abs_sort/2: PREDICATE

No further documentation available for this predicate.

compute_lub/2: PREDICATE

No further documentation available for this predicate.

lub/3: PREDICATE

No further documentation available for this predicate.

glb/3: PREDICATE

No further documentation available for this predicate.

extend/5: PREDICATE

No further documentation available for this predicate.

call_to_success_fact/9: PREDICATE

No further documentation available for this predicate.

call_to_prime_fact/6: PREDICATE

No further documentation available for this predicate.

pos/4: PREDICATE

No further documentation available for this predicate.

script_p_star/3: PREDICATE

No further documentation available for this predicate.

script_p/3: PREDICATE

No further documentation available for this predicate.

unknown_entry/3: PREDICATE

No further documentation available for this predicate.

empty_entry/3: PREDICATE

Usage: `empty_entry(Sg,Vars,Entry)`

Gives the "empty" value in this domain for a given set of variables `Vars`, resulting in the abstract substitution `Entry`. I.e., obtains the abstraction of a substitution in which all variables `Vars` are unbound: free and unaliased. In this domain is the list of singleton lists of variables

- *The following properties should hold at call time:*

  `Sg` is currently a term which is not a free variable. (`nonvar/1`)

  `Vars` is currently a term which is not a free variable. (`nonvar/1`)

  `Entry` is a free variable. (`var/1`)

  `Sg` is a term which represents a goal, i.e., an atom or a structure. (`cgoal/1`)

  `Vars` is a list. (`list/1`)

  `Entry` is an abstract substitution (`absu/1`)

asub_to_native/5: PREDICATE

No further documentation available for this predicate.

input_user_interface/5: PREDICATE

No further documentation available for this predicate.

input_interface/4: PREDICATE

No further documentation available for this predicate.

unknown_call/4: PREDICATE

No further documentation available for this predicate.

less_or_equal/2: PREDICATE

No further documentation available for this predicate.

special_builtin/5: PREDICATE

No further documentation available for this predicate.

success_builtin/6: PREDICATE

No further documentation available for this predicate.

call_to_success_builtin/6: PREDICATE

No further documentation available for this predicate.

32.3 Documentation on multifiles

aidomain/1: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.init_abstract_domain/2: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.amgu/5: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.augment_asub/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.augment_two_asub/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.call_to_entry/10: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.exit_to_prime/8: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.project/6: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.widencall/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.needs/2: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.widen/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.compute_lub/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.compute_clauses_lub/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.compute_clauses_glb/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.fixpoint_covered/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.fixpoint_covered_gfp/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.identical_abstract/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.abs_sort/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.extend/6: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.less_or_equal/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.glb/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.eliminate_equivalent/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.abs_subset/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.call_to_success_fact/10: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.special_builtin/6: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.combined_special_builtin0/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.body_succ_builtin/9: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.split_combined_domain/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.success_builtin/7: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.call_to_success_builtin/7: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.obtain_info/5: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.input_interface/5: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.input_user_interface/6: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.asub_to_native/6: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.concrete/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.unknown_call/5: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.unknown_entry/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.empty_entry/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.part_conc/5: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.multi_part_conc/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.collect_auxinfo_asub/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.rename_auxinfo_asub/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.dom_statistics/2: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.contains_parameters/2: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

32.4 Documentation on imports

This module has the following direct dependencies:

– *Application modules:*
  `preprocess_flags`, `domains`, `sharing_amgu`, `s_grshfr`, `eterms`, `share_aux`.
– *System library modules:*
  `sets`, `sort`, `lsets`, `terms_check`, `terms_vars`, `lists`, `typeslib`.
– *Internal (engine) modules:*
  `term_basic`, `arithmetic`, `atomic_basic`, `basiccontrol`, `exceptions`, `term_compare`, `term_typing`, `debugger_support`, `basic_props`, `hiord_rt`.
– *Packages:*
  `prelude`, `initial`, `condcomp`, `assertions`, `assertions/assertions_basic`, `regtypes`, `isomodes`, `aidomain`.

33 amgu-based sharing domain

Author(s): Jorge Navas.

33.1 Usage and interface

- **Library usage:**
  `:- use_module(domain(sharing_amgu)).`
- **Exports:**
  - *Predicates:*
    `call_to_entry/9`, `augment_asub/3`, `augment_two_asub/3`.
  - *Multifiles:*
    `aidomain/1`, `aidom.init_abstract_domain/2`, `aidom.amgu/5`, `aidom.augment_asub/4`, `aidom.augment_two_asub/4`, `aidom.call_to_entry/10`, `aidom.exit_to_prime/8`, `aidom.project/6`, `aidom.widencall/4`, `aidom.needs/2`, `aidom.widen/4`, `aidom.compute_lub/3`, `aidom.compute_clauses_lub/4`, `aidom.compute_clauses_glb/4`, `aidom.fixpoint_covered/3`, `aidom.fixpoint_covered_gfp/3`, `aidom.identical_abstract/3`, `aidom.abs_sort/3`, `aidom.extend/6`, `aidom.less_or_equal/3`, `aidom.glb/4`, `aidom.eliminate_equivalent/3`, `aidom.abs_subset/3`, `aidom.call_to_success_fact/10`, `aidom.special_builtin/6`, `aidom.combined_special_builtin0/3`, `aidom.body_succ_builtin/9`, `aidom.split_combined_domain/4`, `aidom.success_builtin/7`, `aidom.call_to_success_builtin/7`, `aidom.obtain_info/5`, `aidom.input_interface/5`, `aidom.input_user_interface/6`, `aidom.asub_to_native/6`, `aidom.concrete/4`, `aidom.unknown_call/5`, `aidom.unknown_entry/4`, `aidom.empty_entry/4`, `aidom.part_conc/5`, `aidom.multi_part_conc/4`, `aidom.collect_auxinfo_asub/4`, `aidom.rename_auxinfo_asub/4`, `aidom.dom_statistics/2`, `aidom.contains_parameters/2`.

33.2 Documentation on exports

call_to_entry/9: PREDICATE

No further documentation available for this predicate.

amgu/4: PREDICATE

No further documentation available for this predicate.

augment_asub/3: PREDICATE

No further documentation available for this predicate.

augment_two_asub/3: PREDICATE

No further documentation available for this predicate.

33.3 Documentation on multifiles

aidomain/1: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.init_abstract_domain/2: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.amgu/5: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.augment_asub/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.augment_two_asub/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.call_to_entry/10: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.exit_to_prime/8: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.project/6: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.widencall/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.needs/2: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.widen/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.compute_lub/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.compute_clauses_lub/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.compute_clauses_glb/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.fixpoint_covered/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.fixpoint_covered_gfp/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.identical_abstract/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.abs_sort/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.extend/6: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.less_or_equal/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.glb/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.eliminate_equivalent/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.abs_subset/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.call_to_success_fact/10: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.special_builtin/6: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.combined_special_builtin0/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.body_succ_builtin/9: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.split_combined_domain/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.success_builtin/7: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.call_to_success_builtin/7: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.obtain_info/5: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.input_interface/5: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.input_user_interface/6: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.asub_to_native/6: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.concrete/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.unknown_call/5: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.unknown_entry/4: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.empty_entry/4: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.part_conc/5: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.multi_part_conc/4: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.collect_auxinfo_asub/4: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.rename_auxinfo_asub/4: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.dom_statistics/2: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.contains_parameters/2: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

33.4 Documentation on imports

This module has the following direct dependencies:

- *Application modules:*
  `preprocess_flags`, `domains`, `sharing`, `s_grshfr`, `share_amgu_sets`, `share_amgu_aux`.
- *System library modules:*
  `sets`, `sort`, `terms_vars`, `lsets`, `lists`, `terms_check`.
- *Internal (engine) modules:*
  `term_basic`, `arithmetic`, `atomic_basic`, `basiccontrol`, `exceptions`, `term_compare`, `term_typing`, `debugger_support`, `basic_props`, `hiord_rt`.
- *Packages:*
  `prelude`, `initial`, `condcomp`, `assertions`, `assertions/assertions_basic`, `isomodes`, `aidomain`.

34 CLIQUE-Sharing domain

Author(s): Jorge Navas.

34.1 Usage and interface

- **Library usage:**
  `:- use_module(domain(sharing_clique)).`
- **Exports:**
  - *Predicates:*
    `call_to_entry/9`, `exit_to_prime/7`, `extend/5`, `asub_gt/2`, `prune_success/5`, `sharing_possible/4`, `powerset_with_empty_set/2`, `augment_asub/3`, `abs_sort/2`, `identical_abstract/2`, `eliminate_equivalent/2`, `less_or_equal/2`, `clique_part_less_or_equal/2`, `sharing_part_less_or_equal/3`, `call_to_success_fact/9`, `call_to_prime_fact/6`, `compute_lub/2`, `share_clique_lub_cl/3`, `glb/3`, `may_be_var/2`, `myappend/3`, `asub_to_native/5`, `unknown_call/4`, `unknown_entry/3`, `widen/4`, `widen/5`, `success_builtin/6`, `call_to_success_builtin/6`, `clique_make_decomposition/5`, `eliminate_couples_clique/4`, `share_make_reduction/5`, `projected_gvars_clique/3`, `compute_upper_amgu/2`.
  - *Multifiles:*
    `aidomain/1`, `aidom.init_abstract_domain/2`, `aidom.amgu/5`, `aidom.augment_asub/4`, `aidom.augment_two_asub/4`, `aidom.call_to_entry/10`, `aidom.exit_to_prime/8`, `aidom.project/6`, `aidom.widencall/4`, `aidom.needs/2`, `aidom.widen/4`, `aidom.compute_lub/3`, `aidom.compute_clauses_lub/4`, `aidom.compute_clauses_glb/4`, `aidom.fixpoint_covered/3`, `aidom.fixpoint_covered_gfp/3`, `aidom.identical_abstract/3`, `aidom.abs_sort/3`, `aidom.extend/6`, `aidom.less_or_equal/3`, `aidom.glb/4`, `aidom.eliminate_equivalent/3`, `aidom.abs_subset/3`, `aidom.call_to_success_fact/10`, `aidom.special_builtin/6`, `aidom.combined_special_builtin0/3`, `aidom.body_succ_builtin/9`, `aidom.split_combined_domain/4`, `aidom.success_builtin/7`, `aidom.call_to_success_builtin/7`, `aidom.obtain_info/5`, `aidom.input_interface/5`, `aidom.input_user_interface/6`, `aidom.asub_to_native/6`, `aidom.concrete/4`, `aidom.unknown_call/5`, `aidom.unknown_entry/4`, `aidom.empty_entry/4`, `aidom.part_conc/5`, `aidom.multi_part_conc/4`, `aidom.collect_auxinfo_asub/4`, `aidom.rename_auxinfo_asub/4`, `aidom.dom_statistics/2`, `aidom.contains_parameters/2`.

34.2 Documentation on exports

call_to_entry/9: PREDICATE

No further documentation available for this predicate.

exit_to_prime/7: PREDICATE

No further documentation available for this predicate.

amgu/4: PREDICATE

No further documentation available for this predicate.

extend/5: PREDICATE

No further documentation available for this predicate.

asub_gt/2: PREDICATE

No further documentation available for this predicate.

prune_success/5: PREDICATE

No further documentation available for this predicate.

sharing_possible/4: PREDICATE

No further documentation available for this predicate.

powerset_with_empty_set/2: PREDICATE

No further documentation available for this predicate.

augment_asub/3: PREDICATE

No further documentation available for this predicate.

project/5: PREDICATE

No further documentation available for this predicate.

abs_sort/2: PREDICATE

No further documentation available for this predicate.

identical_abstract/2: PREDICATE

No further documentation available for this predicate.

eliminate_equivalent/2: PREDICATE

No further documentation available for this predicate.

less_or_equal/2: PREDICATE

No further documentation available for this predicate.

clique_part_less_or_equal/2: PREDICATE
No further documentation available for this predicate.

sharing_part_less_or_equal/3: PREDICATE
No further documentation available for this predicate.

call_to_success_fact/9: PREDICATE
No further documentation available for this predicate.

call_to_prime_fact/6: PREDICATE
No further documentation available for this predicate.

compute_lub/2: PREDICATE
No further documentation available for this predicate.

share_clique_lub_cl/3: PREDICATE
No further documentation available for this predicate.

glb/3: PREDICATE
No further documentation available for this predicate.

input_user_interface/5: PREDICATE
No further documentation available for this predicate.

input_interface/4: PREDICATE
No further documentation available for this predicate.

may_be_var/2: PREDICATE
No further documentation available for this predicate.

myappend/3: PREDICATE
No further documentation available for this predicate.

asub_to_native/5: PREDICATE
No further documentation available for this predicate.

unknown_call/4: PREDICATE
No further documentation available for this predicate.

empty_entry/3: PREDICATE
No further documentation available for this predicate.

unknown_entry/3: PREDICATE
No further documentation available for this predicate.

widen/4: PREDICATE
No further documentation available for this predicate.

widen/5: PREDICATE
No further documentation available for this predicate.

special_builtin/5: PREDICATE
No further documentation available for this predicate.

success_builtin/6: PREDICATE
No further documentation available for this predicate.

call_to_success_builtin/6: PREDICATE
No further documentation available for this predicate.

clique_make_decomposition/5: PREDICATE
No further documentation available for this predicate.

eliminate_couples_clique/4: PREDICATE
No further documentation available for this predicate.

share_make_reduction/5: PREDICATE
No further documentation available for this predicate.

projected_gvars_clique/3: PREDICATE
No further documentation available for this predicate.

compute_upper_amgu/2: PREDICATE
No further documentation available for this predicate.

34.3 Documentation on multifiles

aidomain/1: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.init_abstract_domain/2: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.amgu/5: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.augment_asub/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.augment_two_asub/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.call_to_entry/10: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.exit_to_prime/8: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.project/6: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.widencall/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.needs/2: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.widen/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.compute_lub/3: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.compute_clauses_lub/4: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.compute_clauses_glb/4: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.fixpoint_covered/3: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.fixpoint_covered_gfp/3: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.identical_abstract/3: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.abs_sort/3: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.extend/6: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.less_or_equal/3: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.glb/4: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.eliminate_equivalent/3: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.abs_subset/3: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.call_to_success_fact/10: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.special_builtin/6: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.combined_special_builtin0/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.body_succ_builtin/9: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.split_combined_domain/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.success_builtin/7: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.call_to_success_builtin/7: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.obtain_info/5: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.input_interface/5: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.input_user_interface/6: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.asub_to_native/6: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.concrete/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.unknown_call/5: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.unknown_entry/4: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.empty_entry/4: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.part_conc/5: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.multi_part_conc/4: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.collect_auxinfo_asub/4: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.rename_auxinfo_asub/4: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.dom_statistics/2: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.contains_parameters/2: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

34.4 Documentation on imports

This module has the following direct dependencies:

- *Application modules:*
  `preprocess_flags`, `domains`, `share_amgu_sets`, `s_grshfr`, `share_clique_aux`, `share_aux`, `share_clique_1_aux`, `sharing`, `share_amgu_aux`.
- *System library modules:*
  `sets`, `sort`, `lsets`, `lists`, `terms_vars`, `terms_check`, `aggregates`, `messages`.
- *Internal (engine) modules:*
  `term_basic`, `arithmetic`, `atomic_basic`, `basiccontrol`, `exceptions`, `term_compare`, `term_typing`, `debugger_support`, `basic_props`, `hiord_rt`.
- *Packages:*
  `prelude`, `initial`, `condcomp`, `assertions`, `assertions/assertions_basic`, `isomodes`, `aidomain`.

35 1-CLIQUE-Sharing domain

Author(s): Jorge Navas.

35.1 Usage and interface

- **Library usage:**
  `:- use_module(domain(sharing_clique_1)).`

35.2 Documentation on multifiles

aidomain/1: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.init_abstract_domain/2: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.amgu/5: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.augment_asub/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.augment_two_asub/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.call_to_entry/10: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.exit_to_prime/8: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.project/6: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.widencall/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.needs/2: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.widen/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.compute_lub/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.compute_clauses_lub/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.compute_clauses_glb/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.fixpoint_covered/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.fixpoint_covered_gfp/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.identical_abstract/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.abs_sort/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.extend/6: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.less_or_equal/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.glb/4: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.eliminate_equivalent/3: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.abs_subset/3: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.call_to_success_fact/10: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.special_builtin/6: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.combined_special_builtin0/3: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.body_succ_builtin/9: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.split_combined_domain/4: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.success_builtin/7: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.call_to_success_builtin/7: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.obtain_info/5: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.input_interface/5: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.input_user_interface/6: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.asub_to_native/6: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.concrete/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.unknown_call/5: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.unknown_entry/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.empty_entry/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.part_conc/5: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.multi_part_conc/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.collect_auxinfo_asub/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.rename_auxinfo_asub/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.dom_statistics/2: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.contains_parameters/2: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

35.3 Documentation on imports

This module has the following direct dependencies:

- *Application modules:*
  `sharing_clique`, `preprocess_flags`, `domains`, `share_aux`, `share_amgu_sets`, `share_amgu_aux`, `s_grshfr`, `sharing`, `share_clique_aux`, `share_clique_1_aux`.
- *System library modules:*
  `sets`, `sort`, `terms_check`, `terms_vars`, `lsets`, `lists`.
- *Internal (engine) modules:*
  `term_basic`, `arithmetic`, `atomic_basic`, `basiccontrol`, `exceptions`, `term_compare`, `term_typing`, `debugger_support`, `basic_props`, `hiord_rt`.
- *Packages:*
  `prelude`, `initial`, `condcomp`, `assertions`, `assertions/assertions_basic`, `isomodes`, `aidomain`.

36 CLIQUE-Sharing+Def domain

Author(s): Jorge Navas.

36.1 Usage and interface

- **Library usage:**
  `:- use_module(domain(sharing_clique_def)).`

36.2 Documentation on multifiles

aidomain/1: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.init_abstract_domain/2: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.amgu/5: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.augment_asub/4: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.augment_two_asub/4: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.call_to_entry/10: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.exit_to_prime/8: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.project/6: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.widencall/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.needs/2: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.widen/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.compute_lub/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.compute_clauses_lub/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.compute_clauses_glb/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.fixpoint_covered/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.fixpoint_covered_gfp/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.identical_abstract/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.abs_sort/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.extend/6: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.less_or_equal/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.glb/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.eliminate_equivalent/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.abs_subset/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.call_to_success_fact/10: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.special_builtin/6: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.combined_special_builtin0/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.body_succ_builtin/9: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.split_combined_domain/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.success_builtin/7: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.call_to_success_builtin/7: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.obtain_info/5: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.input_interface/5: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.input_user_interface/6: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.asub_to_native/6: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.concrete/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.unknown_call/5: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.unknown_entry/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.empty_entry/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.part_conc/5: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.multi_part_conc/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.collect_auxinfo_asub/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.rename_auxinfo_asub/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.dom_statistics/2: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.contains_parameters/2: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

36.3 Documentation on imports

This module has the following direct dependencies:

- *Application modules:*
  sharing_clique, preprocess_flags, domains, def, share_clique_aux, s_grshfr, share_aux.
- *System library modules:*
  sets, sort, messages.
- *Internal (engine) modules:*
  term_basic, arithmetic, atomic_basic, basiccontrol, exceptions, term_compare, term_typing, debugger_support, basic_props, hiord_rt.
- *Packages:*
  prelude, initial, condcomp, assertions, assertions/assertions_basic, isomodes, aidomain.

37 shfret (library)

37.1 Usage and interface

- **Library usage:**
  `:- use_module(domain(shfret)).`
- **Exports:**
  - *Predicates:*
    `shfret_init_abstract_domain/1`, `shfret_call_to_entry/9`, `shfret_exit_to_prime/7`, `shfret_project/5`, `shfret_extend/5`, `shfret_widen/3`, `shfret_widencall/3`, `shfret_compute_lub/2`, `shfret_glb/3`, `shfret_eliminate_equivalent/2`, `shfret_less_or_equal/2`, `shfret_identical_abstract/2`, `shfret_abs_sort/2`, `shfret_call_to_success_fact/9`, `shfret_combined_special_builtin0/2`, `shfret_split_combined_domain/3`, `shfret_input_interface/4`, `shfret_input_user_interface/5`, `shfret_asub_to_native/5`, `shfret_unknown_call/4`, `shfret_unknown_entry/3`, `shfret_empty_entry/3`.
  - *Multifiles:*
    `aidomain/1`, `aidom.init_abstract_domain/2`, `aidom.amgu/5`, `aidom.augment_asub/4`, `aidom.augment_two_asub/4`, `aidom.call_to_entry/10`, `aidom.exit_to_prime/8`, `aidom.project/6`, `aidom.widencall/4`, `aidom.needs/2`, `aidom.widen/4`, `aidom.compute_lub/3`, `aidom.compute_clauses_lub/4`, `aidom.compute_clauses_glb/4`, `aidom.fixpoint_covered/3`, `aidom.fixpoint_covered_gfp/3`, `aidom.identical_abstract/3`, `aidom.abs_sort/3`, `aidom.extend/6`, `aidom.less_or_equal/3`, `aidom.glb/4`, `aidom.eliminate_equivalent/3`, `aidom.abs_subset/3`, `aidom.call_to_success_fact/10`, `aidom.special_builtin/6`, `aidom.combined_special_builtin0/3`, `aidom.body_succ_builtin/9`, `aidom.split_combined_domain/4`, `aidom.success_builtin/7`, `aidom.call_to_success_builtin/7`, `aidom.obtain_info/5`, `aidom.input_interface/5`, `aidom.input_user_interface/6`, `aidom.asub_to_native/6`, `aidom.concrete/4`, `aidom.unknown_call/5`, `aidom.unknown_entry/4`, `aidom.empty_entry/4`, `aidom.part_conc/5`, `aidom.multi_part_conc/4`, `aidom.collect_auxinfo_asub/4`, `aidom.rename_auxinfo_asub/4`, `aidom.dom_statistics/2`, `aidom.contains_parameters/2`.

37.2 Documentation on exports

shfret_init_abstract_domain/1: PREDICATE

No further documentation available for this predicate.

shfret_call_to_entry/9: PREDICATE

No further documentation available for this predicate.

shfret_exit_to_prime/7: PREDICATE

No further documentation available for this predicate.

shfret_project/5: PREDICATE
No further documentation available for this predicate.

shfret_extend/5: PREDICATE
No further documentation available for this predicate.

shfret_widen/3: PREDICATE
No further documentation available for this predicate.

shfret_widencall/3: PREDICATE
No further documentation available for this predicate.

shfret_compute_lub/2: PREDICATE
No further documentation available for this predicate.

shfret_glb/3: PREDICATE
No further documentation available for this predicate.

shfret_eliminate_equivalent/2: PREDICATE
No further documentation available for this predicate.

shfret_less_or_equal/2: PREDICATE
No further documentation available for this predicate.

shfret_identical_abstract/2: PREDICATE
No further documentation available for this predicate.

shfret_abs_sort/2: PREDICATE
No further documentation available for this predicate.

shfret_call_to_success_fact/9: PREDICATE
No further documentation available for this predicate.

shfret_combined_special_builtin0/2: PREDICATE
No further documentation available for this predicate.

shfret_split_combined_domain/3: PREDICATE

No further documentation available for this predicate.

shfret_input_interface/4: PREDICATE

No further documentation available for this predicate.

shfret_input_user_interface/5: PREDICATE

No further documentation available for this predicate.

shfret_asub_to_native/5: PREDICATE

No further documentation available for this predicate.

shfret_unknown_call/4: PREDICATE

No further documentation available for this predicate.

shfret_unknown_entry/3: PREDICATE

No further documentation available for this predicate.

shfret_empty_entry/3: PREDICATE

No further documentation available for this predicate.

37.3 Documentation on multifiles

aidomain/1: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.init_abstract_domain/2: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.amgu/5: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.augment_asub/4: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.augment_two_asub/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.call_to_entry/10: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.exit_to_prime/8: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.project/6: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.widencall/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.needs/2: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.widen/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.compute_lub/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.compute_clauses_lub/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.compute_clauses_glb/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.fixpoint_covered/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.fixpoint_covered_gfp/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.identical_abstract/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.abs_sort/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.extend/6: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.less_or_equal/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.glb/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.eliminate_equivalent/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.abs_subset/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.call_to_success_fact/10: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.special_builtin/6: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.combined_special_builtin0/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.body_succ_builtin/9: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.split_combined_domain/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.success_builtin/7: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.call_to_success_builtin/7: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.obtain_info/5: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.input_interface/5: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.input_user_interface/6: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.asub_to_native/6: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.concrete/4: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.unknown_call/5: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.unknown_entry/4: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.empty_entry/4: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.part_conc/5: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.multi_part_conc/4: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.collect_auxinfo_asub/4: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.rename_auxinfo_asub/4: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.dom_statistics/2: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.contains_parameters/2: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

37.4 Documentation on imports

This module has the following direct dependencies:

- *Application modules:*
  `preprocess_flags`, `domains`, `eterms`, `sharefree`.
- *System library modules:*
  `sets`, `sort`, `lists`.
- *Internal (engine) modules:*
  `term_basic`, `arithmetic`, `atomic_basic`, `basiccontrol`, `exceptions`, `term_compare`, `term_typing`, `debugger_support`, `basic_props`, `hiord_rt`.
- *Packages:*
  `prelude`, `initial`, `condcomp`, `assertions`, `assertions/assertions_basic`, `regtypes`, `basicmodes`, `aidomain`.

38 amgu-based sharing+freeness+linearity domain

Author(s): Jorge Navas.

38.1 Usage and interface

- **Library usage:**
  `:- use_module(domain(shfrlin_amgu)).`

38.2 Documentation on multifiles

aidomain/1: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.init_abstract_domain/2: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.amgu/5: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.augment_asub/4: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.augment_two_asub/4: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.call_to_entry/10: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.exit_to_prime/8: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.project/6: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.widencall/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.needs/2: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.widen/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.compute_lub/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.compute_clauses_lub/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.compute_clauses_glb/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.fixpoint_covered/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.fixpoint_covered_gfp/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.identical_abstract/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.abs_sort/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.extend/6: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.less_or_equal/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.glb/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.eliminate_equivalent/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.abs_subset/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.call_to_success_fact/10: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.special_builtin/6: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.combined_special_builtin0/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.body_succ_builtin/9: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.split_combined_domain/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.success_builtin/7: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.call_to_success_builtin/7: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.obtain_info/5: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.input_interface/5: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.input_user_interface/6: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.asub_to_native/6: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.concrete/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.unknown_call/5: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.unknown_entry/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.empty_entry/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.part_conc/5: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.multi_part_conc/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.collect_auxinfo_asub/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.rename_auxinfo_asub/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.dom_statistics/2: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.contains_parameters/2: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

38.3 Documentation on imports

This module has the following direct dependencies:

- *Application modules:*
  preprocess_flags, domains, sharefree, sharefree_amgu, share_aux, sharefree_amgu_aux, s_grshfr, shfrlin_amgu_aux.
- *System library modules:*
  sets, sort, terms_vars, lists, terms_check.
- *Internal (engine) modules:*
  term_basic, arithmetic, atomic_basic, basiccontrol, exceptions, term_compare, term_typing, debugger_support, basic_props, hiord_rt.
- *Packages:*
  prelude, initial, condcomp, assertions, assertions/assertions_basic, isomodes, aidomain.

39 shfrson (library)

39.1 Usage and interface

- **Library usage:**
  `:- use_module(domain(shfrson)).`

39.2 Documentation on multifiles

aidomain/1: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.init_abstract_domain/2: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.amgu/5: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.augment_asub/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.augment_two_asub/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.call_to_entry/10: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.exit_to_prime/8: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.project/6: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.widencall/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.needs/2: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.widen/4: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.compute_lub/3: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.compute_clauses_lub/4: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.compute_clauses_glb/4: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.fixpoint_covered/3: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.fixpoint_covered_gfp/3: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.identical_abstract/3: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.abs_sort/3: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.extend/6: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.less_or_equal/3: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.glb/4: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.eliminate_equivalent/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.abs_subset/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.call_to_success_fact/10: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.special_builtin/6: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.combined_special_builtin0/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.body_succ_builtin/9: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.split_combined_domain/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.success_builtin/7: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.call_to_success_builtin/7: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.obtain_info/5: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.input_interface/5: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.input_user_interface/6: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.asub_to_native/6: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.concrete/4: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.unknown_call/5: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.unknown_entry/4: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.empty_entry/4: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.part_conc/5: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.multi_part_conc/4: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.collect_auxinfo_asub/4: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.rename_auxinfo_asub/4: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.dom_statistics/2: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.contains_parameters/2: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

39.3 Documentation on imports

This module has the following direct dependencies:

- *Application modules:*
  preprocess_flags, domains, sharefree, sondergaard, s_grshfr, share_aux, plai_errors.
- *System library modules:*
  sets, sort, llists, terms_vars.
- *Internal (engine) modules:*
  term_basic, arithmetic, atomic_basic, basiccontrol, exceptions, term_compare, term_typing, debugger_support.
- *Packages:*
  prelude, initial, condcomp, aidomain.

40 sondergaard (library)

40.1 Usage and interface

- **Library usage:**

  `:- use_module(domain(sondergaard)).`
- **Exports:**
  - *Predicates:*

    `project/5`, `call_to_entry/9`, `exit_to_prime/7`, `abs_sort/2`, `compute_lub/2`, `lub/3`, `glb/3`, `extend/5`, `call_to_success_fact/9`, `call_to_prime_fact/6`, `unknown_call/4`, `unknown_entry/3`, `empty_entry/3`, `asub_to_native/5`, `less_or_equal/2`, `special_builtin/5`, `success_builtin/6`, `call_to_success_builtin/6`, `share_to_son/3`, `son_to_share/4`, `propagate_to_sh/4`, `propagate_to_son/4`.
  - *Multifiles:*

    `aidomain/1`, `aidom.init_abstract_domain/2`, `aidom.amgu/5`, `aidom.augment_asub/4`, `aidom.augment_two_asub/4`, `aidom.call_to_entry/10`, `aidom.exit_to_prime/8`, `aidom.project/6`, `aidom.widencall/4`, `aidom.needs/2`, `aidom.widen/4`, `aidom.compute_lub/3`, `aidom.compute_clauses_lub/4`, `aidom.compute_clauses_glb/4`, `aidom.fixpoint_covered/3`, `aidom.fixpoint_covered_gfp/3`, `aidom.identical_abstract/3`, `aidom.abs_sort/3`, `aidom.extend/6`, `aidom.less_or_equal/3`, `aidom.glb/4`, `aidom.eliminate_equivalent/3`, `aidom.abs_subset/3`, `aidom.call_to_success_fact/10`, `aidom.special_builtin/6`, `aidom.combined_special_builtin0/3`, `aidom.body_succ_builtin/9`, `aidom.split_combined_domain/4`, `aidom.success_builtin/7`, `aidom.call_to_success_builtin/7`, `aidom.obtain_info/5`, `aidom.input_interface/5`, `aidom.input_user_interface/6`, `aidom.asub_to_native/6`, `aidom.concrete/4`, `aidom.unknown_call/5`, `aidom.unknown_entry/4`, `aidom.empty_entry/4`, `aidom.part_conc/5`, `aidom.multi_part_conc/4`, `aidom.collect_auxinfo_asub/4`, `aidom.rename_auxinfo_asub/4`, `aidom.dom_statistics/2`, `aidom.contains_parameters/2`.

40.2 Documentation on exports

project/5: PREDICATE

No further documentation available for this predicate.

call_to_entry/9: PREDICATE

No further documentation available for this predicate.

exit_to_prime/7: PREDICATE

No further documentation available for this predicate.

abs_sort/2: PREDICATE

No further documentation available for this predicate.

compute_lub/2: PREDICATE

No further documentation available for this predicate.

lub/3: PREDICATE

No further documentation available for this predicate.

glb/3: PREDICATE

No further documentation available for this predicate.

extend/5: PREDICATE

No further documentation available for this predicate.

call_to_success_fact/9: PREDICATE

No further documentation available for this predicate.

call_to_prime_fact/6: PREDICATE

No further documentation available for this predicate.

unknown_call/4: PREDICATE

No further documentation available for this predicate.

unknown_entry/3: PREDICATE

No further documentation available for this predicate.

empty_entry/3: PREDICATE

No further documentation available for this predicate.

asub_to_native/5: PREDICATE

No further documentation available for this predicate.

input_user_interface/5: PREDICATE

No further documentation available for this predicate.

input_interface/4: PREDICATE
No further documentation available for this predicate.

less_or_equal/2: PREDICATE
No further documentation available for this predicate.

special_builtin/5: PREDICATE
No further documentation available for this predicate.

success_builtin/6: PREDICATE
No further documentation available for this predicate.

call_to_success_builtin/6: PREDICATE
No further documentation available for this predicate.

share_to_son/3: PREDICATE
No further documentation available for this predicate.

son_to_share/4: PREDICATE
No further documentation available for this predicate.

propagate_to_sh/4: PREDICATE
No further documentation available for this predicate.

propagate_to_son/4: PREDICATE
No further documentation available for this predicate.

40.3 Documentation on multifiles

aidomain/1: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.init_abstract_domain/2: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.amgu/5: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.augment_asub/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.augment_two_asub/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.call_to_entry/10: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.exit_to_prime/8: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.project/6: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.widencall/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.needs/2: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.widen/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.compute_lub/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.compute_clauses_lub/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.compute_clauses_glb/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.fixpoint_covered/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.fixpoint_covered_gfp/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.identical_abstract/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.abs_sort/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.extend/6: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.less_or_equal/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.glb/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.eliminate_equivalent/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.abs_subset/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.call_to_success_fact/10: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.special_builtin/6: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.combined_special_builtin0/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.body_succ_builtin/9: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.split_combined_domain/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.success_builtin/7: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.call_to_success_builtin/7: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.obtain_info/5: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.input_interface/5: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.input_user_interface/6: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.asub_to_native/6: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.concrete/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.unknown_call/5: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.unknown_entry/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.empty_entry/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.part_conc/5: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.multi_part_conc/4: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.collect_auxinfo_asub/4: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.rename_auxinfo_asub/4: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.dom_statistics/2: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.contains_parameters/2: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

40.4 Documentation on imports

This module has the following direct dependencies:

- *Application modules:*
  `preprocess_flags`, `domains`, `sharing`, `s_grshfr`, `share_aux`.
- *System library modules:*
  `sets`, `sort`, `lists`, `llists`, `lsets`, `terms_check`, `terms_vars`.
- *Internal (engine) modules:*
  `term_basic`, `arithmetic`, `atomic_basic`, `basiccontrol`, `exceptions`, `term_compare`, `term_typing`, `debugger_support`, `basic_props`.
- *Packages:*
  `prelude`, `initial`, `condcomp`, `assertions`, `assertions/assertions_basic`, `aidomain`.

41 svterms: eterms + same value (abstract domain)

41.1 Usage and interface

- **Library usage:**
  `:- use_module(domain(svterms)).`
- **Exports:**
  - *Predicates:*
    `svterms_concrete/3`, `svterms_compute_lub/2`, `svterms_compute_lub_el/3`, `svterms_needs/1`, `svterms_widencall/3`, `svterms_widen/3`, `svterms_init_abstract_domain/1`, `svterms_call_to_entry/9`, `svterms_exit_to_prime/7`, `svterms_project/5`, `svterms_abs_sort/2`, `svterms_extend/5`, `svterms_less_or_equal/2`, `svterms_glb/3`, `svterms_unknown_entry/3`, `svterms_empty_entry/3`, `svterms_unknown_call/4`, `svterms_call_to_success_fact/9`, `svterms_special_builtin/5`, `svterms_success_builtin/6`, `svterms_call_to_success_builtin/6`, `svterms_input_user_interface/5`, `svterms_input_interface/4`, `svterms_asub_to_native/5`, `svterms_output_interface/2`, `svterms_collect_auxinfo_asub/3`, `svterms_rename_auxinfo_asub/3`, `svterms_identical_abstract/2`.
  - *Multifiles:*
    `aidomain/1`, `aidom.init_abstract_domain/2`, `aidom.amgu/5`, `aidom.augment_asub/4`, `aidom.augment_two_asub/4`, `aidom.call_to_entry/10`, `aidom.exit_to_prime/8`, `aidom.project/6`, `aidom.widencall/4`, `aidom.needs/2`, `aidom.widen/4`, `aidom.compute_lub/3`, `aidom.compute_clauses_lub/4`, `aidom.compute_clauses_glb/4`, `aidom.fixpoint_covered/3`, `aidom.fixpoint_covered_gfp/3`, `aidom.identical_abstract/3`, `aidom.abs_sort/3`, `aidom.extend/6`, `aidom.less_or_equal/3`, `aidom.glb/4`, `aidom.eliminate_equivalent/3`, `aidom.abs_subset/3`, `aidom.call_to_success_fact/10`, `aidom.special_builtin/6`, `aidom.combined_special_builtin0/3`, `aidom.body_succ_builtin/9`, `aidom.split_combined_domain/4`, `aidom.success_builtin/7`, `aidom.call_to_success_builtin/7`, `aidom.obtain_info/5`, `aidom.input_interface/5`, `aidom.input_user_interface/6`, `aidom.asub_to_native/6`, `aidom.concrete/4`, `aidom.unknown_call/5`, `aidom.unknown_entry/4`, `aidom.empty_entry/4`, `aidom.part_conc/5`, `aidom.multi_part_conc/4`, `aidom.collect_auxinfo_asub/4`, `aidom.rename_auxinfo_asub/4`, `aidom.dom_statistics/2`, `aidom.contains_parameters/2`.

41.2 Documentation on exports

svterms_concrete/3: PREDICATE

No further documentation available for this predicate.

svterms_compute_lub/2: PREDICATE

No further documentation available for this predicate.

svterms_compute_lub_el/3: PREDICATE
No further documentation available for this predicate.

svterms_needs/1: PREDICATE
No further documentation available for this predicate.

svterms_widencall/3: PREDICATE
No further documentation available for this predicate.

svterms_widen/3: PREDICATE
No further documentation available for this predicate.

svterms_init_abstract_domain/1: PREDICATE
No further documentation available for this predicate.

svterms_call_to_entry/9: PREDICATE
No further documentation available for this predicate.

svterms_exit_to_prime/7: PREDICATE
No further documentation available for this predicate.

svterms_project/5: PREDICATE
No further documentation available for this predicate.

svterms_abs_sort/2: PREDICATE
No further documentation available for this predicate.

svterms_extend/5: PREDICATE
No further documentation available for this predicate.

svterms_less_or_equal/2: PREDICATE
No further documentation available for this predicate.

svterms_glb/3: PREDICATE
No further documentation available for this predicate.

svterms_unknown_entry/3: PREDICATE
No further documentation available for this predicate.

svterms_empty_entry/3: PREDICATE
No further documentation available for this predicate.

svterms_unknown_call/4: PREDICATE
No further documentation available for this predicate.

svterms_call_to_success_fact/9: PREDICATE
No further documentation available for this predicate.

svterms_special_builtin/5: PREDICATE
No further documentation available for this predicate.

svterms_success_builtin/6: PREDICATE
No further documentation available for this predicate.

svterms_call_to_success_builtin/6: PREDICATE
No further documentation available for this predicate.

svterms_input_user_interface/5: PREDICATE
No further documentation available for this predicate.

svterms_input_interface/4: PREDICATE
No further documentation available for this predicate.

svterms_asub_to_native/5: PREDICATE
No further documentation available for this predicate.

svterms_output_interface/2: PREDICATE
No further documentation available for this predicate.

svterms_collect_auxinfo_asub/3: PREDICATE
No further documentation available for this predicate.

svterms_rename_auxinfo_asub/3: PREDICATE

No further documentation available for this predicate.

svterms_identical_abstract/2: PREDICATE

No further documentation available for this predicate.

41.3 Documentation on multifiles

aidomain/1: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.init_abstract_domain/2: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.amgu/5: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.augment_asub/4: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.augment_two_asub/4: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.call_to_entry/10: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.exit_to_prime/8: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.project/6: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.widencall/4: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.needs/2: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.widen/4: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.compute_lub/3: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.compute_clauses_lub/4: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.compute_clauses_glb/4: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.fixpoint_covered/3: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.fixpoint_covered_gfp/3: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.identical_abstract/3: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.abs_sort/3: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.extend/6: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.less_or_equal/3: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.glb/4: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.eliminate_equivalent/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.abs_subset/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.call_to_success_fact/10: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.special_builtin/6: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.combined_special_builtin0/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.body_succ_builtin/9: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.split_combined_domain/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.success_builtin/7: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.call_to_success_builtin/7: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.obtain_info/5: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.input_interface/5: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.input_user_interface/6: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.asub_to_native/6: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.concrete/4: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.unknown_call/5: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.unknown_entry/4: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.empty_entry/4: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.part_conc/5: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.multi_part_conc/4: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.collect_auxinfo_asub/4: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.rename_auxinfo_asub/4: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.dom_statistics/2: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.contains_parameters/2: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

41.4 Documentation on imports

This module has the following direct dependencies:

- *Application modules:*
  preprocess_flags, domains, eterms.
- *System library modules:*
  sets, sort, aggregates, terms_vars, terms_check, lists, assoc, typeslib.
- *Internal (engine) modules:*
  term_basic, arithmetic, atomic_basic, basiccontrol, exceptions, term_compare, term_typing, debugger_support, basic_props, hiord_rt.
- *Packages:*
  prelude, initial, condcomp, assertions, assertions/assertions_basic, regtypes, basicmodes, aidomain.

42 termsd: types with shortening (abstract domain)

Author(s): Claudio Vaucheret, Francisco Bueno, Ciao Development Team.

This module implements the abstract operations of the type domains with widening based on shortening (`shortening_el/3`). An abstract sustitution is a list of `Var:Type` elements, where `Var` is a variable and `Type` is a pure type term [DZ92].

42.1 Usage and interface

- **Library usage:**
  `:- use_module(domain(termsd)).`
- **Exports:**
  - *Predicates:*
    `terms_input_interface/4`, `terms_concrete/3`, `terms_compute_lub/2`, `terms_compute_lub_el/3`, `terms_widencall/3`, `terms_needs/1`, `terms_widen/3`, `terms_init_abstract_domain/1`, `terms_call_to_entry/9`, `variables_are_variable_type/2`, `terms_exit_to_prime/7`, `unify_term_and_type_term/5`, `generate_a_type_assignment/3`, `get_var_types_by_unification/3`, `compute_type/3`, `find_list_entry/3`, `terms_project/5`, `terms_abs_sort/2`, `terms_extend/5`, `terms_less_or_equal/2`, `terms_glb/3`, `terms_unknown_entry/3`, `terms_empty_entry/3`, `terms_unknown_call/4`, `substitution/3`, `variables_are_top_type/2`, `terms_call_to_success_fact/9`, `terms_special_builtin/5`, `terms_success_builtin/6`, `terms_call_to_success_builtin/6`, `precondition_builtin/2`, `postcondition_builtin/4`, `terms_input_user_interface/5`, `terms_asub_to_native/5`, `terms_internal_to_native/3`, `terms_output_interface/2`, `terms_collect_auxinfo_asub/3`, `terms_rename_auxinfo_asub/3`, `terms_identical_abstract/2`.
  - *Multifiles:*
    `aidomain/1`, `aidom.init_abstract_domain/2`, `aidom.amgu/5`, `aidom.augment_asub/4`, `aidom.augment_two_asub/4`, `aidom.call_to_entry/10`, `aidom.exit_to_prime/8`, `aidom.project/6`, `aidom.widencall/4`, `aidom.needs/2`, `aidom.widen/4`, `aidom.compute_lub/3`, `aidom.compute_clauses_lub/4`, `aidom.compute_clauses_glb/4`, `aidom.fixpoint_covered/3`, `aidom.fixpoint_covered_gfp/3`, `aidom.identical_abstract/3`, `aidom.abs_sort/3`, `aidom.extend/6`, `aidom.less_or_equal/3`, `aidom.glb/4`, `aidom.eliminate_equivalent/3`, `aidom.abs_subset/3`, `aidom.call_to_success_fact/10`, `aidom.special_builtin/6`, `aidom.combined_special_`
    `builtin0/3`, `aidom.body_succ_builtin/9`, `aidom.split_combined_domain/4`, `aidom.success_builtin/7`, `aidom.call_to_success_builtin/7`, `aidom.obtain_info/5`, `aidom.input_interface/5`, `aidom.input_user_interface/6`, `aidom.asub_to_native/6`, `aidom.concrete/4`, `aidom.unknown_call/5`, `aidom.unknown_entry/4`, `aidom.empty_entry/4`, `aidom.part_conc/5`, `aidom.multi_part_conc/4`, `aidom.collect_auxinfo_asub/4`, `aidom.rename_auxinfo_asub/4`, `aidom.dom_statistics/2`, `aidom.contains_parameters/2`.

42.2 Documentation on exports

terms_input_interface/4: PREDICATE

No further documentation available for this predicate.

terms_concrete/3: PREDICATE

No further documentation available for this predicate.

terms_compute_lub/2: PREDICATE

Usage: `terms_compute_lub(ListASub,Lub)`

It computes the least upper bound of a set of abstract substitutions. For each two abstract substitutions `ASub1` and `ASub2` in `ListASub`, obtaining the lub is foreach X:Type1 in `ASub1` and X:Type2 in `ASub2`, X:TypeUnion is in Lub, and TypeUnion is the deterministic union of Type1 an Type2.

- *The following properties should hold at call time:*
  - `ListASub` is currently a term which is not a free variable. (`nonvar/1`)
  - `Lub` is a free variable. (`var/1`)
  - `ListASub` is a list of `termsd_asubs`. (`list/2`)
- *The following properties should hold upon exit:*
  - `Lub` is an abstract substitution (`termsd_asub/1`)

terms_compute_lub_el/3: PREDICATE

No further documentation available for this predicate.

terms_widencall/3: PREDICATE

No further documentation available for this predicate.

terms_needs/1: PREDICATE

No further documentation available for this predicate.

terms_widen/3: PREDICATE

Usage: `terms_widen(Prime0,Prime1,NewPrime)`

Induction step on the abstract substitution of a fixpoint. `Prime0` corresponds to non-recursive and `Prime1` to recursive clauses. `NewPrime` is the result of apply one widening operation to the least upper bound of the formers. Widening operations implemented are "shortening" and "restricted shortening" [GdW94, SG94].

- *The following properties should hold at call time:*
  - `Prime0` is currently a term which is not a free variable. (`nonvar/1`)
  - `Prime1` is currently a term which is not a free variable. (`nonvar/1`)
  - `NewPrime` is a free variable. (`var/1`)
  - `Prime0` is an abstract substitution (`termsd_asub/1`)
  - `Prime1` is an abstract substitution (`termsd_asub/1`)
  - `NewPrime` is any term. (`term/1`)
- *The following properties should hold upon exit:*
  - `NewPrime` is an abstract substitution (`termsd_asub/1`)

terms_init_abstract_domain/1: PREDICATE

No further documentation available for this predicate.

terms_call_to_entry/9: PREDICATE

Usage: `terms_call_to_entry(Sv,Sg,Hv,Head,K,Fv,Proj,Entry,ExtraInfo)`

It obtains the abstract substitution `Entry` which results from adding the abstraction of the `Sg` = `Head` to `Proj`, later projecting the resulting substitution onto `Hv`.

This is done as follows:

- If `Sg` and `Head` are identical up to renaming it is just renaming `Proj` and adding the `Fv`
- Otherwise, it will:
  - obtain the abstract substitution from unifying `Sg` and `Head` calling "unify_term_and_type_term"
  - add to this abstract substituion the variables in `Fv` as term type.

The meaning of the variables is:

- `Sg` is the subgoal being analysed.
- `Head` is the Head of the clause being analysed.
- `Fv` is a list of free variables in the body of the clause being considered.
- `Proj` is the abstract substitution `Call` projected onto `Sv`.
- `Entry` is the Abstract entry substitution (i.e. the abstract subtitution obtained after the abstract unification of `Sg` and `Head` projected onto `Hv + Fv`).
- `ExtraInfo` is a flag `yes` when no unification is required, i.e., `Entry=Proj` upto names of vars. and ignoring `Fv`. It is `no` if unification fails.

– *The following properties should hold at call time:*

`Sv` is currently a term which is not a free variable. (`nonvar/1`)
`Sg` is currently a term which is not a free variable. (`nonvar/1`)
`Hv` is currently a term which is not a free variable. (`nonvar/1`)
`Head` is currently a term which is not a free variable. (`nonvar/1`)
`K` is currently a term which is not a free variable. (`nonvar/1`)
`Fv` is currently a term which is not a free variable. (`nonvar/1`)
`Proj` is currently a term which is not a free variable. (`nonvar/1`)
`Entry` is a free variable. (`var/1`)
`ExtraInfo` is a free variable. (`var/1`)
`Sv` is any term. (`term/1`)
`Sg` is a term which represents a goal, i.e., an atom or a structure. (`cgoal/1`)
`Hv` is a list. (`list/1`)
`Head` is a term which represents a goal, i.e., an atom or a structure. (`cgoal/1`)
`K` is any term. (`term/1`)
`Fv` is a list. (`list/1`)
`Proj` is an abstract substitution (`termsd_asub/1`)
`Entry` is any term. (`term/1`)
`ExtraInfo` is any term. (`term/1`)

– *The following properties should hold upon exit:*

`Entry` is an abstract substitution (`termsd_asub/1`)
`termsd_extrainfo(ExtraInfo)` (`termsd_extrainfo/1`)

variables_are_variable_type/2: PREDICATE

Usage: `variables_are_variable_type(Fv,ASub)`

(at the moment) assigns the value top_type to the variables in `Fv` but in the future it must assign the value `var`

– *The following properties should hold at call time:*

`Fv` is currently a term which is not a free variable. (`nonvar/1`)

`ASub` is a free variable. (`var/1`)

`Fv` is a list. (`list/1`)

– *The following properties should hold upon exit:*

`ASub` is an abstract substitution (`termsd_asub/1`)

terms_exit_to_prime/7: PREDICATE

Usage: `terms_exit_to_prime(Sg,Hv,Head,Sv,Exit,ExtraInfo,Prime)`

It computes the prime abstract substitution `Prime`, i.e. the result of going from the abstract substitution over the head variables `Exit`, to the abstract substitution over the variables in the subgoal. It will:

- If `Exit` is '$bottom', `Prime` will be also '$bottom'.
- If `ExtraInfo` = yes (`Head` and `Sg` identical up to renaming) it is just renaming Exit %
- Otherwise, it will:
  - projects `Exit` onto `Hv` obtaining `BPrime`.
  - obtain the abstract substitution from unifying `Sg` and `Head` calling "unify_term_and_type_term"

– *The following properties should hold at call time:*

`Sg` is currently a term which is not a free variable. (`nonvar/1`)

`Hv` is currently a term which is not a free variable. (`nonvar/1`)

`Head` is currently a term which is not a free variable. (`nonvar/1`)

`Sv` is currently a term which is not a free variable. (`nonvar/1`)

`Exit` is currently a term which is not a free variable. (`nonvar/1`)

`ExtraInfo` is currently a term which is not a free variable. (`nonvar/1`)

`Prime` is a free variable. (`var/1`)

`Sg` is a term which represents a goal, i.e., an atom or a structure. (`cgoal/1`)

`Hv` is a list. (`list/1`)

`Head` is a term which represents a goal, i.e., an atom or a structure. (`cgoal/1`)

`Sv` is a list. (`list/1`)

`Exit` is an abstract substitution (`termsd_asub/1`)

`termsd_extrainfo(ExtraInfo)` (`termsd_extrainfo/1`)

`Prime` is any term. (`term/1`)

– *The following properties should hold upon exit:*

`Prime` is an abstract substitution (`termsd_asub/1`)

unify_term_and_type_term/5: PREDICATE

Usage: `unify_term_and_type_term(Term1,Tv,Term2,ASub,NewASub)`

It unifies the term `Term1` to the type term `Term2` obtaining the the abstract substitution TypeAss which is sorted and projeted on `Tv`

- *The following properties should hold at call time:*
  - `Term1` is currently a term which is not a free variable. (`nonvar/1`)
  - `Tv` is currently a term which is not a free variable. (`nonvar/1`)
  - `Term2` is currently a term which is not a free variable. (`nonvar/1`)
  - `ASub` is currently a term which is not a free variable. (`nonvar/1`)
  - `NewASub` is a free variable. (`var/1`)
  - `Term1` is a term which represents a goal, i.e., an atom or a structure. (`cgoal/1`)
  - `Tv` is a list. (`list/1`)
  - `Term2` is a term which represents a goal, i.e., an atom or a structure. (`cgoal/1`)
  - `ASub` is an abstract substitution (`termsd_asub/1`)
  - `NewASub` is any term. (`term/1`)
- *The following properties should hold upon exit:*
  - `NewASub` is an abstract substitution (`termsd_asub/1`)

generate_a_type_assignment/3: PREDICATE

No further documentation available for this predicate.

get_var_types_by_unification/3: PREDICATE

No further documentation available for this predicate.

compute_type/3: PREDICATE

Usage: `compute_type(Term,Type,Types)`

Given the term `Term` and a type `Type`, computes the type corresponding to each variable in `Term`, and store it in the data structure `Types`. `Types` have the structure [vt(Var1, TypeList1), ..., vt(VarN, TypeListN)|_], where TypeListN is the list of types corresponding to variable VarN.

- *The following properties should hold at call time:*
  - `Type` is currently a term which is not a free variable. (`nonvar/1`)

find_list_entry/3: PREDICATE

No further documentation available for this predicate.

terms_project/5: PREDICATE

Usage: `terms_project(Sg,Vars,HvFv_u,Asub,Proj)`

`Proj` is the result of eliminating from `Asub` all `X:Value` such that `X` is not in `Vars`. `HvFv_u` may be a list or `not_provided_HvFv_u` and is not used by this predicate.

- *The following properties should hold at call time:*
  `Sg` is currently a term which is not a free variable. (`nonvar/1`)
  `Vars` is currently a term which is not a free variable. (`nonvar/1`)
  `HvFv_u` is currently a term which is not a free variable. (`nonvar/1`)
  `Asub` is currently a term which is not a free variable. (`nonvar/1`)
  `Proj` is a free variable. (`var/1`)
  `Sg` is any term. (`term/1`)
  `Vars` is a list. (`list/1`)
  `HvFv_u` is any term. (`term/1`)
  `Asub` is an abstract substitution (`termsd_asub/1`)
  `Proj` is any term. (`term/1`)
- *The following properties should hold upon exit:*
  `Proj` is an abstract substitution (`termsd_asub/1`)

terms_abs_sort/2: PREDICATE

Usage: `terms_abs_sort(Asub,Asub_s)`

It sorts the set of `X:Type` in `Asub` containing `Asub_s`

- *The following properties should hold at call time:*
  `Asub` is currently a term which is not a free variable. (`nonvar/1`)
  `Asub_s` is a free variable. (`var/1`)
  `Asub` is an abstract substitution (`termsd_asub/1`)
- *The following properties should hold upon exit:*
  `Asub_s` is an abstract substitution (`termsd_asub/1`)

terms_extend/5: PREDICATE

Usage: `terms_extend(Sg,Prime,Sv,Call,Succ)`

If `Prime` = '$bottom', `Succ` = '$bottom' otherwise, `Succ` is computed updating the values of `Call` with those in `Prime`

- *The following properties should hold at call time:*
  `Sg` is currently a term which is not a free variable. (`nonvar/1`)
  `Prime` is currently a term which is not a free variable. (`nonvar/1`)
  `Sv` is currently a term which is not a free variable. (`nonvar/1`)
  `Call` is currently a term which is not a free variable. (`nonvar/1`)
  `Succ` is a free variable. (`var/1`)
  `Sg` is any term. (`term/1`)
  `Prime` is an abstract substitution (`termsd_asub/1`)
  `Sv` is a list. (`list/1`)
  `Call` is an abstract substitution (`termsd_asub/1`)
  `Succ` is any term. (`term/1`)
- *The following properties should hold upon exit:*
  `Succ` is an abstract substitution (`termsd_asub/1`)

terms_less_or_equal/2: PREDICATE

Usage: `terms_less_or_equal(ASub0,ASub1)`

Succeeds if `ASub1` is more general or equal to `ASub0`. it's assumed the two abstract substitutions are defined on the same variables

- *The following properties should hold at call time:*
  - `ASub0` is currently a term which is not a free variable. (`nonvar/1`)
  - `ASub1` is currently a term which is not a free variable. (`nonvar/1`)
  - `ASub0` is an abstract substitution (`termsd_asub/1`)
  - `ASub1` is an abstract substitution (`termsd_asub/1`)

terms_glb/3: PREDICATE

Usage: `terms_glb(ASub0,ASub1,Glb)`

`Glb` is the great lower bound of `ASub0` and `ASub1`

- *The following properties should hold at call time:*
  - `ASub0` is currently a term which is not a free variable. (`nonvar/1`)
  - `ASub1` is currently a term which is not a free variable. (`nonvar/1`)
  - `Glb` is a free variable. (`var/1`)
  - `ASub0` is an abstract substitution (`termsd_asub/1`)
  - `ASub1` is an abstract substitution (`termsd_asub/1`)
  - `Glb` is any term. (`term/1`)
- *The following properties should hold upon exit:*
  - `Glb` is an abstract substitution (`termsd_asub/1`)

terms_unknown_entry/3: PREDICATE

Usage: `terms_unknown_entry(Sg,Qv,Call)`

Gives the "top" value for the variables involved in a literal whose definition is not present, and adds this top value to Call. In this domain the top value is X:term forall X in the set of variables

- *The following properties should hold at call time:*
  - `Sg` is currently a term which is not a free variable. (`nonvar/1`)
  - `Qv` is currently a term which is not a free variable. (`nonvar/1`)
  - `Call` is a free variable. (`var/1`)
  - `Sg` is a term which represents a goal, i.e., an atom or a structure. (`cgoal/1`)
  - `Qv` is a list. (`list/1`)
  - `Call` is any term. (`term/1`)
- *The following properties should hold upon exit:*
  - `Call` is an abstract substitution (`termsd_asub/1`)

terms_empty_entry/3: PREDICATE

Usage: `terms_empty_entry(Sg,Vars,Entry)`

Gives the "empty" value in this domain for a given set of variables `Vars`, resulting in the abstract substitution `Entry`. I.e., obtains the abstraction of a substitution in which all variables `Vars` are unbound: free and unaliased. In this domain the empty value is giving the variable type to each variable

- *The following properties should hold at call time:*
  `Sg` is currently a term which is not a free variable. (`nonvar/1`)
  `Vars` is currently a term which is not a free variable. (`nonvar/1`)
  `Entry` is a free variable. (`var/1`)
  `Sg` is a term which represents a goal, i.e., an atom or a structure. (`cgoal/1`)
  `Vars` is a list. (`list/1`)
  `Entry` is any term. (`term/1`)
- *The following properties should hold upon exit:*
  `Entry` is an abstract substitution (`termsd_asub/1`)

terms_unknown_call/4: PREDICATE

Usage: `terms_unknown_call(Sg,Vars,Call,Succ)`

Gives the "top" value for the variables involved in a literal whose definition is not present, and adds this top value to `Call`

- *The following properties should hold at call time:*
  `Sg` is currently a term which is not a free variable. (`nonvar/1`)
  `Vars` is currently a term which is not a free variable. (`nonvar/1`)
  `Call` is currently a term which is not a free variable. (`nonvar/1`)
  `Succ` is a free variable. (`var/1`)
  `Sg` is a term which represents a goal, i.e., an atom or a structure. (`cgoal/1`)
  `Vars` is a list. (`list/1`)
  `Call` is an abstract substitution (`termsd_asub/1`)
  `Succ` is any term. (`term/1`)
- *The following properties should hold upon exit:*
  `Succ` is an abstract substitution (`termsd_asub/1`)

substitution/3: PREDICATE

No further documentation available for this predicate.

variables_are_top_type/2: PREDICATE

Usage: `variables_are_top_type(Fv,ASub)`

It assigns the value top_type to the variables in `Fv` and return the abstract substitution `ASub`

- *The following properties should hold at call time:*
  `Fv` is currently a term which is not a free variable. (`nonvar/1`)
  `ASub` is a free variable. (`var/1`)
  `Fv` is a list. (`list/1`)
- *The following properties should hold upon exit:*
  `ASub` is an abstract substitution (`termsd_asub/1`)

terms_call_to_success_fact/9: PREDICATE

Usage: `terms_call_to_success_fact(Sg,Hv,Head,K,Sv,Call,Proj,Prime,Succ)`

Specialized version of call_to_entry + exit_to_prime + extend for facts

- *The following properties should hold at call time:*
  - `Sg` is currently a term which is not a free variable. (`nonvar/1`)
  - `Hv` is currently a term which is not a free variable. (`nonvar/1`)
  - `Head` is currently a term which is not a free variable. (`nonvar/1`)
  - `K` is currently a term which is not a free variable. (`nonvar/1`)
  - `Sv` is currently a term which is not a free variable. (`nonvar/1`)
  - `Call` is currently a term which is not a free variable. (`nonvar/1`)
  - `Proj` is currently a term which is not a free variable. (`nonvar/1`)
  - `Prime` is a free variable. (`var/1`)
  - `Succ` is a free variable. (`var/1`)
  - `Sg` is a term which represents a goal, i.e., an atom or a structure. (`cgoal/1`)
  - `Hv` is a list. (`list/1`)
  - `Head` is a term which represents a goal, i.e., an atom or a structure. (`cgoal/1`)
  - `K` is any term. (`term/1`)
  - `Sv` is a list. (`list/1`)
  - `Call` is an abstract substitution (`termsd_asub/1`)
  - `Proj` is an abstract substitution (`termsd_asub/1`)
  - `Prime` is any term. (`term/1`)
  - `Succ` is any term. (`term/1`)
- *The following properties should hold upon exit:*
  - `Prime` is an abstract substitution (`termsd_asub/1`)
  - `Succ` is an abstract substitution (`termsd_asub/1`)

terms_special_builtin/5: PREDICATE

Usage: `terms_special_builtin(SgKey,Sg,Subgoal,Type,Condvars)`

`Type` is a flag indicating what is the abstraction of builtin `SgKey` and to which variables `Condvars` of the goal `Sg` it affects.

- *The following properties should hold at call time:*
  - `SgKey` is currently a term which is not a free variable. (`nonvar/1`)
  - `Sg` is currently a term which is not a free variable. (`nonvar/1`)
  - `Subgoal` is currently a term which is not a free variable. (`nonvar/1`)
  - `Type` is a free variable. (`var/1`)
  - `Condvars` is a free variable. (`var/1`)

terms_success_builtin/6: PREDICATE

No further documentation available for this predicate.

terms_call_to_success_builtin/6: PREDICATE

No further documentation available for this predicate.

precondition_builtin/2: PREDICATE

No further documentation available for this predicate.

postcondition_builtin/4: PREDICATE

No further documentation available for this predicate.

terms_input_user_interface/5: PREDICATE

Usage: `terms_input_user_interface(InputUser,Qv,ASub,Sg,MaybeCallASub)`

Obtains the abstract substitution ASub from user supplied information.

- *The following properties should hold at call time:*
  - `Qv` is currently a term which is not a free variable. (`nonvar/1`)
  - `ASub` is a free variable. (`var/1`)
  - `Sg` is currently a term which is not a free variable. (`nonvar/1`)
  - `MaybeCallASub` is currently a term which is not a free variable. (`nonvar/1`)

terms_asub_to_native/5: PREDICATE

Usage: `terms_asub_to_native(ASub,Qv,OutFlag,OutputUser,Comps)`

Transforms abstract substitution `ASub` to user friendly format. Record relevant symbols for output if `OutFlag` is `yes`.

- *The following properties should hold at call time:*
  - `ASub` is currently a term which is not a free variable. (`nonvar/1`)
  - `Qv` is currently a term which is not a free variable. (`nonvar/1`)
  - `OutFlag` is currently a term which is not a free variable. (`nonvar/1`)
  - `OutputUser` is a free variable. (`var/1`)
  - `Comps` is a free variable. (`var/1`)

terms_internal_to_native/3: PREDICATE

No further documentation available for this predicate.

terms_output_interface/2: PREDICATE

No further documentation available for this predicate.

terms_collect_auxinfo_asub/3: PREDICATE

No further documentation available for this predicate.

terms_rename_auxinfo_asub/3: PREDICATE

No further documentation available for this predicate.

terms_identical_abstract/2: PREDICATE

No further documentation available for this predicate.

42.3 Documentation on multifiles

aidomain/1: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.init_abstract_domain/2: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.amgu/5: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.augment_asub/4: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.augment_two_asub/4: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.call_to_entry/10: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.exit_to_prime/8: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.project/6: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.widencall/4: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.needs/2: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.widen/4: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.compute_lub/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.compute_clauses_lub/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.compute_clauses_glb/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.fixpoint_covered/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.fixpoint_covered_gfp/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.identical_abstract/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.abs_sort/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.extend/6: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.less_or_equal/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.glb/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.eliminate_equivalent/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.abs_subset/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.call_to_success_fact/10: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.special_builtin/6: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.combined_special_builtin0/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.body_succ_builtin/9: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.split_combined_domain/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.success_builtin/7: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.call_to_success_builtin/7: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.obtain_info/5: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.input_interface/5: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.input_user_interface/6: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.asub_to_native/6: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.concrete/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.unknown_call/5: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.unknown_entry/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.empty_entry/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.part_conc/5: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.multi_part_conc/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.collect_auxinfo_asub/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.rename_auxinfo_asub/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.dom_statistics/2: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.contains_parameters/2: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

42.4 Documentation on imports

This module has the following direct dependencies:

- *Application modules:*
  `preprocess_flags`, `domains`.
- *System library modules:*
  `sets`, `sort`, `messages`, `terms_vars`, `terms_check`, `lists`, `typeslib`, `p_unit`, `assoc`.
- *Internal (engine) modules:*
  `term_basic`, `arithmetic`, `atomic_basic`, `basiccontrol`, `exceptions`, `term_compare`, `term_typing`, `debugger_support`, `basic_props`, `hiord_rt`.
- *Packages:*
  `prelude`, `initial`, `condcomp`, `assertions`, `assertions/assertions_basic`, `regtypes`, `basicmodes`, `aidomain`.

43 top_path_sharing (library)

43.1 Usage and interface

- **Library usage:**
  `:- use_module(domain(top_path_sharing)).`
- **Exports:**
  - *Predicates:*
    `path_to_shfr/3`.
  - *Multifiles:*
    `aidomain/1`, `aidom.init_abstract_domain/2`, `aidom.amgu/5`, `aidom.augment_asub/4`, `aidom.augment_two_asub/4`, `aidom.call_to_entry/10`, `aidom.exit_to_prime/8`, `aidom.project/6`, `aidom.widencall/4`, `aidom.needs/2`, `aidom.widen/4`, `aidom.compute_lub/3`, `aidom.compute_clauses_lub/4`, `aidom.compute_clauses_glb/4`, `aidom.fixpoint_covered/3`, `aidom.fixpoint_covered_gfp/3`, `aidom.identical_abstract/3`, `aidom.abs_sort/3`, `aidom.extend/6`, `aidom.less_or_equal/3`, `aidom.glb/4`, `aidom.eliminate_equivalent/3`, `aidom.abs_subset/3`, `aidom.call_to_success_fact/10`, `aidom.special_builtin/6`, `aidom.combined_special_builtin0/3`, `aidom.body_succ_builtin/9`, `aidom.split_combined_domain/4`, `aidom.success_builtin/7`, `aidom.call_to_success_builtin/7`, `aidom.obtain_info/5`, `aidom.input_interface/5`, `aidom.input_user_interface/6`, `aidom.asub_to_native/6`, `aidom.concrete/4`, `aidom.unknown_call/5`, `aidom.unknown_entry/4`, `aidom.empty_entry/4`, `aidom.part_conc/5`, `aidom.multi_part_conc/4`, `aidom.collect_auxinfo_asub/4`, `aidom.rename_auxinfo_asub/4`, `aidom.dom_statistics/2`, `aidom.contains_parameters/2`.

43.2 Documentation on exports

path_to_shfr/3: PREDICATE

No further documentation available for this predicate.

43.3 Documentation on multifiles

aidomain/1: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.init_abstract_domain/2: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.amgu/5: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.augment_asub/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.augment_two_asub/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.call_to_entry/10: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.exit_to_prime/8: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.project/6: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.widencall/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.needs/2: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.widen/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.compute_lub/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.compute_clauses_lub/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.compute_clauses_glb/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.fixpoint_covered/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.fixpoint_covered_gfp/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.identical_abstract/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.abs_sort/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.extend/6: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.less_or_equal/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.glb/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.eliminate_equivalent/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.abs_subset/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.call_to_success_fact/10: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.special_builtin/6: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.combined_special_builtin0/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.body_succ_builtin/9: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.split_combined_domain/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.success_builtin/7: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.call_to_success_builtin/7: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.obtain_info/5: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.input_interface/5: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.input_user_interface/6: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.asub_to_native/6: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.concrete/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.unknown_call/5: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.unknown_entry/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.empty_entry/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.part_conc/5: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.multi_part_conc/4: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.collect_auxinfo_asub/4: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.rename_auxinfo_asub/4: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.dom_statistics/2: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.contains_parameters/2: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

43.4 Documentation on imports

This module has the following direct dependencies:

- *Application modules:*
  `preprocess_flags`, `domains`, `s_eqs`, `share_aux`, `plai_errors`.
- *System library modules:*
  `sets`, `sort`, `idlists`, `keys`, `lists`, `lsets`, `terms_check`, `terms_vars`.
- *Internal (engine) modules:*
  `term_basic`, `arithmetic`, `atomic_basic`, `basiccontrol`, `exceptions`, `term_compare`, `term_typing`, `debugger_support`, `basic_props`, `io_basic`.
- *Packages:*
  `prelude`, `initial`, `condcomp`, `assertions`, `assertions/assertions_basic`, `aidomain`.

44 Available program transformations

44.0.1 Program specialization

- `simp` This transformation tries to explote analysis information in order to *simplify* the program as much as possible. It includes optimizations such as abstract executability of literals, removal of useless clauses, and unfolding of literals for predicates which are defined by just a fact or a single clause with just one literal in its body (a *bridge*). It also propagates failure backwards in a clause as long as such propagation is safe.
- `spec` This transformation performs the same optimizations as `simp` but it also performs multiple specialization when this improves the possibilities of optimization. The starting point for this transformation is not a program annotated with analysis information, as in the case above, but rather an *expanded program* which corresponds to the analysis graph computed by multi-variant abstract interpretation. A minimization algorithm is used in order to guarantee that the resulting program is minimal in the sense that further collapsing versions would represent losing opportunities for optimization.
- `vers` This transformation has in common with `spec` that it takes as starting point the *expanded program* which corresponds to the analysis graph computed by abstract interpretation. However, this transformation performs no optimizations and does not minimize the program. As a result, it generates the expanded program.

44.0.2 Partial evaluation

- `codegen` This generates the specialized program resulting from partial evaluation, obtained by unfolding goals during analysis. The kind of unfolding performed is governed by the `comp_rule` flag, as follows:
  - `leftmost` unfolds the leftmost clause literal;
  - `eval_builtin` selects for unfolding first builtins which can be evaluated;
  - `local_emb` tries to select first atoms which do not endanger the embedding ordering or evaluable builtins whenever possible;
  - `jump_builtin` selects the leftmost goal but can 'jump' over (ignore) builtins when they are not evaluable. A main difference with the other computation rules is that unfolding is performed 'in situ', i.e., without reordering the atoms in the clause.
  - `safe_jb` same as `jump_builtin` with the difference that it only jumps over a call to a builtin iff the call is safe [APG06] (i.e., it is error free, binding insensitive and side effect free).
  - `bind_ins_jb` same as `safe_jb` with the difference that it only jumps over a call to a builtin iff the call is binding insensitive and side effect free.
  - `no_sideff_jb` same as `bind_ins_jb` with the difference that it only jumps over a call to a builtin iff it is side effect free.

  Unfolding is performed continuously on the already unfolded clauses, until a condition for stopping the process is satisfied. This condition is stablished by the local control policy, governed by the `local_control` flag, as follows:
  - `inst` allows goal instantiation but no actual unfolding is performed.
  - `orig` returns the clauses in the original program for the corresponding predicate.
  - `det` allows unfolding while derivations are deterministic and stops them when a non-deterministic branch is required. Note that this may not be terminating.
  - `det_la` same as `det`, but with look-ahead. It can perform a number of non-deterministic steps in the hope that the computation will turn deterministic. This number is determined by flag `unf_depth`.
  - `depth` always performs the same number of unfolding steps for every call pattern. The number is determined by flag `unf_depth`.
  - `first_sol` explores the SLD tree width-first and keeps on unfolding until a first solution is found. It can be non-terminating.
  - `first_sol_d` same as above, but allows terminating when a given depth bound is reached without obtaining any solution. The bound is determined by `unf_depth`.
  - `all_sol` tries to generate all solutions by exploring the whole SLD tree. This strategy only terminates if the SLD is finite.
  - `hom_emb` keeps on unfolding until the selected atom is homeomorphically embedded in an atom previously selected for unfolding.
  - `hom_emb_anc` same as before, but only takes into account previously selected atoms which are ancestors of the currently selected atom.
  - `hom_emb_as` same as before, but efficiently implemented by using a stack to store ancestors.
  - `df_hom_emb_as` same as before, but traverses the SLD tree on a depth-first fashion (all strategies above use wide-first search). This allows better performance.
  - `df_tree_hom_emb` same as above, but does not use the efficient stack-based implementation for ancestors.
  - `df_hom_emb` same as above, but compares with all previously selected atoms, and not only ancestors. It is like `hom_emb` but with depth-first traversal.
- `global_control` In order to guarantee termination of the partial evaluation process, it is often required to abstract away information before unfolding. This is usually known as global control. This flag can have the following values:
  - `off` unfolds always;
  - `id` unfolds patterns which are not equal (modulo renaming) to a formerly analyzed pattern.
  - `inst` unfolds patterns which are not an instance of a previous pattern.
  - `hom_emb` unfolds patterns which are not covered under the homeomorphic embedding ordering [Leu98].
  - `hom_emb_num` same as `hom_emb`, but also considers that any number embeds any other number.

  Only `hom_emb` guarantees termination. However, `id` and `inst` are more efficient, and terminating in many practical cases.
- `arg_filtering` This transformation removes from program literals static values which are not needed any longer in the resulting program. This is typically the case when some information is known at compile-time about the run-time values of arguments.
- `codegen_af` This performs `codegen` and `arg_filtering` in a single traversal of the code. Good for efficiency.

44.0.3 Code size reduction

- `slicing` This transformation is very useful for debugging programs since it isolates those predicates that are reachable from a given goal. The goals used are those exported by the module. The 'slice' being obtained is controlled by the following local control policies (described above): `df_hom_emb_as`, `df_hom_emb`, `df_tree_hom_emb`. It is also necessary to analyze the program with any of the currently available analyses for partial evaluation. Slicing is also very useful in order to perform other software engineering tasks, such as program understanding, maintenance, specialization, code reuse, etc.

44.0.4 Program parallelization

Parallelization is performed by considering goals the execution of which can be deemed as *independent* [HR95, GHM00] under certain conditions. Parallel expressions (possibly conditional) are built from such goals, in the following fashions:

- `mel` exploits parallel expressions which preserve the ordering of literals in the clauses;
- `cdg` tries to exploit every possible parallel expression, without preserving the initial ordering;
- `udg` is as above, but only exploits unconditional parallel expressions [MBdlBH99];
- `urlp` exploits unconditional parallel expressions for NSIAP with *a posteriori* conditions [CH94].
- `crlp` exploits conditional parallel expressions for NSIAP with *a posteriori* conditions.
- `granul` This transformation allows to perform run-time task granularity control of parallelized code (see [LGHD96]), so that the program will decide at run-time whether to run parallel expressions or not. The decision is based on the value of flag `granularity_threshold`.

44.0.5 Abstract partial deduction

Partial deduction (or partial evaluation) is a program transformation technique which specializes the program w.r.t. information known at compile-time. In CiaoPP this is performed during analysis of the program, so that not only concrete information but also abstract information (from the analysis) can be used for specialization. With analysis domain `pd` (and `pdb`) only concrete values will be used; with other analysis domains the domain abstract values inferred will also be used. This feature is governed by the following flags:

- `abs_spec_defs` to exploit abstract substitutions in order to:
  - `rem` try to eliminate clauses which are incompatible with the inferred substitution at each unfolding step;
  - `exec` perform abstract executability of atoms;
  - `all` do both.
- `part_concrete` to try to convert abstract information into concrete information if possible, so that:
  - `mono` one concrete atom is obtained;
  - `multi` multiple atoms are allowed when the information in the abstract substitution is disjunctive.
- `rem_use_cls` to identify clauses which are incompatible with the abstract call substitution and remove them:
  - `pre` prior to performing any unfolding steps;
  - `post` after performing unfolding steps;
  - `both` both before and after performing unfolding steps.
- `filter_nums` to filter away during partial evaluation numbers which:
  - `safe` are not safe, i.e., do not appear in the original program, or
  - `on` all numbers.

45 Available fixpoints

45.0.1 Analysis with PLAI

Most of the analyses of CiaoPP are performed with the PLAI (Programming in Logic with Abstract Interpretation) framework [BGH94]. This framework is based on the computation of a fixed point for the information being inferred. Such a fixed point computation is governed by flag `fixpoint`, whose values are:

- `plai` for the classical fixed point computation [MH89a];
- `dd` for an incremental fixed point computation [HPMS00];
- `di` for the *depth independent* fixed point algorithm of [PH96];
- `check_di` .

45.0.2 Inter-modular analysis

In inter-modular analysis CiaoPP takes into account the results of analyzing a module when other modules in the same program are analyzed. Thus, it collects analysis results (success patterns) for calls to predicates in other modules to improve the analysis of a given module. It also collects calls (call patterns) that are issued by the given module to other modules to reconsider them during analysis of such other modules.

Such flow of analysis information between modules while being analyzed can be performed when analyzing one single module. The information flow then affects only the modules imported by it. New call patterns will be taken into account when/if it is the turn for such imported modules to be analyzed. Improved success patterns will only be reused when/if the importing module is reanalyzed. However, CiaoPP can also iterate continuously over the set of modules of a given program, transferring the information from one module to others, and deciding which modules to analyze at which moment. This will be done until an inter-modular fixed point is reached in the analysis of the whole program (whereas analysis is performed one-module-at-a-time, anyway).

Inter-modular analysis is enabled with flag `intermod`. During inter-modular analysis there are several possible choices for selecting success patterns and call patterns. For example, when a success pattern is required for a given call pattern to an imported predicate, and there exist several that could be used, but none of them fit exactly with the given call pattern. Also, if, in that same case, there are no success patterns that fit (in which case CiaoPP has to make an *initial guess*). Finally, when there are new call patterns to a given module obtained during analysis of the modules that import it, which of them to use as entry points should be decided. All these features are governed by the following flags:

- `intermod` to activate inter-modular analysis.
  - `off` disables inter-modular analysis. This is the default value.
  - `on` enables inter-modular analysis.
  - `auto` allows the analysis of a modular program, using `intermod:auto_analyze/2-3` with the main module of the program, iterating through the module graph until an inter-modular fixed point is reached. This value is set automatically by CiaoPP, and it should not be set by the user.
- `success_policy` to obtain success information for given call patterns to imported predicates.
  - `best` selects the success pattern which corresponds to the best over-approximation of the sought call pattern; if there are several non-comparable best over-approximations, one of them is chosen randomly.
  - `first` selects the first success pattern which corresponds to a call pattern which is an over-approximation of the sought call pattern.
  - `all` computes the greatest lower bound of the success patterns that correspond to over-approximating call patterns.
  - `top` selects `Top` (no information) as answer pattern for any call pattern.
  - `botfirst` selects the first success pattern which corresponds to a call pattern which is an under-approximation of the sought call pattern.
  - `botbest` selects the success pattern which corresponds to the best under-approximation of the sought call pattern; if there are several non-comparable best under-approximations, one of them is chosen randomly.
  - `botall` computes the least upper bound of the success patterns that correspond to under-approximating call patterns.
  - `bottom` selects `Bottom` (failure) as answer pattern for any call pattern.
- `initial_guess` to obtain an initial guess for the success pattern corresponding to a call pattern to an imported predicate when there is none that fully matches.
  - `botfirst` selects the success pattern already computed corresponding to the first call pattern which is an under-approximation of the given call pattern.
  - `botbest` selects the success pattern corresponding to the call pattern which best under-approximates the given call pattern (if there are several, non-comparable call patterns, one of them is selected randomly).
  - `botall` computes the least upper bound of the success patterns that correspond to under-approximating call patterns.
  - `bottom` selects `Bottom` as initial guess for any call pattern.
- `entry_policy` to obtain entry call patterns for exported predicates.
  - `all` selects all entry call patterns for the current module which have not been analyzed yet, either from entry assertions found in the source code, or from the analysis of other modules that import the current module.
  - `top_level` is only meaningful during `auto` inter-modular analysis, and it is set automatically by CiaoPP. If the current module is the top-level module (the main module of the modular program being analyzed), the entry policy behaves like `all`. In any other case, it selects entry call patterns for the current module from the analysis of other modules that import it, ignoring entry assertions found in the source code.
  - `force` forces the analysis of all entries of the module (from both the module source code and calling modules), even if they have been already analyzed.
  - `force_assrt` forces the analysis of all entries coming from the module source code, but does not analyze entries relative to calling modules, even if they need to be (re)analyzed.
- `punit_boundary` to indicate which modules must be also analyzed when a modular user program is analyzed.
  - `off` disables the analysis of any Ciao system library.
  - `on` enables the analysis of all Ciao system libraries.
  - `no_engine` enables the analysis of Ciao system libraries which are not engine libraries.
  - `bundle` limits the modular analysis to the modules within the bundle of the entry module.
- `use_check_assrt` to indicate that check assertions for imported predicates will be used as trust assertions. This is specially interesting when performing intermodular compile-time checking.
  - `off` disables the use of check assertions as trust assertions for imported predicates.
  - `on` enables the use of check assertions as trust assertions.

46 Other commands

47 Display information about CiaoPP files

Author(s): Isabel Garcia-Contreras.

Stability: [**devel**] Currently the subject of active development and/or research. Functionality may be limited and API and/or functionality may change without warning or deprecation period. Not recommended yet for use in production.

This program outputs information about CiaoPP files.

47.1 Usage

```
ciaopp-dump <action> [<opts>] <files>

where the possible options are:

  -h
     Print this information

  show <path>
     Print info of the dump file or registry directory located in <path>.

  cmp <path1> <path2> <domain>
     Compares the analysis results of path1 and path2.
     May be directories or .dump files.

  fmt <filename>[.dump]
     Outputs the analysis result in dot format.

  ctcheck <modulename>.pl <filename>[.dump] <domain>
     Checks assertions agains the analysis in <filename>.dump with <domain>.

  report reach <file>[.dump]
     Deadcode and failure report of a CiaoPP analysis dumped with the
     incremental option, and analyzed with the pp_flag fact_info=on.

  stats [--print-header] <filenames>[.dump]
     Counts simple characteristics of analysis in the files.

     --print-header: prints the names of the characteristics displayed

  syntactic [--print-header] <filenames>[.pl]
     Counts simple syntactic characteristics of the source in the files.

     --print-header: prints the names of the characteristics displayed
```

48 CiaoPP batch processing

Author(s): Isabel Garcia-Contreras.

This is the command line interface for the batch analysis `ciaopp_batch(ciaopp_batch)` library.

Usage: `ciaopp-batch AbsInt [Paths]`

Analyze a list of directories:

```
?- use_module(ciaopp_batch(ciaopp_batch)).
?- analysis_start([Dir1, Dir2], []).
```

To see more options see page ⟨undefined⟩ [Analysis management predicates], page ⟨undefined⟩.

Generate a report of an analyzed directory:

The following command generates a directory `allure-results` in the directory from which the command is run. The diretory will contain an `xml` file for each of the domains that the directory was analyzed for.

```
$ ciaopp_batch_report Dir
```

Visualize a report:

The analysis report in xml can be processed with the allure tool. Once the tool is installed, visualize the results by running in the same directory as the results were generated

```
$ allure serve
```

49 XML report for ciaopp-batch output

Author(s): Isabel Garcia-Contreras.

This command generates a report in xml format compatible with junit xml (). It can then be used to visualize the results of launching ciaopp_batch with the allure tool (see), as well as in gitlab (.

50 PART II - The Assertion Language and Its Use

Author(s): The Ciao Development Team.

51 Using assertions for preprocessing programs

Author(s): Francisco Bueno.

This chapter explains the use of assertions to specify a program behaviour and properties expected to hold of the program. It also clarifies the role of assertion-related declarations so that a program can be statically preprocessed with CiaoPP.

CiaoPP starts a preprocessing session from a piece of code, annotated with assertions. The code can be either a complete self-contained program or part of a larger program (e.g., a module, or a user file which is only partial). The assertions annotating the code describe some properties which the programmer requires to hold of the program. Assertions are used also to describe to the static analyzer some properties of the interface of the code being preprocessed at a given session with other parts of the program that code belongs to. In addition, assertions can be used to provide information to the static analyzer, in order to guide it, and also to control specialization and other program transformations.

This chapter explains the use of assertions in describing to CiaoPP: (1) the program specification, (2) the program interface, and (3) additional information that might help static preprocessing of the program.

In the following, the Ciao assertion language is briefly described and heavily used. In Chapter 52 [The Ciao assertion language], page 335, a complete reference description of assertions is provided. More detailed explanations of the language can be found in [PBH00].

This chapter also introduces and uses properties, and among them (regular) types. See Chapter 55 [Basic data types and properties], page 357, for a concrete reference of (some of) the Ciao properties. See Chapter 54 [Declaring regular types], page 351, for a presentation of the Ciao type language and an explanation on how you can write your own properties and types.

Most of the predicates used below which are not defined belong to the *ISO-Prolog* standard [DEDC96]. The builtin (or primitive) constraints used have also become more or less de-facto standard. For detailed descriptions of particular constraint logic programming builtins refer for example to the CHIP [COS96], PrologIV [PRO], and Ciao [BCC04] manuals.

51.1 Assertions

Predicate assertions can be used to declare properties of the execution states at the time of calling a predicate and upon predicate success. Also, properties of the computation of the calls to a predicate can be declared.

Assertions may be qualified by keywords `check` or `trust`. Assertions qualified with the former—or not qualifed—are known as check assertions; those qualified with the latter are known as trust assertions. Check assertions state the programmer's intention about the program and are used by the debugger to check for program inconsistencies. On the contrary, trust assertions are "trusted" by CiaoPP tools.

- The specification of a program is made of all check assertions for the program predicates.

51.1.1 Properties of success states

They are similar in nature to the *postconditions* used in program verification. They can be expressed in our assertion language using the basic assertion:

```
:- success Goal => Postcond.
```

This assertion should be interpreted as, "for any call of the form `Goal` which succeeds, on success `Postcond` should also hold" .

Note that, in contrast to other programming paradigms, calls to a predicate may either succeed or fail. The postcondition stated in a `success` assertion only refers to successful executions.

51.1.2 Restricting assertions to a subset of calls

Sometimes we are interested in properties which refer not to all invocations of a predicate, but rather to a subset of them. With this aim we allow the addition of preconditions (`Precond`) to predicate assertions as follows: '`Goal : Precond`'.

For example, `success` assertions can be restricted and we obtain an assertion of the form:

```
:- success Goal : Precond => Postcond.
```

which should be interpreted as, "for any call of the form `Goal` for which `Precond` holds, if the call succeeds then on success `Postcond` should also hold".

51.1.3 Properties of call states

It is also possible to use assertions to describe properties about the calls for a predicate which may appear at run-time. An assertion of the form:

```
:- calls Goal : Cond.
```

must be interpreted as, "all calls of the form `Goal` should satisfy `Cond`".

51.1.4 Properties of the computation

Many other properties which refer to the computation of the predicate (rather than the input-output behaviour) are not easily expressible using `calls` and `success` predicate assertions only. Examples of properties of the computation which we may be interested in are: non-failure, termination, determinacy, non-suspension, etc.

This sort of properties are expressed by an assertion of the form:

```
:- comp Goal : Precond + Comp-prop.
```

which must be interpreted as, "for any call of the form `Goal` for which `Precond` holds, `Comp-prop` should also hold for the computation of `Goal`". Again, the field '`: Precond`' is optional.

51.1.5 Compound assertions

In order to facilitate the writing of assertions, a compound predicate assertion can be used as syntactic sugar for the above mentioned basic assertions. Each compound assertion is translated into one or several basic assertions, depending on how many of the fields in the compound assertion are given. The compound assertion is as follows.

```
:- pred Pred : Precond => Postcond + Comp-prop.
```

Each such compound assertion corresponds to: a `success` assertion of the form:

```
:- success Pred : Precond => Postcond.
```

if the `pred` assertion has a `=>` field (and a `:` field). It also corresponds to a `comp` assertion of the form:

```
:- comp Pred : Precond + Comp-prop.
```

if the `pred` assertion has a `+` field (and a `:` field).

All compound assertions given for the same predicate correspond to a single `calls` assertion. This `calls` assertion states as properties of the calls to the predicate a disjunction of the properties stated by the different compound assertions in their `:` field. Thus, it is of the form:

```
:- calls Pred : ( Precond1 ; ... ; Precondn ).
```

for all the `Precondi` in the `:` fields of (all) the different `pred` assertions.

Note that when compound assertions are used, `calls` assertions are always implicitly generated. If you do not want the `calls` assertion to be generated (for example because the set of assertions available does not cover all possible uses of the predicate) basic `success` or `comp` assertions rather than compound (`pred`) assertions should be used.

51.1.6 Examples of compound assertions

Consider the classical qsort program for sorting lists. We can use the following assertion in order to require that the output of procedure `qsort` be a list:

```
:- success qsort(A,B) => list(B).
```

Alternatively, we may require that if `qsort` is called with a list in the first argument position and the call succeeds, then on success the second argument position should also be a list. This is declared as follows:

```
:- success qsort(A,B) : list(A) => list(B).
```

The difference with respect to the previous assertion is that `B` is only expected to be a list on success of predicate `qsort/2` if `A` was a list at the call.

In addition, we may also require that in all calls to predicate `qsort` the first argument should be a list. The following assertion will do:

```
:- calls qsort(A,B) : list(A).
```

The `qsort` procedure should be able to sort all lists. Thus, we also require that all calls to it that have a list in the first argument and a variable in the second argument do not fail:

```
:- comp qsort(A,B) : (list(A) , var(B)) + not_fails.
```

Instead of the above basic assertions, the following compound one could be given:

```
:- pred qsort(A,B) : (list(A) , var(B)) => list(B) + not_fails.
```

which will be equivalent to:

```
:- calls qsort(A,B) : (list(A), var(B)).
:- success qsort(A,B) : (list(A), var(B)) => list(B).
:- comp qsort(A,B) : (list(A) , var(B)) + not_fails.
```

This will not allow to call `qsort` with anything else than a variable as second argument. If this use of `qsort` is expected, one should have added the assertion:

```
:- pred qsort(A,B) : list(A) => list(B).
```

which, together with the above one, will imply:

```
:- calls qsort(A,B) : ((list(A), var(B)) ; list(A)).
```

Then it is only required that `A` be a list.

51.2 Properties

Whereas each kind of assertion indicates *when*, i.e., in which states or sequences of states, to check the given properties, the properties themselves define *what* to check. Properties are used to say things such as “X is a list of integers,” “Y is ground,” “p(X) does not fail,” etc. and in Ciao they are logic predicates, in the sense that the evaluation of each property either succeeds or fails. The failure or success of properties typically needs to be determined at the time when the assertions in which they appear are checked. Assertions can be checked both at compile-time by CiaoPP and at run-time by Ciao itself (after the instrumentation of the program by CiaoPP). In this section we will concentrate exclusively on run-time checking.

A property may be a predefined predicate in the language (such as `integer(X)`) or constraint (such as `X>5`). Properties may include extra-logical predicates such as `var(X)`). Also, expressions built using conjunctions of properties,[1] or, in principle, any predicate defined by the user, using the full underlying (C)LP language. As an example, consider defining the predicate `sorted(B)` and using it as a postcondition to check that a more involved sorting algorithm such as `qsort(A,B)` produces correct results.

[1] Although disjunctions are also supported, we restrict our attention to only conjunctions.

While user-defined properties allow for properties that are as general as allowed by the full source language syntax, some limitations are useful in practice. Essentially, the behaviour of the program should not change in a fundamental way depending on whether the run-time tests are being performed or not. For example, turning on run-time checking should not introduce non-termination in a program which terminates without run-time checking. To this end, it is required that the user ensure that the execution of properties terminate for any possible initial state. Also, checking a property should not change the answers computed by the program or produce unexpected side-effects. Regarding computed answers, in principle properties are not allowed to further instantiate their arguments or add new constraints. Regarding side-effects, it is required that the code defining the property does not perform input/output, add/delete clauses, etc. which may interfere with the program behaviour. It is the user's responsibility to only use predicates meeting these conditions as properties. The user is required to identify in a special way the predicates which he or she has determined to be legal properties. This is done by means of a declaration of the form

```
:- prop Spec.
```

where `Spec` is a predicate specification in the form `PredName/Arity`.

Given the classes of assertions presented previously, there are two fundamental classes of properties. The properties used in the `Cond` of calls assertions, `Postcond` of success assertions, and `Precond` of success and comp assertions refer to a particular execution state and we refer to them as *properties of execution states*. The properties used in the `Comp-prop` part of comp assertions refer to a sequence of states and we refer to them as *properties of computations*.

Basic properties, including instantiation and compatibility state properties, types, and properties of computations (all discussed in Chapter 54 [Declaring regular types], page 351) are documented in Chapter 55 [Basic data types and properties], page 357.

51.3 Preprocessing units

The preprocessing unit is the piece of code that is made available to CiaoPP at a given preprocessing session. Normally, this is a file, but not all the code of a program is necessarily contained in one single file: in order to statically manipulate the code in a file, CiaoPP needs to know the interactions of this code with other pieces of the program—probably scattered over other files—, as well as what the user's interaction with the code will be upon execution. This is also done through the use of assertions.

If the preprocessing unit is self-contained the only interaction of its code (apart from calling the builtin predicates of the language) is with the user. The user's interaction with the program consists in querying the program. The predicates that may be directly queried by the user are entry points to the preprocessing unit.

Entry points can be declared in two ways: using a module declaration specifying the entry points, or using one entry declaration for each entry point. If entry declarations are used, instead of, or in addition to, the module declaration, they can also state properties which will hold at the time the predicate is called.

However, if the preprocessing unit is not self-contained, but only part of a larger program, then other interactions may occur. The interactions of the preprocessing unit include: the user's queries, calls from other parts of the program to the unit code, calls to the unit code from unit code which does not appear explicitely in the unit text, and calls from the unit code to other parts of the program.

First, other parts of the program can call predicates defined in the preprocessing unit. CiaoPP needs to know this information. It must be declared by specifying additional entry points, together with those corresponding to the user's queries.

Second, the preprocessing unit itself may contain meta-calls which may call any unspecified predicate. All predicates that may be called in such a way should be declared also as entry points.

Additional entry points also occur when there are predicates defined in the preprocessing unit which can be dynamically modified. In this case the code dynamically added can contain new predicate calls. These calls should be declared also as entry points.

Note that *all* entry points to the preprocessing unit should be declared: entry points including query calls that the user may issue to the program, or another part of the program can issue to the unit, but also *dynamic calls*: goals that may be run within the unit which do not appear explicitely in the unit text, i.e., from meta-predicates or from dynamic clauses which may be asserted during execution. In all cases, *entry* declarations are used to declare entry points.[2]

Third, the unit code may call predicates defined in other parts of the program. The code defining such predicates is termed *foreign code*, since it is foreign to the preprocessing unit. It is important that CiaoPP knows information about how calls to foreign code will succeed (if they succeed), in order to improve its accuracy. This can be done using *trust* declarations.

Also, trust declarations can be used to provide the preprocessor with extra information. They can be used to describe calls to predicates defined within the preprocessing unit, in addition to those describing foreign code. This can improve the information available to the preprocessor and thus help it in its task. Trust declarations state properties that the programmer knows to hold of the program.

The builtin predicates is one particular case of predicates the definitions of which are never contained in the program itself. Therefore, preprocessing units never contain code to define the builtins that they use. However, the Ciao Program Precompiler makes no assumptions on the underlying language (except that it is constraint logic programming). Thus, all information on the behaviour of the language builtins should be made available to it by means of assertions (although this does not concern the application programmer who is going to preprocess a unit, rather it concerns the system programmer when installing the Ciao Program Precompiler).

The rest of this document summarizes how assertions can be used to declare the preprocessing unit interactions. It shows the use of entry and trust declarations in preprocessing programs with CiaoPP.[3]

51.4 Foreign code

A program preprocessing unit may make use of predicates defined in other parts of the program. Such predicates are foreign to the preprocessing unit, i.e., their code is not in the unit itself. In this case, CiaoPP needs to know which is the effect that such predicates may cause on the execution of the predicates defined in the unit. For this purpose, trust declarations are used.

Foreign code includes predicates defined in other modules which are used by the preprocessing unit, predicates defined in other files which do not form part of the preprocessing unit but which are called by it, builtin predicates[4] used by the code in the preprocessing unit, and code written in a foreign language which will be linked with the program. All foreign calls (except to builtin predicates) need to be declared.[5]

---

[2] When the language supports a module system, entry points are implicitely declared by the exported predicates. In this case entry declarations are only used for local predicates if there are dynamic calls.

[3] This manual concentrates on one particular use of the declarations for solving problems related to compile-time program analysis. However, there are other possible solutions. For a complete discussion of these see [BCHP96].

[4] However, builtin predicates are usually taken care of by the system programmer, and the preprocessor, once installed, already “knows” them.

[5] However, if the language supports a module system, and the preprocessor is used in modular analysis operation mode, trust declarations are imported from other modules and do not need to be declared in the preprocessing unit.

- The effect of calls to foreign predicates may be declared by using trust declarations for such predicates.

Trust declarations have the following form:

```
:- trust success Goal : ( Prop, ..., Prop )
                      => ( Prop, ..., Prop ).
```

where `Goal` is an atom of the foreign predicate, with all arguments single distinct variables, and `Prop` is an atom which declares a property of one (or several) of the goal variables.

The first list of properties states the information at the time of calling the goal and the second one at the time of success of the goal. Thus, such a trust assertion declares that for any call to the predicate where the properties in the first list hold, those of the second will also hold upon success of the call.

Simplified versions of trust assertions can also be used, much the same as with entry declarations. See ⟨undefined⟩ [Assertions], page ⟨undefined⟩.

Trust declarations are a means to provide the preprocessor with extra information about the program states. This information is guaranteed to hold, and for this reason the preprocessor *trusts* it. Therefore, it should be used with great care, since if it is wrong the precompilation of your program will possibly be wrong.

51.4.1 Examples of trust assertions

The following annotations describe the behavior of the predicate `p/2` for two possible call patterns:

```
:- trust success p/2 : def * free => def * def.
:- trust success p/2 : free * def => free * def.
```

This would allow performing the analysis even if the code for `p/2` is not present. In that case the corresponding success information in the annotation can be used ("trusted") as success substitution.

In addition, trust declarations can be used to improve the results of compile-time program analysis when they are imprecise. This may improve the accuracy of the debugging, possibly allowing it to find more bugs.

51.5 Dynamic predicates

Predicate definitions can be augmented, reduced, and modified during program execution. This is done through the database manipulation builtins, which include `assert`, `retract`, `abolish`, and `clause`. These builtins (with the exception of `clause`) dynamically manipulate the program itself by adding to or removing clauses from it. Predicates that can be affected by such builtins are called dynamic predicates.

There are at least two possible classes of dynamic predicates which behave differently from the point of view of static manipulation. First, clauses can be asserted and/or retracted to maintain an information database that the program uses. In this case, usually only facts are asserted. Second, full clauses can be asserted for predicates which are also called within the program.

The first class of dynamic predicates are declared by data declarations. The second class by dynamic declarations. The form of both declarations is as follows:

```
:- data Spec, ..., Spec.
:- dynamic Spec, ..., Spec.
```

where `Spec` is a predicate specification in the form `PredName/Arity`.

- Dynamic predicates which are called must be declared by using a dynamic declaration.

Of course, the preprocessor cannot know of the effect that dynamic clauses added to the definition of a predicate may cause in the execution of that predicate. However, this effect can be described to the preprocessor by adding trust declarations for the dynamic predicates.

- The effect of calls to predicates which are dynamically modified may be declared by using trust declarations for such predicates.

51.6 Entry points

In a preprocessing session (at least) one entry point to the preprocessing unit is required. It plays a role during preprocessing similar to that of the query that is given to the program to run. Several entry points may be given. Entry points are given to the preprocessor by means of entry or module declarations.

If the preprocessing unit is a module, only the exported predicates can be queried. If the preprocessing unit is not a module, all of its predicates can be queried: all the unit predicates may be entry points to it. Entry declarations can then be used by the programmer to specify additional information about the properties that hold of the arguments of a predicate call when that predicate is queried.

Note that if the unit is not a module all of its predicates are considered entry points to the preprocessor. However, if the unit incorporates some entry declarations the preprocessor will act as if the predicates declared were the only entry points (the preprocessing session being valid for a particular use of the unit code—that specified by the entry declarations given).

- All predicates that can be queried by the user and all predicates that can be called from parts of the program which do not explicitely appear in the preprocessing unit should (but need not) be declared as entry points by using entry declarations.

The entry declaration has the following form:

```
:- entry Goal : ( Prop, ..., Prop ).
```

where `Goal` is an atom of the predicate that may be called, with all arguments single distinct variables, and `Prop` is an atom which declares a property of one (or several) of the goal variables. The list of properties is optional.

There are alternative formats in which the properties can be given: as the arguments of `Goal` itself, or as keywords of the declaration. For a complete reference of the syntax of assertions, see ⟨undefined⟩ [Assertions], page ⟨undefined⟩.

51.6.1 Examples of entry declarations

Consider the following program:

```
append([], L, L).
append([H|T], L, [H|R]) :- append(T, L, R).
```

It may be called in a classical way with the first two arguments bound to lists, and the third argument a free variable. This can be annotated in any of the following three ways:

```
:- entry append(X,Y,Z) : ( list(X), list(Y), var(Z) ).
:- entry append/3 : list * list * var.
:- entry append(list,list,var).
```

Assume you have the following program:

```
p(X,Y):- q(X,Y,Z).
q(X,Y,Z):- X = f(Y,Z), Y + Z = 3.
```

Assume that `p/2` is the only entry point. If you include the following declaration:

```
:- entry p/2.
```

or, equivalently,

```
:- entry p(X,Y).
```

the code will be preprocessed as if goal `p(X,Y)` was called with the most general call pattern (i.e., as if `X` and `Y` may have any two values, or no value at all—the variables being free).

However, if you know that `p/2` will always be called with the first argument uniquely defined and the second unconstrained, you can then provide more accurate information by introducing one of the following declarations:

```
:- entry p(X,Y) : ( def(X), free(Y) ).
:- entry p(def,free).
```

Now assume that `p/2` will always be called with the first argument bound to the compound term `f(A,B)` where `A` is definite and `B` is unconstrained, and the second argument of `p/2` is unconstrained. The entry declaration for this call pattern is:

```
:- entry p(X,Y) : ( X=f(A,B), def(A), free(B), free(Y) ).
```

If both call patterns are possible, the most accurate approach is to include both entry declarations in the preprocessing unit. The preprocessor will then analyze the program for each declaration. Another alternative is to include an entry declaration which approximates both call patterns, such as one of the following two:

```
:- entry p(X,Y) : free(Y).
:- entry p(X,free).
```

which state that `Y` is known to be free, but nothing is known of `X` (since it may or may not be definite).

51.7 Modules

Modules provide for encapsulation of code, in such a way that (some) predicates defined in a module can be used by other parts of the program (possibly other modules), but other (auxiliary) predicates can not. The predicates that can be used are exported by the module defining them and imported by the module(s) which use(s) them. Thus, modules provide for a natural declaration of the allowed entry points to a piece of a program.

A module is identified by a module declaration at the beginning of the file defining that module. The module declaration has the following form:

```
:- module(Name, [ Spec,...,Spec ] ).
```

where the module is named `Name` and it exports the predicates in the different `Spec`'s.

Note that such a module declaration is equivalent, for the purpose of static preprocessing, to as many entry declarations of the form:

```
:- entry Spec.
```

as there are exported `Spec`'s.

51.8 Dynamic calls

In addition to entry points there are other calls that may occur from within a piece of code which do not explicitely appear in the code itself. Among these are metacalls, callbacks, and calls from clauses which are asserted during program execution.

Metacalls are literals which call one of their arguments at run-time, converting at the time of the call a term into a goal. Predicates in this class are not only `call`, but also `bagof`, `findall`, `setof`, negation by failure, and `once` (single solution).

Metacalls may be static, and this kind of calls need not be declared. A static metacall is, for example, `once(p(X))`, where the predicate being called is statically identifiable (since it appears in the code). On the other hand, metacalls of the form `call(Y)` are dynamic, since the predicate being called will only be determined at runtime.[6]

Callbacks are also metacalls. A callback occurs when a piece of a program uses a different program module (or object) in such a way that it provides to that module the call that it should issue upon return. Callbacks, much the same as metacalls, can be either dynamic or static. Only the predicates of the preprocessing unit which can be dynamically called-back need be declared.

Clauses that are asserted during program execution correspond to code which is dynamically created; thus, the preprocessor cannot be aware of such code during a (compile-time) preprocessing session. The calls that may appear from the body of a clause which is dynamically created and asserted are also dynamic calls.

- All dynamic calls must be declared by using entry declarations for the predicates that can be called in a dynamic way.

51.8.1 Examples of dynamic calls

Consider a program where you use the `bagof` predicate to collect all solutions to a goal, and the program call looks like:

```
p(X,...) :- ..., bagof(P,X,L), ...
```

However, you know that, upon execution, only the predicates `p/2` and `q/3` will be called by `bagof`, i.e., `X` will only be bound to terms with functors `p/2` and `q/3`. Moreover, such terms will have all of their arguments constrained to definite values. This information should be given to the preprocessor using the declarations:

```
:- entry p(def,def).
:- entry q(def,def,def).
```

Assume you have a graphics library predicate `menu_create/5` which creates a graphic menu. The call must specify, among other things, the name of the menu, the menu items, and the menu handler, i.e., a predicate which should be called upon the selection of a menu item. The predicate is used as:

```
top :- ..., menu_create(Menu,0,Items,Callback,[]), ...
```

but the program is coded so that there are only two menu handlers: `app_menu/2` and `edit_menu/2`. The first one handles menu items of the type `app_item` and the second one items of the type `edit_item`. This should be declared with:

```
:- entry app_menu(gnd,app_item).
:- entry edit_menu(gnd,edit_item).
```

[6] However, sometimes analysis techniques can be used to transform dynamic metacalls into static ones.

Let a program have a dynamic predicate `dyn_calls/1` to which the program asserts clauses, such that these clauses do only have in their bodies calls to predicates `p/2` and `q/3`. This should be declared with:

```
:- entry p/2.
:- entry q/3.
```

Moreover, if the programmer knows that every call to `dyn_calls/1` which can appear in the program is such that upon its execution the calls to `p/2` and `q/3` have all of their arguments constrained to definite values, then the two entry declarations at the beginning of the examples may be used.

51.9 An overview

To process programs with the Ciao Program Precompiler the following guidelines might be useful:

1. Add

   ```
   :- use_package(assertions).
   ```

   to your program.
2. Declare your specification of the program using calls, success, comp, or pred assertions.
3. Use entry declarations to declare all entry points to your program.
4. The preprocessor will notify you during the session of certain program points where a meta-call appears that may call unknown (at compile-time) predicates.

   Add entry declarations for all the predicates that may be dynamically called at such program points.
5. Use data or dynamic declarations to declare all predicates that may be dynamically modified.
6. Add entry declarations for the dynamic calls that may occur from the code that the program may dynamically assert.
7. Optionally, you can interact with the preprocessor using trust assertions.

   For example, the preprocessor will notify you during the session of certain program points where a call appears to an unknown (at compile-time) predicate.

   Add trust declarations for such predicates.

52 The Ciao assertion language

Author(s): Manuel Hermenegildo, Francisco Bueno, German Puebla.

The `assertions` package adds a number of new declaration definitions and new operator definitions which allow including program assertions in user programs. Such assertions can be used to describe predicates, properties, modules, applications, etc. These descriptions can contain formal specifications (such as sets of preconditions, post-conditions, or descriptions of computations) as well as machine-readable textual comments.

This module is part of the `assertions` library. It defines the basic code-related assertions, i.e., those intended to be used mainly by compilation-related tools, such as the static analyzer or the run-time test generator.

Here we document mainly the use of assertions for providing specifications for predicates and other program elements, as well as the locations within assertions where machine-readable documentation strings can be placed. The commands that can be used in the documentation strings and other directives that can be used to provide additional machine-readable comments are described in the autodocumenter (`lpdoc` [Knu84, Her99]) manual.

There are two kinds of assertions: predicate assertions and program point assertions. Predicate assertions are placed as directives in the source code, i.e., preceded by "`:-`". Program point assertions are placed as literals in clause bodies. Additional documentation on the syntax and fields of predicate assertions can be found in the Chapter 53 [Types and properties related to assertions], page 345 module.

52.1 Getting more information

This documentation is intended to provide information at a "reference manual" level. For more tutorial introductions to the assertion language and more examples please see [BCHP96, HPB99, PBH00, HPBLG05, HBC12] and the `ciaopp` tutorial. % The assertion language as implemented in this library essentially follows these documents, although, due to its evolution, it may differ in some details. The purpose of this manual is to document precisely what the implementation of the library supports at any given point in time.

52.2 Some attention points

- **Formatting commands within text strings:** many of the predicates defined in these modules include arguments intended for providing textual information. This includes titles, descriptions, comments, etc. The type of this argument is a character string. In order for the automatic generation of documentation to work correctly, this character string should adhere to certain conventions. See the description of the `docstring/1` type/grammar for details.
- **Referring to variables:** In order for the automatic documentation system to work correctly, variable names (for example, when referring to arguments in the head patterns of *pred* declarations) must be surrounded by an `@var` command. For example, `@var{VariableName}` should be used for referring to the variable "VariableName", which will appear then formatted as follows: `VariableName`. See the description of the `docstring/1` type/grammar for details.

52.3 Usage and interface

- **Library usage:**
  `:- use_package(assertions).` or `:- module(...,...,[assertions]).`. Use the `assertions_basic` package to disable the implicit use of the `basic_props` module.
- **Exports:**
  - *Predicates:*
    `check/1`, `trust/1`, `true/1`, `false/1`.
- **New operators defined:**
  `=>/2` [975,xfx], `::/2` [978,xfx], `decl/1` [1150,fx], `decl/2` [1150,xfx], `pred/1` [1150,fx], `pred/2` [1150,xfx], `prop/1` [1150,fx], `prop/2` [1150,xfx], `modedef/1` [1150,fx], `calls/1` [1150,fx], `calls/2` [1150,xfx], `success/1` [1150,fx], `success/2` [1150,xfx], `comp/1` [1150,fx], `comp/2` [1150,xfx], `entry/1` [1150,fx], `exit/1` [1150,fx], `exit/2` [1150,xfx], `test/1` [1150,fx], `test/2` [1150,xfx], `texec/1` [1150,fx], `texec/2` [1150,xfx].
- **New declarations defined:**
  `pred/1`, `pred/2`, `calls/1`, `calls/2`, `success/1`, `success/2`, `comp/1`, `comp/2`, `prop/1`, `prop/2`, `test/1`, `test/2`, `texec/1`, `texec/2`, `entry/1`, `exit/1`, `exit/2`, `modedef/1`, `decl/1`, `decl/2`, `doc/2`, `comment/2`.
- **Implicit imports:**
  - *System library modules:*
    `assertions_props`.
  - *Internal (engine) modules:*
    `term_basic`, `arithmetic`, `atomic_basic`, `basiccontrol`, `exceptions`, `term_compare`, `term_typing`, `debugger_support`, `basic_props`.
  - *Packages:*
    `prelude`, `initial`, `condcomp`, `assertions/assertions_basic`.

52.4 Documentation on new declarations

pred/1: DECLARATION

`pred` assertions are the most general type of assertion, and they are used to provide information on the set of admissible calls to a predicate, the set of successes, global properties, and documentation. The body of a `pred` assertion (its only argument) contains properties or comments in different fields following the formats specified by `assrt_body/1`.

There can be more than one of these assertions per predicate, in which case each one represents a possible " mode" of use (usage) of the predicate. The exact scope of the usage is defined by the properties given for calls in the body of each assertion (which should thus distinguish the different usages intended). Predicates are typically specified using a *set* of `pred` assertions, so that together they cover all ways in which the predicate is intended be used (all usages). Each `pred` assertion is translated internally to a `calls` assertion covering all the calling modes and a `success` assertion covering the successes for those calls.

For example, the following assertions would describe all intended modes (and the only modes) of use of a predicate `length/2` (see `lists`):

```
:- pred length(L,N) : list * var => list * integer
    # "Computes the length of L.".
```

```
:- pred length(L,N) : var * integer => list * integer
    # "Outputs L of length N.".
:- pred length(L,N) : list * integer => list * integer
    # "Checks that L is of length N.".
```

Usage: :- `pred(AssertionBody)`.
- *The following properties should hold at call time:*
  `AssertionBody` is an assertion body. (`assrt_body/1`)

pred/2: DECLARATION

A `pred` assertion (see `pred/1`) can be preceded (as most other assertions) by an assertion status (`assrt_status/1`). If no assertion status is present (i.e., in `pred/1` assertions) the status is assumed to be `check`.

For example, the following assertion:

```
:- pred length(L,N) : list * var => list * integer.
```

is equivalent to:

```
:- check pred length(L,N) : list * var => list * integer.
```

Usage: :- `pred(AssertionStatus,AssertionBody)`.
- *The following properties should hold at call time:*
  `AssertionStatus` is an acceptable status for an assertion. (`assrt_status/1`)
  `AssertionBody` is an assertion body. (`assrt_body/1`)

calls/1: DECLARATION

`calls` assertions are similar to `pred/1` assertions but the only provide information about the calls to a predicate. The set of calls assertions for a predicate describe *all* possible calls to the predicate.

For example, the following assertion describes all possible calls to predicate `is/2` (see `arithmetic`):

```
:- calls is(term,arithexpression).
```

Usage: :- `calls(AssertionBody)`.
- *The following properties should hold at call time:*
  `AssertionBody` is a call assertion body. (`c_assrt_body/1`)

calls/2: DECLARATION

A `calls` assertion can be preceded (as most other assertions) by an assertion status (`assrt_status/1`). If no assertion status is present (i.e., in `calls/1` assertions) the status is assumed to be `check`.

Usage: :- `calls(AssertionStatus,AssertionBody)`.
- *The following properties should hold at call time:*
  `AssertionStatus` is an acceptable status for an assertion. (`assrt_status/1`)
  `AssertionBody` is a call assertion body. (`c_assrt_body/1`)

success/1: DECLARATION

`success` assertions specify properties of the answers of a predicate, similarly to the corresponding `success` field of a `pred` assertion. The assertion can be limited to apply to a particular way of calling the predicate if a `calls` field is present. However, unlike `pred` or `calls` assertions, the predicate is not forced to be called only that way.

For example, the following assertion specifies the answers of the `length/2` predicate *if* it is called as in the first mode of usage above (note that the previous pred assertion already conveys such information, however it also restricts the set of admissible `calls`, while the success assertion does not):

```
:- success length(L,N) : list * var => list * integer.
```

Usage: :- `success(AssertionBody)`.

– *The following properties should hold at call time:*

`AssertionBody` is a predicate assertion body. (`s_assrt_body/1`)

success/2: DECLARATION

A `success` assertion can be preceded (as most other assertions) by an assertion status (`assrt_status/1`). If no assertion status is present (i.e., in `success/1` assertions) the status is assumed to be `check`.

Usage: :- `success(AssertionStatus,AssertionBody)`.

– *The following properties should hold at call time:*

`AssertionStatus` is an acceptable status for an assertion. (`assrt_status/1`)

`AssertionBody` is a predicate assertion body. (`s_assrt_body/1`)

comp/1: DECLARATION

`comp` assertions specify global properties of the execution of a predicate, similarly to the corresponding `comp` field of a `pred` assertion. The assertion can be limited to apply to a particular way of calling the predicate if a `calls` field is present. However, unlike `pred` or `calls` assertions, the predicate is not forced to be called only that way.

For example, the following assertion specifies that the computation of `append/3` (see `lists`) will not fail *if* it is called as described (but does not force the predicate to be called only that way):

```
:- comp append(Xs,Ys,Zs) : var * var * var + not_fail.
```

Usage: :- `comp(AssertionBody)`.

– *The following properties should hold at call time:*

`AssertionBody` is a comp assertion body. (`g_assrt_body/1`)

comp/2: DECLARATION

A `comp` assertion can be preceded (as most other assertions) by an assertion status (`assrt_status/1`). If no assertion status is present (i.e., in `comp/1` assertions) the status is assumed to be `check`.

Usage: :- `comp(AssertionStatus,AssertionBody)`.

– *The following properties should hold at call time:*

`AssertionStatus` is an acceptable status for an assertion. (`assrt_status/1`)

`AssertionBody` is a comp assertion body. (`g_assrt_body/1`)

prop/1: DECLARATION

`prop` assertions are similar to a `pred/1` assertions but they flag that the predicate being documented is also a " property."

Properties are standard predicates, but which are *guaranteed to terminate for any possible instantiation state of their argument(s)*, do not perform side-effects which may interfere with the program behaviour, and do not further instantiate their arguments or add new constraints.

Provided the above holds, properties can thus be safely used as run-time checks. The program transformation used in `ciaopp` for run-time checking guarantees the third requirement. It also performs some basic checks on properties which in most cases are enough for the second requirement. However, it is the user's responsibility to guarantee termination of the properties defined. (See also Chapter 54 [Declaring regular types], page 351 for some considerations applicable to writing properties.)

The set of properties is thus a strict subset of the set of predicates. Note that properties, in addition to being used to describe characteristics of arguments in assertions, they can also be executed (called) as any other predicates.

Usage: :- `prop(AssertionBody)`.

− *The following properties should hold at call time:*
  `AssertionBody` is an assertion body. (`assrt_body/1`)

prop/2: DECLARATION

This assertion is similar to a `prop/1` assertion but it is explicitely qualified. Non-qualified `prop/1` assertions are assumed the qualifier `check`.

Usage: :- `prop(AssertionStatus,AssertionBody)`.

− *The following properties should hold at call time:*
  `AssertionStatus` is an acceptable status for an assertion. (`assrt_status/1`)
  `AssertionBody` is an assertion body. (`assrt_body/1`)

test/1: DECLARATION

This assertion is similar to a success assertion but it specifies a concrete test case to be run in order verify (partially) that the predicate is working as expected. For example, the following test will verify that the length predicate works well for the particular list given:

```
:- test length(L,N) : ( L = [1,2,5,2] ) => ( N = 4 ).
```

Usage: :- `test(AssertionBody)`.

− *The following properties should hold at call time:*
  `AssertionBody` is a predicate assertion body. (`s_assrt_body/1`)

test/2: DECLARATION

This assertion is similar to a `test/1` assertion but it is explicitely qualified with an assertion status. Non-qualified `test/1` assertions are assumed to have `check` status. In this context, check means that the test should be executed when the developer runs the test battery.

Usage: :- `test(AssertionStatus,AssertionBody)`.

- *The following properties should hold at call time:*
  `AssertionStatus` is an acceptable status for an assertion. (`assrt_status/1`)
  `AssertionBody` is a predicate assertion body. (`s_assrt_body/1`)

texec/1: DECLARATION

This assertion is similar to a `calls/1` assertion but it is used to provide input data and execution commands for run-time testing.

Usage: :- `texec(AssertionBody)`.

- *The following properties should hold at call time:*
  `AssertionBody` is a call assertion body. (`c_assrt_body/1`)

texec/2: DECLARATION

This assertion is similar to a `texec/1` assertion but it is explicitely qualified with an assertion status. Non-qualified `texec/1` assertions are assumed to have `check` status.

Usage: :- `texec(AssertionStatus,AssertionBody)`.

- *The following properties should hold at call time:*
  `AssertionStatus` is an acceptable status for an assertion. (`assrt_status/1`)
  `AssertionBody` is a call assertion body. (`c_assrt_body/1`)

entry/1: DECLARATION

`entry` assertions provide information about the *external* calls to a predicate. They are identical syntactically to a `calls/1` assertion. However, they describe only the external calls, i.e., the calls to the exported predicates of a module from outside the module, or calls to the predicates in a non-modular file from other files (or the user).

These assertions are *trusted* by the compiler, i.e., they are similar to writing a `trust calls` assertion (except for referring only to the external calls). As a result, if they are erroneous they can introduce bugs in programs. Thus, `entry` assertions should be written with care.

An important use of these assertions is in providing information to the compiler which it may not be able to infer from the program. The main use is in providing information on the ways in which exported predicates of a module will be called from outside the module. This will greatly improve the precision of the analyzer, which otherwise has to assume that the arguments that exported predicates receive are any arbitrary term.

The distinction between external and internal calls is not always relevant and in those cases the use of `trust calls` assertions is preferred. Because of this, `entry` assertions may be deprecated in the future, since the distinction between external and internal calls can also be achieved by means of a bridge predicate.

Usage: :- `entry(AssertionBody)`.

- *The following properties should hold at call time:*
  `AssertionBody` is a call assertion body. (`c_assrt_body/1`)

exit/1: DECLARATION

This type of assertion provides information about the answers that an (exported) predicate provides for *external* calls. It is identical syntactically to a `success/1` assertion. However, it describes only external answers, i.e., answers to the exported predicates of a module from outside the module, or answers to the predicates in a non-modular file from other files (or the user). The described answers may be conditioned to a particular way of calling the predicate. E.g.:

```
:- exit length(L,N) : list * var => list * integer.
```

These assertions are *trusted* by the compiler, i.e., they are similar to writing a `trust success` assertion (except for referring only to the external calls). As a result, if they are erroneous they can introduce bugs in programs. Thus, `exit` assertions should be written with care.

The distinction between external and internal calls is not always relevant and in those cases the use of `trust success` assertions is preferred. Because of this, `entry` assertions may be deprecated in the future, since the distinction between external and internal calls can also be achieved by means of a bridge predicate.

Usage: :- `exit(AssertionBody)`.

− *The following properties should hold at call time:*

`AssertionBody` is a predicate assertion body. (`s_assrt_body/1`)

exit/2: DECLARATION

exit assertion This assertion is similar to an `exit/1` assertion but it is explicitely qualified with an assertion status. Non-qualified `exit/1` assertions are assumed the qualifier `check`.

Usage: :- `exit(AssertionStatus,AssertionBody)`.

− *The following properties should hold at call time:*

`AssertionStatus` is an acceptable status for an assertion. (`assrt_status/1`)

`AssertionBody` is a predicate assertion body. (`s_assrt_body/1`)

modedef/1: DECLARATION

This assertion is used to define modes. A mode defines in a compact way a set of call and success properties. Once defined, modes can be applied to predicate arguments in assertions. The meaning of this application is that the call and success properties defined by the mode hold for the argument to which the mode is applied. Thus, a mode is conceptually a "property macro."

The syntax of mode definitions is similar to that of `pred` declarations. For example, the following set of assertions:

```
:- modedef +A : nonvar(A) # "A is bound upon predicate entry.".

:- pred p(+A,B) : integer(A) =>  ground(B).
```

is equivalent to:

```
:- pred p(A,B) : (nonvar(A),integer(A)) =>  ground(B)
   # "A is bound upon predicate entry.".
```

Usage: :- `modedef(AssertionBody)`.

− *The following properties should hold at call time:*

`AssertionBody` is an assertion body. (`assrt_body/1`)

decl/1: DECLARATION

This assertion is similar to a `pred/1` assertion but it is used to describe declarations instead of predicates.

Usage: :- `decl(AssertionBody)`.

- *The following properties should hold at call time:*

  `AssertionBody` is an assertion body. (`assrt_body/1`)

decl/2: DECLARATION

This assertion is similar to a `decl/1` assertion but it also has a status. Non-qualified `decl/1` assertions are assumed to have status `check`.

Usage: :- `decl(AssertionStatus,AssertionBody)`.

- *The following properties should hold at call time:*

  `AssertionStatus` is an acceptable status for an assertion. (`assrt_status/1`)

  `AssertionBody` is an assertion body. (`assrt_body/1`)

doc/2: DECLARATION

Usage: :- `doc(Pred,Comment)`.

Documentation . This assertion provides a text `Comment` for a given predicate `Pred`, as well as other directives for the documenter.

- *The following properties should hold at call time:*

  `Pred` is a head pattern. (`head_pattern/1`)

  `Comment` is a text comment with admissible documentation commands. The usual formatting commands that are applicable in comment strings are defined by `stringcommand/1`. See the `lpdoc` manual for documentation on comments. (`docstring/1`)

comment/2: DECLARATION

Usage: :- `comment(Pred,Comment)`.

An alias for `doc/2` (deprecated, for compatibility with older versions).

- *The following properties should hold at call time:*

  `Pred` is a head pattern. (`head_pattern/1`)

  `Comment` is a text comment with admissible documentation commands. The usual formatting commands that are applicable in comment strings are defined by `stringcommand/1`. See the `lpdoc` manual for documentation on comments. (`docstring/1`)

- *The following properties should hold globally:*

  DEPRECATED. (`deprecated/1`)

52.5 Documentation on exports

check/1: PREDICATE

Usage: `check(PropertyConjunction)`

This assertion provides information on a clause program point (position in the body of a clause). Calls to a `check/1` assertion can appear in the body of a clause in any place where a literal can normally appear. The property defined by `PropertyConjunction` should hold in all the run-time stores corresponding to that program point. See also Chapter 64 [Run-time checking of assertions], page 407.

- *The following properties should hold at call time:*

  `PropertyConjunction` is either a term or a *conjunction* of terms. The main functor and arity of each of those terms corresponds to the definition of a property. The first argument of each such term is a variable which appears as a head argument. (`property_conjunction/1`)

trust/1: PREDICATE

Usage: `trust(PropertyConjunction)`

This assertion also provides information on a clause program point. It is identical syntactically to a `check/1` assertion. However, the properties stated are not taken as something to be checked but are instead *trusted* by the compiler. While the compiler may in some cases detect an inconsistency between a `trust/1` assertion and the program, in all other cases the information given in the assertion will be taken to be true. As a result, if these assertions are erroneous they can introduce bugs in programs. Thus, `trust/1` assertions should be written with care.

An important use of these assertions is in providing information to the compiler which it may not be able to infer from the program (either because the information is not present or because the analyzer being used is not precise enough). In particular, providing information on external predicates which may not be accessible at the time of compiling the module can greatly improve the precision of the analyzer. This can be easily done with trust assertion.

- *The following properties should hold at call time:*

  `PropertyConjunction` is either a term or a *conjunction* of terms. The main functor and arity of each of those terms corresponds to the definition of a property. The first argument of each such term is a variable which appears as a head argument. (`property_conjunction/1`)

true/1: PREDICATE

Usage: `true(PropertyConjunction)`

This assertion is identical syntactically to a `check/1` assertion. However, the properties stated have been proved to hold by the analyzer. Thus, these assertions often represent the analyzer output.

- *The following properties should hold at call time:*

  `PropertyConjunction` is either a term or a *conjunction* of terms. The main functor and arity of each of those terms corresponds to the definition of a property. The first argument of each such term is a variable which appears as a head argument. (`property_conjunction/1`)

false/1: PREDICATE

Usage: `false(PropertyConjunction)`

This assertion is identical syntactically to a `check/1` assertion. However, the properties stated have been proved not to hold by the analyzer. Thus, these assertions often represent the analyzer output.

- *The following properties should hold at call time:*

  `PropertyConjunction` is either a term or a *conjunction* of terms. The main functor and arity of each of those terms corresponds to the definition of a property. The first argument of each such term is a variable which appears as a head argument. (`property_conjunction/1`)

53 Types and properties related to assertions

Author(s): Manuel Hermenegildo.

This module is part of the `assertions` library. It provides the formal definition of the syntax of several forms of assertions and describes their meaning. It does so by defining types and properties related to the assertions themselves. The text describes, for example, the overall fields which are admissible in the bodies of assertions, where properties can be used inside these bodies, how to combine properties for a given predicate argument (e.g., conjunctions) , etc. and provides some examples.

53.1 Usage and interface

- **Library usage:**
  `:- use_module(library(assertions/assertions_props)).`
- **Exports:**
  - *Properties:*
    `head_pattern/1`, `nabody/1`, `docstring/1`.
  - *Regular Types:*
    `assrt_body/1`, `complex_arg_property/1`, `property_conjunction/1`, `property_starterm/1`, `complex_goal_property/1`, `dictionary/1`, `c_assrt_body/1`, `s_assrt_body/1`, `g_assrt_body/1`, `assrt_status/1`, `assrt_type/1`, `predfunctor/1`, `propfunctor/1`.

53.2 Documentation on exports

assrt_body/1: REGTYPE

This predicate defines the different types of syntax admissible in the bodies of `pred/1`, `decl/1`, etc. assertions. Such a body is of the form:

```
Pr [:: DP] [: CP] [=> AP] [+ GP] [# CO]
```

where (fields between [...] are optional):

- `Pr` is a head pattern (`head_pattern/1`) which describes the predicate or property and possibly gives some implicit call/answer information.
- `DP` is a (possibly empty) complex argument property (`complex_arg_property/1`) which expresses properties which are compatible with the predicate, i.e., instantiations made by the predicate are *compatible* with the properties in the sense that applying the property at any point would not make it fail.
- `CP` is a (possibly empty) complex argument property (`complex_arg_property/1`) which applies to the *calls* to the predicate.
- `AP` is a (possibly empty) complex argument property (`complex_arg_property/1`) which applies to the *answers* to the predicate (if the predicate succeeds). These only apply if the (possibly empty) properties given for calls in the assertion hold.
- `GP` is a (possibly empty) complex goal property (`complex_goal_property/1`) which applies to the *whole execution* of a call to the predicate. These only apply if the (possibly empty) properties given for calls in the assertion hold.

- `CO` is a comment string (`docstring/1`). This comment only applies if the (possibly empty) properties given for calls in the assertion hold. The usual formatting commands that are applicable in comment strings can be used (see `stringcommand/1`).

  See the `lpdoc` manual for documentation on assertion comments.

Usage: `assrt_body(X)`

`X` is an assertion body.

head_pattern/1: PROPERTY

A head pattern can be a predicate name (functor/arity) (`predname/1`) or a term. Thus, both `p/3` and `p(A,B,C)` are valid head patterns. In the case in which the head pattern is a term, each argument of such a term can be:

- A variable. This is useful in order to be able to refer to the corresponding argument positions by name within properties and in comments. Thus, `p(Input,Parameter,Output)` is a valid head pattern.
- A variable, as above, but preceded by a “ mode.” This mode determines in a compact way certain call or answer properties. For example, the head pattern `p(Input,+Parameter,Output)` is valid, as long as `+/1` is declared as a mode.

  Some useful mode definitions can be found in the `basicmodes` and `isomodes` libraries. Users can define modes using the `modedef/1` declaration.
- Any term. In this case this term determines the instantiation state of the corresponding argument position of the predicate calls to which the assertion applies.
- A ground term preceded by a “ mode.” The ground term determines a property of the corresponding argument. The mode determines if it applies to the calls and/or the successes. The actual property referred to is that given by the term but with one more argument added at the beginning, which is a new variable which, in a rewriting of the head pattern, appears at the argument position occupied by the term. For example, the head pattern `p(Input,+list(int),Output)` is valid for mode `+/1` defined in `library(isomodes)`, and equivalent in this case to having the head pattern `p(Input,A,Output)` and stating that the property `list(int,A)` holds for the calls of the predicate.
- Any term preceded by a “ mode.” In this case, only one variable is admitted, it has to be the first argument of the mode, and it represents the argument position. I.e., it plays the role of the new variable mentioned above. Thus, no rewriting of the head pattern is performed in this case. For example, the head pattern `p(Input,+(Parameter,list(int)),Output)` is valid for mode `+/2` defined in `library(isomodes)`, and equivalent in this case to having the head pattern `p(Input,Parameter,Output)` and stating that the property `list(int,Parameter)` holds for the calls of the predicate.

Usage: `head_pattern(Pr)`

`Pr` is a head pattern.

complex_arg_property/1: REGTYPE

`complex_arg_property(Props)`

`Props` is a (possibly empty) complex argument property. Such properties can appear in two formats, which are defined by `property_conjunction/1` and `property_starterm/1` respectively. The two formats can be mixed provided they are not in the same field of an assertion. I.e., the following is a valid assertion:

`:- pred foo(X,Y) : nonvar * var => (ground(X),ground(Y)).`

Usage: `complex_arg_property(Props)`

`Props` is a (possibly empty) complex argument property

property_conjunction/1: REGTYPE

This type defines the first, unabridged format in which properties can be expressed in the bodies of assertions. It is essentially a conjunction of properties which refer to variables. The following is an example of a complex property in this format:

- `(integer(X),list(integer,Y))`: `X` has the property `integer/1` and `Y` has the property `list/2`, with second argument `integer`.

Usage: `property_conjunction(Props)`

`Props` is either a term or a *conjunction* of terms. The main functor and arity of each of those terms corresponds to the definition of a property. The first argument of each such term is a variable which appears as a head argument.

property_starterm/1: REGTYPE

This type defines a second, compact format in which properties can be expressed in the bodies of assertions. A `property_starterm/1` is a term whose main functor is `*/2` and, when it appears in an assertion, the number of terms joined by `*/2` is exactly the arity of the predicate it refers to. A similar series of properties as in `property_conjunction/1` appears, but the arity of each property is one less: the argument position to which they refer (first argument) is left out and determined by the position of the property in the `property_starterm/1`. The idea is that each element of the `*/2` term corresponds to a head argument position. Several properties can be assigned to each argument position by grouping them in curly brackets. The following is an example of a complex property in this format:

- `integer * list(integer)`: the first argument of the procedure (or function, or ...) has the property `integer/1` and the second one has the property `list/2`, with second argument `integer`.
- `{integer,var} * list(integer)`: the first argument of the procedure (or function, or ...) has the properties `integer/1` and `var/1` and the second one has the property `list/2`, with second argument `integer`.

Usage: `property_starterm(Props)`

`Props` is either a term or several terms separated by `*/2`. The main functor of each of those terms corresponds to that of the definition of a property, and the arity should be one less than in the definition of such property. All arguments of each such term are ground.

complex_goal_property/1: REGTYPE

`complex_goal_property(Props)`

`Props` is a (possibly empty) complex goal property. Such properties can be either a term or a *conjunction* of terms. The main functor and arity of each of those terms corresponds to the definition of a property. Such properties apply to all executions of all goals of the predicate which comply with the assertion in which the `Props` appear.

The arguments of the terms in `Props` are implicitely augmented with a first argument which corresponds to a goal of the predicate of the assertion in which the `Props` appear. For example, the assertion

```
:- comp var(A) + not_further_inst(A).
```

has property `not_further_inst/1` as goal property, and establishes that in all executions of `var(A)` it should hold that `not_further_inst(var(A),A)`.

Usage: `complex_goal_property(Props)`

`Props` is either a term or a *conjunction* of terms. The main functor and arity of each of those terms corresponds to the definition of a property. A first implicit argument in such terms identifies goals to which the properties apply.

nabody/1: PROPERTY

Usage: `nabody(ABody)`

`ABody` is a normalized assertion body.

dictionary/1: REGTYPE

Usage: `dictionary(D)`

`D` is a dictionary of variable names.

c_assrt_body/1: REGTYPE

This predicate defines the different types of syntax admissible in the bodies of `call/1`, `entry/1`, etc. assertions. The following are admissible:

```
Pr : CP [# CO]
```

where (fields between [...] are optional):

- `CP` is a (possibly empty) complex argument property (`complex_arg_property/1`) which applies to the *calls* to the predicate.
- `CO` is a comment string (`docstring/1`). This comment only applies if the (possibly empty) properties given for calls in the assertion hold. The usual formatting commands that are applicable in comment strings can be used (see `stringcommand/1`).

The format of the different parts of the assertion body are given by `n_assrt_body/5` and its auxiliary types.

Usage: `c_assrt_body(X)`

`X` is a call assertion body.

s_assrt_body/1: REGTYPE

This predicate defines the different types of syntax admissible in the bodies of `pred/1`, `func/1`, etc. assertions. The following are admissible:

```
Pr : CP => AP # CO
Pr : CP => AP
Pr => AP # CO
Pr => AP
```

where:

- `Pr` is a head pattern (`head_pattern/1`) which describes the predicate or property and possibly gives some implicit call/answer information.
- `CP` is a (possibly empty) complex argument property (`complex_arg_property/1`) which applies to the *calls* to the predicate.
- `AP` is a (possibly empty) complex argument property (`complex_arg_property/1`) which applies to the *answers* to the predicate (if the predicate succeeds). These only apply if the (possibly empty) properties given for calls in the assertion hold.
- `CO` is a comment string (`docstring/1`). This comment only applies if the (possibly empty) properties given for calls in the assertion hold. The usual formatting commands that are applicable in comment strings can be used (see `stringcommand/1`).

The format of the different parts of the assertion body are given by `n_assrt_body/5` and its auxiliary types.

Usage: `s_assrt_body(X)`

`X` is a predicate assertion body.

g_assrt_body/1: REGTYPE

This predicate defines the different types of syntax admissible in the bodies of `comp/1` assertions. The following are admissible:

```
Pr : CP + GP # CO
Pr : CP + GP
Pr + GP # CO
Pr + GP
```

where:

- `Pr` is a head pattern (`head_pattern/1`) which describes the predicate or property and possibly gives some implicit call/answer information.
- `CP` is a (possibly empty) complex argument property (`complex_arg_property/1`) which applies to the *calls* to the predicate.
- `GP` contains (possibly empty) complex goal property (`complex_goal_property/1`) which applies to the *whole execution* of a call to the predicate. These only apply if the (possibly empty) properties given for calls in the assertion hold.
- `CO` is a comment string (`docstring/1`). This comment only applies if the (possibly empty) properties given for calls in the assertion hold. The usual formatting commands that are applicable in comment strings can be used (see `stringcommand/1`).

The format of the different parts of the assertion body are given by `n_assrt_body/5` and its auxiliary types.

Usage: `g_assrt_body(X)`

`X` is a comp assertion body.

assrt_status/1: REGTYPE

The types of assertion status. They have the same meaning as the program-point assertions, and are as follows:

```
assrt_status(true).
assrt_status(false).
assrt_status(check).
assrt_status(checked).
```

```
assrt_status(trust).
```

Usage: `assrt_status(X)`

`X` is an acceptable status for an assertion.

assrt_type/1: REGTYPE

The admissible kinds of assertions:

```
assrt_type(pred).
assrt_type(prop).
assrt_type(decl).
assrt_type(func).
assrt_type(calls).
assrt_type(success).
assrt_type(comp).
assrt_type(entry).
assrt_type(exit).
assrt_type(test).
assrt_type(texec).
assrt_type(modedef).
```

Usage: `assrt_type(X)`

`X` is an admissible kind of assertion.

predfunctor/1: REGTYPE

Usage: `predfunctor(X)`

`X` is a type of assertion which defines a predicate.

propfunctor/1: REGTYPE

Usage: `propfunctor(X)`

`X` is a type of assertion which defines a *property*.

docstring/1: PROPERTY

Usage: `docstring(String)`

`String` is a text comment with admissible documentation commands. The usual formatting commands that are applicable in comment strings are defined by `stringcommand/1`. See the `lpdoc` manual for documentation on comments.

53.3 Documentation on imports

This module has the following direct dependencies:

– *Internal (engine) modules:*
  `term_basic`, `arithmetic`, `atomic_basic`, `basiccontrol`, `exceptions`, `term_compare`, `term_typing`, `debugger_support`, `basic_props`.
– *Packages:*
  `prelude`, `initial`, `condcomp`, `assertions`, `assertions/assertions_basic`, `regtypes`.

54 Declaring regular types

Author(s): Manuel Hermenegildo, Pedro López, Francisco Bueno.

This library package adds declarations and new operator definitions which provide simple syntactic sugar to write regular type definitions in source code. Regular types are just properties which have the additional characteristic of being regular types (`basic_props:regtype/1`), defined below.

For example, this library package allows writing:

```
:- regtype tree(X) # "X is a tree.".
```

instead of the more cumbersome:

```
:- prop tree(X) + regtype # "X is a tree.".
```

Regular types can be used as properties to describe predicates and play an essential role in program debugging (see the Ciao Prolog preprocessor (`ciaopp`) manual).

In this chapter we explain some general considerations worth taking into account when writing properties in general, not just regular types.

54.1 Defining properties

Given the classes of assertions in the Ciao assertion language, there are two fundamental classes of properties. Properties used in assertions which refer to execution states (i.e., `calls/1`, `success/1`, and the like) are called *properties of execution states*. Properties used in assertions related to computations (i.e., `comp/1`) are called *properties of computations*. Different considerations apply when writing a property of the former or of the latter kind.

Consider a definition of the predicate `string_concat/3` which concatenates two character strings (represented as lists of ASCII codes):

```
string_concat([],L,L).
string_concat([X|Xs],L,[X|NL]):- string_concat(Xs,L,NL).
```

Assume that we would like to state in an assertion that each argument "is a list of integers." However, we must decide which one of the following two possibilities we mean exactly: "the argument is *instantiated* to a list of integers" (let us call this property `instantiated_to_intlist/1`), or "if any part of the argument is instantiated, this instantiation must be compatible with the argument being a list of integers" (we will call this property `compatible_with_intlist/1`). For example, `instantiated_to_intlist/1` should be true for the terms `[]` and `[1,2]`, but should not for `X`, `[a,2]`, and `[X,2]`. In turn, `compatible_with_intlist/1` should be true for `[]`, `X`, `[1,2]`, and `[X,2]`, but should not be for `[X|1]`, `[a,2]`, and `1`. We refer to properties such as `instantiated_to_intlist/1` above as *instantiation properties* and to those such as `compatible_with_intlist/1` as *compatibility properties* (corresponding to the traditional notions of "instantiation types" and "compatibility types").

It turns out that both of these notions are quite useful in practice. In the example above, we probably would like to use `compatible_with_intlist/1` to state that on success of `string_concat/3` all three argument must be compatible with lists of integers in an assertion like:

```
:- success string_concat(A,B,C) => ( compatible_with_intlist(A),
                                     compatible_with_intlist(B),
                                     compatible_with_intlist(C) ).
```

With this assertion, no error will be flagged for a call to `string_concat/3` such as `string_concat([20],L,R)`, which on success produces the resulting atom `string_concat([20],L,[20|L])`, but a call `string_concat([],a,R)` would indeed flag an error.

On the other hand, and assuming that we are running on a Prolog system, we would probably like to use `instantiated_to_intlist/1` for `sumlist/2` as follows:

```
:- calls sumlist(L,N) : instantiated_to_intlist(L).

sumlist([],0).
sumlist([X|R],S) :- sumlist(R,PS), S is PS+X.
```

to describe the type of calls for which the program has been designed, i.e., those in which the first argument of `sumlist/2` is indeed a list of integers.

The property `instantiated_to_intlist/1` might be written as in the following (Prolog) definition:

```
:- prop instantiated_to_intlist/1.

instantiated_to_intlist(X) :-
       nonvar(X), instantiated_to_intlist_aux(X).

instantiated_to_intlist_aux([]).
instantiated_to_intlist_aux([X|T]) :-
       integer(X), instantiated_to_intlist(T).
```

(Recall that the Prolog builtin `integer/1` itself implements an instantiation check, failing if called with a variable as the argument.)

The property `compatible_with_intlist/1` might in turn be written as follows (also in Prolog):

```
:- prop compatible_with_intlist/1.

compatible_with_intlist(X) :- var(X).
compatible_with_intlist(X) :-
       nonvar(X), compatible_with_intlist_aux(X).

compatible_with_intlist_aux([]).
compatible_with_intlist_aux([X|T]) :-
       int_compat(X), compatible_with_intlist(T).

int_compat(X) :- var(X).
int_compat(X) :- nonvar(X), integer(X).
```

Note that these predicates meet the criteria for being properties and thus the `prop/1` declaration is correct.

Ensuring that a property meets the criteria for "not affecting the computation" can sometimes make its coding somewhat tedious. In some ways, one would like to be able to write simply:

```
intlist([]).
intlist([X|R]) :- int(X), intlist(R).
```

(Incidentally, note that the above definition, provided that it suits the requirements for being a property and that `int/1` is a regular type, meets the criteria for being a regular type. Thus, it could be declared `:- regtype intlist/1`.)

But note that (independently of the definition of `int/1`) the definition above is not the correct instantiation check, since it would succeed for a call such as `intlist(X)`. In fact, it is not strictly correct as a compatibility property either, because, while it would fail or succeed as expected, it would perform instantiations (e.g., if called with `intlist(X)` it would bind `X` to `[]`). In practice, it is convenient to provide some run-time support to aid in this task.

The run-time support of the Ciao system (see Chapter 64 [Run-time checking of assertions], page 407) ensures that the execution of properties is performed in such a way that properties written as above can be used directly as instantiation checks. Thus, writing:

```
:- calls sumlist(L,N) : intlist(L).
```

has the desired effect. Also, the same properties can often be used as compatibility checks by writing them in the assertions as `compat(Property)` (`basic_props:compat/1`). Thus, writing:

```
:- success string_concat(A,B,C) => ( compat(intlist(A)),
                                      compat(intlist(B)),
                                      compat(intlist(C)) ).
```

also has the desired effect.

As a general rule, the properties that can be used directly for checking for compatibility should be *downwards closed*, i.e., once they hold they will keep on holding in every state accessible in forwards execution. There are certain predicates which are inherently *instantiation* checks and should not be used as *compatibility* properties nor appear in the definition of a property that is to be used with `compat`. Examples of such predicates (for Prolog) are `==`, `ground`, `nonvar`, `integer`, `atom`, `>`, etc. as they require a certain instantiation degree of their arguments in order to succeed.

In contrast with properties of execution states, *properties of computations* refer to the entire execution of the call(s) that the assertion relates to. One such property is, for example, `not_fails/1` (note that although it has been used as in `:- comp append(Xs,Ys,Zs) + not_fails`, it is in fact read as `not_fails(append(Xs,Ys,Zs))`; see `assertions_props:complex_goal_property/1`). For this property, which should be interpreted as "execution of the predicate either succeeds at least once or loops," we can use the following predicate `not_fails/1` for run-time checking:

```
not_fails(Goal):-
      if( call(Goal),
          true,             %% then
          warning(Goal) ). %% else
```

where the `warning/1` (library) predicate simply prints a warning message.

In this simple case, implementation of the predicate is not very difficult using the (non-standard) `if/3` builtin predicate present in many Prolog systems.

However, it is not so easy to code predicates which check other properties of the computation and we may in general need to program a meta-interpreter for this purpose.

54.2 Usage and interface

- **Library usage:**
  `:- use_package(regtypes).`
  or
  `:- module(...,...,[regtypes]).`
- **New operators defined:**
  `regtype/1` [1150,fx], `regtype/2` [1150,xfx].
- **New declarations defined:**
  `regtype/1`, `regtype/2`.
- **Implicit imports:**
  - *System library modules:*
    `assertions_props`.
  - *Internal (engine) modules:*
    `term_basic`, `arithmetic`, `atomic_basic`, `basiccontrol`, `exceptions`, `term_compare`, `term_typing`, `debugger_support`, `basic_props`.
  - *Packages:*
    `prelude`, `initial`, `condcomp`, `assertions`, `assertions/assertions_basic`.

54.3 Documentation on new declarations

regtype/1: DECLARATION

This assertion is similar to a prop assertion but it flags that the property being documented is also a " regular type." Regular types are properties whose definitions are *regular programs* (see below). This allows for example checking whether it is in the class of types supported by the regular type checking and inference modules.

A regular program is defined by a set of clauses, each of the form:

```
p(v_1, ..., v_n, x)  :- body_1, ..., body_k.
```

where:

1. `x` is a term whose variables (which are called *term variables*) are unique, i.e., it is not allowed to introduce equality constraints between the variables of `x`.
   For example, `p(f(X, Y)) :- ...` is valid, but `p(f(X, X)) :- ...` is not.
2. in all clauses defining `p/n+1` the terms `x` do not unify except maybe for one single clause in which `x` is a variable.
3. `n` >= 0 and `p/n` is a *parametric type functor* (whereas the predicate defined by the clauses is `p/n+1`).
4. `v_1`, ..., `v_n` are unique variables, which are called *parametric variables*.
5. Each `body_i` is of the form:
   1. `t(z)` where `z` is one of the *term variables* and `t` is a *regular type expression*;
   2. `q(t_1, ..., t_m, y)` where `m` >= 0, `q/m` is a *parametric type functor*, not in the set of functors `=/2`, `^/2`, `./3`.
      `t_1, ..., t_m` are *regular type expressions*, and `y` is a *term variable*.
6. Each term variable occurs at most once in the clause's body (and should be as the last argument of a literal).

A *regular type expression* is either a parametric variable or a parametric type functor applied to some of the parametric variables.

A parametric type functor is a regular type, defined by a regular program, or a basic type. Basic types are defined in Chapter 55 [Basic data types and properties], page 357.

The set of regular types is thus a well defined subset of the set of properties. Note that types can be used to describe characteristics of arguments in assertions and they can also be executed (called) as any other predicates.

Usage: :- `regtype(AssertionBody)`.

– *The following properties should hold at call time:*

  `AssertionBody` is an assertion body. (`assrt_body/1`)

regtype/2: DECLARATION

This assertion is similar to a `regtype/1` assertion but it is explicitely qualified. Non-qualified `regtype/1` assertions are assumed the qualifier `check`. Note that checking regular type definitions should be done with the `ciaopp` preprocessor.

Usage: :- `regtype(AssertionStatus,AssertionBody)`.

– *The following properties should hold at call time:*

  `AssertionStatus` is an acceptable status for an assertion. (`assrt_status/1`)

  `AssertionBody` is an assertion body. (`assrt_body/1`)

55 Basic data types and properties

Author(s): Daniel Cabeza, Manuel Hermenegildo.

This library contains the set of basic properties used by the builtin predicates, and which constitute the basic data types and properties of the language. They are intented to be used as properties in assertions (for both runtime and compile time checking).

Note that most of those properties can be used directly from goals as type testing builtins, and they should be correct when terms are sufficiently instantiated. Calling with uninstantiated terms have undefined behaviour (failure or non termination, depending on the type). Please use the run-time checking facilities (`inst/2` and `compat/2`) to ensure well-defined behaviour. For low-level instantiation checks we encourage the use `term_typing` builtins.

55.1 Usage and interface

- **Library usage:**

  This module is automatically imported by the `assertions` package. It can be imported explicitly with the `:- use_module(engine(basic_props))` directive.
- **Exports:**
  - *Predicates:*

    `check/1`, `trust/1`, `true/1`, `false/1`.
  - *Properties:*

    `callable/1`, `member/2`, `compat/2`, `inst/2`, `iso/1`, `deprecated/1`, `srcloc/4`, `example/1`, `not_further_inst/2`, `sideff/2`, `regtype/1`, `native/1`, `native/2`, `rtcheck/1`, `rtcheck/2`, `no_rtcheck/1`, `eval/1`, `equiv/2`, `bind_ins/1`, `error_free/1`, `memo/1`, `filter/2`, `pe_type/1`.
  - *Regular Types:*

    `term/1`, `int/1`, `nnegint/1`, `flt/1`, `num/1`, `atm/1`, `struct/1`, `gnd/1`, `gndstr/1`, `constant/1`, `cgoal/1`, `internal_module_id/1`, `operator_specifier/1`, `list/1`, `list/2`, `nlist/2`, `sequence/2`, `sequence_or_list/2`, `character_code/1`, `string/1`, `bytelist/1`, `predname/1`, `atm_or_atm_list/1`, `flag_values/1`.

55.2 Documentation on exports

term/1: REGTYPE

The most general type (includes all possible terms).

Usage: `term(X)`

`X` is any term.

- *The following properties hold globally:*

  This predicate is understood natively by CiaoPP. (`native/1`)

Other properties:

`term(X)`

- *The following properties hold globally:*

  `term(X)` is side-effect `free`. (`sideff/2`)

`term(X)`

- *The following properties hold globally:*
  `term(X)` is evaluable at compile-time. (`eval/1`)

`term(X)`
- *The following properties hold globally:*
  `term(X)` is equivalent to `true`. (`equiv/2`)

int/1: REGTYPE

The type of integers. The range of integers is `[-2^2147483616, 2^2147483616)`. Thus for all practical purposes, the range of integers can be considered infinite.

Usage: `int(T)`

`T` is an integer.
- *The following properties hold globally:*
  This predicate is understood natively by CiaoPP. (`native/1`)

Other properties:

`int(T)`
- *The following properties hold globally:*
  `int(T)` is side-effect `free`. (`sideff/2`)

`int(T)`
- *If the following properties hold at call time:*
  `T` is currently a term which is not a free variable. (`nonvar/1`)
  then the following properties hold globally:
  `int(T)` is evaluable at compile-time. (`eval/1`)
  All calls of the form `int(T)` are deterministic. (`is_det/1`)

`int(T)`
- *The following properties hold upon exit:*
  `T` is an integer. (`int/1`)
- *The following properties hold globally:*
  Indicates the type of test that a predicate performs. Required by the nonfailure analyisis. (`test_type/2`)

nnegint/1: REGTYPE

The type of non-negative integers, i.e., natural numbers.

Usage: `nnegint(T)`

`T` is a non-negative integer.
- *The following properties hold globally:*
  This predicate is understood natively by CiaoPP. (`native/1`)

Other properties:

`nnegint(T)`
- *The following properties hold globally:*
  `nnegint(T)` is side-effect `free`. (`sideff/2`)

`nnegint(T)`

- *If the following properties hold at call time:*
  `T` is currently a term which is not a free variable. (`nonvar/1`)
  then the following properties hold globally:
  `nnegint(T)` is evaluable at compile-time. (`eval/1`)

`nnegint(T)`
- *The following properties hold upon exit:*
  `T` is a non-negative integer. (`nnegint/1`)
- *The following properties hold globally:*
  Indicates the type of test that a predicate performs. Required by the nonfailure analyisis. (`test_type/2`)

flt/1: REGTYPE

The type of floating-point numbers. The range of floats is the one provided by the C `double` type, typically `[4.9e-324, 1.8e+308]` (plus or minus). There are also three special values: Infinity, either positive or negative, represented as `1.0e1000` and `-1.0e1000`; and Not-a-number, which arises as the result of indeterminate operations, represented as `0.Nan`

Usage: `flt(T)`

`T` is a float.
- *The following properties hold globally:*
  This predicate is understood natively by CiaoPP. (`native/1`)

Other properties:

`flt(T)`
- *The following properties hold globally:*
  `flt(T)` is side-effect `free`. (`sideff/2`)

`flt(T)`
- *If the following properties hold at call time:*
  `T` is currently a term which is not a free variable. (`nonvar/1`)
  then the following properties hold globally:
  `flt(T)` is evaluable at compile-time. (`eval/1`)
  All calls of the form `flt(T)` are deterministic. (`is_det/1`)

`flt(T)`
- *The following properties hold upon exit:*
  `T` is a float. (`flt/1`)
- *The following properties hold globally:*
  Indicates the type of test that a predicate performs. Required by the nonfailure analyisis. (`test_type/2`)

num/1: REGTYPE

The type of numbers, that is, integer or floating-point.

Usage: `num(T)`

`T` is a number.

- *The following properties hold globally:*
  This predicate is understood natively by CiaoPP. (`native/1`)

Other properties:

`num(T)`

- *The following properties hold globally:*
  `num(T)` is side-effect `free`. (`sideff/2`)
  `num(T)` is binding insensitive. (`bind_ins/1`)

`num(T)`

- *If the following properties hold at call time:*
  `T` is currently a term which is not a free variable. (`nonvar/1`)
  then the following properties hold globally:
  `num(T)` is evaluable at compile-time. (`eval/1`)
  All calls of the form `num(T)` are deterministic. (`is_det/1`)

`num(T)`

- *The following properties hold upon exit:*
  `T` is a number. (`num/1`)
- *The following properties hold globally:*
  Indicates the type of test that a predicate performs. Required by the nonfailure analyisis. (`test_type/2`)

atm/1: REGTYPE

The type of atoms, or non-numeric constants. The size of atoms is unbound.

Usage: `atm(T)`

`T` is an atom.

- *The following properties hold globally:*
  This predicate is understood natively by CiaoPP. (`native/1`)

Other properties:

`atm(T)`

- *The following properties hold globally:*
  `atm(T)` is side-effect `free`. (`sideff/2`)

`atm(T)`

- *If the following properties hold at call time:*
  `T` is currently a term which is not a free variable. (`nonvar/1`)
  then the following properties hold globally:
  `atm(T)` is evaluable at compile-time. (`eval/1`)
  All calls of the form `atm(T)` are deterministic. (`is_det/1`)

`atm(T)`

- *The following properties hold upon exit:*
  `T` is an atom. (`atm/1`)
- *The following properties hold globally:*
  Indicates the type of test that a predicate performs. Required by the nonfailure analyisis. (`test_type/2`)

struct/1: REGTYPE

The type of compound terms, or terms with non-zeroary functors. By now there is a limit of 255 arguments.

Usage: `struct(T)`

`T` is a compound term.

- *The following properties hold globally:*
  This predicate is understood natively by CiaoPP. (`native/1`)

Other properties:

`struct(T)`

- *The following properties hold globally:*
  `struct(T)` is side-effect `free`. (`sideff/2`)

`struct(T)`

- *If the following properties hold at call time:*
  `T` is currently a term which is not a free variable. (`nonvar/1`)
  then the following properties hold globally:
  `struct(T)` is evaluable at compile-time. (`eval/1`)

`struct(T)`

- *The following properties hold upon exit:*
  `T` is a compound term. (`struct/1`)

gnd/1: REGTYPE

The type of all terms without variables.

Usage: `gnd(T)`

`T` is ground.

- *The following properties hold globally:*
  This predicate is understood natively by CiaoPP. (`native/1`)

Other properties:

`gnd(T)`

- *The following properties hold globally:*
  `gnd(T)` is side-effect `free`. (`sideff/2`)

`gnd(T)`

- *If the following properties hold at call time:*
  `T` is currently ground (it contains no variables). (`ground/1`)
  then the following properties hold globally:
  `gnd(T)` is evaluable at compile-time. (`eval/1`)
  All calls of the form `gnd(T)` are deterministic. (`is_det/1`)

`gnd(T)`

- *The following properties hold upon exit:*
  `T` is ground. (`gnd/1`)
- *The following properties hold globally:*
  Indicates the type of test that a predicate performs. Required by the nonfailure analyisis. (`test_type/2`)

gndstr/1: REGTYPE

Usage: `gndstr(T)`

`T` is a ground compound term.

- *The following properties hold globally:*
  This predicate is understood natively by CiaoPP. (`native/1`)

Other properties:

`gndstr(T)`
- *The following properties hold globally:*
  `gndstr(T)` is side-effect `free`. (`sideff/2`)

`gndstr(T)`
- *If the following properties hold at call time:*
  `T` is currently ground (it contains no variables). (`ground/1`)
  then the following properties hold globally:
  `gndstr(T)` is evaluable at compile-time. (`eval/1`)
  All calls of the form `gndstr(T)` are deterministic. (`is_det/1`)

`gndstr(T)`
- *The following properties hold upon exit:*
  `T` is a ground compound term. (`gndstr/1`)

constant/1: REGTYPE

Usage: `constant(T)`

`T` is an atomic term (an atom or a number).

Other properties:

`constant(T)`
- *The following properties hold globally:*
  `constant(T)` is side-effect `free`. (`sideff/2`)

`constant(T)`
- *If the following properties hold at call time:*
  `T` is currently a term which is not a free variable. (`nonvar/1`)
  then the following properties hold globally:
  `constant(T)` is evaluable at compile-time. (`eval/1`)
  All calls of the form `constant(T)` are deterministic. (`is_det/1`)

`constant(T)`
- *The following properties hold upon exit:*
  `T` is an atomic term (an atom or a number). (`constant/1`)

cgoal/1: REGTYPE

Usage: `cgoal(T)`

`T` is a term which represents a goal, i.e., an atom or a structure.

Other properties:

`cgoal(T)`

- *The following properties hold globally:*
  `cgoal(T)` is side-effect `free`. (`sideff/2`)

`cgoal(T)`
- *If the following properties hold at call time:*
  `T` is currently a term which is not a free variable. (`nonvar/1`)
  then the following properties hold globally:
  `cgoal(T)` is evaluable at compile-time. (`eval/1`)
  All calls of the form `cgoal(T)` are deterministic. (`is_det/1`)

`cgoal(T)`
- *The following properties hold upon exit:*
  `T` is currently a term which is not a free variable. (`nonvar/1`)

callable/1: PROPERTY

Usage: `callable(T)`

`T` is instantiated with an atom or a structure (also known as callable).

Other properties:

`callable(T)`
- *The following properties hold globally:*
  `callable(T)` is side-effect `free`. (`sideff/2`)

`callable(T)`
- *If the following properties hold at call time:*
  `T` is currently a term which is not a free variable. (`nonvar/1`)
  then the following properties hold globally:
  `callable(T)` is evaluable at compile-time. (`eval/1`)
  All calls of the form `callable(T)` are deterministic. (`is_det/1`)

`callable(T)`
- *The following properties hold upon exit:*
  `T` is currently a term which is not a free variable. (`nonvar/1`)

internal_module_id/1: REGTYPE

For a user file it is a term user/1 with an argument different for each user file, for other modules is just the name of the module (as an atom).

Usage: `internal_module_id(M)`

`M` is an internal module identifier

operator_specifier/1: REGTYPE

The type and associativity of an operator is described by the following mnemonic atoms:

`xfx` Infix, non-associative: it is a requirement that both of the two subexpressions which are the arguments of the operator must be of *lower* precedence than the operator itself.

`xfy` Infix, right-associative: only the first (left-hand) subexpression must be of lower precedence; the right-hand subexpression can be of the *same* precedence as the main operator.

`yfx` Infix, left-associative: same as above, but the other way around.

`fx` Prefix, non-associative: the subexpression must be of *lower* precedence than the operator.

`fy` Prefix, associative: the subexpression can be of the *same* precedence as the operator.

`xf` Postfix, non-associative: the subexpression must be of *lower* precedence than the operator.

`yf` Postfix, associative: the subexpression can be of the *same* precedence as the operator.

Usage: `operator_specifier(X)`

`X` specifies the type and associativity of an operator.

Other properties:

`operator_specifier(X)`

− *The following properties hold globally:*

`operator_specifier(X)` is side-effect `free`. (`sideff/2`)

`operator_specifier(X)`

− *If the following properties hold at call time:*

`X` is currently a term which is not a free variable. (`nonvar/1`)

then the following properties hold globally:

`operator_specifier(X)` is evaluable at compile-time. (`eval/1`)

All calls of the form `operator_specifier(X)` are deterministic. (`is_det/1`)

Goal `operator_specifier(X)` produces `7` solutions. (`relations/2`)

`operator_specifier(T)`

− *The following properties hold upon exit:*

`T` specifies the type and associativity of an operator. (`operator_specifier/1`)

list/1: REGTYPE

A list is formed with successive applications of the functor `'.'/2`, and its end is the atom `[]`. Defined as

```
list([]).
list([_1|L]) :-
    list(L).
```

Usage: `list(L)`

`L` is a list.

Other properties:

`list(L)`

− *The following properties hold globally:*

`list(L)` is side-effect `free`. (`sideff/2`)

`list(L)`

- *If the following properties hold at call time:*
  L is currently ground (it contains no variables). (ground/1)
  then the following properties hold globally:
  list(L) is evaluable at compile-time. (eval/1)
  All calls of the form list(L) are deterministic. (is_det/1)

list(T)

- *The following properties hold upon exit:*
  T is a list. (list/1)

list/2: REGTYPE

list(T,L)

L is a list, and for all its elements, T holds.

Usage: list(T,L)

L is a list of Ts.

Meta-predicate with arguments: list(pred(1),?).

Other properties:

list(T,L)

- *The following properties hold globally:*
  list(T,L) is side-effect free. (sideff/2)

list(T,L)

- *If the following properties hold at call time:*
  T is currently ground (it contains no variables). (ground/1)
  L is currently ground (it contains no variables). (ground/1)
  then the following properties hold globally:
  list(T,L) is evaluable at compile-time. (eval/1)

list(T,X)

- *The following properties hold upon exit:*
  X is a list. (list/1)

nlist/2: REGTYPE

Usage: nlist(T,L)

L is T or a nested list of Ts. Note that if T is term, this type is equivalent to term, this fact explains why we do not have an nlist/1 type.

Meta-predicate with arguments: nlist(pred(1),?).

Other properties:

nlist(T,L)

- *The following properties hold globally:*
  nlist(T,L) is side-effect free. (sideff/2)

nlist(T,L)

- *If the following properties hold at call time:*
  `T` is currently ground (it contains no variables). (`ground/1`)
  `L` is currently ground (it contains no variables). (`ground/1`)
  then the following properties hold globally:
  `nlist(T,L)` is evaluable at compile-time. (`eval/1`)

`nlist(T,X)`
- *The following properties hold upon exit:*
  `X` is any term. (`term/1`)

member/2: PROPERTY

Usage: `member(X,L)`

`X` is an element of `L`.

Other properties:

`member(X,L)`
- *The following properties hold globally:*
  `member(X,L)` is side-effect `free`. (`sideff/2`)
  `member(X,L)` is binding insensitive. (`bind_ins/1`)

`member(X,L)`
- *If the following properties hold at call time:*
  `L` is a list. (`list/1`)
  then the following properties hold globally:
  `member(X,L)` is evaluable at compile-time. (`eval/1`)

`member(_X,L)`
- *The following properties hold upon exit:*
  `L` is a list. (`list/1`)

`member(X,L)`
- *If the following properties hold at call time:*
  `L` is currently ground (it contains no variables). (`ground/1`)
  then the following properties hold upon exit:
  `X` is currently ground (it contains no variables). (`ground/1`)

sequence/2: REGTYPE

A sequence is formed with zero, one, or more occurrences of the operator `','/2`. For example, `a, b, c` is a sequence of three atoms, `a` is a sequence of one atom.

Usage: `sequence(T,S)`

`S` is a sequence of `T`s.

Meta-predicate with arguments: `sequence(pred(1),?)`.

Other properties:

`sequence(T,S)`
- *The following properties hold globally:*
  `sequence(T,S)` is side-effect `free`. (`sideff/2`)

`sequence(T,S)`
- *If the following properties hold at call time:*
  `T` is currently ground (it contains no variables). (`ground/1`)
  `S` is currently ground (it contains no variables). (`ground/1`)
  then the following properties hold globally:
  `sequence(T,S)` is evaluable at compile-time. (`eval/1`)

`sequence(T,E)`
- *The following properties hold upon exit:*
  `T` is currently ground (it contains no variables). (`ground/1`)
  `E` is currently a term which is not a free variable. (`nonvar/1`)

sequence_or_list/2: REGTYPE

Usage: `sequence_or_list(T,S)`

`S` is a sequence or list of `T`s.

Meta-predicate with arguments: `sequence_or_list(pred(1),?)`.

Other properties:

`sequence_or_list(T,S)`
- *The following properties hold globally:*
  `sequence_or_list(T,S)` is side-effect `free`. (`sideff/2`)

`sequence_or_list(T,S)`
- *If the following properties hold at call time:*
  `T` is currently ground (it contains no variables). (`ground/1`)
  `S` is currently ground (it contains no variables). (`ground/1`)
  then the following properties hold globally:
  `sequence_or_list(T,S)` is evaluable at compile-time. (`eval/1`)

`sequence_or_list(T,E)`
- *The following properties hold upon exit:*
  `T` is currently ground (it contains no variables). (`ground/1`)
  `E` is currently a term which is not a free variable. (`nonvar/1`)

character_code/1: REGTYPE

Usage: `character_code(T)`

`T` is an integer which is a character code.

Other properties:

`character_code(T)`
- *The following properties hold globally:*
  `character_code(T)` is side-effect `free`. (`sideff/2`)

`character_code(T)`
- *If the following properties hold at call time:*
  `T` is currently a term which is not a free variable. (`nonvar/1`)
  then the following properties hold globally:
  `character_code(T)` is evaluable at compile-time. (`eval/1`)

`character_code(I)`

– *The following properties hold upon exit:*

`I` is an integer which is a character code. (`character_code/1`)

string/1: REGTYPE

A string is a list of character codes. The usual syntax for strings `"string"` is allowed, which is equivalent to `[0's,0't,0'r,0'i,0'n,0'g]` or `[115,116,114,105,110,103]`. There is also a special Ciao syntax when the list is not complete: `"st"||R` is equivalent to `[0's,0't|R]`.

Usage: `string(T)`

`T` is a string (a list of character codes).

Other properties:

`string(T)`

– *The following properties hold globally:*

`string(T)` is side-effect `free`. (`sideff/2`)

`string(T)`

– *If the following properties hold at call time:*

`T` is currently ground (it contains no variables). (`ground/1`)

then the following properties hold globally:

`string(T)` is evaluable at compile-time. (`eval/1`)

`string(T)`

– *The following properties hold upon exit:*

`T` is a string (a list of character codes). (`string/1`)

bytelist/1: REGTYPE

Usage: `bytelist(T)`

`T` is list of bytes.

Other properties:

`bytelist(T)`

– *The following properties hold globally:*

`bytelist(T)` is side-effect `free`. (`sideff/2`)

`bytelist(T)`

– *If the following properties hold at call time:*

`T` is currently ground (it contains no variables). (`ground/1`)

then the following properties hold globally:

`bytelist(T)` is evaluable at compile-time. (`eval/1`)

`bytelist(T)`

– *The following properties hold upon exit:*

`T` is list of bytes. (`bytelist/1`)

predname/1: REGTYPE

`predname(P)`

`P` is a Name/Arity structure denoting a predicate name:

```
predname(P/A) :-
    atm(P),
    nnegint(A).
```

Usage: `predname(P)`

`P` is a predicate name.

Other properties:

`predname(P)`

- *The following properties hold globally:*
  `predname(P)` is side-effect `free`. (`sideff/2`)

`predname(P)`

- *If the following properties hold at call time:*
  `P` is currently ground (it contains no variables). (`ground/1`)
  then the following properties hold globally:
  `predname(P)` is evaluable at compile-time. (`eval/1`)

`predname(P)`

- *The following properties hold upon exit:*
  `P` is a predicate name. (`predname/1`)

atm_or_atm_list/1: REGTYPE

Usage: `atm_or_atm_list(T)`

`T` is an atom or a list of atoms.

Other properties:

`atm_or_atm_list(T)`

- *The following properties hold globally:*
  `atm_or_atm_list(T)` is side-effect `free`. (`sideff/2`)

`atm_or_atm_list(T)`

- *If the following properties hold at call time:*
  `T` is currently ground (it contains no variables). (`ground/1`)
  then the following properties hold globally:
  `atm_or_atm_list(T)` is evaluable at compile-time. (`eval/1`)

`atm_or_atm_list(T)`

- *The following properties hold upon exit:*
  `T` is an atom or a list of atoms. (`atm_or_atm_list/1`)

compat/2: PROPERTY

This property captures the notion of type or property compatibility. The instantiation or constraint state of the term is compatible with the given property, in the sense that assuming that imposing that property on the term does not render the store inconsistent.

For example, terms `X` (i.e., a free variable), `[Y|Z]`, and `[Y,Z]` are all compatible with the regular type `list/1`, whereas the terms `f(a)` and `[1|2]` are not.

Usage: `compat(Term,Prop)`

`Term` is *compatible* with `Prop`.

Meta-predicate with arguments: `compat(?,pred(1))`.

Other properties:

`compat(Term,Prop)`

- *If the following properties hold at call time:*
  `Term` is currently ground (it contains no variables). (`ground/1`)
  `Prop` is currently ground (it contains no variables). (`ground/1`)
  then the following properties hold globally:
  `compat(Term,Prop)` is evaluable at compile-time. (`eval/1`)

inst/2: PROPERTY

Usage: `inst(Term,Prop)`

`Term` is instantiated enough to satisfy `Prop`.

Meta-predicate with arguments: `inst(?,pred(1))`.

Other properties:

`inst(Term,Prop)`

- *The following properties hold globally:*
  `inst(Term,Prop)` is side-effect `free`. (`sideff/2`)

`inst(Term,Prop)`

- *If the following properties hold at call time:*
  `Term` is currently ground (it contains no variables). (`ground/1`)
  `Prop` is currently ground (it contains no variables). (`ground/1`)
  then the following properties hold globally:
  `inst(Term,Prop)` is evaluable at compile-time. (`eval/1`)

iso/1: PROPERTY

Specifies that the predicate and usage marked with this global property complies with the ISO-Prolog standard.

Usage: `iso(G)`

Complies with the ISO-Prolog standard.

Meta-predicate with arguments: `iso(goal)`.

Other properties:

`iso(G)`

- *The following properties hold globally:*
  `iso(G)` is side-effect `free`. (`sideff/2`)

deprecated/1: PROPERTY

Specifies that the predicate marked with this global property has been deprecated, i.e., its use is not recommended any more since it will be deleted at a future date. Typically this is done because its functionality has been superseded by another predicate.

Usage: `deprecated(G)`

DEPRECATED.

Meta-predicate with arguments: `deprecated(goal)`.

Other properties:

`deprecated(G)`

− *The following properties hold globally:*

`deprecated(G)` is side-effect `free`. (`sideff/2`)

srcloc/4: PROPERTY

This annotation (pseudo-property) is used to mark that the element (normally assertion) in which it appears was originally between lines `LB`-`LE` of source file `Src`.

Usage: `srcloc(Goal,Src,LB,LE)`

− *The following properties should hold globally:*

`srcloc(Goal,Src,LB,LE)` is side-effect `free`. (`sideff/2`)

Meta-predicate with arguments: `srcloc(?,?,?,?)`.

example/1: PROPERTY

This pseudo-property can be used as a qualifier for test assertions. Specifies that the test assertion in which it appears is actually an example and should be included as such in the manual by the auto-documeter.

Usage: `example(G)`

EXAMPLE.

Meta-predicate with arguments: `example(goal)`.

Other properties:

`example(G)`

− *The following properties hold globally:*

`example(G)` is side-effect `free`. (`sideff/2`)

not_further_inst/2: PROPERTY

Usage: `not_further_inst(G,V)`

`V` is not further instantiated.

Meta-predicate with arguments: `not_further_inst(goal,?)`.

Other properties:

`not_further_inst(G,V)`

− *The following properties hold globally:*

`not_further_inst(G,V)` is side-effect `free`. (`sideff/2`)

`not_further_inst(G,V)` is not checked during run-time checking. (`no_rtcheck/1`)

sideff/2: PROPERTY

`sideff(G,X)`

Declares that `G` is side-effect free (if its execution has no observable result other than its success, its failure, or its abortion), soft (if its execution may have other observable results which, however, do not affect subsequent execution, e.g., input/output), or hard (e.g., assert/retract).

Usage: `sideff(G,X)`

`G` is side-effect `X`.

- *If the following properties should hold at call time:*

  `G` is a term which represents a goal, i.e., an atom or a structure. (`cgoal/1`)

  `X` is an element of `[free,soft,hard]`. (`member/2`)

Meta-predicate with arguments: `sideff(goal,?)`.

Other properties:

`sideff(G,X)`

- *The following properties hold globally:*

  This predicate is understood natively by CiaoPP. (`native/1`)

  `sideff(G,X)` is side-effect `free`. (`sideff/2`)

  `sideff(G,X)` is not checked during run-time checking. (`no_rtcheck/1`)

regtype/1: PROPERTY

Usage: `regtype(G)`

Defines a regular type.

Meta-predicate with arguments: `regtype(goal)`.

Other properties:

`regtype(G)`

- *The following properties hold globally:*

  `regtype(G)` is side-effect `free`. (`sideff/2`)

native/1: PROPERTY

Usage: `native(Pred)`

This predicate is understood natively by CiaoPP.

Meta-predicate with arguments: `native(goal)`.

Other properties:

`native(P)`

- *The following properties hold globally:*

  `native(P)` is side-effect `free`. (`sideff/2`)

native/2: PROPERTY

Usage: `native(Pred,Key)`

This predicate is understood natively by CiaoPP as `Key`.

Meta-predicate with arguments: `native(goal,?)`.

Other properties:

`native(P,K)`

– *The following properties hold globally:*

`native(P,K)` is side-effect `free`. (`sideff/2`)

rtcheck/1: PROPERTY

Usage: `rtcheck(G)`

Equivalent to rtcheck(G, complete).

– *If the following properties should hold at call time:*

`G` is a term which represents a goal, i.e., an atom or a structure. (`cgoal/1`)

Meta-predicate with arguments: `rtcheck(goal)`.

Other properties:

`rtcheck(G)`

– *The following properties hold globally:*

`rtcheck(G)` is side-effect `free`. (`sideff/2`)

rtcheck/2: PROPERTY

Usage: `rtcheck(G,Status)`

The runtime check of this property is `Status`.

– *If the following properties should hold at call time:*

`G` is a term which represents a goal, i.e., an atom or a structure. (`cgoal/1`)

`Status` is the status of the runtime-check implementation for a given property. Valid values are:

- complete: The implementation of the run-time check for this property is complete. i.e., an error is reported always if the property is violated. Default.
- incomplete: The current run-time check is incomplete, i.e., it is possible that no error is reported even if the property is violated.
- unimplemented: No run-time checker has been implemented (yet) for the property.
- unknown: It has not been determined yet whether the current implementation of the run-time checker is complete or not.
- impossible: The property cannot be checked at run time (for theoretical or practical reasons).

(`rtc_status/1`)

Meta-predicate with arguments: `rtcheck(goal,?)`.

Other properties:

`rtcheck(G,Status)`

– *The following properties hold globally:*

`rtcheck(G,Status)` is side-effect `free`. (`sideff/2`)

no_rtcheck/1: PROPERTY

This comp pseudo-property is used to declare that the assertion in which it appears should not be checked at run-time. Equivalent to `rtcheck(G, S)` with `S` unimplemented, impossible, etc.

Usage: `no_rtcheck(G)`

`G` is not checked during run-time checking.

- *If the following properties should hold at call time:*
  `G` is a term which represents a goal, i.e., an atom or a structure. (`cgoal/1`)

Meta-predicate with arguments: `no_rtcheck(goal)`.

Other properties:

`no_rtcheck(G)`

- *The following properties hold globally:*
  `no_rtcheck(G)` is side-effect `free`. (`sideff/2`)

eval/1: PROPERTY

Usage: `eval(Goal)`

`Goal` is evaluable at compile-time.

Meta-predicate with arguments: `eval(goal)`.

equiv/2: PROPERTY

Usage: `equiv(Goal1,Goal2)`

`Goal1` is equivalent to `Goal2`.

Meta-predicate with arguments: `equiv(goal,goal)`.

bind_ins/1: PROPERTY

Usage: `bind_ins(Goal)`

`Goal` is binding insensitive.

Meta-predicate with arguments: `bind_ins(goal)`.

error_free/1: PROPERTY

Usage: `error_free(Goal)`

`Goal` is error free.

Meta-predicate with arguments: `error_free(goal)`.

memo/1: PROPERTY

Usage: `memo(Goal)`

`Goal` should be memoized (not unfolded).

Meta-predicate with arguments: `memo(goal)`.

filter/2: PROPERTY

Usage: `filter(Vars,Goal)`

`Vars` should be filtered during global control).

flag_values/1: REGTYPE

Usage: `flag_values(X)`

Define the valid flag values

pe_type/1: PROPERTY

Usage: `pe_type(Goal)`

`Goal` will be filtered in partial evaluation time according to the PE types defined in the assertion.

Meta-predicate with arguments: `pe_type(goal)`.

check/1: PREDICATE

No further documentation available for this predicate.

trust/1: PREDICATE

No further documentation available for this predicate.

true/1: PREDICATE

No further documentation available for this predicate.

false/1: PREDICATE

No further documentation available for this predicate.

55.3 Documentation on imports

This module has the following direct dependencies:

– *System library modules:*
  `native_props`, `terms_check`.
– *Internal (engine) modules:*
  `term_basic`, `arithmetic`, `atomic_basic`, `basiccontrol`, `exceptions`, `term_compare`, `term_typing`, `debugger_support`, `hiord_rt`.
– *Packages:*
  `prelude`, `initial`, `condcomp`, `assertions`, `assertions/assertions_basic`, `nortchecks`, `nativeprops`.

56 Properties which are native to analyzers

Author(s): Francisco Bueno, Manuel Hermenegildo, Pedro López, Edison Mera, Amadeo Casas, Jose F. Morales.

This library contains a set of properties which are natively understood by the different program analyzers of `ciaopp`. They are used by `ciaopp` on output and they can also be used as properties in assertions.

Some of the properties can also be used as runtime-checks. See `native_props_rtc` for the runtime-check implementation of such properties.

57 Properties related to sharing/aliasing, groundness

These properties are part of the `native_props` library.

57.1 Usage and interface

- **Library usage:**
  `:- use_module(library(assertions/native_props))`
  or also as a package `:- use_package(nativeprops)`.
  Note the slightly different names of the library and the package.
- **Exports:**
  - *Properties:*
    `mshare/1`, `mshare/2`, `indep/2`, `indep/1`, `covered/2`, `linear/1`, `ivar/1`, `nonground/1`, `clique/1`, `clique_1/1`.

57.2 Documentation on exports

mshare/1: PROPERTY

`mshare(X)`

`X` contains all *sharing sets* [JL88, MH89b] which specify the possible variable occurrences in the terms to which the variables involved in the clause may be bound. Sharing sets are a compact way of representing groundness of variables and dependencies between variables. This representation is however generally difficult to read for humans. For this reason, this information is often translated to `ground/1`, `indep/1` and `indep/2` properties, which are easier to read.

Usage: `mshare(X)`

The sharing pattern for the variables in the clause is `X`.

- *The following properties should hold globally:*
  This predicate is understood natively by CiaoPP as `sharing(X)`. (`native/2`)
  `mshare(X)` is not checked during run-time checking. (`no_rtcheck/1`)

mshare/2: PROPERTY

Usage: `mshare(Xs,Xss)`

The sharing pattern for the variables `Xs` is `Xss`.

- *The following properties should hold globally:*
  This predicate is understood natively by CiaoPP as `sharing(Xs,Xss)`. (`native/2`)
  `mshare(Xs,Xss)` is not checked during run-time checking. (`no_rtcheck/1`)

indep/2: PROPERTY

Usage: `indep(X,Y)`

`X` and `Y` do not have variables in common.

- *The following properties should hold globally:*
  This predicate is understood natively by CiaoPP as `indep([[X,Y]])`. (`native/2`)

indep/1: PROPERTY

Usage: `indep(X)`

The variables in the the pairs in `X` are pairwise independent.

- *The following properties should hold globally:*
  This predicate is understood natively by CiaoPP as `indep(X)`. (`native/2`)

covered/2: PROPERTY

`covered(X,Y)`

All variables occuring in `X` occur also in `Y`. Used by the non-strict independence-based annotators.

Usage: `covered(X,Y)`

`X` is covered by `Y`.

- *The following properties should hold globally:*
  This predicate is understood natively by CiaoPP. (`native/1`)

linear/1: PROPERTY

`linear(X)`

`X` is bound to a term which is linear, i.e., if it contains any variables, such variables appear only once in the term. For example, `[1,2,3]` and `f(A,B)` are linear terms, while `f(A,A)` is not.

Usage: `linear(X)`

`X` is instantiated to a linear term.

- *The following properties should hold globally:*
  This predicate is understood natively by CiaoPP. (`native/1`)

ivar/1: PROPERTY

`ivar(X)`

`X` is a free variable independent of the rest of the variables appearing in the head of the predicate.

For a predicate `p(X0, ..., Xn)`, `ivar(X0)` conceptually expands to `(var(X0), indep([[X0,X1], ..., [X0,Xn]]))`.

Usage: `ivar(X)`

`X` is a free independent variable.

nonground/1: PROPERTY

Usage: `nonground(X)`

`X` is not ground.

- *The following properties should hold globally:*
  This predicate is understood natively by CiaoPP as `not_ground(X)`. (`native/2`)

clique/1: PROPERTY

`clique(X)`

`X` is a set of variables of interest, much the same as a sharing group but `X` represents all the sharing groups in the powerset of those variables. Similar to a sharing group, a clique is often translated to `ground/1`, `indep/1`, and `indep/2` properties.

Usage: `clique(X)`

The clique sharing pattern is `X`.

- *The following properties should hold globally:*

  This predicate is understood natively by CiaoPP as `clique(X)`. (`native/2`)

  `clique(X)` is not checked during run-time checking. (`no_rtcheck/1`)

clique_1/1: PROPERTY

`clique_1(X)`

`X` is a set of variables of interest, much the same as a sharing group but `X` represents all the sharing groups in the powerset of those variables but disregarding the singletons. Similar to a sharing group, a clique_1 is often translated to `ground/1`, `indep/1`, and `indep/2` properties.

Usage: `clique_1(X)`

The 1-clique sharing pattern is `X`.

- *The following properties should hold globally:*

  This predicate is understood natively by CiaoPP as `clique_1(X)`. (`native/2`)

  `clique_1(X)` is not checked during run-time checking. (`no_rtcheck/1`)

57.3 Documentation on imports

This module has the following direct dependencies:

- *Internal (engine) modules:*

  `term_basic`, `arithmetic`, `atomic_basic`, `basiccontrol`, `exceptions`, `term_compare`, `term_typing`, `debugger_support`, `basic_props`.

- *Packages:*

  `prelude`, `initial`, `condcomp`, `assertions`, `assertions/assertions_basic`, `regtypes`.

58 Properties related to determinacy, failure, choice-points

These properties are part of the `native_props` library.

58.1 Usage and interface

- **Library usage:**
  `:- use_module(library(assertions/native_props))`
  or also as a package `:- use_package(nativeprops)`.
  Note the slightly different names of the library and the package.
- **Exports:**
  - *Properties:*
    `det/1`, `fails/1`, `semidet/1`, `multi/1`, `nondet/1`, `mut_exclusive/1`, `not_mut_exclusive/1`, `possibly_not_mut_exclusive/1`, `covered/1`, `not_covered/1`, `possibly_not_covered/1`, `no_choicepoints/1`, `leaves_choicepoints/1`, `is_det/1`, `non_det/1`, `possibly_nondet/1`, `not_fails/1`, `possibly_fails/1`.

58.2 Documentation on exports

det/1: PROPERTY

`det(X)`

Calls of the form `X` are deterministic, i.e., produce exactly one solution (or do not terminate).

Note that it can still leave choice points after its execution, but when backtracking into these, it can only fail or go into an infinite loop.

These properties are inferred and checked natively by CiaoPP using the domains and techniques of [LGBH05, LGBH10, DLGH97, BLGH04].

Usage: `det(X)`

Calls of the form `X` are deterministic.

Meta-predicate with arguments: `det(goal)`.

fails/1: PROPERTY

`fails(X)`

Calls of the form `X` fail.

Usage: `fails(X)`

Calls of the form `X` fail.

- *The following properties should hold globally:*
  This predicate is understood natively by CiaoPP. (`native/1`)

Meta-predicate with arguments: `fails(goal)`.

semidet/1: PROPERTY

`semidet(X)`

Calls of the form `X` are semi-deterministic, i.e., produce at most one solution (or do not terminate).

Note that it can still leave choice points after its execution, but when backtracking into these, it can only fail or go into an infinite loop.

These properties are inferred and checked natively by CiaoPP using the domains and techniques of [LGBH05, LGBH10, DLGH97, BLGH04].

Usage: `semidet(X)`

Calls of the form `X` are semi-deterministic.

Meta-predicate with arguments: `semidet(goal)`.

multi/1: PROPERTY

`multi(X)`

Calls of the form `X` are multi-deterministic, i.e., they produce one or more solutions and do not fail.

Usage: `multi(X)`

Calls of the form `X` are multi-deterministic.

Meta-predicate with arguments: `multi(goal)`.

nondet/1: PROPERTY

`nondet(X)`

Nothing is ensured about failure and determinacy of calls to `X`. This is the default when no information is given for a predicate, so this property does not need to be stated explicitly.

Usage: `nondet(X)`

Calls of the form `X` are non-deterministic.

- *The following properties should hold globally:*
  `nondet(X)` is not checked during run-time checking. (`no_rtcheck/1`)

Meta-predicate with arguments: `nondet(goal)`.

mut_exclusive/1: PROPERTY

`mut_exclusive(X)`

For any call of the form `X` there is at most one clause whose test (guard) succeeds, i.e., clause tests are pairwise exclusive. Note that determinacy is the transitive closure (to all called predicates) of this property. This property is inferred and checked natively by CiaoPP using the domains and techniques of [LGBH05, LGBH10].

Usage: `mut_exclusive(X)`

For any call of the form `X` at most one clause test succeeds.

- *The following properties should hold globally:*
  The runtime check of this property is `unimplemented`. (`rtcheck/2`)

Meta-predicate with arguments: `mut_exclusive(goal)`.

not_mut_exclusive/1: PROPERTY

`not_mut_exclusive(X)`

For calls of the form `X` more than one clause test may succeed. I.e., clause tests are not disjoint for some call.

Usage: `not_mut_exclusive(X)`

For some calls of the form `X` more than one clause test may succeed.

- *The following properties should hold globally:*

  The runtime check of this property is `unimplemented`. (`rtcheck/2`)

Meta-predicate with arguments: `not_mut_exclusive(goal)`.

possibly_not_mut_exclusive/1: PROPERTY

`possibly_not_mut_exclusive(X)`

Mutual exclusion of the clause tests for calls of the form `X` cannot be ensured. This is the default when no information is given for a predicate, so this property does not need to be stated explicitly.

Usage: `possibly_not_mut_exclusive(X)`

Mutual exclusion is not ensured for calls of the form `X`.

- *The following properties should hold globally:*

  `possibly_not_mut_exclusive(X)` is not checked during run-time checking. (`no_rtcheck/1`)

Meta-predicate with arguments: `possibly_not_mut_exclusive(goal)`.

covered/1: PROPERTY

`covered(X)`

For any call of the form `X` there is at least one clause whose test (guard) succeeds (i.e., all the calls of the form `X` are covered). Note that nonfailure is the transitive closure (to all called predicates) of this property. [DLGH97, BLGH04].

Usage: `covered(X)`

All the calls of the form `X` are covered.

- *The following properties should hold globally:*

  The runtime check of this property is `unimplemented`. (`rtcheck/2`)

Meta-predicate with arguments: `covered(goal)`.

not_covered/1: PROPERTY

`not_covered(X)`

There is some call of the form `X` for which there is no clause whose test succeeds [DLGH97].

Usage: `not_covered(X)`

Not all of the calls of the form `X` are covered.

- *The following properties should hold globally:*

  The runtime check of this property is `unimplemented`. (`rtcheck/2`)

Meta-predicate with arguments: `not_covered(goal)`.

possibly_not_covered/1: PROPERTY

`possibly_not_covered(X)`

Covering is not ensured for any call of the form `X`. In other words, nothing can be ensured about covering of such calls.

Usage: `possibly_not_covered(X)`

Covering is not ensured for calls of the form `X`.

- *The following properties should hold globally:*

  `possibly_not_covered(X)` is not checked during run-time checking. (`no_rtcheck/1`)

Meta-predicate with arguments: `possibly_not_covered(goal)`.

no_choicepoints/1: PROPERTY

Usage: `no_choicepoints(X)`

A call to `X` does not leave new choicepoints.

Meta-predicate with arguments: `no_choicepoints(goal)`.

leaves_choicepoints/1: PROPERTY

Usage: `leaves_choicepoints(X)`

A call to `X` leaves new choicepoints.

Meta-predicate with arguments: `leaves_choicepoints(goal)`.

is_det/1: PROPERTY

`is_det(X)`

All calls of the form `X` are deterministic, i.e., produce at most one solution (or do not terminate). In other words, if `X` succeeds, it can only succeed once. It can still leave choice points after its execution, but when backtracking into these, it can only fail or go into an infinite loop. This property is inferred and checked natively by CiaoPP using the domains and techniques of [LGBH05, LGBH10].

Usage: `is_det(X)`

All calls of the form `X` are deterministic.

Meta-predicate with arguments: `is_det(goal)`.

non_det/1: PROPERTY

`non_det(X)`

All calls of the form `X` are non-deterministic, i.e., they always produce more than one solution.

Usage: `non_det(X)`

All calls of the form `X` are non-deterministic.

Meta-predicate with arguments: `non_det(goal)`.

possibly_nondet/1: PROPERTY

`possibly_nondet(X)`

Non-determinism is not ensured for calls of the form `X`. In other words, nothing can be ensured about determinacy of such calls. This is the default when no information is given for a predicate, so this property does not need to be stated explicitly.

Usage: `possibly_nondet(X)`

Non-determinism is not ensured for calls of the form `X`.

– *The following properties should hold globally:*
  `possibly_nondet(X)` is not checked during run-time checking. (`no_rtcheck/1`)

Meta-predicate with arguments: `possibly_nondet(goal)`.

not_fails/1: PROPERTY

`not_fails(X)`

Calls of the form `X` produce at least one solution (succeed), or do not terminate. This property is inferred and checked natively by CiaoPP using the domains and techniques of [DLGH97, BLGH04].

Usage: `not_fails(X)`

All the calls of the form `X` do not fail.

– *The following properties should hold globally:*
  This predicate is understood natively by CiaoPP. (`native/1`)

Meta-predicate with arguments: `not_fails(goal)`.

possibly_fails/1: PROPERTY

`possibly_fails(X)`

Non-failure is not ensured for any call of the form `X`. In other words, nothing can be ensured about non-failure nor termination of such calls.

Usage: `possibly_fails(X)`

Non-failure is not ensured for calls of the form `X`.

– *The following properties should hold globally:*
  `possibly_fails(X)` is not checked during run-time checking. (`no_rtcheck/1`)

Meta-predicate with arguments: `possibly_fails(goal)`.

58.3 Documentation on imports

This module has the following direct dependencies:

– *Internal (engine) modules:*
  `term_basic`, `arithmetic`, `atomic_basic`, `basiccontrol`, `exceptions`, `term_compare`, `term_typing`, `debugger_support`, `basic_props`.
– *Packages:*
  `prelude`, `initial`, `condcomp`, `assertions`, `assertions/assertions_basic`, `regtypes`.

59 Properties related to cardinality and exact solutions

These properties are part of the `native_props` library.

59.1 Usage and interface

- **Library usage:**
  `:- use_module(library(assertions/native_props))`
  or also as a package `:- use_package(nativeprops)`.
  Note the slightly different names of the library and the package.
- **Exports:**
  - *Properties:*
    `cardinality/3`, `num_solutions/2`, `relations/2`, `finite_solutions/1`, `solutions/2`.

59.2 Documentation on exports

cardinality/3: PROPERTY

Usage: `cardinality(Goal,Lower,Upper)`

`Goal` has a number of solutions between `Lower` and `Upper`.

- *The following properties should hold globally:*
  `cardinality(Goal,Lower,Upper)` is not checked during run-time checking. (`no_rtcheck/1`)

Meta-predicate with arguments: `cardinality(goal,?,?)`.

num_solutions/2: PROPERTY

Usage 1: `num_solutions(X,N)`

Calls of the form `X` have `N` solutions, i.e., `N` is the cardinality of the solution set of `X`.

- *If the following properties should hold at call time:*
  `X` is a term which represents a goal, i.e., an atom or a structure. (`cgoal/1`)
  `N` is an integer. (`int/1`)

Usage 2: `num_solutions(Goal,Check)`

For a call to `Goal`, `Check(X)` succeeds, where `X` is the number of solutions.

- *If the following properties should hold at call time:*
  `Goal` is a term which represents a goal, i.e., an atom or a structure. (`cgoal/1`)
  `Check` is a term which represents a goal, i.e., an atom or a structure. (`cgoal/1`)

relations/2: PROPERTY

`relations(X,N)`

Calls of the form `X` produce `N` solutions, i.e., `N` is the cardinality of the solution set of `X`.

Usage: `relations(X,N)`

Goal `X` produces `N` solutions.

- *If the following properties should hold at call time:*
  X is a term which represents a goal, i.e., an atom or a structure. (`cgoal/1`)
  N is an integer. (`int/1`)
  then the following properties should hold globally:
  The runtime check of this property is `unimplemented`. (`rtcheck/2`)

Meta-predicate with arguments: `relations(goal,?)`.

finite_solutions/1: PROPERTY

`finite_solutions(X)`

Calls of the form X produce a finite number of solutions [DLGH97].

Usage: `finite_solutions(X)`

All the calls of the form X have a finite number of solutions.

- *The following properties should hold globally:*
  `finite_solutions(X)` is not checked during run-time checking. (`no_rtcheck/1`)

Meta-predicate with arguments: `finite_solutions(goal)`.

solutions/2: PROPERTY

Usage: `solutions(Goal,Sols)`

Goal `Goal` produces the solutions listed in `Sols`.

- *If the following properties should hold at call time:*
  `Goal` is a term which represents a goal, i.e., an atom or a structure. (`cgoal/1`)
  `Sols` is a list. (`list/1`)

59.3 Documentation on imports

This module has the following direct dependencies:

- *Internal (engine) modules:*
  `term_basic`, `arithmetic`, `atomic_basic`, `basiccontrol`, `exceptions`, `term_compare`, `term_typing`, `debugger_support`, `basic_props`.
- *Packages:*
  `prelude`, `initial`, `condcomp`, `assertions`, `assertions/assertions_basic`, `regtypes`.

60 Properties related to exceptions and signals

These properties are part of the `native_props` library.

60.1 Usage and interface

- **Library usage:**
  `:- use_module(library(assertions/native_props))`
  or also as a package `:- use_package(nativeprops)`.
  Note the slightly different names of the library and the package.
- **Exports:**
  - *Properties:*
    `exception/1`, `exception/2`, `possible_exceptions/2`, `no_exception/1`, `no_exception/2`, `signal/1`, `signal/2`, `possible_signals/2`, `no_signal/1`, `no_signal/2`.

60.2 Documentation on exports

exception/1: PROPERTY

Usage: `exception(Goal)`

Calls of the form `Goal` will throw an (unspecified) exception.

Meta-predicate with arguments: `exception(goal)`.

exception/2: PROPERTY

Usage: `exception(Goal,E)`

Calls to `Goal` will throw an exception that unifies with `E`.

Meta-predicate with arguments: `exception(goal,?)`.

possible_exceptions/2: PROPERTY

Usage: `possible_exceptions(Goal,Es)`

Calls of the form `Goal` may throw exceptions, but only the ones that unify with the terms listed in `Es`.

- *If the following properties should hold at call time:*
  `Es` is a list. (`list/1`)
  then the following properties should hold globally:
  The runtime check of this property is `unimplemented`. (`rtcheck/2`)

Meta-predicate with arguments: `possible_exceptions(goal,?)`.

no_exception/1: PROPERTY

Usage: `no_exception(Goal)`

Calls of the form `Goal` do not throw any exception.

Meta-predicate with arguments: `no_exception(goal)`.

no_exception/2: PROPERTY

Usage: `no_exception(Goal,E)`

Calls of the form `Goal` do not throw any exception that unifies with `E`.

Meta-predicate with arguments: `no_exception(goal,?)`.

signal/1: PROPERTY

Usage: `signal(Goal)`

Calls to `Goal` will send an (unspecified) signal.

Meta-predicate with arguments: `signal(goal)`.

signal/2: PROPERTY

Usage: `signal(Goal,E)`

Calls to `Goal` will send a signal that unifies with `E`.

Meta-predicate with arguments: `signal(goal,?)`.

possible_signals/2: PROPERTY

Usage: `possible_signals(Goal,Es)`

Calls of the form `Goal` may generate signals, but only the ones that unify with the terms listed in `Es`.

- *The following properties should hold globally:*
  The runtime check of this property is `unimplemented`. (`rtcheck/2`)

Meta-predicate with arguments: `possible_signals(goal,?)`.

no_signal/1: PROPERTY

Usage: `no_signal(Goal)`

Calls of the form `Goal` do not send any signal.

Meta-predicate with arguments: `no_signal(goal)`.

no_signal/2: PROPERTY

Usage: `no_signal(Goal,E)`

Calls of the form `Goal` do not send any signals that unify with `E`.

Meta-predicate with arguments: `no_signal(goal,?)`.

60.3 Documentation on imports

This module has the following direct dependencies:

- *Internal (engine) modules:*
  term_basic, arithmetic, atomic_basic, basiccontrol, exceptions, term_compare, term_typing, debugger_support, basic_props.
- *Packages:*
  prelude, initial, condcomp, assertions, assertions/assertions_basic, regtypes.

61 Properties related to side effects

These properties are part of the `native_props` library.

61.1 Usage and interface

- **Library usage:**
  `:- use_module(library(assertions/native_props))`
  or also as a package `:- use_package(nativeprops)`.
  Note the slightly different names of the library and the package.
- **Exports:**
  - *Properties:*
    `sideff_pure/1`, `sideff_soft/1`, `sideff_hard/1`.

61.2 Documentation on exports

sideff_pure/1: PROPERTY

Usage: `sideff_pure(X)`

`X` is pure, i.e., has no side-effects.

- *The following properties should hold globally:*
  `sideff_pure(X)` is not checked during run-time checking. (`no_rtcheck/1`)

Meta-predicate with arguments: `sideff_pure(goal)`.

sideff_soft/1: PROPERTY

Usage: `sideff_soft(X)`

`X` has *soft side-effects*, i.e., those not affecting program execution (e.g., input/output).

- *The following properties should hold globally:*
  `sideff_soft(X)` is not checked during run-time checking. (`no_rtcheck/1`)

Meta-predicate with arguments: `sideff_soft(goal)`.

sideff_hard/1: PROPERTY

Usage: `sideff_hard(X)`

`X` has *hard side-effects*, i.e., those that might affect program execution (e.g., assert/retract).

- *The following properties should hold globally:*
  `sideff_hard(X)` is not checked during run-time checking. (`no_rtcheck/1`)

Meta-predicate with arguments: `sideff_hard(goal)`.

61.3 Documentation on imports

This module has the following direct dependencies:

- *Internal (engine) modules:*
  term_basic, arithmetic, atomic_basic, basiccontrol, exceptions, term_compare, term_typing, debugger_support, basic_props.
- *Packages:*
  prelude, initial, condcomp, assertions, assertions/assertions_basic, regtypes.

62 Properties related to polyhedral constraints

These properties are part of the `native_props` library.

62.1 Usage and interface

- **Library usage:**
  `:- use_module(library(assertions/native_props))`
  or also as a package `:- use_package(nativeprops)`.
  Note the slightly different names of the library and the package.
- **Exports:**
  - *Properties:*
    `constraint/1`.

62.2 Documentation on exports

constraint/1: PROPERTY

`constraint(C)`

`C` contains a list of linear (in)equalities that relate variables and `int` values. For example, `[A < B + 4]` is a constraint while `[A < BC + 4]` or `[A = 3.4, B >= C]` are not. Used by polyhedra-based analyses.

Usage: `constraint(C)`

`C` is a list of linear equations.

- *The following properties should hold globally:*
  This predicate is understood natively by CiaoPP. (`native/1`)

62.3 Documentation on imports

This module has the following direct dependencies:

- *Internal (engine) modules:*
  `term_basic`, `arithmetic`, `atomic_basic`, `basiccontrol`, `exceptions`, `term_compare`, `term_typing`, `debugger_support`, `basic_props`.
- *Packages:*
  `prelude`, `initial`, `condcomp`, `assertions`, `assertions/assertions_basic`, `regtypes`.

63 Properties related to data sizes, cost, termination

These properties are part of the `native_props` library.

63.1 Usage and interface

- **Library usage:**
  `:- use_module(library(assertions/native_props))`
  or also as a package `:- use_package(nativeprops)`.
  Note the slightly different names of the library and the package.
- **Exports:**
  - *Properties:*
    `resource_id/1`, `size/2`, `size/3`, `size/4`, `size_lb/2`, `size_ub/2`, `size_o/2`, `size_metric/3`, `size_metric/4`, `steps_lb/2`, `steps_ub/2`, `steps/2`, `steps_o/2`, `rsize/2`, `costb/4`, `cost/4`, `terminates/1`.
  - *Regular Types:*
    `approx/1`, `cost_expression/1`, `agg_expression/1`, `numeric_constant/1`, `size_term/1`, `indexvar/1`, `number_lattice/1`, `measure_t/1`.

63.2 Documentation on exports

approx/1: REGTYPE

The types of approximations (bounding) supported by the size and cost analyses (see also `resources_basic.pl`).

- `ub`: for upper bounds.
- `lb`: for lower bounds.
- `exact`: for an exact expression.
- `o`: for the big-O.

Usage: `approx(X)`

`X` represents an approximation.

resource_id/1: PROPERTY

A resource is a numerical property that varies (is used) throughout the execution of a piece of code. Resources can be predefined (i.e., in a library) or user-defined. Examples are computational (resolution) steps, time spent, energy consumed, memory usage, bytes sent over a wire, procedure calls, database operations, files left open, monetary units spent, disk space used, etc. See also `resources_decl`.

Usage: `resource_id(X)`

`X` is the name/identifier of a resource, either an atom or a compound term.

cost_expression/1: REGTYPE

A cost expression is a symbolic function representing the cost of executing a piece of code in terms of the sizes of its input arguments and possibly other parameters. It is a term built from elements of the lattice of real numbers (see `number_lattice`), numerical constants (see `numeric_constant/1`), size metrics (see `size_metric`) and the following functors:

- `- /1`: sign reversal.
- `+ /1`: identity.
- `-- /1`: decrement by one.
- `++ /1`: increment by one.
- `+ /2`: addition.
- `- /2`: subtraction.
- `* /2`: multiplication.
- `/ /2`: division.
- `** /2`: exponentiation.
- `exp/2`: exponentiation (DEPRECATED use `** /2`).
- `log/2`: logarithm given the base.
- `log10/1`: 10-base logarithm.
- `log2/1`: 2-base logarithm.
- `exp/1`: exponential (e to the power of).
- `log/1`: natural logarithm (base e).
- `sqrt/1`: square root.
- `fact/1`: factorial.
- `max/2`: maximum.
- `min/2`: minimum.
- `sum(Index,LowerBound,UpperBound,Exp)`: summation of all `Exp`s obtained by instantiating index variable `Index` from `LowerBound` to `UpperBound`.
- `prod(Index,LowerBound,UpperBound,Exp)`: product of all `Exp`s obtained by instantiating index variable `Index` from `LowerBound` to `UpperBound`.

Usage: `cost_expression(X)`

`X` is a cost expression.

agg_expression/1: REGTYPE

An aggregation expression is an expression appearing in an aggregation function (`sum/4`, `prod/4`). An aggregation expression can be a regular cost expression (see `cost_expression/1`) or it may include index variables as well (see `indexvar/1`).

Usage: `agg_expression(X)`

`X` is an aggregation expression.

numeric_constant/1: REGTYPE

A numeric constant is a fixed, well-defined real number.

Currently supported constants:

- `e`: Euler's number.

Usage: `numeric_constant(X)`

`X` represents a numeric constant.

size_term/1: REGTYPE

A size term is a term in a cost expression (see `cost_expression/1`) representing the size of a term in a given metric (see `measure_t/1`).

Usage: `size_term(X)`

`X` represents the size of a term.

indexvar/1: REGTYPE

A variable used as an index in an aggregation expression.

Usage: `indexvar(X)`

`X` is an index variable.

number_lattice/1: REGTYPE

A lattice containing real numbers, `inf` and `bot`.

Usage: `number_lattice(X)`

`X` is a number, `inf` or `bot`.

size/2: PROPERTY

`size(X,Y)`

The exact size (for any approximation) of term `X` is given by expression `Y`, which may depend on the size of other terms. `size(X,Y)` is equivalent to `size(exact,X,Y)`.

Usage: `size(X,Y)`

`Y` is the size of argument `X`, for any approximation.

- *If the following properties should hold at call time:*

  `X` is any term. (`term/1`)

  `Y` is a cost expression. (`cost_expression/1`)

  then the following properties should hold globally:

  `size(X,Y)` is not checked during run-time checking. (`no_rtcheck/1`)

size/3: PROPERTY

`size(A,X,Y)`

The size of term `X` for the approximation `A` is given by expression `Y`, which may depend on the size of other terms.

Usage: `size(A,X,Y)`

`Y` is the size of argument `X`, for the approximation `A`.

- *If the following properties should hold at call time:*

  `A` represents an approximation. (`approx/1`)

  `X` is any term. (`term/1`)

  `Y` is a cost expression. (`cost_expression/1`)

  then the following properties should hold globally:

  `size(A,X,Y)` is not checked during run-time checking. (`no_rtcheck/1`)

size/4: PROPERTY

`size(A,M,X,Y)`

The size of term `X`, measured in metric `M`, for the approximation `A` is given by expression `Y`, which may depend on the size of other terms.

Usage: `size(A,M,X,Y)`

`Y` is the size of argument `X` measured in `M`, for the approximation `A`.

- *If the following properties should hold at call time:*

  `A` represents an approximation. (`approx/1`)

  `M` is a term size metric. (`measure_t/1`)

  `X` is any term. (`term/1`)

  `Y` is a cost expression. (`cost_expression/1`)

  then the following properties should hold globally:

  `size(A,M,X,Y)` is not checked during run-time checking. (`no_rtcheck/1`)

size_lb/2: PROPERTY

`size_lb(X,Y)`

The minimum size of the terms to which the argument `X` is bound is given by the expression `Y`. Various measures can be used to determine the size of an argument, e.g., list-length, term-size, term-depth, integer-value, etc. [DL93, LGHD96]. See `measure_t/1`.

Usage: `size_lb(X,Y)`

`Y` is a lower bound on the size of argument `X`.

- *If the following properties should hold at call time:*

  `X` is any term. (`term/1`)

  `Y` is a cost expression. (`cost_expression/1`)

  then the following properties should hold globally:

  `size_lb(X,Y)` is not checked during run-time checking. (`no_rtcheck/1`)

size_ub/2: PROPERTY

`size_ub(X,Y)`

The maximum size of the terms to which the argument `X` is bound is given by the expression `Y`. Various measures can be used to determine the size of an argument, e.g., list-length, term-size, term-depth, integer-value, etc. [DL93, LGHD96]. See `measure_t/1`.

Usage: `size_ub(X,Y)`

`Y` is a upper bound on the size of argument `X`.

- *If the following properties should hold at call time:*

  `X` is any term. (`term/1`)

  `Y` is a cost expression. (`cost_expression/1`)

  then the following properties should hold globally:

  `size_ub(X,Y)` is not checked during run-time checking. (`no_rtcheck/1`)

size_o/2: PROPERTY

Usage: `size_o(X,Y)`

The size of argument `X` is in the order of expression `Y`.

- *If the following properties should hold at call time:*

  `X` is any term. (`term/1`)

  `Y` is a cost expression. (`cost_expression/1`)

  then the following properties should hold globally:

  `size_o(X,Y)` is not checked during run-time checking. (`no_rtcheck/1`)

size_metric/3: PROPERTY

Usage: `size_metric(Head,Var,Metric)`

`Metric` is the measure used to determine the size of the terms that `Var` is bound to, for any type of approximation.

- *If the following properties should hold at call time:*

  `Head` is a term which represents a goal, i.e., an atom or a structure. (`cgoal/1`)

  `Var` is any term. (`term/1`)

  `Metric` is a term size metric. (`measure_t/1`)

  then the following properties should hold globally:

  `size_metric(Head,Var,Metric)` is not checked during run-time checking. (`no_rtcheck/1`)

Meta-predicate with arguments: `size_metric(goal,?,?)`.

size_metric/4: PROPERTY

Usage: `size_metric(Head,Approx,Var,Metric)`

`Metric` is the measure used to determine the size of the terms that variable `Var` bound to, for the approximation `Approx`.

- *If the following properties should hold at call time:*

  `Head` is a term which represents a goal, i.e., an atom or a structure. (`cgoal/1`)

  `Approx` represents an approximation. (`approx/1`)

  `Var` is any term. (`term/1`)

  `Metric` is a term size metric. (`measure_t/1`)

  then the following properties should hold globally:

  `size_metric(Head,Approx,Var,Metric)` is not checked during run-time checking. (`no_rtcheck/1`)

Meta-predicate with arguments: `size_metric(goal,?,?,?)`.

measure_t/1: REGTYPE

The types of term size measures currently supported in size and cost analysis (see also in `resources_basic.pl`).

- `int`: The size of the term (which is an integer) is the integer value itself.
- `length`: The size of the term (which is a list) is its length.
- `size`: The size is the overall of the term (number of subterms).

- `depth([_|_])`: The size of the term is its depth.
- `void`: Used to indicate that the size of this argument should be ignored.

Usage: `measure_t(X)`

`X` is a term size metric.

steps_lb/2: PROPERTY

`steps_lb(X,Y)`

The minimum computation (in resolution steps) spent by any call of the form `X` is given by the expression `Y` [DLGHL97, LGHD96]

Usage: `steps_lb(X,Y)`

`Y` is a lower bound on the cost of any call of the form `X`.

- *If the following properties should hold at call time:*
  `X` is a term which represents a goal, i.e., an atom or a structure. (`cgoal/1`)
  `Y` is a cost expression. (`cost_expression/1`)
  then the following properties should hold globally:
  `steps_lb(X,Y)` is not checked during run-time checking. (`no_rtcheck/1`)

Meta-predicate with arguments: `steps_lb(goal,?)`.

steps_ub/2: PROPERTY

`steps_ub(X,Y)`

The maximum computation (in resolution steps) spent by any call of the form `X` is given by the expression `Y` [DL93, LGHD96].

Usage: `steps_ub(X,Y)`

`Y` is a upper bound on the cost of any call of the form `X`.

- *If the following properties should hold at call time:*
  `X` is a term which represents a goal, i.e., an atom or a structure. (`cgoal/1`)
  `Y` is a cost expression. (`cost_expression/1`)
  then the following properties should hold globally:
  `steps_ub(X,Y)` is not checked during run-time checking. (`no_rtcheck/1`)

Meta-predicate with arguments: `steps_ub(goal,?)`.

steps/2: PROPERTY

`steps(X,Y)`

The computation (in resolution steps) spent by any call of the form `X` is given by the expression `Y`

Usage: `steps(X,Y)`

`Y` is the cost (number of resolution steps) of any call of the form `X`.

- *If the following properties should hold at call time:*
  `X` is a term which represents a goal, i.e., an atom or a structure. (`cgoal/1`)
  `Y` is a cost expression. (`cost_expression/1`)
  then the following properties should hold globally:
  `steps(X,Y)` is not checked during run-time checking. (`no_rtcheck/1`)

Meta-predicate with arguments: `steps(goal,?)`.

steps_o/2: PROPERTY

`steps_o(X,Y)`

The big O expression of the computation (in resolution steps) spent by any call of the form `X` is given by the expression `Y`

Usage: `steps_o(X,Y)`

`Y` is the complexity order of the cost of any call of the form `X`.

- *If the following properties should hold at call time:*
  `X` is a term which represents a goal, i.e., an atom or a structure. (`cgoal/1`)
  `Y` is a cost expression. (`cost_expression/1`)
  then the following properties should hold globally:
  `steps_o(X,Y)` is not checked during run-time checking. (`no_rtcheck/1`)

Meta-predicate with arguments: `steps_o(goal,?)`.

rsize/2: PROPERTY

Usage: `rsize(Var,SizeDescr)`

`Var` has its size defined by `SizeDescr`.

- *The following properties should hold globally:*
  `rsize(Var,SizeDescr)` is not checked during run-time checking. (`no_rtcheck/1`)

costb/4: PROPERTY

`costb(Goal,Resource,Lower,Upper)`

`Lower` (res. `Upper`) is a (safe) lower (res. upper) bound on the cost of the computation of `Goal` in terms of `Resource` units.

Usage: `costb(Goal,Resource,Lower,Upper)`

`Lower` (resp. `Upper`) is a (safe) lower (resp. upper) bound on the cost of the computation of `Goal` expressed in terms of `Resource` units.

- *If the following properties should hold at call time:*
  `Goal` is a term which represents a goal, i.e., an atom or a structure. (`cgoal/1`)
  `Resource` is the name/identifier of a resource, either an atom or a compound term. (`resource_id/1`)
  `Lower` is a cost expression. (`cost_expression/1`)
  `Upper` is a cost expression. (`cost_expression/1`)
  then the following properties should hold globally:
  `costb(Goal,Resource,Lower,Upper)` is not checked during run-time checking. (`no_rtcheck/1`)

Meta-predicate with arguments: `costb(goal,?,?,?)`.

cost/4: PROPERTY

`cost(Goal,Approx,Resource,Expr)`

`Expr` is a (safe) upper or lower bounds, depending on the value of `Approx` (see `approx/1`), of the cost of computation of the goal `Goal` in terms of `Resource` units.

Usage: `cost(Goal,Approx,Resource,Expr)`

`Expr` is a safe upper or lower bounds (depending on `Approx`) of the cost of computing `Goal` in terms of `Resource` units.

- *If the following properties should hold at call time:*
  `Goal` is a term which represents a goal, i.e., an atom or a structure. (`cgoal/1`)
  `Approx` represents an approximation. (`approx/1`)
  `Resource` is the name/identifier of a resource, either an atom or a compound term. (`resource_id/1`)
  `Expr` is a cost expression. (`cost_expression/1`)
  then the following properties should hold globally:
  `cost(Goal,Approx,Resource,Expr)` is not checked during run-time checking. (`no_rtcheck/1`)

Meta-predicate with arguments: `cost(goal,?,?,?)`.

terminates/1: PROPERTY

`terminates(X)`

Calls of the form `X` always terminate.

Usage: `terminates(X)`

All calls of the form `X` terminate.

- *If the following properties should hold at call time:*
  `X` is a term which represents a goal, i.e., an atom or a structure. (`cgoal/1`)
  then the following properties should hold globally:
  `terminates(X)` is not checked during run-time checking. (`no_rtcheck/1`)

Meta-predicate with arguments: `terminates(goal)`.

63.3 Documentation on imports

This module has the following direct dependencies:

- *Internal (engine) modules:*
  `term_basic`, `arithmetic`, `atomic_basic`, `basiccontrol`, `exceptions`, `term_compare`, `term_typing`, `debugger_support`, `basic_props`.
- *Packages:*
  `prelude`, `initial`, `condcomp`, `assertions`, `assertions/assertions_basic`, `regtypes`.

64 Run-time checking of assertions

Author(s): Edison Mera, Nataliia Stulova.

Stability: **[beta]** Most of the functionality is there but it is still missing some testing and/or verification.

This package provides a complete implementation of run-time checks of predicate assertions. The program is instrumented to check such assertions at run time, and in case a property does not hold, the error is reported. Note that there is also an older package called rtchecks, by David Trallero. The advantage of this one is that it can be used independently of CiaoPP and also has updated functionality.

There are two main applications of run-time checks:

- To improve debugging of certain predicates, specifying some expected behavior that is checked at run-time with the assertions.
- To avoid manual implementation of run-time checks that should be done in some predicates, leaving the code clean and understandable.

64.1 Run-time checks configuration with flags

The run-time checks can be configured using prolog flags. Below we itemize the valid prolog flags with its values and a brief explanation of the meaning:

- `rtchecks_level`
  - `exports`: Only use rtchecks for external calls of the exported predicates.
  - `inner` : Use also rtchecks for internal calls. Default.
- `rtchecks_trust`
  - `no` : Disable rtchecks for trust assertions.
  - `yes` : Enable rtchecks for trust assertions. Default.
- `rtchecks_entry`
  - `no` : Disable rtchecks for entry assertions.
  - `yes` : Enable rtchecks for entry assertions. Default.
- `rtchecks_exit`
  - `no` : Disable rtchecks for exit assertions.
  - `yes` : Enable rtchecks for exit assertions. Default.
- `rtchecks_test`
  - `no` : Disable rtchecks for test assertions. Default.
  - `yes` : Enable rtchecks for test assertions. Used for debugging purposes, but is better to use the unittest library.
- `rtchecks_asrloc` Controls the usage of locators for the assertions in the error messages. The locator says the file and lines that contains the assertion that had failed. Valid values are:
  - `no` : Disabled.
  - `yes` : Enabled. Default.
- `rtchecks_predloc` Controls the usage of locators for the predicate that caused the run-time check error. The locator says the first clause of the predicate that the violated assertion refers to.
  - `no` : Disabled.
  - `yes` : Enabled, Default.
- `rtchecks_callloc`

    - `no` : Do not show the stack of predicates that caused the failure
    - `predicate`: Show the stack of predicates that caused the failure. Instrument it in the predicate. Default.
    - `literal` : Show the stack of predicates that caused the failure. Instrument it in the literal. This mode provides more information, because reports also the literal in the body of the predicate.
- `rtchecks_namefmt`
    - `long` : Show the name of predicates, properties and the values of the variables. Default.
    - `short` : Only show the name of the predicate in a reduced format.

64.2 Custom implementation for property checks

Note: this is a new feature and under active development. The documentation may be partial/obsolete.

The run-time check instrumentation can use custom property checks implementations. This is useful when defining native properties (without a clause-based definition supported by the default instrumentation) or when the generic run-time checks are not efficient enough. The custom implementation can be used specifically in run-time checks, while keeping a declarative version (which might be easier to read or is better understood by some static analyzer).

To provide the custom property implementation for run-time checking it is recommended to edit two files (suppose the original property is defined in a module `foo.pl`):

- the `foo_rtc.pl` module that contains the custom property implementation and is placed in the same folder as `foo.pl`. Make sure the custom property implementation is exported from both modules.
- `$CIAOROOT/core/lib/rtchecks/rtchecks_rt_propimpl.pl`, a database file that stores the links between different property implementations in the format of declarations '`:- rtc_impl(ModOr:PropOr/ArityOr, ModRt:PropRt/ArityRt).`' where `ModOr` and `ModRt` are names of the two modules that contain the original and the custom property definitions, `PropOr` and `PropRt` are the two different property implementations with respective arities `ArityOr` and `ArityRt`.

After these edits the `rtchecks` library needs to be rebuilt.

For an example of a system library that uses this feature see the `assertions/native_props` library.

64.3 Usage and interface

- **Library usage:**

  :- module(...,...,[...,rtchecks]).
- **Implicit imports:**
  - *Internal (engine) modules:*

    `term_basic`, `arithmetic`, `atomic_basic`, `basiccontrol`, `exceptions`, `term_compare`, `term_typing`, `debugger_support`, `basic_props`.
  - *Packages:*

    `prelude`, `initial`, `condcomp`, `assertions`, `assertions/assertions_basic`.

64.4 Other information

Note: the `assertions` package must always be included together with the `rtchecks` package (see `core/lib/compiler/global_module_options.pl` for details).

65 Applying assertions during fixpoint

Author(s): Isabel Garcia-Contreras.

This module contains the logic of what assertions need to be applied depending on the analysis step.

This module reads the values of the following flags:

- `fixp_stick_to_success`: whether to use the value of the success assertions instead of what was inferred for the fixpoint (not working for eterms).

  For a module:

  ```
  :- module(_,[p/1],[assertions]).

  :- pred p(X) => atm(X).
  p(X) :- X = a.
  ```

  If `fixp_stick_to_success` is set to `on`, the analysis result will be:

  ```
  :- module(_,[p/1],[assertions]).

  :- true pred p(X) => atm(X).
  p(X) :- X = a.
  ```

  Because the value of the assertion is used as it is. If it is set to `off`, it will use the assertion only when it allows to gain precision (useless in this case).

  ```
  :- module(_,[p/1],[assertions,regtypes]).

  :- true pred p(X) => rta(X).
  p(X) :- X = a.

  :- regtype rta/1.
  rta(a).
  ```

- `fixp_stick_to_calls`: **not implemented yet**, whether to use the value of the calls assertion instead of what was inferred by the fixpoint. This is harder than the success case since the heads of the analysis information may not be normalized.

  For example in the following module:

  ```
  :- module(_,[main/0,p/1],[assertions]).

  main :-
      p(a).

  :- pred p(X) : atm(X).
  p(X).
  ```

  Since the plai does not normalize heads it is imposible to generalize `p(a)` to `p(X) : atm(X)`, the variables that the abstract substitution approximates must be in the head. However `p(X) : rta(X)` can be generalized only by changing the abstract substitution because the variable is already in the head.

  The fixpoint would have to be modified so that applying assertions can change the goals

- `use_check_as_trust`: To activate the runtime semantics of the check assertions, i.e., use them as trust in the fixpoint.

Program point assertions are not processed here.

65.1 Usage and interface

- **Library usage:**
  `:- use_module(ciaopp(plai/apply_assertions)).`
- **Exports:**
  - *Predicates:*
    `call_asr/5`, `success_asr/6`, `cleanup_applied_assertions/1`, `apply_assrt_call_to_success/8`, `apply_assrt_exit/7`, `apply_assrt_no_source/6`, `get_call_assertions_asub/5`, `get_succ_assertion_asubs/7`, `add_success_asr/6`, `abs_normalize/8`, `get_applicable_status/3`.

65.2 Documentation on exports

call_asr/5: PREDICATE

No further documentation available for this predicate. The predicate is of type *data*.

success_asr/6: PREDICATE

No further documentation available for this predicate. The predicate is of type *data*.

cleanup_applied_assertions/1: PREDICATE

Cleans the cached assertions

apply_assrt_call_to_success/8: PREDICATE

Usage: `apply_assrt_call_to_success(AbsInt,Sg,Sv,Proj0,HvFv_u,Call,NewProj,NewCall)`

`NewProj` and `NewCall` are the result of applying the assertions that correspond to the `call_to_success` analysis point.

Currently, the implementation has only one option which is applying `true` and `trust` call assertions, and `true` success assertions (that are applicable to `Sg`:`Proj`).

- *The following properties should hold at call time:*
  - `AbsInt` is currently a term which is not a free variable. (`nonvar/1`)
  - `Sg` is currently a term which is not a free variable. (`nonvar/1`)
  - `Sv` is currently a term which is not a free variable. (`nonvar/1`)
  - `Proj0` is currently a term which is not a free variable. (`nonvar/1`)
  - `HvFv_u` is currently a term which is not a free variable. (`nonvar/1`)
  - `Call` is currently a term which is not a free variable. (`nonvar/1`)
  - `NewProj` is a free variable. (`var/1`)
  - `NewCall` is a free variable. (`var/1`)
  - `AbsInt` is an atom. (`atm/1`)
  - `Sv` is a list. (`list/1`)
  - `HvFv_u` is a list. (`list/1`)

- *The following properties should hold globally:*
  All the calls of the form `apply_assrt_call_to_success(AbsInt,Sg,Sv,Proj0,HvFv_u,Call,NewProj,NewCall)` do not fail. (`not_fails/1`)

apply_assrt_exit/7: PREDICATE

Usage: `apply_assrt_exit(AbsInt,Sg,Sv,Proj0,LPrime,NLPrime,Applied)`

- *The following properties should hold at call time:*
  `AbsInt` is currently a term which is not a free variable. (`nonvar/1`)
  `Sg` is currently a term which is not a free variable. (`nonvar/1`)
  `Sv` is currently a term which is not a free variable. (`nonvar/1`)
  `Proj0` is currently a term which is not a free variable. (`nonvar/1`)
  `LPrime` is currently a term which is not a free variable. (`nonvar/1`)
  `NLPrime` is a free variable. (`var/1`)
  `AbsInt` is an atom. (`atm/1`)
  `Sv` is a list. (`list/1`)
  `LPrime` is a list. (`list/1`)
- *The following properties should hold upon exit:*
  `NLPrime` is a list. (`list/1`)
- *The following properties should hold globally:*
  All the calls of the form `apply_assrt_exit(AbsInt,Sg,Sv,Proj0,LPrime,NLPrime,Applied)` do not fail. (`not_fails/1`)
  All calls of the form `apply_assrt_exit(AbsInt,Sg,Sv,Proj0,LPrime,NLPrime,Applied)` are deterministic. (`is_det/1`)

apply_assrt_no_source/6: PREDICATE

Usage: `apply_assrt_no_source(SgKey,AbsInt,Sg,Sv,Proj,Prime)`

Succeeds if there are applicable assertions to `Sg:Proj` understood by domain `AbsInt` and `Prime` is the answer pattern.

- *The following properties should hold at call time:*
  `SgKey` is currently a term which is not a free variable. (`nonvar/1`)
  `AbsInt` is currently a term which is not a free variable. (`nonvar/1`)
  `Sg` is currently a term which is not a free variable. (`nonvar/1`)
  `Sv` is currently a term which is not a free variable. (`nonvar/1`)
  `Proj` is currently a term which is not a free variable. (`nonvar/1`)
  `Prime` is a free variable. (`var/1`)
  `SgKey` is an atom. (`atm/1`)
  `AbsInt` is an atom. (`atm/1`)
  `Sv` is a list. (`list/1`)
- *The following properties should hold upon exit:*
  `Prime` is currently a term which is not a free variable. (`nonvar/1`)

get_call_assertions_asub/5: PREDICATE

Usage: `get_call_assertions_asub(Head,SgKey,Sts,AbsInt,AsrProj)`

- *The following properties should hold at call time:*
  `Head` is currently a term which is not a free variable. (`nonvar/1`)
  `AsrProj` is a free variable. (`var/1`)
  `Sts` is a list. (`list/1`)
  `SgKey` is an atom. (`atm/1`)
  `AbsInt` is an atom. (`atm/1`)
- *The following properties should hold upon exit:*
  `AsrProj` is currently a term which is not a free variable. (`nonvar/1`)

get_succ_assertion_asubs/7: PREDICATE

Usage: `get_succ_assertion_asubs(SgKey,Head,Hv,Sts,AbsInt,Proj,Prime)`

This predicate succeeds if there are assertions that are applicable to `Head`:`Proj` (normalized). If no `Sts` is specified all trusted status are returned

- *The following properties should hold at call time:*
  `SgKey` is currently a term which is not a free variable. (`nonvar/1`)
  `Head` is currently a term which is not a free variable. (`nonvar/1`)
  `Hv` is currently a term which is not a free variable. (`nonvar/1`)
  `Sts` is currently a term which is not a free variable. (`nonvar/1`)
  `AbsInt` is currently a term which is not a free variable. (`nonvar/1`)
  `Proj` is currently a term which is not a free variable. (`nonvar/1`)
  `Prime` is a free variable. (`var/1`)
  `SgKey` is an atom. (`atm/1`)
  `Hv` is a list. (`list/1`)
  `Sts` is a list. (`list/1`)
  `AbsInt` is an atom. (`atm/1`)
- *The following properties should hold upon exit:*
  `Prime` is currently a term which is not a free variable. (`nonvar/1`)

add_success_asr/6: PREDICATE

Usage: `add_success_asr(SgKey,Head,Sts,AbsInt,AsrProj,AsrPrime)`

Joins an abstract success pattern from an assertion with a previous value for the call pattern `Head`:`AsrProj`.

- *The following properties should hold at call time:*
  `SgKey` is currently a term which is not a free variable. (`nonvar/1`)
  `Head` is currently a term which is not a free variable. (`nonvar/1`)
  `Sts` is currently a term which is not a free variable. (`nonvar/1`)
  `AbsInt` is currently a term which is not a free variable. (`nonvar/1`)
  `AsrProj` is currently a term which is not a free variable. (`nonvar/1`)
  `AsrPrime` is currently a term which is not a free variable. (`nonvar/1`)
  `SgKey` is an atom. (`atm/1`)
  `Sts` is a list. (`list/1`)
  `AbsInt` is an atom. (`atm/1`)

- *The following properties should hold globally:*
  All the calls of the form `add_success_asr(SgKey,Head,Sts,AbsInt,AsrProj,AsrPrime)` do not fail. (`not_fails/1`)

abs_normalize/8: PREDICATE

Usage: `abs_normalize(AbsInt,Sg,Sv,ASub0,Head,Hv,ASub,ExtraInfo)`

- *The following properties should hold at call time:*
  `AbsInt` is currently a term which is not a free variable. (`nonvar/1`)
  `Sg` is currently a term which is not a free variable. (`nonvar/1`)
  `Sv` is currently a term which is not a free variable. (`nonvar/1`)
  `ASub0` is currently a term which is not a free variable. (`nonvar/1`)
  `ASub` is a free variable. (`var/1`)
  `ExtraInfo` is a free variable. (`var/1`)
  `AbsInt` is an atom. (`atm/1`)
  `Sv` is a list. (`list/1`)
- *The following properties should hold upon exit:*
  `ASub` is currently a term which is not a free variable. (`nonvar/1`)
- *The following properties should hold globally:*
  All the calls of the form `abs_normalize(AbsInt,Sg,Sv,ASub0,Head,Hv,ASub,ExtraInfo)` do not fail. (`not_fails/1`)

get_applicable_status/3: PREDICATE

No further documentation available for this predicate.

65.3 Documentation on imports

This module has the following direct dependencies:

- *Application modules:*
  `preprocess_flags`, `frontend_driver`, `fixpo_ops`, `apply_assertions_old`, `domains`, `plai_errors`.
- *System library modules:*
  `datafacts_rt`, `native_props`, `terms_vars`, `aggregates`, `lists`, `assrt_lib`, `program_keys`, `p_unit`, `p_unit_db`.
- *Internal (engine) modules:*
  `term_basic`, `arithmetic`, `atomic_basic`, `basiccontrol`, `exceptions`, `term_compare`, `term_typing`, `debugger_support`, `basic_props`, `hiord_rt`.
- *Packages:*
  `prelude`, `initial`, `condcomp`, `assertions`, `assertions/assertions_basic`, `isomodes`, `datafacts`, `nativeprops`, `.(notrace)`.

66 PART III - Extending CiaoPP

Author(s): The Ciao Development Team.

67 Adding a new analysis domain to CiaoPP

Author(s): The Ciao Development Team.

One of the most relevant features of the CiaoPP system is that it allows the addition of new analysis domains to the abstract interpretation-based framework in an easy way.

The next Chapter (Chapter 68 [Plug-in points for abstract domains], page 421) describes the module `domains.pl`, the general interface used by the analyzer for accessing the operations of the different domains implemented in the system. The developer of a new domain must edit this module, adding the necessary clauses to link the general interface with the specific operations of the new domain.

The procedure of adding a new domain is illustrated in Chapter 16 [Simple groundness abstract domain], page 103 by means of a simple abstract domain for groundness information inference. It includes a list of the predicates that need to be implemented by the developer in order to add the abstract domain to CiaoPP.

68 Plug-in points for abstract domains

Author(s): Maria Garcia de la Banda, Francisco Bueno, Jose F. Morales (aidomain package).

This module contains the predicates for selecting the abstract operations that correspond to an analysis domain. The selection depends on the name of the domain given as first argument to all predicates. Whenever a new domain is added to the system, a new clause for each predicate exported here will be needed to call the corresponding domain operation in the domain module. Some local operations used but not exported by this module would have to be defined, too. See the following chapter for an example domain module.

Adding an analysis domain to PLAI requires only changes in this module. However, in order for other CiaoPP operations to work, you may need to change other modules. See, for example, module `infer_dom`.

In this chapter, arguments referred to as `Sv`, `Hv`, `Fv`, `Qv`, `Vars` are lists of program variables and are supposed to always be sorted. Abstract substitutions are referred to as `ASub`, and are also supposed sorted (except where indicated), although this depends on the domain.

68.1 Variable naming convention for CiaoPP domains

Both in the PLAI fixpoints and domains, for simplicity, we use the following variable name meanings:

- `AbsInt` : identifier of the abstract interpreter being used.
- `Sg` : Subgoal being analysed.
- `SgKey` : Subgoal key (represented by functor/arity).
- `Head` : Head of the clause being analysed.
- `Sv` : Subgoal variables.
- `Hv` : Head variables.
- `Fv` : Free variables in the body of the clause being considered.
- `Vars` : Any possible set of variables.
- `Call` : Abstract call substitution.
- `Proj` : `Call` projected onto `Sv`.
- `Entry` : Abstract entry substitution (i.e. the abstract subtitution obtained after the abstract unification of `Sg` and `Head` projected onto `Hv + Fv`).
- `Exit` : Abstract exit substitution (i.e. the abstract subtitution obtained after the analysis of the clause being considered projected onto `Hv`).
- `Prime` : Abstract prime substitution (i.e. the abstract subtitution obtained after the analysis of the clause being considered projected onto `Sv`).
- `Succ` : Abstract success substitution (i.e. the abstract subtitution obtained after the analysis of the clause being considered extended to the variables of the clause in which Sg appears).
- `ASub` : Any possible abstract substitution.
- `R_flag` : Flag which represents the recursive characteristics of a predicate. It will be "nr" in case the predicate be non recursive. Otherwise it will be r (recursive).
- `List` : (can be represented as `OldList`,`List`,`AddList`,`IdList`,`NewList`) current the list of nodes which a given node depends on.
- `_s` : The suffix `_s` means that the term to which the variable is bound to has been sorted. By default they are always sorted thus `_s` is added only when it appears neccessary to say it explicitely.
- `_uns` : The suffix `_uns` means that the term to which the variable is bound is not sorted.
- `ExtraInfo`: Info computed during the `call_to_entry` that can be reused during the `exit_to_prime` step.

68.2 Usage and interface

- **Library usage:**
  `:- use_module(ciaopp(plai/domains)).`
- **Exports:**
  - *Predicates:*
    `init_abstract_domain/2`, `amgu/5`, `augment_asub/4`, `augment_two_asub/4`, `call_to_entry/10`, `exit_to_prime/8`, `project/5`, `project/6`, `needs/2`, `widencall/4`, `dual_widencall/4`, `widen/4`, `dual_widen/4`, `normalize_asub/3`, `compute_lub/3`, `compute_glb/3`, `identical_proj/5`, `identical_proj_1/7`, `identical_abstract/3`, `abs_sort/3`, `extend/6`, `less_or_equal_proj/5`, `less_or_equal/3`, `glb/4`, `eliminate_equivalent/3`, `abs_subset/3`, `call_to_success_fact/10`, `special_builtin/6`, `body_succ_builtin/9`, `success_builtin/7`, `obtain_info/5`, `info_to_asub/7`, `full_info_to_asub/5`, `asub_to_info/5`, `asub_to_native/6`, `concrete/4`, `unknown_call/5`, `unknown_entry/4`, `empty_entry/4`, `part_conc/5`, `multi_part_conc/4`, `collect_types_in_abs/4`, `rename_types_in_abs/4`, `dom_statistics/2`, `abstract_instance/5`, `contains_parameters/2`, `absub_eliminate_equivalent/3`, `absub_fixpoint_covered/3`, `body_builtin/9`, `undef_call_to_success_builtin/2`.
  - *Multifiles:*
    `aidomain/1`, `aidom.init_abstract_domain/2`, `aidom.amgu/5`, `aidom.augment_asub/4`, `aidom.augment_two_asub/4`, `aidom.call_to_entry/10`, `aidom.exit_to_prime/8`, `aidom.project/6`, `aidom.widencall/4`, `aidom.needs/2`, `aidom.widen/4`, `aidom.compute_lub/3`, `aidom.compute_clauses_lub/4`, `aidom.compute_clauses_glb/4`, `aidom.fixpoint_covered/3`, `aidom.fixpoint_covered_gfp/3`, `aidom.identical_abstract/3`, `aidom.abs_sort/3`, `aidom.extend/6`, `aidom.less_or_equal/3`, `aidom.glb/4`, `aidom.eliminate_equivalent/3`, `aidom.abs_subset/3`, `aidom.call_to_success_fact/10`, `aidom.special_builtin/6`, `aidom.combined_special_`
    `builtin0/3`, `aidom.body_succ_builtin/9`, `aidom.split_combined_domain/4`, `aidom.success_builtin/7`, `aidom.call_to_success_builtin/7`, `aidom.obtain_info/5`, `aidom.input_interface/5`, `aidom.input_user_interface/6`, `aidom.asub_to_native/6`, `aidom.concrete/4`, `aidom.unknown_call/5`, `aidom.unknown_entry/4`, `aidom.empty_entry/4`, `aidom.part_conc/5`, `aidom.multi_part_conc/4`, `aidom.collect_auxinfo_asub/4`, `aidom.rename_auxinfo_asub/4`, `aidom.dom_statistics/2`, `aidom.contains_parameters/2`.

68.3 Documentation on exports

init_abstract_domain/2: PREDICATE

`init_abstract_domain(AbsInt,PushedFlags)`

Initializes abstract domain `AbsInt`. Tells the list of modified (pushed) PP flags to pop afterwards.

amgu/5: PREDICATE

Usage: `amgu(AbsInt,Sg,Head,ASub,AMGU)`

Perform the abstract unification `AMGU` between `Sg` and `Head` given an initial abstract substitution `ASub` and abstract domain `AbsInt`.

- *The following properties should hold at call time:*
  `AbsInt` is currently a term which is not a free variable. (`nonvar/1`)
  `Sg` is currently a term which is not a free variable. (`nonvar/1`)
  `Head` is currently a term which is not a free variable. (`nonvar/1`)
  `ASub` is currently a term which is not a free variable. (`nonvar/1`)
  `AMGU` is a free variable. (`var/1`)
  `AbsInt` is an atom. (`atm/1`)
- *The following properties should hold globally:*
  All the calls of the form `amgu(AbsInt,Sg,Head,ASub,AMGU)` do not fail. (`not_fails/1`)

augment_asub/4: PREDICATE

`augment_asub(AbsInt,ASub,Vars,ASub0)`

Augment the abstract substitution `ASub` adding the variables `Vars` and then resulting the abstract substitution `ASub0`.

augment_two_asub/4: PREDICATE

`augment_two_asub(AbsInt,ASub0,ASub1,ASub)`

`ASub` is an abstract substitution resulting of augmenting two abstract substitutions: `ASub0` and `ASub1` whose domains are disjoint.

call_to_entry/10: PREDICATE

Usage:

`call_to_entry(AbsInt,Sv,Sg,Hv,Head,ClauseKey,Fv,Proj,Entry,ExtraInfo)`

Obtains the abstract substitution `Entry` which results from adding the abstraction of the unification `Sg` = `Head` to abstract substitution `Proj` (the call substitution for `Sg` projected on its variables `Sv`) and then projecting the resulting substitution onto `Hv` (the variables of `Head`) plus `Fv` (the free variables of the relevant clause). `ExtraInfo` is information which may be reused later in other abstract operations.

Assumtions: This predicate assumes that the variables in `Sv` and `Proj` are sorted (see sortedness for each domain).

.

- *The following properties should hold at call time:*
  `AbsInt` is currently a term which is not a free variable. (`nonvar/1`)
  `Sv` is currently a term which is not a free variable. (`nonvar/1`)
  `Sg` is currently a term which is not a free variable. (`nonvar/1`)
  `Hv` is currently a term which is not a free variable. (`nonvar/1`)
  `Head` is currently a term which is not a free variable. (`nonvar/1`)
  `ClauseKey` is currently a term which is not a free variable. (`nonvar/1`)
  `Fv` is currently a term which is not a free variable. (`nonvar/1`)
  `Proj` is currently a term which is not a free variable. (`nonvar/1`)
  `Entry` is a free variable. (`var/1`)

ExtraInfo is a free variable. (var/1)
AbsInt is an atom. (atm/1)
Sv is a list. (list/1)
Hv is a list. (list/1)
Fv is a list. (list/1)

- *The following properties should hold globally:*
  All the calls of the form call_to_entry(AbsInt,Sv,Sg,Hv,Head,ClauseKey,Fv,Proj,Entry,ExtraInfo) do not fail. (not_fails/1)
  All calls of the form call_to_entry(AbsInt,Sv,Sg,Hv,Head,ClauseKey,Fv,Proj,Entry,ExtraInfo) are deterministic. (is_det/1)

exit_to_prime/8: PREDICATE

Usage: exit_to_prime(AbsInt,Sg,Hv,Head,Sv,Exit,ExtraInfo,Prime)

Computes the abstract substitution Prime which results from adding the abstraction of the unification Sg = Head to abstract substitution Exit (the exit substitution for a clause Head projected over its variables Hv), projecting the resulting substitution onto Sv.

- *The following properties should hold at call time:*
  AbsInt is currently a term which is not a free variable. (nonvar/1)
  Sg is currently a term which is not a free variable. (nonvar/1)
  Hv is currently a term which is not a free variable. (nonvar/1)
  Head is currently a term which is not a free variable. (nonvar/1)
  Sv is currently a term which is not a free variable. (nonvar/1)
  Exit is currently a term which is not a free variable. (nonvar/1)
  Prime is a free variable. (var/1)
  AbsInt is an atom. (atm/1)
  Hv is a list of vars. (list/2)
  Sv is a list of vars. (list/2)
- *The following properties should hold globally:*
  All the calls of the form exit_to_prime(AbsInt,Sg,Hv,Head,Sv,Exit,ExtraInfo,Prime) do not fail. (not_fails/1)

project/5: PREDICATE

Usage: project(AbsInt,Vars,HvFv_u,ASub,Proj)

Projects the abstract substitution ASub onto the variables of list Vars resulting in the projected abstract substitution Proj.

- *The following properties should hold at call time:*
  AbsInt is currently a term which is not a free variable. (nonvar/1)
  Vars is currently a term which is not a free variable. (nonvar/1)
  HvFv_u is currently a term which is not a free variable. (nonvar/1)
  ASub is currently a term which is not a free variable. (nonvar/1)

`Proj` is a free variable. (`var/1`)
`AbsInt` is an atom. (`atm/1`)
`Vars` is a list. (`list/1`)
`HvFv_u` is a list. (`list/1`)
`ASub` is any term. (`term/1`)
`Proj` is any term. (`term/1`)

– *The following properties should hold globally:*

All the calls of the form `project(AbsInt,Vars,HvFv_u,ASub,Proj)` do not fail. (`not_fails/1`)

All calls of the form `project(AbsInt,Vars,HvFv_u,ASub,Proj)` are deterministic. (`is_det/1`)

project/6: PREDICATE

Usage: `project(AbsInt,Sg,Vars,HvFv_u,ASub,Proj)`

Projects the abstract substitution `ASub` onto the variables of list `Vars` resulting in the projected abstract substitution `Proj`.

– *The following properties should hold at call time:*

`AbsInt` is currently a term which is not a free variable. (`nonvar/1`)
`Sg` is currently a term which is not a free variable. (`nonvar/1`)
`Vars` is currently a term which is not a free variable. (`nonvar/1`)
`HvFv_u` is currently a term which is not a free variable. (`nonvar/1`)
`ASub` is currently a term which is not a free variable. (`nonvar/1`)
`Proj` is a free variable. (`var/1`)
`AbsInt` is an atom. (`atm/1`)
`Sg` is any term. (`term/1`)
`Vars` is a list. (`list/1`)
`HvFv_u` is a list. (`list/1`)
`ASub` is any term. (`term/1`)
`Proj` is any term. (`term/1`)

– *The following properties should hold globally:*

All the calls of the form `project(AbsInt,Sg,Vars,HvFv_u,ASub,Proj)` do not fail. (`not_fails/1`)

All calls of the form `project(AbsInt,Sg,Vars,HvFv_u,ASub,Proj)` are deterministic. (`is_det/1`)

needs/2: PREDICATE

Usage: `needs(AbsInt,Op)`

Succeeds if `AbsInt` needs operation `Op` for correctness/termination. It must only be used for checking, not enumerating. The supported operations are: `widen` (whether widening is necessary for termination), `clauses_lub` (whether the lub must be performed over the abstract substitution split by clase), `split_combined_domain` (whether the domain contains information of several domains and it needs to be split), and `aux_info` (whether the information in the abstract substitutions is not complete and an external solver may be needed, currently only used when outputing the analysis in a `.dump` file)

- *The following properties should hold at call time:*
  `AbsInt` is currently a term which is not a free variable. (`nonvar/1`)
  `Op` is currently a term which is not a free variable. (`nonvar/1`)
- *The following properties should hold globally:*
  All calls of the form `needs(AbsInt,Op)` are deterministic. (`is_det/1`)

widencall/4: PREDICATE

Usage: `widencall(AbsInt,ASub0,ASub1,ASub)`

`ASub` is the result of widening abstract substitution `ASub0` and `ASub1`, which are supposed to be consecutive call patterns in a fixpoint computation.

This predicate is allowed to fail and it fails if the domain does not define a widening on calls.

- *The following properties should hold at call time:*
  `AbsInt` is currently a term which is not a free variable. (`nonvar/1`)
  `ASub0` is currently a term which is not a free variable. (`nonvar/1`)
  `ASub1` is currently a term which is not a free variable. (`nonvar/1`)
  `ASub` is a free variable. (`var/1`)
  `AbsInt` is an atom. (`atm/1`)

dual_widencall/4: PREDICATE

`dual_widencall(AbsInt,ASub0,ASub1,ASub)`

`ASub` is the result of dual widening abstract substitution `ASub0` and `ASub1`, which are supposed to be consecutive call patterns in a fixpoint computation.

widen/4: PREDICATE

Usage: `widen(AbsInt,ASub0,ASub1,ASub)`

`ASub` is the result of widening abstract substitution `ASub0` and `ASub1`, which are supposed to be consecutive approximations to the same abstract value. I.e., `less_or_equal(AbsInt,ASub0,ASub1)` succeeds.

- *The following properties should hold at call time:*
  `AbsInt` is currently a term which is not a free variable. (`nonvar/1`)
  `ASub0` is currently a term which is not a free variable. (`nonvar/1`)
  `ASub1` is currently a term which is not a free variable. (`nonvar/1`)
  `ASub` is a free variable. (`var/1`)
  `AbsInt` is an atom. (`atm/1`)
- *The following properties should hold globally:*
  All the calls of the form `widen(AbsInt,ASub0,ASub1,ASub)` do not fail. (`not_fails/1`)

dual_widen/4: PREDICATE

`dual_widen(AbsInt,ASub0,ASub1,ASub)`

`ASub` is the result of dual widening abstract substitution `ASub0` and `ASub1`, which are supposed to be consecutive approximations to the same abstract value.

normalize_asub/3: PREDICATE

`normalize_asub(AbsInt,ASub0,ASub1)`

`ASub1` is the result of normalizing abstract substitution `ASub0`. This is required in some domains, specially to perform the widening.

compute_lub/3: PREDICATE

Usage: `compute_lub(AbsInt,ListASub,LubASub)`

`LubASub` is the least upper bound of the abstract substitutions in list `ListASub`.

- *The following properties should hold at call time:*
  `AbsInt` is currently a term which is not a free variable. (`nonvar/1`)
  `ListASub` is currently a term which is not a free variable. (`nonvar/1`)
  `LubASub` is a free variable. (`var/1`)
  `AbsInt` is an atom. (`atm/1`)
  `ListASub` is a list. (`list/1`)
  `LubASub` is any term. (`term/1`)
- *The following properties should hold globally:*
  All the calls of the form `compute_lub(AbsInt,ListASub,LubASub)` do not fail. (`not_fails/1`)
  All calls of the form `compute_lub(AbsInt,ListASub,LubASub)` are deterministic. (`is_det/1`)

compute_glb/3: PREDICATE

`compute_glb(AbsInt,ListASub,GlbASub)`

`GlbASub` is the greatest lower bound of the abstract substitutions in list `ListASub`.

identical_proj/5: PREDICATE

Usage: `identical_proj(AbsInt,Sg,Proj,Sg1,Proj1)`

Abstract patterns `Sg`:`Proj` and `Sg1`:`Proj1` are equivalent in domain `AbsInt`. Note that `Proj` is assumed to be already sorted.

This predicate unifies `Sg` and `Sg1`.

- *The following properties should hold at call time:*
  `AbsInt` is currently a term which is not a free variable. (`nonvar/1`)
  `Sg` is currently a term which is not a free variable. (`nonvar/1`)
  `Proj` is currently a term which is not a free variable. (`nonvar/1`)
  `Sg1` is currently a term which is not a free variable. (`nonvar/1`)
  `Proj1` is currently a term which is not a free variable. (`nonvar/1`)
  `AbsInt` is an atom. (`atm/1`)

identical_proj_1/7: PREDICATE

Usage: `identical_proj_1(AbsInt,Sg,Proj,Sg1,Proj1,Prime1,Prime2)`

Abstract patterns `Sg`:`Proj` and `Sg1`:`Proj1` are equivalent in domain `AbsInt`. Note that `Proj` is assumed to be already sorted. It is different from `identical_proj/5` because it can be true although `Sg` and `Sg1` are not variant

- *The following properties should hold at call time:*

  `AbsInt` is currently a term which is not a free variable. (`nonvar/1`)
  `Sg` is currently a term which is not a free variable. (`nonvar/1`)
  `Proj` is currently a term which is not a free variable. (`nonvar/1`)
  `Sg1` is currently a term which is not a free variable. (`nonvar/1`)
  `Proj1` is currently a term which is not a free variable. (`nonvar/1`)
  `Prime1` is currently a term which is not a free variable. (`nonvar/1`)
  `Prime2` is currently a term which is not a free variable. (`nonvar/1`)

identical_abstract/3: PREDICATE

Usage: `identical_abstract(AbsInt,ASub1,ASub2)`

Succeeds if, in the particular abstract domain, the two abstract substitutions `ASub1` and `ASub2` are defined on the same variables and are equivalent.

- *The following properties should hold at call time:*

  `AbsInt` is currently a term which is not a free variable. (`nonvar/1`)
  `ASub1` is currently a term which is not a free variable. (`nonvar/1`)
  `ASub2` is currently a term which is not a free variable. (`nonvar/1`)
  `AbsInt` is an atom. (`atm/1`)

abs_sort/3: PREDICATE

Usage: `abs_sort(AbsInt,ASub_u,ASub)`

`ASub` is the result of sorting abstract substitution `ASub_u`.

- *The following properties should hold at call time:*

  `AbsInt` is currently a term which is not a free variable. (`nonvar/1`)
  `ASub_u` is currently a term which is not a free variable. (`nonvar/1`)
  `AbsInt` is an atom. (`atm/1`)

- *The following properties should hold globally:*

  All the calls of the form `abs_sort(AbsInt,ASub_u,ASub)` do not fail. (`not_fails/1`)
  All calls of the form `abs_sort(AbsInt,ASub_u,ASub)` are deterministic. (`is_det/1`)

extend/6: PREDICATE

Usage: `extend(AbsInt,Sg,Prime,Sv,Call,Succ)`

`Succ` is the extension the information given by `Prime` (success abstract substitution over the goal variables `Sv`) to the rest of the variables of the clause in which the goal occurs (those over which abstract substitution `Call` is defined on). I.e., it is like a conjunction of the information in `Prime` and `Call`, except that they are defined over different sets of variables, and that `Prime` is a successor substitution to `Call` in the execution of the program.

- *The following properties should hold at call time:*
  `AbsInt` is currently a term which is not a free variable. (`nonvar/1`)
  `Sg` is currently a term which is not a free variable. (`nonvar/1`)
  `Prime` is currently a term which is not a free variable. (`nonvar/1`)
  `Sv` is currently a term which is not a free variable. (`nonvar/1`)
  `Call` is currently a term which is not a free variable. (`nonvar/1`)
  `Succ` is a free variable. (`var/1`)
  `AbsInt` is an atom. (`atm/1`)
- *The following properties should hold globally:*
  All the calls of the form `extend(AbsInt,Sg,Prime,Sv,Call,Succ)` do not fail. (`not_fails/1`)
  All calls of the form `extend(AbsInt,Sg,Prime,Sv,Call,Succ)` are deterministic. (`is_det/1`)

less_or_equal_proj/5: PREDICATE

Usage: `less_or_equal_proj(AbsInt,Sg,Proj,Sg1,Proj1)`

Abstract pattern `Sg`:`Proj` is less general or equivalent to abstract pattern `Sg1`:`Proj1` in domain `AbsInt`.

- *The following properties should hold at call time:*
  `AbsInt` is currently a term which is not a free variable. (`nonvar/1`)
  `Sg` is currently a term which is not a free variable. (`nonvar/1`)
  `Proj` is currently a term which is not a free variable. (`nonvar/1`)
  `Sg1` is currently a term which is not a free variable. (`nonvar/1`)
  `Proj1` is currently a term which is not a free variable. (`nonvar/1`)
  `AbsInt` is an atom. (`atm/1`)

less_or_equal/3: PREDICATE

Usage: `less_or_equal(AbsInt,ASub0,ASub1)`

Succeeds if `ASub1` is more general or equivalent to `ASub0`.

- *The following properties should hold at call time:*
  `AbsInt` is currently a term which is not a free variable. (`nonvar/1`)
  `ASub0` is currently a term which is not a free variable. (`nonvar/1`)
  `ASub1` is currently a term which is not a free variable. (`nonvar/1`)
  `AbsInt` is an atom. (`atm/1`)

glb/4: PREDICATE

Usage: `glb(AbsInt,ASub0,ASub1,GlbASub)`

`GlbASub` is the greatest lower bound of abstract substitutions `ASub0` and `ASub1`.

- *The following properties should hold at call time:*
  `AbsInt` is currently a term which is not a free variable. (`nonvar/1`)
  `ASub0` is currently a term which is not a free variable. (`nonvar/1`)
  `ASub1` is currently a term which is not a free variable. (`nonvar/1`)
  `GlbASub` is a free variable. (`var/1`)
  `AbsInt` is an atom. (`atm/1`)

- *The following properties should hold globally:*
  All the calls of the form `glb(AbsInt,ASub0,ASub1,GlbASub)` do not fail. (`not_fails/1`)
  All calls of the form `glb(AbsInt,ASub0,ASub1,GlbASub)` are deterministic. (`is_det/1`)

eliminate_equivalent/3: PREDICATE

`eliminate_equivalent(AbsInt,TmpLSucc,LSucc)`

The list `LSucc` is reduced wrt the list `TmpLSucc` in that it does not contain abstract substitutions which are equivalent.

abs_subset/3: PREDICATE

`abs_subset(AbsInt,LASub1,LASub2)`

Succeeds if each abstract substitution in list `LASub1` is equivalent to some abstract substitution in list `LASub2`.

call_to_success_fact/10: PREDICATE

Usage: `call_to_success_fact(AbsInt,Sg,Hv,Head,K,Sv,Call,Proj,Prime,Succ)`

Specialized version of call_to_entry + entry_to_exit + exit_to_prime for a fact `Head`.

- *The following properties should hold at call time:*
  `AbsInt` is currently a term which is not a free variable. (`nonvar/1`)
  `Sg` is currently a term which is not a free variable. (`nonvar/1`)
  `Hv` is currently a term which is not a free variable. (`nonvar/1`)
  `Head` is currently a term which is not a free variable. (`nonvar/1`)
  `K` is currently a term which is not a free variable. (`nonvar/1`)
  `Sv` is currently a term which is not a free variable. (`nonvar/1`)
  `Call` is currently a term which is not a free variable. (`nonvar/1`)
  `Proj` is currently a term which is not a free variable. (`nonvar/1`)
  `Prime` is a free variable. (`var/1`)
  `Succ` is a free variable. (`var/1`)
  `AbsInt` is an atom. (`atm/1`)
- *The following properties should hold globally:*
  All the calls of the form `call_to_success_fact(AbsInt,Sg,Hv,Head,K,Sv,Call,Proj,Prime,Succ)` do not fail. (`not_fails/1`)

special_builtin/6: PREDICATE

Usage: `special_builtin(AbsInt,SgKey,Sg,Subgoal,Type,Condvars)`

Predicate `Sg` is considered a "builtin" of type `Type` in domain `AbsInt`. Types are domain dependent. Domains may have two different ways to treat these predicates: see `body_succ_builtin/9`.

- *The following properties should hold at call time:*

  `AbsInt` is currently a term which is not a free variable. (`nonvar/1`)

  `SgKey` is currently a term which is not a free variable. (`nonvar/1`)

  `Type` is a free variable. (`var/1`)

  `Condvars` is a free variable. (`var/1`)

  `AbsInt` is an atom. (`atm/1`)

body_succ_builtin/9: PREDICATE

Usage: `body_succ_builtin(AbsInt,Type,Sg,Vs,Sv,Hv,Call,Proj,Succ)`

Specialized version of call_to_entry + entry_to_exit + exit_to_prime + extend for predicate `Sg` considered a "builtin" of type `Type` in domain `AbsInt`. Whether a predicate is "builtin" in a domain is determined by `special_builtin/5`. There are two different ways to treat these predicates, depending on `Type`: `success_builtin` handles more usual types of "builtins", `call_to_success_builtin` handles particular predicates. The later is called when `Type` is of the form `special(SgKey)`.

- *The following properties should hold at call time:*

  `AbsInt` is currently a term which is not a free variable. (`nonvar/1`)

  `Type` is currently a term which is not a free variable. (`nonvar/1`)

  `Sg` is currently a term which is not a free variable. (`nonvar/1`)

  `Sv` is currently a term which is not a free variable. (`nonvar/1`)

  `Hv` is currently a term which is not a free variable. (`nonvar/1`)

  `Call` is currently a term which is not a free variable. (`nonvar/1`)

  `Proj` is currently a term which is not a free variable. (`nonvar/1`)

  `Succ` is a free variable. (`var/1`)

  `AbsInt` is an atom. (`atm/1`)
- *The following properties should hold globally:*

  All the calls of the form `body_succ_builtin(AbsInt,Type,Sg,Vs,Sv,Hv,Call,Proj,Succ)` do not fail. (`not_fails/1`)

success_builtin/7: PREDICATE

Usage: `success_builtin(AbsInt,Type,Sv,Condvars,HvFv_u,Call,Succ)`

`Succ` is the success substitution on domain `AbsInt` for a call `Call` to a goal of a "builtin" (domain dependent) type `Type` with variables `Sv`. `Condvars` can be used to transfer some information from `special_builtin/5`.

- *The following properties should hold at call time:*

  `AbsInt` is an atom. (`atm/1`)
- *The following properties should hold globally:*

  All the calls of the form `success_builtin(AbsInt,Type,Sv,Condvars,HvFv_u,Call,Succ)` do not fail. (`not_fails/1`)

obtain_info/5: PREDICATE

Usage: `obtain_info(AbsInt,Prop,Vars,ASub,Info)`

Obtains variables `Info` for which property `Prop` holds given abstract substitution `ASub` on variables `Vars` for domain `AbsInt`.

- *The following properties should hold at call time:*
  - `AbsInt` is currently a term which is not a free variable. (`nonvar/1`)
  - `Prop` is currently a term which is not a free variable. (`nonvar/1`)
  - `Vars` is currently a term which is not a free variable. (`nonvar/1`)
  - `ASub` is currently a term which is not a free variable. (`nonvar/1`)
  - `Info` is a free variable. (`var/1`)
  - `AbsInt` is an atom. (`atm/1`)

info_to_asub/7: PREDICATE

`info_to_asub(AbsInt,Kind,InputUser,Qv,ASub,Sg,MaybeCallASub)`

Obtains the abstract substitution `ASub` on variables `Qv` for domain `AbsInt` from the user supplied information `InputUser` refering to properties on `Qv`. It works by calling `input_interface/5` on each property of `InputUser` which is a native property, so that they are accumulated, and then calls `input_user_interface/6`.

full_info_to_asub/5: PREDICATE

`full_info_to_asub(AbsInt,InputUser,Qv,ASub,Sg)`

Same as `info_to_asub(AbsInt,InputUser,Qv,ASub)` except that it fails if some property in `InputUser` is not native or not relevant to the domain `AbsInt`.

asub_to_info/5: PREDICATE

`asub_to_info(AbsInt,ASub,Qv,OutputUser,CompProps)`

Transforms an abstract substitution `ASub` on variables `Qv` for a domain `AbsInt` to a list of state properties `OutputUser` and computation properties `CompProps`, such that properties are visible in the preprocessing unit. It fails if `ASub` represents bottom. It works by calling `asub_to_native/6`.

asub_to_native/6: PREDICATE

`asub_to_native(AbsInt,ASub,Qv,OutFlag,NativeStat,NativeComp)`

`NativeStat` and `NativeComp` are the list of native (state and computational, resp.) properties that are the concretization of abstract substitution `ASub` on variables `Qv` for domain `AbsInt`. These are later translated to the properties which are visible in the preprocessing unit.

concrete/4: PREDICATE

`concrete(AbsInt,Var,ASub,List)`

`List` are (all) the terms to which `Var` can be bound in the concretization of `ASub`, if they are a finite number of finite terms. Otherwise, the predicate fails.

unknown_call/5: PREDICATE

Usage: `unknown_call(AbsInt,Sg,Vars,Call,Succ)`

`Succ` is the result of adding to `Call` the "topmost" abstraction in domain `AbsInt` of the variables `Vars` involved in a literal `Sg` whose definition is not present in the preprocessing unit. I.e., it is like the conjunction of the information in `Call` with the top for a subset of its variables.

- *The following properties should hold at call time:*
  - `AbsInt` is currently a term which is not a free variable. (`nonvar/1`)
  - `Sg` is currently a term which is not a free variable. (`nonvar/1`)
  - `Vars` is currently a term which is not a free variable. (`nonvar/1`)
  - `Call` is currently a term which is not a free variable. (`nonvar/1`)
  - `Succ` is a free variable. (`var/1`)
  - `AbsInt` is an atom. (`atm/1`)
  - `Sg` is a term which represents a goal, i.e., an atom or a structure. (`cgoal/1`)
  - `Vars` is a list. (`list/1`)
- *The following properties should hold globally:*
  - All the calls of the form `unknown_call(AbsInt,Sg,Vars,Call,Succ)` do not fail. (`not_fails/1`)

unknown_entry/4: PREDICATE

Usage: `unknown_entry(AbsInt,Sg,Vars,Entry)`

`Entry` is the "topmost" abstraction in domain `AbsInt` of variables `Vars` corresponding to literal `Sg`.

- *The following properties should hold at call time:*
  - `AbsInt` is currently a term which is not a free variable. (`nonvar/1`)
  - `Sg` is currently a term which is not a free variable. (`nonvar/1`)
  - `Vars` is currently a term which is not a free variable. (`nonvar/1`)
  - `Entry` is a free variable. (`var/1`)
  - `AbsInt` is an atom. (`atm/1`)
  - `Vars` is a list. (`list/1`)
- *The following properties should hold globally:*
  - All the calls of the form `unknown_entry(AbsInt,Sg,Vars,Entry)` do not fail. (`not_fails/1`)

empty_entry/4: PREDICATE

Usage: `empty_entry(AbsInt,Sg,Vars,Entry)`

`Entry` is the "empty" abstraction in domain `AbsInt` of variables `Vars`. I.e., it is the abstraction of a substitution on `Vars` in which all variables are unbound: free and unaliased.

- *The following properties should hold at call time:*
  - `AbsInt` is currently a term which is not a free variable. (`nonvar/1`)
  - `Sg` is currently a term which is not a free variable. (`nonvar/1`)
  - `Vars` is currently a term which is not a free variable. (`nonvar/1`)
  - `Entry` is a free variable. (`var/1`)

`AbsInt` is an atom. (`atm/1`)

`Sg` is a term which represents a goal, i.e., an atom or a structure. (`cgoal/1`)

`Vars` is a list. (`list/1`)

`Entry` is any term. (`term/1`)

– *The following properties should hold globally:*

All the calls of the form `empty_entry(AbsInt,Sg,Vars,Entry)` do not fail. (`not_fails/1`)

part_conc/5: PREDICATE

`part_conc(AbsInt,Sg,Subs,NSg,NSubs)`

This operation returns in `NSg` an instance of `Sg` in which the deterministic structure information available in `Subs` is materialized. The substitution `NSubs` refers to the variables in `NSg`.

multi_part_conc/4: PREDICATE

`multi_part_conc(AbsInt,Sg,Subs,List)`

Similar to part_conc but it gives instantiations of goals even in the case types are not deterministic, it generates a `List` of pairs of goals and substitutions. It stops unfolding types as soon as they are recursive.

collect_types_in_abs/4: PREDICATE

Usage: `collect_types_in_abs(ASub,AbsInt,Types,Tail)`

Collects the type symbols occurring in `ASub` of domain `AbsInt` in a difference list `Types`-`Tail`.

– *The following properties should hold at call time:*

`ASub` is currently a term which is not a free variable. (`nonvar/1`)

`AbsInt` is currently a term which is not a free variable. (`nonvar/1`)

– *The following properties should hold globally:*

All calls of the form `collect_types_in_abs(ASub,AbsInt,Types,Tail)` are deterministic. (`is_det/1`)

All the calls of the form `collect_types_in_abs(ASub,AbsInt,Types,Tail)` do not fail. (`not_fails/1`)

rename_types_in_abs/4: PREDICATE

Usage: `rename_types_in_abs(ASub0,AbsInt,Dict,ASub1)`

Renames the type symbols occurring in `ASub0` of domain `AbsInt` for the corresponding symbols as in (avl-tree) `Dict` yielding `ASub1`.

– *The following properties should hold at call time:*

`ASub0` is currently a term which is not a free variable. (`nonvar/1`)

`AbsInt` is currently a term which is not a free variable. (`nonvar/1`)

`Dict` is currently a term which is not a free variable. (`nonvar/1`)

- *The following properties should hold globally:*
  All calls of the form `rename_types_in_abs(ASub0,AbsInt,Dict,ASub1)` are deterministic. (`is_det/1`)
  All the calls of the form `rename_types_in_abs(ASub0,AbsInt,Dict,ASub1)` do not fail. (`not_fails/1`)

dom_statistics/2: PREDICATE

`dom_statistics(AbsInt,Info)`

Obtains in list `Info` statistics about the results of the abstract interpreter `AbsInt`.

abstract_instance/5: PREDICATE

Usage: `abstract_instance(AbsInt,Sg1,Proj1,Sg2,Proj2)`

The pair <Sg1,Proj1> is an abstract instance of the pair <Sg2,Proj2>, i.e., the concretization of <Sg1,Proj1> is included in the concretization of <Sg2,Proj2>.

contains_parameters/2: PREDICATE

`contains_parameters(AbsInt,Subst)`

True if an abstract substitution `Subst` contains type parameters

absub_eliminate_equivalent/3: PREDICATE

No further documentation available for this predicate.

absub_fixpoint_covered/3: PREDICATE

No further documentation available for this predicate.

body_builtin/9: PREDICATE

No further documentation available for this predicate.

undef_call_to_success_builtin/2: PREDICATE

No further documentation available for this predicate.

68.4 Documentation on multifiles

aidomain/1: PREDICATE

`aidomain(AbsInt)`

Declares that `AbsInt` identifies an abstract domain. The predicate is *multifile*.

aidom.init_abstract_domain/2: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.amgu/5: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.augment_asub/4: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.augment_two_asub/4: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.call_to_entry/10: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.exit_to_prime/8: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.project/6: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.widencall/4: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.needs/2: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.widen/4: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.compute_lub/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.compute_clauses_lub/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.compute_clauses_glb/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.fixpoint_covered/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.fixpoint_covered_gfp/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.identical_abstract/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.abs_sort/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.extend/6: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.less_or_equal/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.glb/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.eliminate_equivalent/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.abs_subset/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.call_to_success_fact/10: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.special_builtin/6: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.combined_special_builtin0/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.body_succ_builtin/9: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.split_combined_domain/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.success_builtin/7: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.call_to_success_builtin/7: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.obtain_info/5: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.input_interface/5: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.input_user_interface/6: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.asub_to_native/6: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.concrete/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

aidom.unknown_call/5: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.unknown_entry/4: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.empty_entry/4: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.part_conc/5: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.multi_part_conc/4: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.collect_auxinfo_asub/4: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.rename_auxinfo_asub/4: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.dom_statistics/2: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidom.contains_parameters/2: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

68.5 Documentation on internals

call_to_success_builtin/7: PREDICATE

Usage: `call_to_success_builtin(AbsInt,Type,Sg,Sv,Call,Proj,Succ)`

`Succ` is the success substitution on domain `AbsInt` for a call `Call` to a goal `Sg` with variables `Sv` considered of a `"builtin"` (domain dependent) type `Type`. `Proj` is `Call` projected on `Sv`.

− *The following properties should hold globally:*

All the calls of the form `call_to_success_builtin(AbsInt,Type,Sg,Sv,Call,Proj,Succ)` do not fail. (`not_fails/1`)

input_interface/5: PREDICATE

`input_interface(AbsInt,Prop,Kind,Struc0,Struc1)`

`Prop` is a native property that is relevant to domain `AbsInt` (i.e., the domain knows how to fully –+`Kind`=perfect– or approximately –-`Kind`=approx– abstract it) and `Struct1` is a (domain defined) structure resulting of adding the (domain dependent) information conveyed by `Prop` to structure `Struct0`. This way, the properties relevant to a domain are being accumulated.

input_user_interface/6: PREDICATE

`input_user_interface(AbsInt,Struct,Qv,ASub,Sg,MaybeCallASub)`

`ASub` is the abstraction in `AbsInt` of the information collected in `Struct` (a domain defined structure) on variables `Qv`.

68.6 Documentation on imports

This module has the following direct dependencies:

- *Application modules:*
  `fixpo_ops`, `low_level_props`, `preprocess_flags`, `pd`, `pdb`, `fr_top`, `frdef`, `lsign`, `lsigndiff`, `gr`, `def`, `sharing`, `sharefree`, `sharefree_non_var`, `shfret`, `shareson`, `shfrson`, `sondergaard`, `sharing_amgu`, `sharefree_amgu`, `shfrlin_amgu`, `sharing_clique`, `sharing_clique_1`, `sharefree_clique`, `sharing_clique_def`, `sharefree_clique_def`, `aeq`, `depthk`, `top_path_sharing`, `termsd`, `ptypes`, `eterms`, `etermsvar`, `svterms`, `deftypes`, `nonrel_intervals`, `nfdet`, `nfplai`, `detplai`.
- *System library modules:*
  `native_props`, `p_unit`, `terms_check`, `terms_vars`, `native_props_rtc`, `sets`, `sort`, `lists`, `messages`.
- *Internal (engine) modules:*
  `term_basic`, `arithmetic`, `atomic_basic`, `basiccontrol`, `exceptions`, `term_compare`, `term_typing`, `debugger_support`, `basic_props`, `hiord_rt`.
- *Packages:*
  `prelude`, `initial`, `condcomp`, `assertions`, `assertions/assertions_basic`, `regtypes`, `isomodes`, `nativeprops`, `ciaopp(ciaopp_options)`, `aidomain`.

69 PART IV - CiaoPP Internals

CiaoPP is provided as a module that can be either loaded in the shell or imported into a user program module. There are predicates that activate each option, a predicate to load the file to be processed, and predicates to perform the action of each task of CiaoPP.

This document describes the structure of the code of CiaoPP, its components (each performing a task), and their functionality, from the point of view of the implementor, not of the user.

69.1 How to use CiaoPP in your own preprocessor

If you implement a program for preprocessing Ciao programs, you can take advantage of the facilities provided by CiaoPP. The basic facilities are described in the following three chapters. In particular, the chapter on module `driver` explains how to integrate your tool and CiaoPP to make them work together.

One particularly useful feature of CiaoPP is that it provides for the correct reading of an input program to be preprocessed (this is more tricky than it seems, because of all of the language features that the program might be using). Once a program is read in by CiaoPP, your tool can process it as plain Prolog. The chapter on module `p_unit` explains how a program is (correctly) read in, and provides an interface so that you can retrieve information about that program.

Depending on the kind of preprocessing you want to do, you may want to use other parts of CiaoPP. If your program is doing program transformations, then the chapters on `program_keys` and `tr_syntax` might be helpful. If you want to use analysis information to help the transformations, then the chapter on `infer` is the one to read. If your program is a program analyzer based on abstract interpretation, then have a look at the chapters on `plai` and `domains`. Program analyses not based on abstract interpretation (and other possible tools) may be a bit harder to incorporate: you might need to develop first your skills with the CiaoPP code...

69.2 Other information

The work of CiaoPP is focused on one-module-at-a-time. The file being preprocessed at a given time is called the *current module* . If the file is of the user module, it is considered as a module which exports all its predicates.

The preprocessing unit is the code of the current module together with the set of assertions / properties which allows a correct and complete interpretation of the assertions present for the current module. Assertions and properties are usually cached in auxiliary files, which are called the .asr files.

69.3 Basic structure of the code

The code of CiaoPP is divided into *components*, each of which provides a particular functionality. The top-level organization of the main components is as follows:

Each component is addressed by a particular qualifier, which makes reference to modules independent of the particular library paths. These qualifiers are defined in file `paths.pl`. The qualifier for the top-level components is `ciaopp`.

The component `p_unit` is addressed by qualifier `program` and is in charge of processing the .asr files, gathering together the code for the preprocessing unit of the current module.

The component `tr_syntax` is addressed by qualifier `syntax` and implements simple source code syntactic transformations (like removing disjunctions and the like).

The component `infer` is addressed by qualifier `infer` and is in charge of serving all known semantic information on the current module, from any analysis and/or assertion.

The component `plai` is addressed by qualifier `plai` and implements abstract interpretation based analysis.

The component `infernf` is addressed by qualifier `infernf` and implements the non-failure analysis.

The component `infercost` is addressed by qualifier `infercost` and implements the cost analysis.

70 Frontend driver (monolithic)

This module provides the main entry points for loading source programs (in a suitable form for performing analysis and transformations) and print them back as source.

70.1 Adding new frontend

(to be written)

70.2 Usage and interface

- **Library usage:**
  `:- use_module(ciaopp(frontend_driver)).` This module is loaded by default in the CiaoPP toplevel and reexported from the `ciaopp` module.
- **Exports:**
  - *Predicates:*
    `supported_language/1`, `language_extension/2`, `language_output_extension/2`, `module/1`, `module/2`, `entry_assertion/3`, `assrt_used_as_entry/2`, `is_library/1`, `cleanup_history/0`, `push_history/1`, `pop_history/1`, `get_output_path/2`, `output/0`, `output/1`, `output/2`, `output_ext/1`, `output_by_ext/2`, `write_one_type/2`, `ensure_libcache_loaded/0`, `gen_and_load_libcache/0`.
  - *Multifiles:*
    `hook_compact_global_prop/2`, `hook_compact_calls_prop/2`, `hook_native_prop/2`, `hook_pp_info_clause/3`, `hook_pp_info_lit/4`, `hook_legal_regtype/1`, `hook_insert_regtype/2`, `hook_post_init_regtypes/0`, `hook_cleanup_lib_regtypes/0`, `hook_restore_lib_regtypes/1`, `hook_save_lib_regtypes/1`, `file_search_path/2`, `library_directory/1`.

70.3 Documentation on exports

supported_language/1: PREDICATE

`supported_language(L)`

Indicates that a translation is available from language `L` to Ciao Prolog language.

language_extension/2: PREDICATE

`language_extension(L,E)`

`E` is an extension (including the dot) for files that must be translated from language `L` to Ciao.

language_output_extension/2: PREDICATE

`language_output_extension(L,E)`

`E` is the extension of the file produced as output by the Ciao printer for language `L`.

module/1: PREDICATE

Usage 1: `module(FileName)`

Reads the code of `FileName` and its preprocessing unit, and sets it as the current module.

– *The following properties should hold at call time:*

`FileName` is currently a term which is not a free variable. (`nonvar/1`)

`FileName` is a source name. (`sourcename/1`)

Usage 2: `module(FileNameList)`

Reads the code of the list of file names `FileNameList` (and their preprocessing units), and sets them as the current modules.

– *The following properties should hold at call time:*

`FileNameList` is currently a term which is not a free variable. (`nonvar/1`)

`FileNameList` is a list of `atm`s. (`list/2`)

module/2: PREDICATE

Usage: `module(Ms,Info)`

Same as `module/1` but it also prints the time required to read the module and returns extra information (including the time) on its second argument.

– *The following properties should hold at call time:*

`Ms` is currently a term which is not a free variable. (`nonvar/1`)

`Info` is a free variable. (`var/1`)

entry_assertion/3: PREDICATE

Usage: `entry_assertion(Goal,Call,Name)`

There is an entry assertion for `Goal` with call pattern `Call`, uniquely identifiable by `Name`.

– *The following properties should hold upon exit:*

`Goal` is a term which represents a goal, i.e., an atom or a structure. (`cgoal/1`)

assrt_used_as_entry/2: PREDICATE

Usage: `assrt_used_as_entry(Sg,AssrtType)`

Assertion with head `Sg` and assertion type `AssrtType` is used as an analysis entry (depending on values of `entry_points_auto` and `entry_calls_scope`).

– *The following properties should hold at call time:*

`Sg` is a term which represents a goal, i.e., an atom or a structure. (`cgoal/1`)

`AssrtTYpe` is an atom. (`atm/1`)

is_library/1: PREDICATE

Usage: `is_library(Base)`

– *The following properties should hold at call time:*

`Base` is currently a term which is not a free variable. (`nonvar/1`)

cleanup_history/0: PREDICATE

Usage:

Cleanup history items.

push_history/1: PREDICATE

Usage: `push_history(X)`

Push history item `X`.

- *The following properties should hold at call time:*

  `X` is currently instantiated to an atom. (`atom/1`)

pop_history/1: PREDICATE

Usage: `pop_history(X)`

Pop history item `X`.

- *The following properties should hold at call time:*

  `X` is currently instantiated to an atom. (`atom/1`)

get_output_path/2: PREDICATE

Usage: `get_output_path(UseHistory,Path)`

Default output file name based. Encode analysis/transformation history if `UseHistory` is `yes`.

- *The following properties should hold upon exit:*

  `UseHistory` is an atom. (`atm/1`)

  `Path` is an atom. (`atm/1`)

output/0: PREDICATE

Usage:

Outputs the current module preprocessing state with the default output options. Equivalent to `output(File,[])`.

output/1: PREDICATE

Usage: `output(File)`

Outputs to `File` the current module preprocessing state with the default output options. If unbound, `File` is unified with a default name (encoding the preprocessing steps). Equivalent to `output(File,[])`.

output/2: PREDICATE

Usage: `output(File,Opts)`

Outputs the current module preprocessing state to a file `File`. If unbound, `File` is unified with a default name (encoding the preprocessing steps). The output format (which should be valid for the loaded program) is guessed from the file extension. `Opts` is a list of options for this predicate (see `output_option/1`)

- *The following properties should hold at call time:*

  `Opts` is currently a term which is not a free variable. (`nonvar/1`)

output_ext/1: PREDICATE

Usage: `output_ext(Ext)`

Extension `Ext` is a supported output extension.

- *The following properties should hold at call time:*

  `Ext` is an atom. (`atm/1`)

output_by_ext/2: PREDICATE

Usage: `output_by_ext(Ext,File)`

Produce output `File` for the given extension `Ext`.

- *The following properties should hold at call time:*

  `Ext` is an atom. (`atm/1`)

  `File` is a source name. (`sourcename/1`)

write_one_type/2: PREDICATE

No further documentation available for this predicate.

ensure_libcache_loaded/0: PREDICATE

Usage:

Ensure that the libcache is loaded (if `use_libcache` is enabled and the cache is generated) (see `p_unit:load_libcache/1`).

gen_and_load_libcache/0: PREDICATE

Usage:

Generate and load the libcache (see `p_unit:gen_libcache/1`).

get_data/2: (UNDOC_REEXPORT)

Imported from `p_dump` (see the corresponding documentation for details).

clean_ciaopp_db/1: (UNDOC_REEXPORT)

Imported from `p_dump` (see the corresponding documentation for details).

clean_all_ciaopp_db/0: (UNDOC_REEXPORT)

Imported from `p_dump` (see the corresponding documentation for details).

dump/2: (UNDOC_REEXPORT)

Imported from `p_dump` (see the corresponding documentation for details).

dump_dir/1: (UNDOC_REEXPORT)
Imported from `p_dump` (see the corresponding documentation for details).

add_to_db/1: (UNDOC_REEXPORT)
Imported from `p_dump` (see the corresponding documentation for details).

show_dump/1: (UNDOC_REEXPORT)
Imported from `p_dump` (see the corresponding documentation for details).

restore/2: (UNDOC_REEXPORT)
Imported from `p_dump` (see the corresponding documentation for details).

restore/1: (UNDOC_REEXPORT)
Imported from `p_dump` (see the corresponding documentation for details).

dump/1: (UNDOC_REEXPORT)
Imported from `p_dump` (see the corresponding documentation for details).

70.4 Documentation on multifiles

hook_compact_global_prop/2: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

hook_compact_calls_prop/2: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

hook_native_prop/2: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

hook_pp_info_clause/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

hook_pp_info_lit/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

hook_legal_regtype/1: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

hook_insert_regtype/2: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

hook_post_init_regtypes/0: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

hook_cleanup_lib_regtypes/0: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

hook_restore_lib_regtypes/1: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

hook_save_lib_regtypes/1: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

file_search_path/2: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

The predicate is of type *dynamic*.

library_directory/1: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

The predicate is of type *dynamic*.

70.5 Documentation on imports

This module has the following direct dependencies:

- *Application modules:*

  `ciaopp_log`, `analysis_stats`, `preprocess_flags`, `incanal`, `analyze_driver`, `intermod_ops`, `transform_driver`, `raw_printer`, `p_dump`, `prepare_ai_output`, `infer_db`, `infer_dom`.
- *System library modules:*

  `native_props`, `datafacts_rt`, `p_unit`, `p_printer`, `aggregates`, `unexpand`, `p_unit_db`, `assrt_lib`, `native`, `typeslib`, `system`, `pathnames`, `lists`, `pretty_print`, `messages`, `terms`, `format`, `sort`, `terms_vars`, `formulae`.

- *Internal (engine) modules:*
  term_basic, arithmetic, atomic_basic, basiccontrol, exceptions, term_compare, term_typing, debugger_support, basic_props, hiord_rt, internals, runtime_control, stream_basic, io_basic.
- *Packages:*
  prelude, initial, condcomp, assertions, assertions/assertions_basic, basicmodes, regtypes, nativeprops, hiord, datafacts, ciaopp(ciaopp_options), compiler/p_unit/p_unit_argnames, argnames.

71 Advanced output of preprocessor

Author(s): Isabel Garcia-Contreras.

Raw printer outputs the analysis results of ciaopp without preprocessing any information, as it is represented internally in CiaoPP

71.1 How to use it

```
?- set_pp_flag(output_lang, raw).

yes
% make analysis
?- module(foo).
{Loading current module from foo.pl
{loaded in 390.268 msec.}
}

yes
?- analyze(eterms).
{Analyzing foo.pl
{preprocessed for the plai fixpoint in 1.736 msec.}
{analyzed by plai using eterms with local-control off in 5.106 msec.}
}

?- output.
{written file .../foo_eterms_co.pl}

yes
?- % open file or C-c C-v in emacs
```

The output will be generated as a module importing package raw. The idea is to be able to load this analysis (containing the translations to avoid recomputation), but it not implemented yet.

71.2 Usage and interface

- **Library usage:**
  `:- use_module(ciaopp(raw_printer)).`
- **Exports:**
  - *Predicates:*
    `raw_output/1`, `show_trans_clauses/0`, `show_analysis/0`, `show_registry_info/0`, `show_global_answer_table/1`, `show_change_list/0`.

71.3 Documentation on exports

raw_output/1: PREDICATE

Usage: `raw_output(File)`

Shows the current state of the plai databases in file `File`, including:

- Transformed clauses.
- Inferred assertions (completes).
- Information at program point (memo_tables).
- State of the types library.
- *The following properties should hold at call time:*
  `File` is an open stream. (`stream/1`)

show_trans_clauses/0: PREDICATE

Usage:

Shows the code transformed to be used by the fixpoints (given by `trans_clause/3`).

show_analysis/0: PREDICATE

Usage:

Shows all the information inferred by ciaopp.

show_registry_info/0: PREDICATE

Shows information about the modular analysis registries.

show_global_answer_table/1: PREDICATE

Usage: `show_global_answer_table(AbsInt)`

Shows de global answer table for modular analysis with domain `AbsInt`

show_change_list/0: PREDICATE

Shows the list of changes that need computation in the fixpoint dd.

71.4 Documentation on imports

This module has the following direct dependencies:

- *Application modules:*
  `preprocess_flags`, `transform`, `plai_db`, `intermod_db`, `plai`, `fixpo_dd`.
- *System library modules:*
  `datafacts_rt`, `write`, `format`, `lists`, `aggregates`, `assrt_write`, `assrt_lib`, `typeslib`, `program_keys`, `p_unit`, `p_unit_db`.
- *Internal (engine) modules:*
  `term_basic`, `arithmetic`, `atomic_basic`, `basiccontrol`, `exceptions`, `term_compare`, `term_typing`, `debugger_support`, `basic_props`, `hiord_rt`, `stream_basic`, `io_basic`.
- *Packages:*
  `prelude`, `initial`, `condcomp`, `assertions`, `assertions/assertions_basic`, `isomodes`, `datafacts`.

72 Analyze driver (monolithic)

This module provides the main entry points for performing analysis and assertion checking. It requires loading the program before (e.g., with `frontend_driver`).

72.1 Adding new analyses

To include a new analysis, add a clause for `analyze/2` (and for `analysis/1`).

As an alternative, you can add clauses for the multifile predicates `analysis/4` and `analysis/1`, directly in your own sources.

See the file `examples/Extending/myanalyzer.pl` in the source directory for an example of this.

72.2 Usage and interface

- **Library usage:**
  `:- use_module(ciaopp(analyze_driver)).` This module is loaded by default in the CiaoPP toplevel and reexported from the `ciaopp` module.
- **Exports:**
  - *Predicates:*
    `assert_domain/1`, `last_domain_used/1`, `analyze/1`, `analyze/2`, `analyze1/2`, `acheck_summary/1`, `acheck_summary/2`, `get_assert_count/1`, `acheck/0`, `acheck/1`, `acheck/2`, `clean_analysis_info/0`, `clean_analysis_info0/0`, `cleanup_for_codegen/0`.
  - *Regular Types:*
    `ctcheck_sum/1`.
  - *Multifiles:*
    `lazy_analysis/1`, `analysis_module/2`, `loaded_analysis/1`, `analysis/4`, `analysis/1`, `aidomain/1`.

72.3 Documentation on exports

assert_domain/1: PREDICATE

No further documentation available for this predicate.

last_domain_used/1: PREDICATE

No further documentation available for this predicate.

analyze/1: PREDICATE

Usage: `analyze(Analysis)`

Analyzes the current module with `Analysis`. If the intermod flag is not off, this predicate may call `module/1`.

- *The following properties should hold at call time:*
  `Analysis` is currently a term which is not a free variable. (`nonvar/1`)
  `Analysis` is a valid analysis identifier. (`analysis/1`)
- *The following properties should hold globally:*
  All the calls of the form `analyze(Analysis)` do not fail. (`not_fails/1`)
  A call to `analyze(Analysis)` does not leave new choicepoints. (`no_choicepoints/1`)

analyze/2: PREDICATE

Usage: `analyze(Analysis,Info)`

Same as analyze(`Analysis`) but in `Info` returns statistics (time and memory).

- *The following properties should hold at call time:*
  `Analysis` is currently a term which is not a free variable. (`nonvar/1`)
  `Info` is a free variable. (`var/1`)

analyze1/2: PREDICATE

No further documentation available for this predicate.

ctcheck_sum/1: REGTYPE

A regular type, defined as follows:

```
ctcheck_sum(ok).
ctcheck_sum(warning).
ctcheck_sum(error).
```

acheck_summary/1: PREDICATE

Usage: `acheck_summary(S)`

Checks assertions w.r.t. analysis information. Upon success `S` is bound to: ok (the compile-time checking process has generated no error nor warning), warning (compile-time checking has not generated any error, but there has been at least one warning) or error (at least one error has been produced).

- *The following properties should hold at call time:*
  `S` is a free variable. (`var/1`)
- *The following properties should hold upon exit:*
  `ctcheck_sum(S)` (`ctcheck_sum/1`)

acheck_summary/2: PREDICATE

Usage: `acheck_summary(Info,S)`

Same as acheck_summary(`S`) but in `Info` returns assertion checking statistics.

- *The following properties should hold at call time:*
  `Info` is a free variable. (`var/1`)
  `S` is a free variable. (`var/1`)

get_assert_count/1: PREDICATE

No further documentation available for this predicate.

acheck/0: PREDICATE

Usage:

Checks assertions w.r.t. analysis information, obtains from `domain/1` which analyses were run.

acheck/1: PREDICATE

Usage: `acheck(AbsInt)`

Checks assertions using the analysis information of `AbsInt`. The analysis must be present in CiaoPP (via analysis or restore dump).

acheck/2: PREDICATE

Usage 1: `acheck(AbsInt,ModList)`

- *The following properties should hold at call time:*
  - `AbsInt` is currently a term which is not a free variable. (`nonvar/1`)
  - `ModList` is currently a term which is not a free variable. (`nonvar/1`)
  - `AbsInt` is a list of `atm`s. (`list/2`)
  - `ModList` is a list of `atm`s. (`list/2`)

Usage 2: `acheck(AbsInt,MaybeModList)`

If `MaybeModList` is the atom `all`, all modules in the current punit are considered. If it is a list of **module names**, only the assertions or predicates in those modules are checked.

- *The following properties should hold at call time:*
  - `AbsInt` is currently a term which is not a free variable. (`nonvar/1`)
  - `MaybeModList` is currently a term which is not a free variable. (`nonvar/1`)
  - `AbsInt` is a list of `atm`s. (`list/2`)
  - `MaybeModList` is an atom. (`atm/1`)

clean_analysis_info/0: PREDICATE

Usage:

Cleans all analysis info but keep the program as wether it would be just read.

clean_analysis_info0/0: PREDICATE

No further documentation available for this predicate.

cleanup_for_codegen/0: PREDICATE

No further documentation available for this predicate.

trace_fixp/1: PREDICATE

Usage 1: `trace_fixp(X)`

Mode for setting the current flag to a single value.

- *The following properties should hold at call time:*
  `trace_option(X)` (`trace_option/1`)
- *The following properties should hold globally:*
  `X` is not further instantiated. (`not_further_inst/2`)

Usage 2: `trace_fixp(X)`

Mode for setting the current flag to several values.

- *The following properties should hold at call time:*
  `X` is a list of `trace_option`s. (`list/2`)
- *The following properties should hold globally:*
  `X` is not further instantiated. (`not_further_inst/2`)

72.4 Documentation on multifiles

lazy_analysis/1: PREDICATE

Specifies an analysis that can be lazy loaded The predicate is *multifile*.

analysis_module/2: PREDICATE

Given an analysis, returns the module that contains the implementation (or entry point) of a lazy loadable analysis. Such module must implement the analysis/4 multifile clause. The predicate is *multifile*.

loaded_analysis/1: PREDICATE

Helps to determine if an analysis was already loaded. Must be defined in the same module specified by analysis_module/2. The predicate is *multifile*.

analysis/4: PREDICATE

Usage: `analysis(Analysis,Clauses,Dictionaries,Info)`

Performs `Analysis` on program `Clauses`.

- *The following properties should hold at call time:*
  `Analysis` is a valid analysis identifier. (`analysis/1`)

The predicate is *multifile*.

analysis/1: PROPERTY

Usage: `analysis(Analysis)`

`Analysis` is a valid analysis identifier.

The predicate is *multifile*.

aidomain/1: PREDICATE

See the chapter on `domains`. The predicate is *multifile*.

72.5 Documentation on imports

This module has the following direct dependencies:

- *Application modules:*
  ciaopp_log, analysis_stats, preprocess_flags, infer_db, frontend_driver, trace_fixp, incanal, intermod, plai, vartypes, inferseff, infer_dom, ctchecks_pred, ctchecks_pp, ctchecks_common, preproc_errors, intermod_ops.
- *System library modules:*
  native_props, datafacts_rt, compiler, p_unit, p_unit_db, typeslib, counters, aggregates.
- *Internal (engine) modules:*
  term_basic, arithmetic, atomic_basic, basiccontrol, exceptions, term_compare, term_typing, debugger_support, basic_props, hiord_rt, messages_basic.
- *Packages:*
  prelude, initial, condcomp, assertions, assertions/assertions_basic, basicmodes, regtypes, nativeprops, datafacts, ciaopp(ciaopp_options), ciaopp(analysis_register).

73 Transform driver (monolithic)

This module provides the main entry points for for performing program transformations. It requires loading the program before (e.g., with `frontend_driver`).

73.1 Adding new transformation

To include a new program transformation, add a clause for `transform/2` (and for `transformation/1`).

As an alternative, you can add clauses for the multifile predicates `transformation/4` and `transformation/1`, directly in your own sources.

See the file `examples/Extending/myspecializer.pl` in the source directory for an example of this.

73.2 Usage and interface

- **Library usage:**
  `:- use_module(ciaopp(transform_driver))`. This module is loaded by default in the CiaoPP toplevel and reexported from the `ciaopp` module.
- **Exports:**
  - *Predicates:*
    `transform/1`, `transform/2`.
  - *Multifiles:*
    `transformation/4`, `transformation/1`.

73.3 Documentation on exports

transform/1: PREDICATE

Usage 1: `transform(Trans)`

Returns on backtracking all available program transformation identifiers.

- *The following properties should hold at call time:*
  `Trans` is a free variable. (`var/1`)
- *The following properties should hold upon exit:*
  `Trans` is a valid transformation identifier. (`transformation/1`)

Usage 2: `transform(Trans)`

Performs transformation `Trans` on the current module.

- *The following properties should hold at call time:*
  `Trans` is currently a term which is not a free variable. (`nonvar/1`)
  `Trans` is a valid transformation identifier. (`transformation/1`)

transform/2: PREDICATE

Usage: `transform(Trans,Info)`

Same as transform(`Trans`) but returns information that can be used to check the results of the transformation.

- *The following properties should hold at call time:*
  Trans is currently a term which is not a free variable. (nonvar/1)
  Info is a free variable. (var/1)

73.4 Documentation on multifiles

transformation/4: PREDICATE

Usage: transformation(Transformation,Clauses,Dictionaries,Info)

Performs Transformation on program Clauses.

- *The following properties should hold at call time:*
  Transformation is a valid transformation identifier. (transformation/1)

The predicate is *multifile*.

transformation/1: PROPERTY

Usage: transformation(Transformation)

Transformation is a valid transformation identifier.

The predicate is *multifile*.

73.5 Documentation on imports

This module has the following direct dependencies:

- *Application modules:*
  preprocess_flags, ciaopp_log, frontend_driver, normalize_args, re_analysis, spec, spec_multiple, codegen, slicing, arg_filtering, unfold_non_rec, analyze_driver.
- *System library modules:*
  native_props, datafacts_rt, p_unit_db, p_unit.
- *Internal (engine) modules:*
  term_basic, arithmetic, atomic_basic, basiccontrol, exceptions, term_compare, term_typing, debugger_support, basic_props, hiord_rt, messages_basic.
- *Packages:*
  prelude, initial, condcomp, assertions, assertions/assertions_basic, basicmodes, regtypes, nativeprops, datafacts, ciaopp(ciaopp_options).

74 Modular driver

Author(s): The Ciao Development Team, Isabel Garcia-Contreras.

Stability: [**devel**] Currently the subject of active development and/or research. Functionality may be limited and API and/or functionality may change without warning or deprecation period. Not recommended yet for use in production.

This module provides intermodular analysis to CiaoPP (high-level).

Global compilation options of intermodular analysis are available in `intermod_options`. Edit this file to activate tracing or run-time checks.

74.1 Usage and interface

- **Library usage:**

  `:- use_module(ciaopp(plai/intermod)).`
- **Exports:**
  - *Predicates:*

    `cleanup_intermod/0`, `intermod_analyze/2`, `intermod_analyze/3`, `intermod_ctcheck/2`, `intermod_transform/3`, `intermod_simp_libs/2`, `intermod_simp_libs/3`, `inductive_ctcheck/2`, `inductive_ctcheck/4`, `inductive_ctcheck_summary/3`, `intermod_ctcheck_opt/2`, `intermod_ctcheck_opt/3`.
  - *Properties:*

    `valid_mod_analysis/1`.
  - *Multifiles:*

    `dump_flags_list/2`, `aidomain/1`, `transformation/1`, `library_directory/1`, `file_search_path/2`.

74.2 Documentation on exports

cleanup_intermod/0: PREDICATE

Usage:

Cleans up the internal database of the intermodular analysis global level.

valid_mod_analysis/1: PROPERTY

Usage 1: `valid_mod_analysis(Domain)`

Succeeds if `Domain` is a valid analysis domain for modular analysis

Usage 2: `valid_mod_analysis(DomainList)`

Succeeds if the domains in `DomainList` are valid analysis domains for modular analysis

intermod_analyze/2: PREDICATE

No further documentation available for this predicate.

intermod_analyze/3: PREDICATE

No further documentation available for this predicate.

intermod_ctcheck/2: PREDICATE

Usage: `intermod_ctcheck(AbsInts,Modules)`

Assuming that modular fixpoint already reached. Only supporting ctchecks of pred assertions (not program point)

- *The following properties should hold at call time:*
  `AbsInts` is currently a term which is not a free variable. (`nonvar/1`)
  `Modules` is currently a term which is not a free variable. (`nonvar/1`)
  `AbsInts` is a list of `atm`s. (`list/2`)
  `Modules` is a list of `atm`s. (`list/2`)
- *The following properties should hold globally:*
  All the calls of the form `intermod_ctcheck(AbsInts,Modules)` do not fail. (`not_fails/1`)

intermod_transform/3: PREDICATE

Usage: `intermod_transform(AbsInt,Trans,TopLevel)`

Performs transformation `Trans` of the program unit which has `TopLevel` as top-level module, using `AbsInt` to get information about the program.

- *The following properties should hold at call time:*
  `AbsInt` is currently a term which is not a free variable. (`nonvar/1`)
  `Trans` is currently a term which is not a free variable. (`nonvar/1`)
  `TopLevel` is currently a term which is not a free variable. (`nonvar/1`)

intermod_simp_libs/2: PREDICATE

Usage: `intermod_simp_libs(TopLevel,Dir)`

Generates a copy of the program represented by `TopLevel` and the libraries used (except those in `engine`) in `Dir`, and removes dead-code from both user modules and libraries.

- *The following properties should hold at call time:*
  `TopLevel` is currently a term which is not a free variable. (`nonvar/1`)
  `Dir` is currently a term which is not a free variable. (`nonvar/1`)

intermod_simp_libs/3: PREDICATE

No further documentation available for this predicate.

inductive_ctcheck/2: PREDICATE

No further documentation available for this predicate.

inductive_ctcheck/4: PREDICATE

No further documentation available for this predicate.

inductive_ctcheck_summary/3: PREDICATE

No further documentation available for this predicate.

intermod_ctcheck_opt/2: PREDICATE

Usage: `intermod_ctcheck_opt(AbsInt,TopLevel)`

Performs CT assertion checking of the program unit which has `TopLevel` as a top-level module, using `AbsInt` to get information about the program (exploits order of the modules).

- *The following properties should hold at call time:*

  `AbsInt` is currently a term which is not a free variable. (`nonvar/1`)

  `TopLevel` is currently a term which is not a free variable. (`nonvar/1`)

intermod_ctcheck_opt/3: PREDICATE

No further documentation available for this predicate.

74.3 Documentation on multifiles

dump_flags_list/2: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

aidomain/1: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

transformation/1: PROPERTY

Usage: `transformation(Transformation)`

`Transformation` is a valid transformation identifier.

The predicate is *multifile*.

library_directory/1: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

The predicate is of type *dynamic*.

file_search_path/2: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

The predicate is of type *dynamic*.

74.4 Documentation on imports

This module has the following direct dependencies:

- *Application modules:*
  mem_usage, intermod_db, intermod_schedule, intermod_punit, intermod_entry, intermod_ops, frontend_driver, analyze_driver, re_analysis, plai, spec, codegen, plai_db, fixpo_ops, preprocess_flags, p_dump, ciaopp_log, raw_printer, ctchecks_pred, unfold_times, analysis_stats.
- *System library modules:*
  native_props, datafacts_rt, dynamic_rt, p_unit, aux_filenames, p_unit_db, counters, system, process, system_extra, terms, c_itf, lists, llists, pathnames, aggregates, hiordlib.
- *Internal (engine) modules:*
  term_basic, arithmetic, atomic_basic, basiccontrol, exceptions, term_compare, term_typing, debugger_support, basic_props, hiord_rt, internals, stream_basic, runtime_control, messages_basic.
- *Packages:*
  prelude, initial, condcomp, assertions, assertions/assertions_basic, basicmodes, nativeprops, fsyntax, dynamic, datafacts, spec(nomem_usage).

75 Database for intermodular analysis driver

This module contains the predicates used as data base (state) of intermodular analysis. This includes the program unit graph, i.e., the imported/exported modules graph; and the global analysis graph (registry), i.e., the intermodular equivalent of complete/7 in plai_db, that contains the relations of the predicates in the boundaries of the modules, i.e., how exported predicates call imported predicates.

75.1 Usage and interface

- **Library usage:**
  `:- use_module(ciaopp(plai/intermod_db)).`
- **Exports:**
  - *Predicates:*
    `add_main_module/1`, `main_module/2`, `punit_module/2`, `local_ana_module/2`, `registry/3`, `registry_header/2`, `module_depth/2`, `imported_module/2`, `caller_module/2`, `src_changed/1`, `add_src_changed/1`, `unset_src_changed/1`, `clean_program_structure/0`, `set_main_module/1`, `set_punit_modules/1`, `set_local_ana_modules/1`, `get_registry/4`, `add_registry/2`, `cleanup_registry/1`, `reg_version/1`, `registry_header_format/2`, `mod_typedb/2`, `add_mod_typedb/2`, `enum_mod_typedb/1`.

75.2 Documentation on exports

add_main_module/1: PREDICATE

No further documentation available for this predicate.

main_module/2: PREDICATE

Usage: `main_module(Base,Mod)`

Succeeds if `Base` is an entry module of the program unit to be analyzed.

The predicate is of type *data*.

punit_module/2: PREDICATE

Usage: `punit_module(Base,Mod)`

Succeeds if `Base` is a module of the program unit to be analyzed (if reachable).

The predicate is of type *data*.

local_ana_module/2: PREDICATE

Usage: `local_ana_module(Base,Mod)`

Succeeds if `File` is a module of the program unit that is being analyzed in this iteration of the intermodular analysis.

The predicate is of type *data*.

registry/3: PREDICATE

Usage: `registry(Key,Module,Registry)`

Data predicate to locally store information about the registry of one or several modules. `Module` is the name of the module for which `Registry` is an entry in the registry file. It corresponds to the *global answer table* as it is described in [PCH03], or other auxiliary information (e.g., types).

The predicate is of type *data*.

registry_header/2: PREDICATE

Usage: `registry_header(Module,HeaderTerm)`

`HeaderTerm` is a term read from the registry header of module `Module`. Data predicate to store the header terms of every registry file read. The list of registry header terms depends on the registry file version, and is stored in `registry_header_format/2`

The predicate is of type *data*.

module_depth/2: PREDICATE

No further documentation available for this predicate. The predicate is of type *data*.

imported_module/2: PREDICATE

Usage: `imported_module(Module,Base)`

Enumerates the modules imported by the current module

The predicate is of type *data*.

caller_module/2: PREDICATE

Usage: `caller_module(Module,BaseName)`

List of caller modules to be processed.

The predicate is of type *data*.

src_changed/1: PREDICATE

No further documentation available for this predicate. The predicate is of type *data*.

add_src_changed/1: PREDICATE

No further documentation available for this predicate.

unset_src_changed/1: PREDICATE

No further documentation available for this predicate.

clean_program_structure/0: PREDICATE

No further documentation available for this predicate.

set_main_module/1: PREDICATE

Usage: `set_main_module(File)`

- *The following properties should hold at call time:*
  `File` is currently a term which is not a free variable. (`nonvar/1`)
- *The following properties should hold globally:*
  All the calls of the form `set_main_module(File)` do not fail. (`not_fails/1`)

set_punit_modules/1: PREDICATE

Usage: `set_punit_modules(File)`

- *The following properties should hold at call time:*
  `File` is currently a term which is not a free variable. (`nonvar/1`)
  `File` is a list. (`list/1`)
- *The following properties should hold globally:*
  All the calls of the form `set_punit_modules(File)` do not fail. (`not_fails/1`)

set_local_ana_modules/1: PREDICATE

Usage: `set_local_ana_modules(Mods)`

- *The following properties should hold at call time:*
  `Mods` is a list. (`list/1`)
- *The following properties should hold globally:*
  All the calls of the form `set_local_ana_modules(Mods)` do not fail. (`not_fails/1`)
  All calls of the form `set_local_ana_modules(Mods)` are deterministic. (`is_det/1`)

get_registry/4: PREDICATE

No further documentation available for this predicate.

add_registry/2: PREDICATE

No further documentation available for this predicate.

cleanup_registry/1: PREDICATE

No further documentation available for this predicate.

reg_version/1: PREDICATE

Usage: `reg_version(Version)`

Contains a version number which identifies the registry files associated with this version of the assertion library. Should be changed every time changes are made which render registry files incompatible, since this forces recomputation of all such files.

– *The following properties should hold upon exit:*

`Version` is an atom. (`atm/1`)

registry_header_format/2: PREDICATE

Usage: `registry_header_format(Version,TermList)`

`TermList` is the list of terms which must appear in the header of version `Version` registry files, excluding the version number term itself.

– *The following properties should hold at call time:*

`Version` is an atom. (`atm/1`)

– *The following properties should hold upon exit:*

`TermList` is a list. (`list/1`)

mod_typedb/2: PREDICATE

Usage: `mod_typedb(Module,TypeDef)`

Data predicate to locally store information about the types used in the registry of one or several modules. `Module` is the name of the module for which the type definition `TypeDef` is referenced in the registry file. The original definition of `TypeDef` may not reside in `Module`, but in a related module.

– *The following properties should hold upon exit:*

`Module` is an atom. (`atm/1`)

`TypeDef` is any term. (`term/1`)

add_mod_typedb/2: PREDICATE

No further documentation available for this predicate.

enum_mod_typedb/1: PREDICATE

No further documentation available for this predicate.

75.3 Documentation on imports

This module has the following direct dependencies:

– *System library modules:*

`native_props`, `datafacts_rt`, `streams`, `aux_filenames`, `program_keys`.

– *Internal (engine) modules:*

`term_basic`, `arithmetic`, `atomic_basic`, `basiccontrol`, `exceptions`, `term_compare`, `term_typing`, `debugger_support`, `basic_props`, `hiord_rt`.

– *Packages:*

`prelude`, `initial`, `condcomp`, `assertions`, `assertions/assertions_basic`, `isomodes`, `regtypes`, `nativeprops`, `datafacts`.

76 Intermodular analysis scheduler

This module implements different intermodular scheduling policies. This affects the order in which modular iteration is performed.

76.1 Usage and interface

- **Library usage:**
  `:- use_module(ciaopp(plai/intermod_schedule)).`
- **Exports:**
  - *Predicates:*
    `modules_analyzed/1`, `reset_queue/0`, `push/2`, `pop/2`, `naive_pending_modules/1`, `force_analysis/1`, `cleanup_intermod_scheduling/0`, `naive_module_order/1`, `iterations/1`, `module_processed/2`, `setup_scheduling/4`, `is_naive_scheduling/1`, `add_naive_pending_modules/1`.

76.2 Documentation on exports

modules_analyzed/1: PREDICATE

No further documentation available for this predicate. The predicate is of type *data*.

reset_queue/0: PREDICATE

Usage:

Empties the queue.

push/2: PREDICATE

Usage 1: `push(Element,Priority)`

Pushes a new element `Element` with priority `Priority` into the priority queue. If `Element` is already in the queue, it is not duplicated, but its priority is changed to the maximum prioriy of the already existing element and the priority of the new element.

- *The following properties should hold at call time:*

  `Element` is currently a term which is not a free variable. (`nonvar/1`)

  `Priority` is currently a term which is not a free variable. (`nonvar/1`)

  `Element` is an atom. (`atm/1`)

  `Priority` is an integer. (`int/1`)

Usage 2: `push(ElementList,Priority)`

Pushes a set of new elements `ElementList` with priority `Priority` into the priority queue. If any element of `ElementList` is already in the queue, it is not duplicated, but its priority is changed to the maximum prioriy of the already existing element and the priority of the new element.

- *The following properties should hold at call time:*
  `ElementList` is currently a term which is not a free variable. (`nonvar/1`)
  `Priority` is currently a term which is not a free variable. (`nonvar/1`)
  `ElementList` is a list of `atm`s. (`list/2`)
  `Priority` is currently instantiated to an integer. (`integer/1`)

pop/2: PREDICATE

Usage: `pop(Element,Priority)`

Pops the element `Element` with highest priority from the priority queue.

- *The following properties should hold at call time:*
  `Element` is a free variable. (`var/1`)
  `Priority` is a free variable. (`var/1`)
- *The following properties should hold upon exit:*
  `Element` is an atom. (`atm/1`)
  `Priority` is an integer. (`int/1`)

naive_pending_modules/1: PREDICATE

No further documentation available for this predicate. The predicate is of type *data*.

force_analysis/1: PREDICATE

No further documentation available for this predicate. The predicate is of type *data*.

cleanup_intermod_scheduling/0: PREDICATE

No further documentation available for this predicate.

naive_module_order/1: PREDICATE

No further documentation available for this predicate. The predicate is of type *data*.

iterations/1: PREDICATE

No further documentation available for this predicate. The predicate is of type *data*.

module_processed/2: PREDICATE

Usage: `module_processed(Module,AlreadyProcessed)`

Lists the modules in the program unit and whether they are already processed or not.

The predicate is of type *data*.

setup_scheduling/4: PREDICATE

Usage: `setup_scheduling(SchedPolicy,Domains,TopLevel,ModList)`

This predicate sets up some stuff in specific global scheduling policies. Implemented policies are as follows:

- `abs_depth_first` Starting from the modules which require analysis (those in `ModList`, it uses the depth in the intermodule dependency graph as priority, and analyzes the modules with higher priority first.
- `naive_top_down` Traverses the list of modules in the intermodule graph in top-down order, and analyzes the modules which have pending patterns. Initially, the modules with pending patterns are the ones in `ModList`. It stops when in a module traversal there is no module with pending patterns.
- `naive_bottom_up` The same as `naive_top_down`, but the list of modules is in reverse top-down order.
- `top_down_preanalysis` The same as `naive_top_down`, but initially all modules are scheduled for analysis. This scheduling policy is intended for using an entry policy different of `main_module` (for example, `all`).
- `bottom_up_preanalysis` The same as `top_down_preanalysis`, but a bottom-up traversal of the intermodule graph is made.

is_naive_scheduling/1: PREDICATE

No further documentation available for this predicate.

add_naive_pending_modules/1: PREDICATE

Usage:

- *The following properties should hold at call time:*

  `Arg1` is a list. (`list/1`)
- *The following properties should hold globally:*

  All the calls of the form `add_naive_pending_modules(Arg1)` do not fail. (`not_fails/1`)

76.3 Documentation on imports

This module has the following direct dependencies:

- *Application modules:*

  `intermod_db`, `intermod_punit`.
- *System library modules:*

  `native_props`, `datafacts_rt`, `lists`, `sets`.
- *Internal (engine) modules:*

  `term_basic`, `arithmetic`, `atomic_basic`, `basiccontrol`, `exceptions`, `term_compare`, `term_typing`, `debugger_support`, `basic_props`, `hiord_rt`.
- *Packages:*

  `prelude`, `initial`, `condcomp`, `assertions`, `assertions/assertions_basic`, `isomodes`, `nativeprops`, `datafacts`.

77 Program structure for intermodular analysis

This module contains the functionality to compute the intermodular graph of a program, i.e., the dependencies of its modules.

The predicate `compute_intermodule_graph/1` calls the Ciao compiler to obtain which modules are in the intermodular punit, which is controlled by the flag `punit_boundary`. This is done by:

- `process_changed_module/1`: Is executed if the module changed (w.r.t. the `.itf` file.
- `check_stop_one_module/1`: Decides whether the compiler needs to keep processing the modules imported by the current module.
- `redo_unchanged_module/1`: Is executed if there were no changes since the last time the module was **compiled**, if always fails because it is asking whether to process the file **again**.

77.1 Usage and interface

- **Library usage:**
  `:- use_module(ciaopp(plai/intermod_punit)).`
- **Exports:**
  - *Predicates:*
    `compute_punit_modules/3`, `get_punit_modules/1`, `get_punit_included_files/1`, `get_all_module_cycles/2`, `module_is_processable/1`, `ensure_registry_file/3`, `read_registry_file/3`, `write_registry_file/3`, `open_mode/3`, `change_open_mode/2`, `save_registry_info/3`, `save_registry_info/2`, `compute_imported_modules/0`, `get_imported_module_base/2`, `get_modules_analyzed/1`, `set_modules_analyzed/1`.

77.2 Documentation on exports

compute_punit_modules/3: PREDICATE

Usage: `compute_punit_modules(TopLevel,ModList,Error)`

`ModList` is the list of modules to analyze. This list is formed by the modules which have their .reg file outdated, or if the module is not completely analyzed (some of the entries in the .reg file are marked or invalid). For those modules which have no .reg file, the *parents* of the module are included in the list (as there are no call patterns for those modules without .reg file).

The structure of the elements in the list is a term (`Mod`,`Depth`,`Force`), where `Mod` stands for the module name, `Depth` is the maximum depth without cycles in the intermodular graph, and `Force` marks those modules which must be completely reanalyzed (only useful for the parents of the modules with no reg file).

- *The following properties should hold at call time:*
  `TopLevel` is currently a term which is not a free variable. (`nonvar/1`)
- *The following properties should hold upon exit:*
  `ModList` is currently a term which is not a free variable. (`nonvar/1`)
  `Error` is currently a term which is not a free variable. (`nonvar/1`)

- *The following properties should hold globally:*

  All the calls of the form `compute_punit_modules(TopLevel,ModList,Error)` do not fail. (`not_fails/1`)

  All calls of the form `compute_punit_modules(TopLevel,ModList,Error)` are deterministic. (`is_det/1`)

get_punit_modules/1: PREDICATE

Usage: `get_punit_modules(ModList)`

Obtains `ModList`, the list of modules in the program unit that is currently loaded. This list is formed by the modules which appear in the top-down modular graph traversal with registries set to read_write mode (and including library modules if punit_boundary flag is set to 'on'. `compute_punit_modules/2` needs to be called before.

- *The following properties should hold upon exit:*

  `ModList` is currently a term which is not a free variable. (`nonvar/1`)

  `ModList` is a list. (`list/1`)
- *The following properties should hold globally:*

  All the calls of the form `get_punit_modules(ModList)` do not fail. (`not_fails/1`)

  All calls of the form `get_punit_modules(ModList)` are deterministic. (`is_det/1`)

get_punit_included_files/1: PREDICATE

Usage: `get_punit_included_files(IncludeList)`

Returns the list of included files in the program unit. This list includes not only files explicitly included with `:- include` declarations, but also packages used. `compute_punit_modules/2` needs to be called before.

- *The following properties should hold upon exit:*

  `IncludeList` is currently a term which is not a free variable. (`nonvar/1`)

  `IncludeList` is a list of `filename`s. (`list/2`)
- *The following properties should hold globally:*

  All the calls of the form `get_punit_included_files(IncludeList)` do not fail. (`not_fails/1`)

  All calls of the form `get_punit_included_files(IncludeList)` are deterministic. (`is_det/1`)

get_all_module_cycles/2: PREDICATE

Usage: `get_all_module_cycles(TopLevelFile,CycleList)`

Obtains `CycleList`, the list of cycles in the program unit whose top-level module is `TopLevelFile`. A cycle is a ciclic dependency in the module dependency graph. Every element of `CycleList` is a list of the modules which belong to each cycle. Modules not belonging to any cycle are represented as one-element lists. `CycleList` is sorted as a post-order traversal of the inter-cycle dependency graph.

The modules included in `CycleList` are those which appear in the top-down modular graph traversal with registries set to read_write mode (and including library modules if punit_boundary flag is set to 'on'.)

- *The following properties should hold at call time:*
  `TopLevelFile` is currently a term which is not a free variable. (`nonvar/1`)
- *The following properties should hold upon exit:*
  `CycleList` is currently a term which is not a free variable. (`nonvar/1`)
  `CycleList` is a list. (`list/1`)
- *The following properties should hold globally:*
  All the calls of the form `get_all_module_cycles(TopLevelFile,CycleList)` do not fail. (`not_fails/1`)
  All calls of the form `get_all_module_cycles(TopLevelFile,CycleList)` are deterministic. (`is_det/1`)

module_is_processable/1: PREDICATE

Usage: `module_is_processable(Base)`

Succeeds if module in `Base` can be processed by intermodular preprocessing tools. This predicate may have to load the registry file of that module, in order to check that the module has read-write mode.

- *The following properties should hold at call time:*
  `Base` is currently a term which is not a free variable. (`nonvar/1`)

ensure_registry_file/3: PREDICATE

No further documentation available for this predicate.

read_registry_file/3: PREDICATE

Usage: `read_registry_file(Module,Base,Verb)`

Reads the registry file of `Module` and loads it into `registry/2`, erasing any previous registry information for that module. `Base` must be the absolute file name, but excluding file extension.

- *The following properties should hold at call time:*
  `Module` is currently a term which is not a free variable. (`nonvar/1`)
  `Base` is currently a term which is not a free variable. (`nonvar/1`)
  `Verb` is currently a term which is not a free variable. (`nonvar/1`)
  `Module` is an atom. (`atm/1`)
  `Base` is an atom. (`atm/1`)
  `Verb` is an atom. (`atm/1`)
- *The following properties should hold globally:*
  All the calls of the form `read_registry_file(Module,Base,Verb)` do not fail. (`not_fails/1`)
  All calls of the form `read_registry_file(Module,Base,Verb)` are deterministic. (`is_det/1`)

write_registry_file/3: PREDICATE

Usage: `write_registry_file(Base,Module,Verb)`

Writes to disk the registry information stored in memory for module `Module` which has as base file name `Base`.

- *The following properties should hold at call time:*

  `Base` is an atom. (`atm/1`)

  `Module` is an atom. (`atm/1`)

  `Verb` is an atom. (`atm/1`)

- *The following properties should hold globally:*

  All the calls of the form `write_registry_file(Base,Module,Verb)` do not fail. (`not_fails/1`)

  All calls of the form `write_registry_file(Base,Module,Verb)` are deterministic. (`is_det/1`)

open_mode/3: PREDICATE

Usage: `open_mode(Base,Type,OpenMode)`

Module with basename `Base` is of type `Type` and it is opened with mode `OpenMode`. `Type` can be `user` or `library`. `OpenMode` is used to indicate if an imported module's registry can be updated. It can take the values `read_write` or `read_only`.

change_open_mode/2: PREDICATE

Usage: `change_open_mode(Base,OpenMode)`

`OpenMode` is the new open mode of the module with basename `Base`. `OpenMode` can take the values `read_write` and `read_only`.

- *The following properties should hold at call time:*

  `Base` is currently a term which is not a free variable. (`nonvar/1`)

  `OpenMode` is currently a term which is not a free variable. (`nonvar/1`)

save_registry_info/3: PREDICATE

Usage: `save_registry_info(Verb,CurrBase,Info)`

Writes on disk the registry information of the modules related to the current module (`CurrBase`) which have been modified. This information is updated by `gen_registry_info/3`. Even if this predicate is used, `save_registry_info/2` must be used after performing intermodular preprocessing.

- *The following properties should hold at call time:*

  `Verb` is currently a term which is not a free variable. (`nonvar/1`)

  `CurrBase` is currently a term which is not a free variable. (`nonvar/1`)

  `Verb` is an atom. (`atm/1`)

  `CurrBase` is an atom. (`atm/1`)

  `Info` is any term. (`term/1`)

- *The following properties should hold globally:*

  All the calls of the form `save_registry_info(Verb,CurrBase,Info)` do not fail. (`not_fails/1`)

  All calls of the form `save_registry_info(Verb,CurrBase,Info)` are deterministic. (`is_det/1`)

save_registry_info/2: PREDICATE

Usage: `save_registry_info(Verb,Info)`

Writes on disk the registry information of the modules loaded into Ciaopp or related to modules loaded into Ciaopp. This information is updated by `gen_registry_info/3`. This predicate must be called after performing intermodular preprocessing (analysis, specialization...), even if `save_registry_info/3` has been used.

- *The following properties should hold at call time:*
  - `Verb` is currently a term which is not a free variable. (`nonvar/1`)
  - `Verb` is an atom. (`atm/1`)
  - `Info` is any term. (`term/1`)
- *The following properties should hold globally:*
  - All the calls of the form `save_registry_info(Verb,Info)` do not fail. (`not_fails/1`)
  - All calls of the form `save_registry_info(Verb,Info)` are deterministic. (`is_det/1`)

compute_imported_modules/0: PREDICATE

Usage:

Gets the list of imported modules from the current module. This list is obtained from the itf information of the current module, and is stored in `imported_module/1`.

- *The following properties should hold globally:*
  - All the calls of the form `compute_imported_modules` do not fail. (`not_fails/1`)
  - All calls of the form `compute_imported_modules` are deterministic. (`is_det/1`)

get_imported_module_base/2: PREDICATE

No further documentation available for this predicate.

get_modules_analyzed/1: PREDICATE

Usage: `get_modules_analyzed(ModList)`

Returns the list of modules analyzed the last time a modular analysis was executed.

- *The following properties should hold upon exit:*
  - `ModList` is a list. (`list/1`)

set_modules_analyzed/1: PREDICATE

Usage: `set_modules_analyzed(ModList)`

Sets the list of modules which have been analyzed.

- *The following properties should hold at call time:*
  - `ModList` is a list. (`list/1`)

77.3 Documentation on imports

This module has the following direct dependencies:

- *Application modules:*
  preprocess_flags, ciaopp_log, analysis_stats, intermod_db, tarjan, frontend_driver, intermod_ops.
- *System library modules:*
  native_props, datafacts_rt, p_unit_db, p_asr, aux_filenames, c_itf_props, aggregates, ctrlcclean, errhandle, streams, write, read, pathnames, system, fastrw, hiordlib, bundle_paths, c_itf.
- *Internal (engine) modules:*
  term_basic, arithmetic, atomic_basic, basiccontrol, exceptions, term_compare, term_typing, debugger_support, basic_props, hiord_rt, stream_basic, io_basic.
- *Packages:*
  prelude, initial, condcomp, assertions, assertions/assertions_basic, modes, nativeprops, datafacts.

78 Intermodular entry policy

This module provides the entry policy algorithms for modular analysis. The entry policy is determined by the `entry_policy` preprocessing flag.

78.1 Usage and interface

- **Library usage:**
  `:- use_module(ciaopp(plai/intermod_entry)).`
- **Exports:**
  - *Predicates:*
    `check_curr_entry_id/1`, `curr_entry_id/1`, `get_entry_info/3`, `intermod_entry_point/7`, `add_entries_to_registry/1`.

78.2 Documentation on exports

check_curr_entry_id/1: PREDICATE

Usage: `check_curr_entry_id(Id)`

Checks that `Id` was an entry of the last fixpoint analysis.

- *The following properties should hold at call time:*
  `Id` is currently a term which is not a free variable. (`nonvar/1`)

curr_entry_id/1: PREDICATE

Usage: `curr_entry_id(Id)`

Enumerates the `Ids` of the entries of the last fixpoint analysis.

get_entry_info/3: PREDICATE

Usage: `get_entry_info(AbsInt,Sg,Proj)`

Provides on backtracking entry abstract substitutions for the current module in the global level of intermodular analysis. In the case of manual scheduling, this predicate should be called for every module in the program unit. In the case of automatic scheduling, this predicate should only be called with top-level(U).

- *The following properties should hold at call time:*
  `AbsInt` is currently a term which is not a free variable. (`nonvar/1`)
  `Sg` is a free variable. (`var/1`)
  `Proj` is a free variable. (`var/1`)

intermod_entry_point/7: PREDICATE

Usage:

User-specified entry points for intermodular analysis. I.e., according to the policy defined

add_entries_to_registry/1: PREDICATE

Usage: `add_entries_to_registry(AbsInt)`

- *The following properties should hold at call time:*
  `AbsInt` is currently a term which is not a free variable. (`nonvar/1`)
  `AbsInt` is an atom. (`atm/1`)
- *The following properties should hold globally:*
  All the calls of the form `add_entries_to_registry(AbsInt)` do not fail. (`not_fails/1`)

78.3 Documentation on imports

This module has the following direct dependencies:

- *Application modules:*
  `intermod_ops`, `preprocess_flags`, `domains`, `transform`, `apply_assertions`, `intermod_db`.
- *System library modules:*
  `native_props`, `datafacts_rt`, `lists`, `terms_vars`, `counters`, `vndict`, `p_unit`, `program_keys`, `p_unit_db`, `assrt_lib`.
- *Internal (engine) modules:*
  `term_basic`, `arithmetic`, `atomic_basic`, `basiccontrol`, `exceptions`, `term_compare`, `term_typing`, `debugger_support`, `basic_props`, `hiord_rt`.
- *Packages:*
  `prelude`, `initial`, `condcomp`, `assertions`, `assertions/assertions_basic`, `isomodes`, `nativeprops`, `datafacts`.

79 Intermodular success policy

This module implements the policies to compute (possibly) temporary success patterns when modular analysis is performed.

79.1 Usage and interface

- **Library usage:**
  `:- use_module(ciaopp(plai/intermod_success)).`
- **Exports:**
  - *Predicates:*
    `get_success_info/7`, `apply_success_policy/9`.

79.2 Documentation on exports

get_success_info/7: PREDICATE

Usage: `get_success_info(Call,SgKey,Sg,Sv,AbsInt,Prime,PatternsApplied)`

Given a call pattern for an imported predicate defined by `Call` call and `Sg` abstract substitution, `Prime` is the success substitution resulting from the application of the success policy for imported predicates. `PatternsApplied` is instatiated to `no` if there are no applicable patterns. The predicate is allowed to fail if no patterns are found.

- *The following properties should hold at call time:*
  - `Call` is currently a term which is not a free variable. (`nonvar/1`)
  - `SgKey` is currently a term which is not a free variable. (`nonvar/1`)
  - `Sg` is currently a term which is not a free variable. (`nonvar/1`)
  - `Sv` is currently a term which is not a free variable. (`nonvar/1`)
  - `AbsInt` is currently a term which is not a free variable. (`nonvar/1`)
  - `Prime` is a free variable. (`var/1`)
  - `PatternsApplied` is a free variable. (`var/1`)

apply_success_policy/9: PREDICATE

Usage: `apply_success_policy(SuccPolicy,AbsInt,SgKey,Sg,Sv,Proj,Patterns,Prime,PatternsApplied)`

Applies the success policy given as first argument to the list of triples (SgProj,Proj,Succ) `Patterns` w.r.t. `Proj`. If there are no applicable patterns in `Patterns`, it returns either `'$bottom'` or the topmost substitution, depending on the type of the success policy (either it is underapproximating or overapproximating, respectively.) `PatternsApplied` is instatiated to `no` if there are no applicable patterns.

- *The following properties should hold at call time:*
  - `SuccPolicy` is currently a term which is not a free variable. (`nonvar/1`)
  - `AbsInt` is currently a term which is not a free variable. (`nonvar/1`)
  - `SgKey` is currently a term which is not a free variable. (`nonvar/1`)
  - `Sg` is currently a term which is not a free variable. (`nonvar/1`)

`Sv` is currently a term which is not a free variable. (`nonvar/1`)
`Proj` is currently a term which is not a free variable. (`nonvar/1`)
`Patterns` is currently a term which is not a free variable. (`nonvar/1`)
`Prime` is a free variable. (`var/1`)
`PatternsApplied` is a free variable. (`var/1`)

79.3 Documentation on imports

This module has the following direct dependencies:

- *Application modules:*
  `intermod_ops`, `preprocess_flags`, `domains`, `fixpo_ops`, `intermod_db`, `intermod_punit`.
- *System library modules:*
  `datafacts_rt`, `terms_vars`, `aggregates`, `terms_check`, `lists`, `p_unit_db`, `program_keys`.
- *Internal (engine) modules:*
  `term_basic`, `arithmetic`, `atomic_basic`, `basiccontrol`, `exceptions`, `term_compare`, `term_typing`, `debugger_support`, `basic_props`, `hiord_rt`.
- *Packages:*
  `prelude`, `initial`, `condcomp`, `assertions`, `assertions/assertions_basic`, `isomodes`, `datafacts`.

80 Preprocessing Flags

80.1 Usage and interface

- **Library usage:**
  `:- use_module(ciaopp(preprocess_flags)).`
- **Exports:**
  - *Predicates:*
    `push_pp_flag/2`, `pop_pp_flag/1`, `current_pp_flag/2`, `set_pp_flag/2`, `pp_flag/2`, `valid_flag_values/2`, `dump_flags/1`, `save_config/1`, `restore_config/1`, `remove_config/1`, `show_config/1`, `show_configs/0`, `sublist2/2`, `typeanalysis/1`.
  - *Properties:*
    `valid_flag_value/2`.
  - *Regular Types:*
    `pp_flag/1`, `flag_value/1`.
  - *Multifiles:*
    `$is_persistent/2`, `persistent_dir/2`, `persistent_dir/4`, `dump_flags_list/2`.

80.2 Documentation on exports

push_pp_flag/2: PREDICATE

Usage: `push_pp_flag(Flag,Value)`

Sets `Value` for preprocessor flag `Flag`, storing the current value to restore it with `pop_pp_flag/1`.

- *The following properties should hold at call time:*
  `Flag` is a valid preprocessor flag. (`pp_flag/1`)
  `Value` is a valid value for preprocessor flag `Flag`. (`valid_flag_value/2`)

pop_pp_flag/1: PREDICATE

Usage: `pop_pp_flag(Flag)`

Restores the value of the preprocessor flag `Flag` previous to the last non-canceled `push_pp_flag/2` on it.

- *The following properties should hold at call time:*
  `Flag` is a valid preprocessor flag. (`pp_flag/1`)

current_pp_flag/2: PREDICATE

Usage: `current_pp_flag(Name,Value)`

Preprocess flag `Name` has the value `Value`.

- *The following properties should hold at call time:*
  `Name` is a valid preprocessor flag. (`pp_flag/1`)
- *The following properties should hold upon exit:*
  `Value` is a valid value for preprocessor flag `Name`. (`valid_flag_value/2`)

set_pp_flag/2: PREDICATE

Usage: `set_pp_flag(Name,Value)`

Sets `Value` for preprocessor flag `Name`.

– *The following properties should hold at call time:*

`Name` is a valid preprocessor flag. (`pp_flag/1`)

`Value` is a valid value for preprocessor flag `Name`. (`valid_flag_value/2`)

pp_flag/2: PREDICATE

Usage: `pp_flag(Name,Help)`

`Name` is a valid preprocess flag.`Help` is a description of what `Name` does.

pp_flag/1: REGTYPE

The list of valid flags and their values is as follows:

(This list is outdated – see source code for a complete description)

- for the output:
  - `analysis_info` (`off, on`) Whether to output the results of analysis.
  - `point_info` (`off,on`) Whether to output analysis information for program points within clauses.
  - `collapse_ai_vers` (`off, on`) to output all the versions of call/success patterns inferred by analysis or just one version (summing-up all of them).
  - `type_output` (`defined, all`) to output the types inferred for predicates in terms only of types defined by the user or including types inferred anew.
  - `simplify_checks` (`on, off`) to output simplified check assertions after verification or not.
- for analysis:
  - `fixpoint` (`plai, dd, di, check_di, check_di2, check_di3, check_di4`) The kind of fixpoint computation used.
  - `multi_success` (`off, on`) Whether to allow success multivariance.
  - `widen` (`off, on`) Whether to perform widening.
  - `intermod` (`off, on, auto`) The policy for inter-modular analysis.
  - `success_policy` (`best, first, all, top, botfirst, botbest, botall, bottom`) The policy for obtaining success information for imported predicates during inter-modular analysis.
  - `entry_policy` (`all, top_level, force, force_assrt`) The policy for obtaining entry call patterns for exported predicates during inter-modular analysis.
  - `process_libraries` (`on, off, no_engine`) Whether to perform the analysis of Ciao system libraries when a modular user program is analyzed.
  - `initial_guess` (`botfirst, botbest, botall, bottom`) The policy for obtaining initial guess when computing the analysis of a predicate from the current module.
  - `use_check_assrt` (`off, on`) Whether to use check assertions for imported predicates as if they were trust assertions.

- `depth` (a non-negative integer) The maximum depth of abstractions in analyses based on term depth.
- `type_eval` (`on`, `off`) Whether to attempt concrete evaluation of types being inferred.
- `type_precision` (`defined`, `all`) to use during type analysis only types defined by the user or also types inferred anew.
- `entry_points_auto` (`none`, `calls`, `all`) What kind of assertions to use as entry points: `entry/1` will always be used, `calls/1` and `calls/2` will be used if the flag is **not** set to `none`; and `success/1`, `success/2`, `comp/1` and `comp/2` will be used if the flag is set to `all`.
- `entry_calls_scope` (`exported`, `all`) What non-`entry` assertions to use as entry points. If the flag takes the value `exported`, only assertions from exported predicates will be used; if it takes the value `all`, exported and non-exported assertions will be used.

- for partial evaluation:
  - `global_control` (`off`, `id`, `inst`, `hom_emb`) The abstraction function to use to control the creation of new patterns to analyze as a result of unfolding.
  - `comp_rule` (`leftmost`, `safe_jb`, `bind_ins_jb`, `no_sideff_jb`, `jump_builtin`, `eval_builtin`, `local_emb`) The computation rule for the selection of atoms in a goal.
  - `local_control` (`off`, `orig`, `inst`, `det`, `det_la`, `depth`, `first_sol`, `first_sol_d`, `all_sol`, `hom_emb`, `hom_emb_anc`, `hom_emb_as`, `df_hom_emb_as`, `df_tree_hom_emb`, `df_hom_emb`) The unfolding rule to use during partial evaluation.
  - `unf_depth` (a non-negative integer) The depth limit for unfolding.
  - `rem_use_cls` (`off`, `pre`, `post`, `both`) Whether to remove useless clauses.
  - `abs_spec_defs` (`off`, `rem`, `exec`, `all`) Whether to exploit abstract substitutions while obtaining specialized definitions on unfolding.
  - `filter_nums` (`off`, `on`) Whether to filter away numbers in partial evaluation.
  - `exec_unif` (`off`, `on`) Whether to execute unifications during specialization time or not.
  - `pres_inf_fail` (`off`, `on`) Whether infinite failure should be preserved in the specialized program.
  - `part_concrete` (`off`, `mono`, `multi`) The kind of partial concretization to be performed.
- for parallelization and granularity control:
  - `granularity_threshold` (a non-negative integer) The threshold on computational cost at which parallel execution pays off.

Usage: `pp_flag(Flag)`

`Flag` is a valid preprocessor flag.

flag_value/1: REGTYPE

Usage: `flag_value(V)`

`V` is a value for a flag.

valid_flag_value/2: PROPERTY

Usage: `valid_flag_value(Name,Value)`

`Value` is a valid value for preprocessor flag `Name`.

- *If the following properties should hold at call time:*

  `Name` is a valid preprocessor flag. (`pp_flag/1`)

  `Value` is a value for a flag. (`flag_value/1`)

valid_flag_values/2: PREDICATE

Usage: `valid_flag_values(Name,Value_Checker)`

`Value_Checker` is a goal that checks that a value given as first argument of this term is a valid value for `Name`.

- *Call and exit should be compatible with:*

  `Name` is a valid preprocessor flag. (`pp_flag/1`)

  `Value_Checker` is a term which represents a goal, i.e., an atom or a structure. (`cgoal/1`)

dump_flags/1: PREDICATE

Usage: `dump_flags(Name)`

`Name` represent the list of flags to be dumped. To associate a name (key) with a list, use `dump_flags_list`.

- *The following properties should hold at call time:*

  `Name` is an atom. (`atm/1`)

save_config/1: PREDICATE

Usage: `save_config(Name)`

Save the current flags configuration under the `Name` key.

- *The following properties should hold at call time:*

  `Name` is an atom. (`atm/1`)

restore_config/1: PREDICATE

Usage: `restore_config(Name)`

Restores the set of flags saved previously under the name of `Name`.

- *The following properties should hold at call time:*

  `Name` is an atom. (`atm/1`)

remove_config/1: PREDICATE

Usage: `remove_config(Name)`

Remove the configuration stored with the `Name` key.

- *The following properties should hold at call time:*

  `Name` is an atom. (`atm/1`)

show_config/1: PREDICATE

Usage: `show_config(C)`

Show specific configuration values pointed by `C` key.

– *The following properties should hold at call time:*

`C` is an atom. (`atm/1`)

show_configs/0: PREDICATE

Usage:

Show all stored configs.

sublist2/2: PREDICATE

No further documentation available for this predicate.

typeanalysis/1: PREDICATE

No further documentation available for this predicate.

80.3 Documentation on multifiles

$is_persistent/2: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

The predicate is of type *data*.

persistent_dir/2: PREDICATE

Usage 1: `persistent_dir(Key,Dir)`

– *The following properties hold upon exit:*

`Key` is an atom. (`atm/1`)

`Dir` is an atom. (`atm/1`)

Usage 2: `persistent_dir(Key,Dir)`

– *The following properties should hold at call time:*

`Key` is an atom. (`atm/1`)

`Dir` is a free variable. (`var/1`)

Usage 3: `persistent_dir(Key,Dir)`

– *The following properties should hold at call time:*

`Key` is an atom. (`atm/1`)

`Dir` is an atom. (`atm/1`)

The predicate is *multifile*.

The predicate is of type *data*.

persistent_dir/4: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

The predicate is of type *data*.

dump_flags_list/2: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

80.4 Documentation on imports

This module has the following direct dependencies:

- *Application modules:*
  `frontend_driver`.
- *System library modules:*
  `datafacts_rt`, `persdb_rt`, `pathnames`, `system`, `lists`, `aggregates`, `write`, `menu_generator`.
- *Internal (engine) modules:*
  `term_basic`, `arithmetic`, `atomic_basic`, `basiccontrol`, `exceptions`, `term_compare`, `term_typing`, `debugger_support`, `basic_props`, `hiord_rt`, `messages_basic`, `io_basic`.
- *Packages:*
  `prelude`, `initial`, `condcomp`, `assertions`, `assertions/assertions_basic`, `nortchecks`, `regtypes`, `isomodes`, `persdb`, `datafacts`, `persdb/persdb_decl`, `ciaopp(ciaopp_options)`.

81 Debugging PLAI analyses

Author(s): Isabel Garcia-Contreras.

This module is intended for debugging fixpoints and abstract domains. The debugging modes are sorted by difficulty and knowledge required by the programmer.

General comments. To discard errors related with not cleaning properly internal databases it is better not to debug in the ciao/ciaopp toplevel, but rather use the ciaopp command line if possible.

81.1 Debug raw analysis result

Output the analysis result without postprocessing. This is using internal clause representation of ciaopp and internal representation of the abstract domain. For this, simply set the flag `output_lang` to `raw`.

For every **reachable** program point there is one or more `true` annotations, one for every inferred call pattern. In the body, annotations with the same number correspond to the execution of the abstract call with that number. A raw assertion will appear with the same id for such abstract call and success.

Note that this option is specially useful when developing a new abstract domain. If the predicates `asub_to_info/5`, `asub_to_native/6`, etc are not defined, no useful information appears in the usual.

81.2 Inspect analysis graphs

Dump the analysis graph in some <file>:

```
ciaopp ?- use_module(ciaopp(p_dump)).
ciaopp ?- dump(<file>).
```

Use the command `ciaopp-dump` to inspect:

```
$ ciaopp-dump show <file>
     # Print info of the dump <file>.
$ ciaopp-dump fmt <file>
     # Outputs the analysis result in dot format.
$ dot -Tpdf <file>.dot -o <file>.pdf
     # Generates a graph in pdf of the dot analysis format.
```

81.3 Trace fixpoint operations

Output steps to reach fixpoint. It is useful to combine this option with the raw output, as it shows the steps to generate the graphs using the same numbers for ids.

In file `src/plai/fixpoint_options.pl`, uncomment `:- use_package(.(notrace)).`, and recompile:

```
$ ciao build --bin ciaopp
```

A clean may be necessary, due to some undetected changes in incremental compilation of programs with included files.

```
$ ciao clean ciaopp
```

Then set flag `trace_fixp` to `trace` in the `ciaopp` toplevel or activate it in the command line:

```
$ ciaopp -A <file> -ftrace_fixp=trace
```

Additionally, if the flag `timestamp_trace` is set to `on` (can only be set in the `ciaopp` toplevel) a timestamp is printed when tracing. This can help finding perfomance issues.

81.4 Activate run-time checks

Go to any module, add `:- use_package(rtchecks).` and recompile by

```
$ ciao build --bin ciaopp
```

Modules recommended:

- Generic domain interface `src/plai/domains.pl`.
- Any specific domain. of `src/domains/*` (check that it has assertions!).
- Fixpoints: `src/plai/fixpoint_options.pl`.
- Generic fixpoint operations: `src/plai/fixpo_ops.pl`.
- Modular analysis: `src/plai/intermod_options.pl`.
- Incremental analysis: `src/plai/incanal/incanal_options.pl`.
- Assertions during fixpoint: `src/plai/apply_assertions.pl`/`src/plai/apply_assertions_old.pl`.

In `domains.pl` the assertions were written by looking only at some domains, not all of them. It could happen that they are too restrictive. If you find that an assertion does not hold in run-time, please open an issue.

81.5 Debug fixpoint step by step

For this, it is recommended to deactivate run-time checks, as the debugger shows also the code for RT checking.

Write `:- use_package(trace).` in `src/plai/fixpoint_options.pl` and recompile:

```
$ ciao build --bin ciaopp
```

In an emacs shell buffer do `M-x ciao-inferior-mode`. Then execute the ciaopp command to debug

```
$ ciaopp -A <file>
```

81.6 Access PLAI databases directly

```
ciaopp ?- use_module(ciaopp(plai/plai_db)).
ciaopp ?- complete(SgKey,AbsInt,Sg,Proj,Prime,Id,Parents).
ciaopp ?- memo_table(LitKey,AbsInt,Id,Child,Vars_u,Call).
```

82 Incremental analysis (high level)

Author(s): Isabel Garcia-Contreras.

Stability: [**devel**] Currently the subject of active development and/or research. Functionality may be limited and API and/or functionality may change without warning or deprecation period. Not recommended yet for use in production.

This module implements a high level interface of incremental analysis (see `incanal_driver` for the low level interface).

Global compilation options of incremental analysis are available in `incanal_options`. Edit this file to activate tracing or run-time checks.

Incremental analysis only works with fixpoint dd, one abstract domain and multivariance on success off. The assertions of the predicates of this module reflect this.

82.1 Usage

Important: For using incremental analysis some ciaopp flags have to be set:

```
?- set_pp_flag(fixpoint, dd),
   set_pp_flag(incremental, on),
   set_pp_flag(del_strategy, top_down). %% select strategy for incremental deletion
```

1. (Re)Load a list of modules to be analyzed with `incremental_module/1` (or `incremental_module/2` to get statistics). It can be done several times before reanalyzing.
2. (Re)Analyze the code with `incremental_analyze/1` (or `incremental_analyze/2` to get statistics). Internally, the changes will be automatically incrementally applied.

   This predicate depends on the ciaopp flag `del_strategy` for incremental deletion.

Note: If a module is not present in the reloading list, it is assumed that it has been removed from analysis so all modules must be in the input list although they may have not changed.

Tip (under construction!!): If you want to quickly display the information inferred by the analyzer between analyses you can:

```
?- use_module(ciaopp(raw_printer)).
?- show_analysis.
```

82.2 Example

Example of incremental analysis for file `foo.pl`

```
:- module(foo, [p/1], []).

p(1).
p(3).
```

Incremental analysis from Ciao toplevel:

```
?- use_module(ciaopp(plai/incanal)), use_module(ciaopp(preprocess_flags)).

yes
```

```
?- % incremental flags
   set_pp_flag(fixpoint, dd),
   set_pp_flag(incremental, on).

yes
?- incremental_module(foo, T).

T = [time(317.96,[])] ?

yes
?- incremental_analyze(eterms, T).

T = [proc_diff(0.244),add(2.281),delete(0.001)] ?

yes
% reload changes:
?- incremental_module(foo, T).

T = [time(299.822,[])] ?

yes
?- incremental_analyze(eterms, T).
{Incrementally analyzed with dd for eterms }

T = [proc_diff(0.314),add(2.148),delete(2.797)] ?

yes
```

82.3 Side effects

As a result of the incremental analysis, auxiliary files are generated for each analyzed module:

- When loaded a module: `File.itf`.
- When analyzed a module: `File.dump`.

They are used to detect if changes were made in a module. To reset the incremental analysis these files have to be removed.

WARNING: There might be problems with clause locators because Ids generated when transforming the code can be modified.

82.4 Usage and interface

- **Library usage:**
  `:- use_module(ciaopp(plai/incanal)).`
- **Exports:**
  - *Predicates:*
    `incremental_module/1`, `incremental_module/2`, `incremental_analyze/1`, `incremental_analyze/2`, `reset_incremental_analysis_info/0`, `config_incremental_load/1`, `all_assertions_pred/4`.

82.5 Documentation on exports

incremental_module/1: PREDICATE

Usage: `incremental_module(Files)`

Loads and transforms the code of `Files` to the ciaopp code database, computing the differences with the code previously present

- *The following properties should hold at call time:*
  `Files` is a list. (`list/1`)
- *The following properties should hold globally:*
  All the calls of the form `incremental_module(Files)` do not fail. (`not_fails/1`)

incremental_module/2: PREDICATE

Usage: `incremental_module(Files,Stats)`

Same as `incremental_module/1` but returns runtime statistics in `Stats`.

- *The following properties should hold at call time:*
  `Files` is a list. (`list/1`)
- *The following properties should hold upon exit:*
  `Stats` is a list. (`list/1`)
- *The following properties should hold globally:*
  All the calls of the form `incremental_module(Files,Stats)` do not fail. (`not_fails/1`)

incremental_analyze/1: PREDICATE

Usage: `incremental_analyze(AbsInt)`

Applies `Diff` to the current analysis by performing all deletions according to the deletion strategy set the ciaopp flag (`top_down` or `bottom_up`), reanalyzing and adding all clauses and assertions after.

- *The following properties should hold at call time:*
  `AbsInt` is an atom. (`atm/1`)

incremental_analyze/2: PREDICATE

Usage: `incremental_analyze(AbsInt,Stats)`

Same as `incremental_analyze/1` but returns runtime statistics in `Stats`. Statistics are divided in:

- `proc_diff`: Time needed to apply the differences to the incremental database.
- `add`: Time needed for performing the addition of clauses.
- `delete`: Time needed for performing the deletion of clauses according to the selected deletion strategy.

– *The following properties should hold at call time:*
`AbsInt` is an atom. (`atm/1`)
– *The following properties should hold upon exit:*
`Stats` is a list. (`list/1`)
– *The following properties should hold globally:*
All the calls of the form `incremental_analyze(AbsInt,Stats)` do not fail. (`not_fails/1`)

reset_incremental_analysis_info/0: PREDICATE

Usage:

Resets incremental and regular ciaopp internal data structures.

config_incremental_load/1: PREDICATE

Usage: `config_incremental_load(X)`

– *The following properties should hold at call time:*
`load_config(X)` (`load_config/1`)

all_assertions_pred/4: PREDICATE

No further documentation available for this predicate.

82.6 Documentation on imports

This module has the following direct dependencies:

– *Application modules:*
`frontend_driver`, `analyze_driver`, `preprocess_flags`, `fixpo_dd`, `incanal_driver`, `incanal_db`, `incanal_persistent_db`, `tarjan_inc`, `apply_assertions_inc`, `ciaopp_log`, `analysis_stats`.
– *System library modules:*
`datafacts_rt`, `native_props`, `lists`, `aggregates`, `diff`, `pathnames`, `p_unit_db`, `program_keys`.
– *Internal (engine) modules:*
`term_basic`, `arithmetic`, `atomic_basic`, `basiccontrol`, `exceptions`, `term_compare`, `term_typing`, `debugger_support`, `basic_props`, `hiord_rt`.
– *Packages:*
`prelude`, `initial`, `condcomp`, `assertions`, `assertions/assertions_basic`, `regtypes`, `modes`, `datafacts`, `nativeprops`, `ciaopp(plai/notrace)`.

83 Incremental analysis (low level)

Author(s): Isabel Garcia-Contreras (ported from ciaopp-0.8).

This module contains the basic operations for adding or removing clauses to ciaopp (currently working for fixpoint dd) and perform an incremental reanalysis.

The posible actions are:

- Initialize ciaopp for incremental analysis (see `init_empty_inc_analysis`), for clauses to be added/removed from, they should be implemented in a file passed to this predicate.
- Add clauses to analysis (see `td_add_clauses/2`). Adds clauses and performs top down reanalysis.
- Remove clauses from analysis, following a top down strategy (see `td_delete_clauses/2`). Removes clauses and performs top down reanalysis.
- Remove clauses from analysis, following a bottom up strategy (see `bu_delete_clauses/2`). Removes clauses and reanalyzes according to the strategy defined.

83.1 Usage and interface

- **Library usage:**
  `:- use_module(ciaopp(plai/incanal/incanal_driver)).`
- **Exports:**
  - *Predicates:*
    `init_empty_inc_analysis/0`, `init_file_inc_analysis/3`, `td_add_clauses/2`, `td_delete_clauses/2`, `bu_delete_clauses/2`, `td_rec_delete_complete/3`, `module_has_entries/1`, `remove_useless_completes/1`, `run_inc_fixpoint/1`, `incrementally_update_analysis/2`, `process_external_complete_change/6`.

83.2 Documentation on exports

init_empty_inc_analysis/0: PREDICATE

Initializes ciaopp for incremental analysis. **Removes all previous analysis info**.

init_file_inc_analysis/3: PREDICATE

Usage: `init_file_inc_analysis(Files,Cls,Ds)`

Initializes incremental analysis with a file.

- *The following properties should hold at call time:*
  `Files` is a list. (`list/1`)

td_add_clauses/2: PREDICATE

Usage: `td_add_clauses(ClKeys,AbsInt)`

Adds a list of already processed clauses, expressed with their clause id (`ClKeys`) to the code database and updates the analysis of abstract domain `AbsInt` with them.

- *The following properties should hold at call time:*
  `ClKeys` is a list. (`list/1`)
  `AbsInt` is an atom. (`atm/1`)
- *The following properties should hold globally:*
  All the calls of the form `td_add_clauses(ClKeys,AbsInt)` do not fail. (`not_fails/1`)
  All calls of the form `td_add_clauses(ClKeys,AbsInt)` are deterministic. (`is_det/1`)

td_delete_clauses/2: PREDICATE

Usage: `td_delete_clauses(ClKeys,AbsInt)`

Removes clauses with ids of list `ClKeys` for abstract domain `AbsInt` from analysis using top-down deletion strategy and performs a reanalysis.

- *The following properties should hold at call time:*
  `ClKeys` is a list. (`list/1`)
  `AbsInt` is an atom. (`atm/1`)
- *The following properties should hold globally:*
  All the calls of the form `td_delete_clauses(ClKeys,AbsInt)` do not fail. (`not_fails/1`)
  All calls of the form `td_delete_clauses(ClKeys,AbsInt)` are deterministic. (`is_det/1`)

bu_delete_clauses/2: PREDICATE

Usage: `bu_delete_clauses(ClKeys,AbsInt)`

Removes clauses `Cls` for abstract domain `AbsInt` from analysis using the bottom up deletion strategy and performs a reanalysis.

- *The following properties should hold at call time:*
  `ClKeys` is a list. (`list/1`)
  `AbsInt` is an atom. (`atm/1`)
- *The following properties should hold globally:*
  All the calls of the form `bu_delete_clauses(ClKeys,AbsInt)` do not fail. (`not_fails/1`)
  All calls of the form `bu_delete_clauses(ClKeys,AbsInt)` are deterministic. (`is_det/1`)

td_rec_delete_complete/3: PREDICATE

Usage: `td_rec_delete_complete(Id,Key,AbsInt)`

This predicate removes the complete with `Id` in domain `AbsInt` and all its dependent information

- *The following properties should hold at call time:*
  `Id` is currently a term which is not a free variable. (`nonvar/1`)
  `AbsInt` is currently a term which is not a free variable. (`nonvar/1`)
  `plai_db_id(Id)` (`plai_db_id/1`)
  `AbsInt` is an atom. (`atm/1`)

- *The following properties should hold globally:*
  All the calls of the form `td_rec_delete_complete(Id,Key,AbsInt)` do not fail. (`not_fails/1`)

module_has_entries/1: PREDICATE

No further documentation available for this predicate.

remove_useless_completes/1: PREDICATE

No further documentation available for this predicate.

run_inc_fixpoint/1: PREDICATE

Usage: `run_inc_fixpoint(AbsInt)`

This predicate runs fixpoint for abstract domain `AbsInt`. It should to be run after adding information from registry files in intermodular analysis.

- *The following properties should hold at call time:*
  `AbsInt` is currently a term which is not a free variable. (`nonvar/1`)
- *The following properties should hold globally:*
  All the calls of the form `run_inc_fixpoint(AbsInt)` do not fail. (`not_fails/1`)
  All calls of the form `run_inc_fixpoint(AbsInt)` are deterministic. (`is_det/1`)

incrementally_update_analysis/2: PREDICATE

Usage: `incrementally_update_analysis(Mod,AbsInt)`

`Mod` is the module that will be reanalyzed after updating its external completes. To be called in modular analysis, no following analysis needed. This is a top-down strategy update

- *The following properties should hold at call time:*
  `Mod` is currently a term which is not a free variable. (`nonvar/1`)
  `AbsInt` is currently a term which is not a free variable. (`nonvar/1`)
- *The following properties should hold globally:*
  All the calls of the form `incrementally_update_analysis(Mod,AbsInt)` do not fail. (`not_fails/1`)
  All calls of the form `incrementally_update_analysis(Mod,AbsInt)` are deterministic. (`is_det/1`)

process_external_complete_change/6: PREDICATE

Usage: `process_external_complete_change(AbsInt,NewPrime,SgKey,Sg,Id,Proj)`

This predicate updates analysis information given a complete that has changed externally. Typically this is used for updating completes that are outdated as a consequence of modular analysis (the analysis of a module was improved). The complete that has change has key `SgKey` and id `Id`. The new answer for the complete is `NewPrime` and refers to domain `AbsInt`.

Changes in answers are updated as follows:

- If the new answer is more general than the previous one, the update is as simple as replacing the answer in the complete and adding an event.
- If the new answer is more specific or incompatible, the information that depends on it has to be deleted and recomputed. We are removing the information with a top-down deletion strategy, although a bottom-up deletion strategy could more efficient. The fixpoint has to be runned after this deletion.

– *The following properties should hold at call time:*

`AbsInt` is currently a term which is not a free variable. (`nonvar/1`)
`NewPrime` is currently a term which is not a free variable. (`nonvar/1`)
`SgKey` is currently a term which is not a free variable. (`nonvar/1`)
`Sg` is currently a term which is not a free variable. (`nonvar/1`)
`Id` is currently a term which is not a free variable. (`nonvar/1`)
`Proj` is currently a term which is not a free variable. (`nonvar/1`)
`AbsInt` is an atom. (`atm/1`)
`NewPrime` is a list. (`list/1`)
`SgKey` is an atom. (`atm/1`)
`plai_db_id(Id)` (`plai_db_id/1`)

– *The following properties should hold globally:*

All the calls of the form `process_external_complete_change(AbsInt,NewPrime,SgKey,Sg,Id,Proj)` do not fail. (`not_fails/1`)

83.3 Documentation on imports

This module has the following direct dependencies:

– *Application modules:*
`fixpo_dd`, `fixpo_ops`, `transform`, `domains`, `tarjan`, `plai_db`, `plai`, `intermod_entry`, `intermod_ops`, `frontend_driver`, `analyze_driver`, `preprocess_flags`, `analysis_stats`, `ciaopp_log`, `p_dump`, `tarjan_inc`, `plai_db_comparator`, `incanal_deletion`, `incanal_db`, `raw_printer`.

– *System library modules:*
`datafacts_rt`, `native_props`, `aggregates`, `lists`, `terms_vars`, `sort`, `p_unit`, `program_keys`, `p_unit_db`.

– *Internal (engine) modules:*
`term_basic`, `arithmetic`, `atomic_basic`, `basiccontrol`, `exceptions`, `term_compare`, `term_typing`, `debugger_support`, `basic_props`, `hiord_rt`, `runtime_control`, `messages_basic`.

– *Packages:*
`prelude`, `initial`, `condcomp`, `assertions`, `assertions/assertions_basic`, `isomodes`, `datafacts`, `nativeprops`, `ciaopp(plai/notrace)`.

84 Update plai_db after clause deletion (incremental analysis)

Author(s): Isabel Garcia-Contreras (ported from ciaopp-0.8).

This module contains predicates for removing completes and related memo_table data from analysis necessary for recomputing correctly analysis after removing clauses or modifying the answers of the analysis.

84.1 Usage and interface

- **Library usage:**
  `:- use_module(ciaopp(plai/incanal/incanal_deletion)).`
- **Exports:**
  - *Predicates:*
    `remove_clauses_and_related_info_collect_preds/4`, `bottom_up_delete_completes_preds/4`, `bottom_up_delete_complete/5`, `external_calls/4`.

84.2 Documentation on exports

remove_clauses_and_related_info_collect_preds/4: PREDICATE

Usage:
`remove_clauses_and_related_info_collect_preds(ClKeys,AbsInt,Preds,Keys)`

`Cls` is a set of clauses which has have been eliminated from the program. Thus, they must be deleted from the database of clauses used by the analyzer. In addition to the clause, we delete all the memo_tables and memo_lubs for the clauses. In `Preds` and `Keys` we collect the name,arity and keys of predicates for which clauses have been deleted. This will later be used for updating the SCCs.

- *The following properties should hold at call time:*
  `ClKeys` is currently a term which is not a free variable. (`nonvar/1`)
  `AbsInt` is currently a term which is not a free variable. (`nonvar/1`)
  `Preds` is a free variable. (`var/1`)
  `Keys` is a free variable. (`var/1`)
- *The following properties should hold globally:*
  All the calls of the form `remove_clauses_and_related_info_collect_preds(ClKeys,AbsInt,Preds,Keys)` do not fail. (`not_fails/1`)
  All calls of the form `remove_clauses_and_related_info_collect_preds(ClKeys,AbsInt,Preds,Keys)` are deterministic. (`is_det/1`)

bottom_up_delete_completes_preds/4: PREDICATE

Usage: `bottom_up_delete_completes_preds(Keys,SCCs,AbsInt,ExtCompletes)`

`Keys` is the list of keys of predicates for which clauses have been deleted. The completes for such predicates may no longer be accurate enough. Thus we collect the external calls in `ExtCompletes` and erase the corresponding completes from the analysis database.

- *The following properties should hold at call time:*
  `Keys` is currently a term which is not a free variable. (`nonvar/1`)
  `AbsInt` is currently a term which is not a free variable. (`nonvar/1`)
  `ExtCompletes` is a free variable. (`var/1`)
- *The following properties should hold globally:*
  All the calls of the form `bottom_up_delete_completes_preds(Keys,SCCs,AbsInt,ExtCompletes)` do not fail. (`not_fails/1`)

bottom_up_delete_complete/5: PREDICATE

Usage: `bottom_up_delete_complete(SgKey,Id,SCC,AbsInt,ExtCompletes)`
- *The following properties should hold at call time:*
  `SgKey` is currently a term which is not a free variable. (`nonvar/1`)
  `Id` is currently a term which is not a free variable. (`nonvar/1`)
  `SCC` is currently a term which is not a free variable. (`nonvar/1`)
  `AbsInt` is currently a term which is not a free variable. (`nonvar/1`)
  `ExtCompletes` is a free variable. (`var/1`)
- *The following properties should hold globally:*
  All the calls of the form `bottom_up_delete_complete(SgKey,Id,SCC,AbsInt,ExtCompletes)` do not fail. (`not_fails/1`)

external_calls/4: PREDICATE

No further documentation available for this predicate.

84.3 Documentation on imports

This module has the following direct dependencies:
- *Application modules:*
  `tarjan`, `domains`, `plai_db`, `transform`, `plai_db_comparator`.
- *System library modules:*
  `native_props`, `datafacts_rt`, `program_keys`, `sort`, `aggregates`, `lists`.
- *Internal (engine) modules:*
  `term_basic`, `arithmetic`, `atomic_basic`, `basiccontrol`, `exceptions`, `term_compare`, `term_typing`, `debugger_support`, `basic_props`, `hiord_rt`.
- *Packages:*
  `prelude`, `initial`, `condcomp`, `assertions`, `assertions/assertions_basic`, `isomodes`, `nativeprops`, `datafacts`, `ciaopp(plai/notrace)`.

85 Database for incremental analysis

Predicates for mantaining the structures needed for incremental analysis

85.1 Usage and interface

- **Library usage:**
  `:- use_module(ciaopp(plai/incanal/incanal_db)).`
- **Exports:**
  - *Predicates:*
    `init_clkeys/0`, `clean_incremental_db/0`, `change_id_clause/2`, `insert_after_clkey/2`, `save_clause_db/0`, `restore_clause_db/0`, `remove_clauses_pred/2`, `add_all_clauses/1`, `add_new_preds/1`, `get_current_clauses/1`, `get_current_assertions/1`, `add_assertion_source_db/1`, `del_assertion_source_db/1`, `add_changed_registry/4`, `get_changed_registry/4`, `clean_incanal_mod_data/0`, `update_inc_clkey/1`, `all_different_mods/0`, `set_loaded_mods/1`, `loaded_mods/1`, `set_analyzed_mods/1`, `analyzed_mods/1`, `fixpoint_id_/2`, `store_current_fixpoint_id/1`, `restore_fixpoint_id/1`.

85.2 Documentation on exports

init_clkeys/0: PREDICATE

This predicate annotates which is the last clause identifier that was assinged by module program_key. This is needed to give new ids to the clauses that are added to the analysis.

clean_incremental_db/0: PREDICATE

Remove databases for incremental analysis.

change_id_clause/2: PREDICATE

Usage: `change_id_clause(Cl,NCl)`

Changes the Ids through `Cl` so that there is no conflict with other clauses already present in the analysis.

insert_after_clkey/2: PREDICATE

Usage: `insert_after_clkey(Id,Cl)`

- *The following properties should hold at call time:*
  `Id` is an atom. (`atm/1`)

save_clause_db/0: PREDICATE

Stores locally the current source database of ciaopp.

restore_clause_db/0: PREDICATE

Restores from a local copy the source database of ciaopp.

remove_clauses_pred/2: PREDICATE

Usage: `remove_clauses_pred(Pred,Cls)`

Removes from the database all clauses for predicate `Pred` of the form `F/A` and returns them in `Cls`.

add_all_clauses/1: PREDICATE

Usage: `add_all_clauses(Cls)`

Asserts the list of clauses `Cls` to the temporal database.

- *The following properties should hold at call time:*
  `Cls` is a list. (`list/1`)

add_new_preds/1: PREDICATE

Usage: `add_new_preds(Preds)`

This predicates adds predicates which were not defined before.

This predicate is needed for the initialization of the counters for Ids.

- *The following properties should hold at call time:*
  `Preds` is currently a term which is not a free variable. (`nonvar/1`)

get_current_clauses/1: PREDICATE

Usage: `get_current_clauses(Cls)`

Returns all clauses currently present in the source database.

- *The following properties should hold at call time:*
  `Cls` is a free variable. (`var/1`)
- *The following properties should hold upon exit:*
  `Cls` is a list. (`list/1`)

get_current_assertions/1: PREDICATE

Usage: `get_current_assertions(Assrts)`

Returns all assertions currently present in the source database.

- *The following properties should hold at call time:*
  `Assrts` is a free variable. (`var/1`)
- *The following properties should hold upon exit:*
  `Assrts` is a list. (`list/1`)

add_assertion_source_db/1: PREDICATE

No further documentation available for this predicate.

del_assertion_source_db/1: PREDICATE

No further documentation available for this predicate.

add_changed_registry/4: PREDICATE

Usage: `add_changed_registry(SgKey,ImMod,Module,Reg)`

This predicate annotates changes that occurs on the registry of predicate `SgKey` exported in `Module`.

- *The following properties should hold at call time:*

  `SgKey` is an atom. (`atm/1`)

  `Module` is an atom. (`atm/1`)

get_changed_registry/4: PREDICATE

Usage: `get_changed_registry(Key,CMod,Mod,Reg)`

The data of answer of predicate `Key` exported by module `Mod` has changed to `Reg`.

- *Call and exit should be compatible with:*

  `Key` is an atom. (`atm/1`)

  `CMod` is an atom. (`atm/1`)

  `Mod` is an atom. (`atm/1`)

  `Reg` is any term. (`term/1`)

clean_incanal_mod_data/0: PREDICATE

Cleans structures needed for incremental modular analysis.

update_inc_clkey/1: PREDICATE

Usage: `update_inc_clkey(NClKey)`

Updates the value of the last id assigned by the program_keys module.

- *The following properties should hold at call time:*

  `NClKey` is an atom. (`atm/1`)

all_different_mods/0: PREDICATE

Flag that determines if the set of modules to be loaded with `incremental_module/2` is disjoint with the set of the previously loaded The predicate is of type *data*.

set_loaded_mods/1: PREDICATE

Usage: `set_loaded_mods(M)`

Sets the context (as a list of loaded modules) of the incremental analysis.

- *The following properties should hold at call time:*

  `M` is a list. (`list/1`)

loaded_mods/1: PREDICATE

Usage: `loaded_mods(Ms)`

Returns the list of currently loaded modules.

- *The following properties should hold upon exit:*

  `Ms` is a list. (`list/1`)

set_analyzed_mods/1: PREDICATE

Usage: `set_analyzed_mods(M)`

Sets the context (as a list of loaded modules) of the incremental analysis.

- *The following properties should hold at call time:*

  `M` is a list. (`list/1`)

analyzed_mods/1: PREDICATE

Usage: `analyzed_mods(Ms)`

Returns the list of currently loaded modules.

- *The following properties should hold upon exit:*

  `Ms` is a list. (`list/1`)

fixpoint_id_/2: PREDICATE

No further documentation available for this predicate. The predicate is of type *data*.

store_current_fixpoint_id/1: PREDICATE

No further documentation available for this predicate.

restore_fixpoint_id/1: PREDICATE

No further documentation available for this predicate.

85.3 Documentation on imports

This module has the following direct dependencies:

- *Application modules:*

  `domains`, `fixpo_ops`, `preprocess_flags`.
- *System library modules:*

  `datafacts_rt`, `aggregates`, `lists`, `hiordlib`, `pathnames`, `p_unit_db`, `program_keys`.
- *Internal (engine) modules:*

  `term_basic`, `arithmetic`, `atomic_basic`, `basiccontrol`, `exceptions`, `term_compare`, `term_typing`, `debugger_support`, `basic_props`, `hiord_rt`.
- *Packages:*

  `prelude`, `initial`, `condcomp`, `assertions`, `assertions/assertions_basic`, `isomodes`, `datafacts`.

86 Persistent database for incremental analysis

This module handles the maintenance of the persistent database needed for resuming a stopped incremental analysis.

86.1 Usage and interface

- **Library usage:**
  `:- use_module(ciaopp(plai/incanal/incanal_persistent_db)).`
- **Exports:**
  - *Predicates:*
    `set_loaded_module/1`, `reset_persistent_db/0`, `load_persistent_if_needed/2`, `top_level_of_files/2`, `has_dump/2`, `set_inc_persistent/1`, `save_persistent_analysis/0`, `clean_persistent_analysis/0`.
  - *Regular Types:*
    `inc_persistent_status/1`.

86.2 Documentation on exports

set_loaded_module/1: PREDICATE

No further documentation available for this predicate.

reset_persistent_db/0: PREDICATE

Removes previous persistent data of an incremental analysis.

load_persistent_if_needed/2: PREDICATE

Usage: `load_persistent_if_needed(Top,Fs)`

Restores the persistent analysis of tiles in list `Fs` if it existed and the flag `inc_persistent` is on.

- *The following properties should hold at call time:*
  `Top` is an atom. (`atm/1`)
  `Fs` is a list. (`list/1`)

top_level_of_files/2: PREDICATE

No further documentation available for this predicate.

has_dump/2: PREDICATE

No further documentation available for this predicate.

set_inc_persistent/1: PREDICATE

Usage: `set_inc_persistent(St)`

This flags controls the storage of persistent information after the analysis.

– *The following properties should hold at call time:*

`St` Defines whether the analysis should be stored persistently. (`inc_persistent_status/1`)

inc_persistent_status/1: REGTYPE

`inc_persistent_status(X)`

`X` can be:

- `on`: Analysis information will be stored in the same location as the analyzed code (as dump files). This is usefull when analysis scheduling involves loading and unloading modules. To perform an analysis from scratch, these files have to be manually removed.
- `off`: No information is stored.

Usage: `inc_persistent_status(St)`

`St` Defines whether the analysis should be stored persistently.

save_persistent_analysis/0: PREDICATE

Usage:

Makes incremental analysis persistent by saving it to disk.

clean_persistent_analysis/0: PREDICATE

Usage:

Removes the persistent incremental analysis from disk.

86.3 Documentation on imports

This module has the following direct dependencies:

– *Application modules:*

`p_dump`, `analysis_stats`, `preprocess_flags`, `ciaopp_log`.

– *System library modules:*

`datafacts_rt`, `system`, `system_extra`, `pathnames`.

– *Internal (engine) modules:*

`term_basic`, `arithmetic`, `atomic_basic`, `basiccontrol`, `exceptions`, `term_compare`, `term_typing`, `debugger_support`, `basic_props`.

– *Packages:*

`prelude`, `initial`, `condcomp`, `assertions`, `assertions/assertions_basic`, `regtypes`, `datafacts`.

87 Incremental computation of tarjan algorithm

Author(s): Isabel Garcia-Contreras (ported from ciaopp-0.8).

This module contains predicates for computing incrementally the strongly connected components of programs using the tarjan algorithm.

87.1 Usage and interface

- **Library usage:**
  `:- use_module(ciaopp(plai/incanal/tarjan_inc)).`
- **Exports:**
  - *Predicates:*
    `inc_add_source_clauses/3`, `rearrange_tarjan_after_deletion/1`, `tarjan_data/1`, `predicates/1`, `rec/1`, `vertexes/1`, `rec_preds/1`.

87.2 Documentation on exports

inc_add_source_clauses/3: PREDICATE

Usage: `inc_add_source_clauses(Cls,Ds,AbsInt)`

Adds incrementally clauses of list `Cls` with their respective dictionaries in `Ds` for abstract domain `AbsInt` to the source code database, and updates the recursivity of the clauses that may have changed .

– *The following properties should hold at call time:*

`Cls` is a list. (`list/1`)

`Ds` is a list. (`list/1`)

`AbsInt` is an atom. (`atm/1`)

rearrange_tarjan_after_deletion/1: PREDICATE

Usage: `rearrange_tarjan_after_deletion(Preds)`

`Preds` is the list of predicates for which clauses have been deleted. This predicate updates the SCCs after deleting a set of clauses.

– *The following properties should hold at call time:*

`Preds` is currently a term which is not a free variable. (`nonvar/1`)

tarjan_data/1: PREDICATE

Computed recursive classes of the tarjan algorithm. The predicate is of type *data*.

predicates/1: PREDICATE

`predicates(Ps)`

`Ps` is a list of the predicates of the program. The predicate is of type *data*.

rec/1: PREDICATE

`rec(P/A)`

Succeeds if `P/A` is a recursive predicate of the program. The predicate is of type *data*.

vertexes/1: PREDICATE

The vertexes of the graph for performing the tarjan algorithm. The predicate is of type *data*.

rec_preds/1: PREDICATE

Usage: `rec_preds(Ps)`

`Ps` is the list of recursive predicates of the loaded program.

– *The following properties should hold upon exit:*

`Ps` is a list. (`list/1`)

87.3 Documentation on imports

This module has the following direct dependencies:

– *Application modules:*

`tarjan`, `transform`, `plai_db`, `plai`, `analysis_stats`.

– *System library modules:*

`datafacts_rt`, `lists`, `sets`, `aggregates`, `sort`, `idlists`, `program_keys`, `p_unit_db`.

– *Internal (engine) modules:*

`term_basic`, `arithmetic`, `atomic_basic`, `basiccontrol`, `exceptions`, `term_compare`, `term_typing`, `debugger_support`, `basic_props`, `hiord_rt`.

– *Packages:*

`prelude`, `initial`, `condcomp`, `assertions`, `assertions/assertions_basic`, `isomodes`, `datafacts`.

88 Diff algorithm

Author(s): Isabel Garcia-Contreras, Jose F. Morales.

This module implements Eugene Myers' Greedy Difference Algorithm described in 'An O(ND) Difference Algorithm and Its Variations' [Mye86].

The implementation is parametric in the comparison predicate. This predicate has to succeed if two elements are considered equal.

88.1 Complexity

This 'greedy' algorithm that computes the number differences has complexity O((N+M)D) in both time and space. With M and N the lenghts of the input lists and D the number of changes.

88.2 Pseudocode

This pseudocode is copied from the original article, where:

- `a` and `b` are the input lists of lengths `N` and `M`,
- `k = x - y` and is used to label diagonals,
- `V[k]` contains the row index of the endpoint of a furthest reaching path in diagonal k.

```
Constant MAX [0,M+N]

Var V: Array [-MAX.. MAX] of Integer

V[1] = 0

For D = 0 to MAX Do
   For k = -D to D in steps of 2 Do
      If k = -D or (k = D and V[k-1] < V[k+1] Then
    x = V[k+1]
      Else
    x = V[k-1] + 1
    y = x - k
      While (x < N, y < M and a(x+1) = b(y+1) Do
    (x,y) = (x+1,y+1)
      V[k] = x
      If (x >= N and y >= M) Then
        Length of an SES is D
        Stop

Length of an SES is greater than MAX
```

88.3 Example

```
?- diff([a,a,b,c], [b,c,d], '=', Diff).

Diff = [del(0,a),del(0,a),ins(2,d)] ?

yes
?-
```

88.4 Patch

The patch operation applies a list of changes expressed as `diff_item/1` to obtain a new list. Note that given two lists L1, L2, if their Diff is applied to L1, it will be obtained L2:

```
?- L1 = [a,a,b,c], L2 = [b,c,d],
   diff(L1, L2, '=', Diff),
   patch(L1, Diff, L2).


Diff = [del(0,a),del(0,a),ins(2,d)],
L1 = [a,a,b,c],
L2 = [b,c,d] ?
yes
?-
```

88.5 Usage and interface

- **Library usage:**
  `:- use_module(library(diff)).`
- **Exports:**
  - *Predicates:*
    `diff/4`, `patch/3`.
  - *Regular Types:*
    `diff_item/1`.

88.6 Documentation on exports

diff/4: PREDICATE

Usage: `diff(Ls1,Ls2,Compare,Diff)`

`Diff` are the changes needed to transform `Ls1` into `Ls2`.

- *The following properties should hold at call time:*
  - `Ls1` is a list. (`list/1`)
  - `Ls2` is a list. (`list/1`)
- *The following properties should hold upon exit:*
  - `Diff` is a list of `diff_item`s. (`list/2`)

Meta-predicate with arguments: `diff(?,?,pred(2),?)`.

patch/3: PREDICATE

Usage: `patch(Ls,Diff,LNew)`

Apply a list of changes (`Diff`) onto a `Ls`.

- *The following properties should hold at call time:*
  - `Ls` is a list. (`list/1`)
  - `diff_item` is a list of `Diff`s. (`list/2`)

- *The following properties should hold upon exit:*
  `LNew` is a list. (`list/1`)

diff_item/1: REGTYPE

`diff_item(X)`

`X` is an insertion denoted with `ins(Pos, Elem)` meaning that `Elem` has to be inserted in position `Pos`, or `del(Pos, Elem)` meaning that `Elem` has to be removed from position `Pos`. It is defined as:

```
diff_item(ins(P,_1)) :-
    nnegint(P).
diff_item(del(P,_1)) :-
    nnegint(P).
```

Usage: `diff_item(X)`

`X` is a single edition in a sequence (insertion or deletion)

88.7 Documentation on imports

This module has the following direct dependencies:

- *System library modules:*
  `lists`.
- *Internal (engine) modules:*
  `term_basic`, `arithmetic`, `atomic_basic`, `basiccontrol`, `exceptions`, `term_compare`, `term_typing`, `debugger_support`, `basic_props`, `hiord_rt`.
- *Packages:*
  `prelude`, `initial`, `condcomp`, `assertions`, `assertions/assertions_basic`, `hiord`, `regtypes`.

89 fixpo_dd (library)

Stability: [**devel**] Currently the subject of active development and/or research. Functionality may be limited and API and/or functionality may change without warning or deprecation period. Not recommended yet for use in production.

89.1 Usage and interface

- **Library usage:**
  `:- use_module(ciaopp(plai/fixpo_dd)).`
- **Exports:**
  - *Predicates:*
    `query/8`, `init_fixpoint/0`, `cleanup_fixpoint/1`, `entry_to_exit/7`, `$change_list/3`, `proj_to_prime_nr/9`, `proj_to_prime_r/8`, `proj_to_prime/9`, `compute/9`, `td_mark_parents_change_list/2`, `add_change/5`, `mark_parents_change_list_scc/3`, `add_change_scc/6`, `fixpoint_compute_change/9`.

89.2 Documentation on exports

query/8: PREDICATE

Usage: `query(AbsInt,QKey,Query,Qv,RFlag,N,Call,Succ)`

The success pattern of `Query` with `Call` is `Succ` in the analysis domain `AbsInt`. The predicate called is identified by `QKey`, and `RFlag` says if it is recursive or not. The goal `Query` has variables `Qv`, and the call pattern is uniquely identified by `N`.

- *The following properties should hold at call time:*
  `AbsInt` is currently a term which is not a free variable. (`nonvar/1`)
  `QKey` is currently a term which is not a free variable. (`nonvar/1`)
  `Query` is currently a term which is not a free variable. (`nonvar/1`)
  `Qv` is currently a term which is not a free variable. (`nonvar/1`)
  `RFlag` is currently a term which is not a free variable. (`nonvar/1`)
  `Call` is currently a term which is not a free variable. (`nonvar/1`)
  `Succ` is a free variable. (`var/1`)
  `AbsInt` is an atom. (`atm/1`)
  `QKey` is an atom. (`atm/1`)
  `Query` is any term. (`term/1`)
  `Qv` is a list. (`list/1`)
  `RFlag` is an atom. (`atm/1`)
  `N` is any term. (`term/1`)
  `Call` is any term. (`term/1`)
  `Succ` is any term. (`term/1`)
- *The following properties should hold globally:*
  All the calls of the form `query(AbsInt,QKey,Query,Qv,RFlag,N,Call,Succ)` do not fail. (`not_fails/1`)

init_fixpoint/0: PREDICATE

Usage:

Cleans up the database of analysis of temporary information.

cleanup_fixpoint/1: PREDICATE

Usage: `cleanup_fixpoint(AbsInt)`

Cleanups the database of analysis, of both temporary as well as permanent information regarding abstract domain `AbsInt`.

- *The following properties should hold globally:*
  All the calls of the form `cleanup_fixpoint(AbsInt)` do not fail. (`not_fails/1`)

entry_to_exit/7: PREDICATE

Usage: `entry_to_exit(Body,Key,Call,Exit,Vars_u,AbsInt,NewN)`

Traverses the body of a clause (first argument) obtaining the Exit for a given Entry (represented by Call, since it is both the Entry of the clause and the Call for the first subgoal).

- *The following properties should hold at call time:*
  `Body` is currently a term which is not a free variable. (`nonvar/1`)
  `Key` is currently a term which is not a free variable. (`nonvar/1`)
  `Call` is currently a term which is not a free variable. (`nonvar/1`)
  `Exit` is a free variable. (`var/1`)
  `Vars_u` is currently a term which is not a free variable. (`nonvar/1`)
  `AbsInt` is currently a term which is not a free variable. (`nonvar/1`)
  `NewN` is currently a term which is not a free variable. (`nonvar/1`)
  `Key` is an atom. (`atm/1`)
  `Vars_u` is a list. (`list/1`)
  `AbsInt` is an atom. (`atm/1`)
- *The following properties should hold globally:*
  All the calls of the form `entry_to_exit(Body,Key,Call,Exit,Vars_u,AbsInt,NewN)` do not fail. (`not_fails/1`)

$change_list/3: PREDICATE

No further documentation available for this predicate. The predicate is of type *data*.

proj_to_prime_nr/9: PREDICATE

Usage: `proj_to_prime_nr(SgKey,Sg,Sv,Call,Proj,AbsInt,Clid,ListPrime,Id)`

This predicate obtains the list of Prime corresponding to each non recursive clause of Sg for a given Call. It first reads those non recursive clauses by means of a bagof and then proccess each one with a loop. If there is no non recursive clause, the answer will be `['$bottom']`.

> `Clid` may be 0 also (see `fixpo_ops:inexistent/1`.

- *The following properties should hold at call time:*
  `SgKey` is currently a term which is not a free variable. (`nonvar/1`)
  `Sg` is currently a term which is not a free variable. (`nonvar/1`)
  `Sv` is currently a term which is not a free variable. (`nonvar/1`)
  `Call` is currently a term which is not a free variable. (`nonvar/1`)
  `Proj` is currently a term which is not a free variable. (`nonvar/1`)
  `AbsInt` is currently a term which is not a free variable. (`nonvar/1`)
  `Clid` is currently a term which is not a free variable. (`nonvar/1`)
  `ListPrime` is a free variable. (`var/1`)
  `Id` is currently a term which is not a free variable. (`nonvar/1`)
  `SgKey` is an atom. (`atm/1`)
  `Sv` is a list of `var`s. (`list/2`)
  `AbsInt` is an atom. (`atm/1`)
  `plai_db_id(Id)` (`plai_db_id/1`)
- *The following properties should hold upon exit:*
  `ListPrime` is a list. (`list/1`)
- *The following properties should hold globally:*
  All the calls of the form `proj_to_prime_nr(SgKey,Sg,Sv,Call,Proj,AbsInt,Clid,ListPrime,Id)` do not fail. (`not_fails/1`)
  All calls of the form `proj_to_prime_nr(SgKey,Sg,Sv,Call,Proj,AbsInt,Clid,ListPrime,Id)` are deterministic. (`is_det/1`)

proj_to_prime_r/8: PREDICATE

Usage:
- *The following properties should hold globally:*
  All the calls of the form `proj_to_prime_r(Arg1,Arg2,Arg3,Arg4,Arg5,Arg6,Arg7,Arg8)` do not fail. (`not_fails/1`)

proj_to_prime/9: PREDICATE

Usage: `proj_to_prime(Clauses,SgKey,Sg,Sv,Call,Proj,AbsInt,Prime,Id)`
- *The following properties should hold at call time:*
  `Clauses` is currently a term which is not a free variable. (`nonvar/1`)
  `SgKey` is currently a term which is not a free variable. (`nonvar/1`)
  `Sg` is currently a term which is not a free variable. (`nonvar/1`)
  `Sv` is currently a term which is not a free variable. (`nonvar/1`)
  `Call` is currently a term which is not a free variable. (`nonvar/1`)
  `Proj` is currently a term which is not a free variable. (`nonvar/1`)
  `AbsInt` is currently a term which is not a free variable. (`nonvar/1`)
  `Id` is currently a term which is not a free variable. (`nonvar/1`)
  `Clauses` is a list. (`list/1`)

`SgKey` is an atom. (`atm/1`)

`Sv` is a list of `var`s. (`list/2`)

`AbsInt` is an atom. (`atm/1`)

– *The following properties should hold globally:*

All the calls of the form `proj_to_prime(Clauses,SgKey,Sg,Sv,Call,Proj,AbsInt,Prime,Id)` do not fail. (`not_fails/1`)

All calls of the form `proj_to_prime(Clauses,SgKey,Sg,Sv,Call,Proj,AbsInt,Prime,Id)` are deterministic. (`is_det/1`)

compute/9: PREDICATE

Usage: `compute(Clauses,SgKey,Sg,Sv,Proj,AbsInt,TempPrime,Prime,Id)`

It analyzes each recursive clause clause.

– *The following properties should hold at call time:*

`Clauses` is currently a term which is not a free variable. (`nonvar/1`)

`SgKey` is currently a term which is not a free variable. (`nonvar/1`)

`Sg` is currently a term which is not a free variable. (`nonvar/1`)

`Sv` is currently a term which is not a free variable. (`nonvar/1`)

`Proj` is currently a term which is not a free variable. (`nonvar/1`)

`AbsInt` is currently a term which is not a free variable. (`nonvar/1`)

`TempPrime` is currently a term which is not a free variable. (`nonvar/1`)

`Prime` is a free variable. (`var/1`)

`Id` is currently a term which is not a free variable. (`nonvar/1`)

`SgKey` is an atom. (`atm/1`)

`Sv` is a list of `var`s. (`list/2`)

`AbsInt` is an atom. (`atm/1`)

`plai_db_id(Id)` (`plai_db_id/1`)

– *The following properties should hold upon exit:*

`Prime` is currently a term which is not a free variable. (`nonvar/1`)

– *The following properties should hold globally:*

All the calls of the form `compute(Clauses,SgKey,Sg,Sv,Proj,AbsInt,TempPrime,Prime,Id)` do not fail. (`not_fails/1`)

All calls of the form `compute(Clauses,SgKey,Sg,Sv,Proj,AbsInt,TempPrime,Prime,Id)` are deterministic. (`is_det/1`)

td_mark_parents_change_list/2: PREDICATE

Usage: `td_mark_parents_change_list(Parents,AbsInt)`

This complete has changed. So we add the change in the $change_list of all parents.

- *The following properties should hold at call time:*

  `Parents` is currently a term which is not a free variable. (`nonvar/1`)

  `AbsInt` is currently a term which is not a free variable. (`nonvar/1`)

  `Parents` is a list. (`list/1`)

  `AbsInt` is an atom. (`atm/1`)
- *The following properties should hold globally:*

  All the calls of the form `td_mark_parents_change_list(Parents,AbsInt)` do not fail. (`not_fails/1`)

  All calls of the form `td_mark_parents_change_list(Parents,AbsInt)` are deterministic. (`is_det/1`)

add_change/5: PREDICATE

Usage: `add_change(Id,Lit_Key,Literal,Parents,AbsInt)`

`Id`: Id of the complete of the predicate that changed `Lit_Key`: Key of the `Literal` of the complete that needs recomputing. `Literal`: `Literal` with the term `F/A/C/L`. `Parents`: Program points in which the literal is called (0 means an entry) `AbsInt`: Abstract Domain.

- *The following properties should hold at call time:*

  `Id` is currently a term which is not a free variable. (`nonvar/1`)

  `Literal` is currently a term which is not a free variable. (`nonvar/1`)

  `Parents` is currently a term which is not a free variable. (`nonvar/1`)

  `AbsInt` is currently a term which is not a free variable. (`nonvar/1`)

  `plai_db_id(Id)` (`plai_db_id/1`)

  `Lit_Key` is an atom. (`atm/1`)
- *The following properties should hold globally:*

  All the calls of the form `add_change(Id,Lit_Key,Literal,Parents,AbsInt)` do not fail. (`not_fails/1`)

mark_parents_change_list_scc/3: PREDICATE

Usage: `mark_parents_change_list_scc(Parents,SCC,AbsInt)`

This complete has changed. So we add the change in the $change_list of all parents. If the parent is in the same SCC then we recursively mark its parents as well.

- *The following properties should hold at call time:*

  `Parents` is currently a term which is not a free variable. (`nonvar/1`)

  `SCC` is currently a term which is not a free variable. (`nonvar/1`)

  `AbsInt` is currently a term which is not a free variable. (`nonvar/1`)

  `Parents` is a list. (`list/1`)

  `SCC` is a list. (`list/1`)

  `AbsInt` is an atom. (`atm/1`)
- *The following properties should hold globally:*

  All the calls of the form `mark_parents_change_list_scc(Parents,SCC,AbsInt)` do not fail. (`not_fails/1`)

  All calls of the form `mark_parents_change_list_scc(Parents,SCC,AbsInt)` are deterministic. (`is_det/1`)

add_change_scc/6: PREDICATE

Usage: `add_change_scc(Id,Lit_Key,Literal,Parents,SCC,AbsInt)`

`Lit_Key` may be free when a clause is marked to analyze from the begining (see second clause of `compute_change`).

- *The following properties should hold at call time:*
  - `Id` is currently a term which is not a free variable. (`nonvar/1`)
  - `Literal` is currently a term which is not a free variable. (`nonvar/1`)
  - `Parents` is currently a term which is not a free variable. (`nonvar/1`)
  - `SCC` is currently a term which is not a free variable. (`nonvar/1`)
  - `AbsInt` is currently a term which is not a free variable. (`nonvar/1`)
  - `Id` is an integer. (`int/1`)
  - `Lit_Key` is an atom. (`atm/1`)
  - `Literal` is any term. (`term/1`)
  - `Parents` is a list. (`list/1`)
  - `SCC` is a list. (`list/1`)
  - `AbsInt` is an atom. (`atm/1`)
- *The following properties should hold globally:*
  - All the calls of the form `add_change_scc(Id,Lit_Key,Literal,Parents,SCC,AbsInt)` do not fail. (`not_fails/1`)
  - All calls of the form `add_change_scc(Id,Lit_Key,Literal,Parents,SCC,AbsInt)` are deterministic. (`is_det/1`)

fixpoint_compute_change/9: PREDICATE

Usage: `fixpoint_compute_change(Changes,SgKey,Sg,Sv,Proj,AbsInt,TempPrime,Prime,Id)`

It obtains the Prime for the recursive predicate Sg with Call (which has been assigned to node Id), and the list of nodes it depends on In doing this it performs an iteration over the recursive clauses of Sg by calling to compute_change/13 and then checks if the fixpoint has been reached by calling to fixpoint/13. Fixpoint is reached if either NewList is empty (it means that Id does not depend on any node) or if Flag is a variable (it means that the information has not been modified within the iteration) LEntryInf is the list of (Entry,ExtraInfo) couples for each Clause. It will be computed in the first iteration and used in subsequent ones

- *The following properties should hold at call time:*
  - `Changes` is currently a term which is not a free variable. (`nonvar/1`)
  - `SgKey` is currently a term which is not a free variable. (`nonvar/1`)
  - `Sg` is currently a term which is not a free variable. (`nonvar/1`)
  - `Sv` is currently a term which is not a free variable. (`nonvar/1`)
  - `Proj` is currently a term which is not a free variable. (`nonvar/1`)
  - `AbsInt` is currently a term which is not a free variable. (`nonvar/1`)
  - `TempPrime` is currently a term which is not a free variable. (`nonvar/1`)
  - `Prime` is a free variable. (`var/1`)
  - `Id` is currently a term which is not a free variable. (`nonvar/1`)
  - `SgKey` is an atom. (`atm/1`)

`Sv` is a list of `vars`. (`list/2`)

`AbsInt` is an atom. (`atm/1`)

`plai_db_id(Id)` (`plai_db_id/1`)

- *The following properties should hold upon exit:*

  `Prime` is currently a term which is not a free variable. (`nonvar/1`)

- *The following properties should hold globally:*

  All the calls of the form `fixpoint_compute_change(Changes,SgKey,Sg,Sv,Proj,AbsInt,TempPrime,Prime,Id)` do not fail. (`not_fails/1`)

  All calls of the form `fixpoint_compute_change(Changes,SgKey,Sg,Sv,Proj,AbsInt,TempPrime,Prime,Id)` are deterministic. (`is_det/1`)

89.3 Documentation on imports

This module has the following direct dependencies:

- *Application modules:*

  `preprocess_flags`, `fixpo_ops`, `domains`, `plai_db`, `transform`, `apply_assertions_old`, `apply_assertions`, `global_control`, `tarjan`, `plai_db_comparator`.

- *System library modules:*

  `native_props`, `datafacts_rt`, `write`, `lists`, `aggregates`, `terms_vars`, `sort`, `sets`, `program_keys`, `messages`.

- *Internal (engine) modules:*

  `term_basic`, `arithmetic`, `atomic_basic`, `basiccontrol`, `exceptions`, `term_compare`, `term_typing`, `debugger_support`, `basic_props`, `hiord_rt`, `messages_basic`.

- *Packages:*

  `prelude`, `initial`, `condcomp`, `assertions`, `assertions/assertions_basic`, `nativeprops`, `datafacts`, `isomodes`, `regtypes`, `ciaopp(ciaopp_options)`, `.(notrace)`, `spec(no_debug)`.

90 The Preprocessing Unit Component

This component is in charge of processing the .asr files, gathering together the code for the preprocessing unit of the current module. The preprocessing unit is the code of the current module together with the set of assertions/properties which allows a precise interpretation of the assertions present for the current module. It serves the rest of the precompiler current module clauses, assertions, (definitions of) properties related, and characteristics of the current module predicates (e.g.: initialization directives, whether a predicate is a meta-predicate, whether it is a "builtin", etc.).

91 The Preprocessing Unit

This component handles data structures for the code of the current module and for all information about it that may be necessary to precompile that module. Since CiaoPP works in a per-module fashion, when preprocessing one module the modules imported by it may not be accessible. However, information on imported predicates may be essential for some manipulations of the current module: e.g., for analyzing it with greater precision. For this reason, information about exported predicates of a module is stored in separate cache files. The cache files of modules imported by the current module are then loaded together with the file of the current module. All the information stored together with the current module is called the preprocessing unit of the current module.

Cache files are used to save assertions that refer to exported predicates, properties exported or properties that are used in assertions for exported predicates, and properties used in the definiton of other relevant properties. (The exact definition of what is cached and how the preprocessing unit is obtained from this caches is given in the chapter for `p_asr`). This facilitates the use within CiaoPP of all information that may be relevant about exported predicates that may be imported into other modules. This information from (user) assertions is extracted from the module code.

Other information that may be relevant is that inferred by CiaoPP on its own. The information from analysis that refers to exported predicates of a module is also cached in separate files, so that the preprocessing of other modules that import such predicates can use the information. Currently, separate files are used for user supplied assertions (the so called .asr file) and for information from analysis (the so called .abs file). Module `p_asr` is in charge of the .asr file and module `p_abs` is in charge of the .abs files.

More information concerning the current module that this component serves includes: initialization directives, entry assertions that may need analysis, whether a predicate is a meta-predicate or a dynamic predicate, whether it is a `"builtin"`, a property, or a native property, etc. In the sections below we explain what is understood by builtin predicate and native property. The complete functionality offered by this component is explained in the following chapters.

91.1 The preprocessing unit

In the following, related files refer to files defining modules from which the current module imports a predicate, whether via direct importation or via re-exportation by an imported module. The *preprocessing unit* consists of:

1. the code (and directives) of the current module,
2. the assertions of the current module,
3. the (imported) properties of related files which are used in the previous assertions,
4. the assertions of related files for predicates imported by the current module,
5. the properties of related files (or imported by them) which are used in the previous assertions,
6. the properties transitively used in the definitions of all the previous properties, whichever file they are defined.

Note that once the preprocessing unit of the current module is loaded, every assertion for a predicate of the current module or imported into the current module can be correctly and completely interpreted, since all the properties that may appear in such assertions (or transitively used in the definitions of such properties) are available. This is the main reason for including the preprocessing unit of the current module instead of simply loading the current module itself alone (and some assertions for imported predicates, probably).

91.2 Builtin predicates

A predicate is builtin if there is information about it built into one of the components of CiaoPP. This is the case, for example, for analyzers. Usually, these predicates are ISO predicates, whose semantics conform with the standard, and thererefore, can be implemented into CiaoPP once and for all. The names used to refer to these predicates are thus usually also the ISO names.

However, in a modular system like Ciao, such builtin predicates may be defined in a module, and can be used (imported) or not by the current module. Whether the predicate is visible or not to the current module is solved by looking at the module interface of the current module. The exact name of the predicate (at the source language level) can be known; for example, ISO `var/1` in CiaoPP is `term_typing:var/1`. This name could also be built into CiaoPP, but this would tight CiaoPP to the particular module structure of the library of a system (Ciao, in this case). Instead, there is a dynamic scheme that makes CiaoPP more flexible: To indicate that a given predicate in a given module corresponds to a builtin predicate, the following kind of assertions are used:

```
:- pred var(X) + native(var(X)).
```

This assertion in module `term_typing` tells CiaoPP that predicate `term_typing:var(X)` corresponds to the predicate builtin into CiaoPP as `var(X)`. Note that the name `var/1` should be used all throughtout CiaoPP to identify that very same predicate (the one in ISO by that name, in this case). This is mandatory.

Note also that with the above scheme, the only real predicate name that is built into CiaoPP is that for the property `native(Pred,Name)`, which currently is precisely `native(Pred,Name)` (which is currently defined in library module `engine(basic_props)`).

91.3 Native properties

A property in Ciao is a predicate. Certain properties are used within CiaoPP in a native way, which means that some component understands the property natively (for example, an analysis for groundness understands the property of being ground). These properties are called *native properties* .

The same problem above for builtin predicates occurs also for native properties, and the same solution has been given to the problem also in this case. To indicate that a given predicate corresponds to a property understood natively by CiaoPP the following kind of assertions are used:

```
:- prop var(X) + native(free(X)).
```

This is interpreted by CiaoPP so that predicate `term_typing:var(X)` corresponds to the native property identified in CiaoPP as `free(X)`. Note that the name `free/1` should be used all throughtout CiaoPP to identify that property. The names of native properties are defined in `native:native/1`.

The (default) predicates that are used as source language level counterparts of the native property names are gathered together in a Ciao library module that currently is `library(assertions/native_props)`.

There is an scheme for translating predicate property names to native property names upon input of the current module, and native property names back to predicate property names upon output that is described in the chapter about module `domains`.

91.4 Organization of the code

The organization of the code of this component is as follows:

Module `p_unit` is in charge of supporting all other CiaoPP components when they require information about the program. Module `p_asr` is in charge of processing the .asr files and gathering together the preprocessing unit for the current module. Module `assrt_norm` is used to normalize assertions. Module `p_abs` is in charge of reading .abs files and writing them. Module `assrt_db` maintains the database of assertions, module `clause_db` the database of clauses, and module `itf_db` the database of information related to the module interface of the current module.

Each of the modules is documented in the following chapters.

92 Preprocessing Unit Information Server

Author(s): The Ciao Development Team.

This module loads the preprocessing unit of a file to be preprocessed and serves all the information related to it to the rest of CiaoPP, including (but not limited to) the source code.

92.1 Usage and interface

- **Library usage:**
  `:- use_module(library(compiler/p_unit)).`
- **Exports:**
  - *Predicates:*
    `preprocessing_unit/3`, `program/2`, `filtered_program_clauses/3`, `replace_program/2`, `native_to_prop/2`, `prop_to_native/2`, `native_to_props_visible/2`, `dynamic_or_unknown_predicate/1`, `get_assertion/2`, `add_assertions/1`, `add_assertion/1`, `add_directive/1`, `erase_directive/1`, `type_of_directive/2`, `pr_key_clean/0`, `pr_key_add/1`, `pr_key_get/1`, `add_defined_pred/2`, `new_predicate/3`, `predicate_names/1`, `multifile_predicate_names/1`, `inject_output_package/1`, `add_output_package/1`, `get_output_package/1`, `add_output_operator/3`, `get_output_operator/3`, `add_comment/1`, `get_comment/1`, `cleanup_comment_db/0`, `cleanup_commented_assrt/0`, `get_commented_assertion/2`, `add_commented_assertions/1`, `add_commented_assertion/1`, `cleanup_punit/0`, `get_call_from_call_assrt/7`, `get_pred_mod_defined/2`, `assertion_set_status/3`, `assertion_set_head/3`, `assertion_set_calls/3`, `assertion_set_success/3`, `assertion_set_comp/3`, `cleanup_libcache/0`, `loaded_libcache/0`, `load_libcache/1`, `gen_libcache/1`.
  - *Multifiles:*
    `define_flag/3`, `hook_compact_global_prop/2`, `hook_compact_calls_prop/2`, `hook_native_prop/2`, `hook_pp_info_clause/3`, `hook_pp_info_lit/4`, `hook_legal_regtype/1`, `hook_insert_regtype/2`, `hook_post_init_regtypes/0`, `hook_cleanup_lib_regtypes/0`, `hook_restore_lib_regtypes/1`, `hook_save_lib_regtypes/1`.

92.2 Documentation on exports

preprocessing_unit/3: PREDICATE

Usage 1: `preprocessing_unit(Fs,Ms,E)`

Loads the preprocessing unit of `Fs` defining `Ms`.

- *The following properties should hold at call time:*
  `Fs` is a list of `filenames`. (`list/2`)
- *The following properties should hold upon exit:*
  `Ms` is a list of `moddescs`. (`list/2`)
  `E` is an atom indicating yes/no. (`switch/1`)

Usage 2: `preprocessing_unit(F,M,E)`

Loads the preprocessing unit of `F` defining `M`.

- *The following properties should hold at call time:*
  `F` is an atom describing the name of a file. (`filename/1`)

- *The following properties should hold upon exit:*
  `M` is a module descriptor. (`moddesc/1`)
  `E` is an atom indicating yes/no. (`switch/1`)

program/2: PREDICATE

Usage: `program(P,D)`

`P` are the clauses (no directives) of the current module and `D` their dictionaries.

- *The following properties should hold at call time:*
  `P` is a free variable. (`var/1`)
  `D` is a free variable. (`var/1`)
- *The following properties should hold upon exit:*
  `P` is a list of `clause`s. (`list/2`)
  `D` is a list of `varnamedict`s. (`list/2`)

filtered_program_clauses/3: PREDICATE

Usage: `filtered_program_clauses(Mods,P,D)`

`P` are the clauses of module `Mod` and `D` their dictionaries.

- *The following properties should hold at call time:*
  `Mods` is currently a term which is not a free variable. (`nonvar/1`)
  `P` is a free variable. (`var/1`)
  `D` is a free variable. (`var/1`)
  `Mods` is a list. (`list/1`)
- *The following properties should hold upon exit:*
  `Mods` is a list. (`list/1`)
  `P` is a list of `clause`s. (`list/2`)
  `D` is a list of `varnamedict`s. (`list/2`)

replace_program/2: PREDICATE

Usage: `replace_program(P,D)`

The database holding the program is updated by first deleting its contents and then adding the clauses in `P` and dictionaries in `D`.

- *The following properties should hold at call time:*
  `P` is a list of `clause`s. (`list/2`)
  `D` is a dictionary of variable names. (`varnamedict/1`)

native_to_prop/2: PREDICATE

Usage: `native_to_prop(NProp,Prop)`

Obtain the user predicate (lit) `Prop` that corresponds to the native property (lit) `NProp`.

- *The following properties should hold at call time:*
  `NProp` is currently a term which is not a free variable. (`nonvar/1`)
  `Prop` is a free variable. (`var/1`)

prop_to_native/2: PREDICATE

Usage: `prop_to_native(Prop,NProp)`

Obtain the native property (lit) `NProp` that corresponds to the (lit) `Prop` user predicate.

- *The following properties should hold at call time:*

  `Prop` is currently a term which is not a free variable. (`nonvar/1`)

  `NProp` is a free variable. (`var/1`)

native_to_props_visible/2: PREDICATE

Usage: `native_to_props_visible(Props,Goals)`

Maps native `Props` into their corresponding `Goals` visible in the current module.

- *The following properties should hold upon exit:*

  `Props` is a list of `cgoals`. (`list/2`)

  `Goals` is a list of `cgoals`. (`list/2`)

dynamic_or_unknown_predicate/1: PREDICATE

Usage: `dynamic_or_unknown_predicate(Goal)`

`Goal` is an atom for a predicate such that all its clauses might not be available or may change in the program unit.

get_assertion/2: PREDICATE

No further documentation available for this predicate.

add_assertions/1: PREDICATE

Usage: `add_assertions(AssrtList)`

Add assertions list `AssrtList` to internal DB.

- *The following properties should hold at call time:*

  `AssrtList` is a list. (`list/1`)

add_assertion/1: PREDICATE

Usage: `add_assertion(Assrt)`

Add assertion `Assrt` to internal DB.

add_directive/1: PREDICATE

Usage: `add_directive(C)`

The directive `C` is added to the program data base. This directive will be considered as read from the program, i.e., analizers, transformations and output will use/show it.

- *The following properties should hold at call time:*

  `C` is any term. (`term/1`)

erase_directive/1: PREDICATE

Usage: `erase_directive(D)`

Erase directive `D` (previously added with `add_directive/1`

- *The following properties should hold at call time:*
  `D` is any term. (`term/1`)

type_of_directive/2: PREDICATE

Usage: `type_of_directive(Type,Body)`

There is a directive of the form `:- Type Body` (of arity one).

pr_key_clean/0: PREDICATE

Usage:

Removes all information about predicate order.

- *The following properties should hold globally:*
  Calls of the form `pr_key_clean` are deterministic. (`det/1`)

pr_key_add/1: PREDICATE

Usage: `pr_key_add(K)`

Add a predicate key (once)

- *The following properties should hold upon exit:*
  `K` is a term which represents a goal, i.e., an atom or a structure. (`cgoal/1`)
- *The following properties should hold globally:*
  Calls of the form `pr_key_add(K)` are deterministic. (`det/1`)

pr_key_get/1: PREDICATE

Usage: `pr_key_get(K)`

Current predicate keys

- *The following properties should hold upon exit:*
  `K` is a term which represents a goal, i.e., an atom or a structure. (`cgoal/1`)
- *The following properties should hold globally:*
  Calls of the form `pr_key_get(K)` are multi-deterministic. (`multi/1`)

add_defined_pred/2: PREDICATE

Usage: `add_defined_pred(ClKey,M)`

Add the necessary data in itf_db (and `pr_key/1`) to define the predicate `ClKey` in the module `M`.

- *The following properties should hold at call time:*
  `ClKey` is any term. (`term/1`)
  `M` is an atom. (`atm/1`)
- *The following properties should hold globally:*
  Calls of the form `add_defined_pred(ClKey,M)` are deterministic. (`det/1`)

new_predicate/3: PREDICATE

Usage: `new_predicate(F,A,NewF)`

Checks wether there is a predicate `F/A` in the program and returns `NewF` so that there is no predicate `NewF/A` in the program.

- *The following properties should hold at call time:*
  `F` is currently a term which is not a free variable. (`nonvar/1`)
  `A` is currently a term which is not a free variable. (`nonvar/1`)
  `NewF` is a free variable. (`var/1`)
- *The following properties should hold globally:*
  Calls of the form `new_predicate(F,A,NewF)` are deterministic. (`det/1`)

predicate_names/1: PREDICATE

Usage:

Returns the specs of predicates defined in the current punit.

- *Call and exit should be compatible with:*
  `Arg1` is a list. (`list/1`)
- *The following properties should hold at call time:*
  `Arg1` is a free variable. (`var/1`)

multifile_predicate_names/1: PREDICATE

Usage:

Returns the specs of predicates defined in the current punit.

- *Call and exit should be compatible with:*
  `Arg1` is a list. (`list/1`)
- *The following properties should hold at call time:*
  `Arg1` is a free variable. (`var/1`)

inject_output_package/1: PREDICATE

Usage: `inject_output_package(A)`

Inject the package `A` in the current program database (including the output package list). The necesary information from these packages is loaded for correct treatment and unexpansion.

- *The following properties should hold at call time:*
  `A` is an atom. (`atm/1`)

add_output_package/1: PREDICATE

Usage: `add_output_package(A)`

Add the package `A` to the output packages list

- *The following properties should hold at call time:*
  `A` is an atom. (`atm/1`)

get_output_package/1: PREDICATE

Usage: `get_output_package(X)`

`X` is a package that will be included in the output (`module` directive).

add_output_operator/3: PREDICATE

Usage: `add_output_operator(Prec,Type,OP)`

Define an operator for output (same arguments as in `op/3`).

– *The following properties should hold at call time:*

`Prec` is an integer. (`int/1`)

`Type` is an atom. (`atm/1`)

`OP` is an atom or a list of atoms. (`atm_or_atm_list/1`)

get_output_operator/3: PREDICATE

Usage: `get_output_operator(Pred,Type,OP)`

Enumerate output operators (same arguments as in `op/3`).

– *The following properties should hold upon exit:*

`Pred` is an integer. (`int/1`)

`Type` is an atom. (`atm/1`)

`OP` is an atom or a list of atoms. (`atm_or_atm_list/1`)

add_comment/1: PREDICATE

Usage: `add_comment(Comment)`

Add comment to the comment db

– *The following properties should hold at call time:*

`Comment` is a string (a list of character codes). (`string/1`)

get_comment/1: PREDICATE

Usage: `get_comment(Comment)`

Retrieves comments from the comment db

– *The following properties should hold upon exit:*

`Comment` is a string (a list of character codes). (`string/1`)

cleanup_comment_db/0: PREDICATE

Usage:

Cleans up the comment db

cleanup_commented_assrt/0: PREDICATE

No further documentation available for this predicate.

get_commented_assertion/2: PREDICATE

No further documentation available for this predicate.

add_commented_assertions/1: PREDICATE

Usage: `add_commented_assertions(A)`

Add the assertions list `A` to the commented assertions DB.

- *The following properties should hold at call time:*
  `A` is a list. (`list/1`)

add_commented_assertion/1: PREDICATE

Usage: `add_commented_assertion(A)`

Add assertion `A` to the commented assertions DB.

- *The following properties should hold at call time:*
  `A` is any term. (`term/1`)

cleanup_punit/0: PREDICATE

Usage:

Clean up all facts that p_unit asserts.

get_call_from_call_assrt/7: PREDICATE

Usage: `get_call_from_call_assrt(Sg,M,Status,Call,Source,LB,LE)`

Returns in `Call`, upon backtracking call patterns from calls assertions related to `Sg`, in module `M`. Also takes care of disjunctions.

get_pred_mod_defined/2: PREDICATE

Usage: `get_pred_mod_defined(Sg,Mod)`

Returns the module where a predicate given by goal `Sg` is defined. If the `Sg` is multifile, the modules where it is defined are enumerated.

- *The following properties should hold at call time:*
  `Sg` is currently a term which is not a free variable. (`nonvar/1`)
  `Mod` is a free variable. (`var/1`)
- *The following properties should hold globally:*
  Calls of the form `get_pred_mod_defined(Sg,Mod)` are non-deterministic. (`nondet/1`)

assertion_set_status/3: PREDICATE

No further documentation available for this predicate.

assertion_set_head/3: PREDICATE

No further documentation available for this predicate.

assertion_set_calls/3: PREDICATE

No further documentation available for this predicate.

assertion_set_success/3: PREDICATE

No further documentation available for this predicate.

assertion_set_comp/3: PREDICATE

No further documentation available for this predicate.

cleanup_libcache/0: PREDICATE

Usage:

Cleans up libcache.

loaded_libcache/0: PREDICATE

Usage:

Checks if the libcache is loaded

load_libcache/1: PREDICATE

Usage: `load_libcache(DataDir)`

Load the preprocessed library cache (specified in the `core/Manifest/core.libcache.pl` module.

gen_libcache/1: PREDICATE

Usage: `gen_libcache(DataDir)`

Generate the preprocessed library cache (specified in the `core/Manifest/core.libcache.pl` module. .

type_of_goal/2: (UNDOC_REEXPORT)

Imported from `p_unit_basic` (see the corresponding documentation for details).

92.3 Documentation on multifiles

define_flag/3: PREDICATE

Usage: `define_flag(Flag,FlagValues,Default)`

- *The following properties hold upon exit:*

  `Flag` is an atom. (`atm/1`)

  Define the valid flag values (`flag_values/1`)

The predicate is *multifile*.

hook_compact_global_prop/2: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

hook_compact_calls_prop/2: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

hook_native_prop/2: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

hook_pp_info_clause/3: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

hook_pp_info_lit/4: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

hook_legal_regtype/1: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

hook_insert_regtype/2: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

hook_post_init_regtypes/0: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

hook_cleanup_lib_regtypes/0: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

hook_restore_lib_regtypes/1: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

hook_save_lib_regtypes/1: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

92.4 Documentation on imports

This module has the following direct dependencies:

- *System library modules:*
  `datafacts_rt`, `native_props`, `c_itf_props`, `messages`, `aggregates`, `c_itf`, `hiordlib`, `lists`, `vndict`, `terms_check`, `assrt_lib`, `p_unit_db`, `program_keys`, `native`, `p_unit_basic`, `p_asr`, `tr_syntax`, `unexpand`, `streams`, `pathnames`, `bundle_paths`, `datadir`.
- *Internal (engine) modules:*
  `term_basic`, `arithmetic`, `atomic_basic`, `basiccontrol`, `exceptions`, `term_compare`, `term_typing`, `debugger_support`, `basic_props`, `hiord_rt`, `internals`, `runtime_control`, `stream_basic`.
- *Packages:*
  `prelude`, `initial`, `condcomp`, `assertions`, `assertions/assertions_basic`, `basicmodes`, `regtypes`, `datafacts`, `hiord`, `nativeprops`, `define_flag`, `compiler/p_unit/p_unit_argnames`, `argnames`.

93 native (library)

This module has the basic procedures for understanding predicates as builtins and properties as native properties.

93.1 Usage and interface

- **Library usage:**
  `:- use_module(library(compiler/p_unit/native)).`
- **Exports:**
  - *Predicates:*
    `builtin/2`, `native_prop_map/3`, `wam_builtin/2`.
  - *Regular Types:*
    `native_prop_term/1`.
  - *Multifiles:*
    `hook_compact_global_prop/2`, `hook_compact_calls_prop/2`, `hook_native_prop/2`, `hook_pp_info_clause/3`, `hook_pp_info_lit/4`, `hook_legal_regtype/1`, `hook_insert_regtype/2`, `hook_post_init_regtypes/0`, `hook_cleanup_lib_regtypes/0`, `hook_restore_lib_regtypes/1`, `hook_save_lib_regtypes/1`.

93.2 Documentation on exports

builtin/2: PREDICATE

checks that a user level (property) predicate is declared native and translates it into the internal representation for that native (builtin or property) predicate.

native_prop_map/3: PREDICATE

`native_prop_map(Prop,P,Vars)`

`Prop` is a property `P`(`Vars`) representing a mapping `map`(`Vars`,`P`)

native_prop_term/1: REGTYPE

Defined as

```
native_prop_term(Prop) :-
    native:native(Prop).
native_prop_term(regtype(Prop)) :-
    cgoal(Prop).
```

Usage: `native_prop_term(Prop)`

`Prop` identifies a native property.

wam_builtin/2: PREDICATE

No further documentation available for this predicate.

93.3 Documentation on multifiles

hook_compact_global_prop/2: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

hook_compact_calls_prop/2: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

hook_native_prop/2: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

hook_pp_info_clause/3: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

hook_pp_info_lit/4: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

hook_legal_regtype/1: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

hook_insert_regtype/2: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

hook_post_init_regtypes/0: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

hook_cleanup_lib_regtypes/0: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

hook_restore_lib_regtypes/1: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

hook_save_lib_regtypes/1: PREDICATE
No further documentation available for this predicate. The predicate is *multifile*.

93.4 Documentation on internals

native/1: REGTYPE

`native(Prop)`

`Prop` is (the internal representation of) a native property: one that CiaoPP knows natively how to manipulate.

```
native(covered(V,L)) :-
    var(V),
    list(var,L).
native(free(L)) :-
    vlist(L).
native(ground(L)) :-
    vlist(L).
native(indep(L)) :-
    list(list(var),L).
native(instance(V,T)) :-
    var(V),
    nonvar(T).
native(linear(L)) :-
    vlist(L).
native(not_free(L)) :-
    vlist(L).
native(not_ground(L)) :-
    vlist(L).
native(sharing(L)) :-
    list(list(var),L).
native(sharing(Xs,Xss)) :-
    list(var,Xs),
    list(list(var),Xss).
native(det).
native(semidet).
native(multi).
native(nondet).
native(covered).
native(fails).
native(is_det).
native(mut_exclusive).
native(non_det).
native(not_covered).
native(not_fails).
native(not_mut_exclusive).
native(sideff(X)) :-
    member(X,[pure,soft,hard]).
native(steps(C)) :-
    cost_expression(C).
native(steps_lb(C)) :-
    cost_expression(C).
native(steps_ub(C)) :-
    cost_expression(C).
native(steps_o(C)) :-
    cost_expression(C).
```

```
native(cost(C)) :-
    cost_expression(C).
native(terminates).
native(size(C)) :-
    cost_expression(C).
native(size(A,B)) :-
    var(A),
    cost_expression(B).
native(size_lb(C)) :-
    cost_expression(C).
native(size_ub(C)) :-
    cost_expression(C).
native(size_o(C)) :-
    cost_expression(C).
native(rsize(A,_B)) :-
    var(A).
native(cardinality(_N,_M)).
native(costb(_R,_N,_M)).
native(X) :-
    native(X,X).
```

vlist/1: REGTYPE

```
vlist(L) :-
    var(L).
vlist(L) :-
    list(var,L).
```

93.5 Documentation on imports

This module has the following direct dependencies:

- *System library modules:*
  native_props, lists.
- *Internal (engine) modules:*
  term_basic, arithmetic, atomic_basic, basiccontrol, exceptions, term_compare, term_typing, debugger_support, basic_props, hiord_rt, runtime_control.
- *Packages:*
  prelude, initial, condcomp, assertions, assertions/assertions_basic, regtypes, hiord, nortchecks, nativeprops.

94 program_keys (library)

This module handles the format of the structures used to represent the program. It is a list of clauses, where each clause is identified by a key, and the clause body is a conjunction of literals, where each literal is also identified by a key.

When creating a new clause for a predicate `H` in the program, the following goal will give you a clause in the correct format:

```
clause_key(H,ClId), rewrite_source_clause(clause(H,Body),ClId,Clause),
```

94.1 Usage and interface

- **Library usage:**

  `:- use_module(library(compiler/p_unit/program_keys)).`
- **Exports:**
  - *Predicates:*

    `cleanup_program_keys/0`, `clause_key/2`, `clauses_keys/2`, `clause_number/2`, `first_key/2`, `last_clause/1`, `make_atom/2`, `inverse_rewrite_source_program_cl/2`, `inverse_rewrite_source_body/2`, `rewrite_source_all_clauses/2`, `rewrite_source_clause/3`, `decode_litkey/5`, `decode_clkey/4`, `decode_entrykey/4`, `decode_predkey/3`, `is_clkey/1`, `is_litkey/1`, `is_entrykey/1`, `is_directive/3`, `is_clause/4`, `orig_clause_id/2`, `lit_ppkey/3`, `clid_of_atomid/2`, `counter/3`, `get_predkey/3`, `get_predkeys/2`, `predkey_from_sg/2`, `get_clkey/4`, `get_litkey/5`.
  - *Properties:*

    `clausebody/1`.
  - *Regular Types:*

    `clause/1`, `literal_ppkey/1`, `litkey/1`, `clkey/1`, `predkey/1`.

94.2 Documentation on exports

cleanup_program_keys/0: PREDICATE

Cleanups internal database.

clause_key/2: PREDICATE

Usage: `clause_key(H,ClId)`

`ClId` is the clause key that an additional clause of predicate `H` should have.

- *The following properties should hold at call time:*

  `H` is a clause head identifier. (`head/1`)

  `ClId` is a free variable. (`var/1`)
- *The following properties should hold upon exit:*

  `ClId` is an atom that uniquely identifies a program clause. (`clkey/1`)

clauses_keys/2: PREDICATE

Usage: `clauses_keys(H,ClIds)`

`ClIds` are the clause keys of all clauses already known for predicate `H`.

- *The following properties should hold at call time:*
  `H` is a term which represents a goal, i.e., an atom or a structure. (`cgoal/1`)
  `ClIds` is a free variable. (`var/1`)
- *The following properties should hold upon exit:*
  `H` is a term which represents a goal, i.e., an atom or a structure. (`cgoal/1`)
  `ClIds` is a list of `clkeys`. (`list/2`)

clause_number/2: PREDICATE

No further documentation available for this predicate.

first_key/2: PREDICATE

No further documentation available for this predicate.

last_clause/1: PREDICATE

Usage: `last_clause(ClId)`

`ClId` is the clause key of the last known clause of the predicate to which it corresponds.

- *The following properties should hold at call time:*
  `ClId` is an atom that uniquely identifies a program clause. (`clkey/1`)

make_atom/2: PREDICATE

No further documentation available for this predicate.

inverse_rewrite_source_program_cl/2: PREDICATE

No further documentation available for this predicate.

inverse_rewrite_source_body/2: PREDICATE

No further documentation available for this predicate.

rewrite_source_all_clauses/2: PREDICATE

`rewrite_source_all_clauses(Clauses,NewClauses)`

`NewClauses` is the result of applying the format transformation to the list of clauses `Clauses`.

rewrite_source_clause/3: PREDICATE

Usage: `rewrite_source_clause(Clause0,ClId,Clause)`

`Clause` is an structure representing the clause `Clause0` identified by `ClId` (does nothing for directives).

- *The following properties should hold at call time:*

  `clause(Clause0)` (`clause/1`)

  `ClId` is an atom that uniquely identifies a program clause. (`clkey/1`)

  `Clause` is a free variable. (`var/1`)

- *The following properties should hold upon exit:*

  `clause(Clause0)` (`clause/1`)

  `ClId` is an atom that uniquely identifies a program clause. (`clkey/1`)

  `clause(Clause)` (`clause/1`)

decode_litkey/5: PREDICATE

Usage: `decode_litkey(AtId,F,A,C,G)`

`AtId` identifies the `G`th literal of the body of the `C`th clause of predicate `F/A`.

- *Call and exit should be compatible with:*

  `AtId` is an atom that uniquely identifies a program point of a program clause body literal. (`litkey/1`)

  `F` is an atom. (`atm/1`)

  `A` is a number. (`num/1`)

  `C` is a number. (`num/1`)

  `G` is a number. (`num/1`)

decode_clkey/4: PREDICATE

Usage: `decode_clkey(ClId,F,A,C)`

`ClId` identifies the `C`th clause of predicate `F/A`.

- *Call and exit should be compatible with:*

  `ClId` is an atom that uniquely identifies a program clause. (`clkey/1`)

  `F` is an atom. (`atm/1`)

  `A` is a number. (`num/1`)

  `C` is a number. (`num/1`)

decode_entrykey/4: PREDICATE

No further documentation available for this predicate.

decode_predkey/3: PREDICATE

Usage: `decode_predkey(PredId,F,A)`

`PredId` identifies the predicate `F/A`.

- *Call and exit should be compatible with:*

  `PredId` is an atom that uniquely identifies a program predicate. (`predkey/1`)

  `F` is an atom. (`atm/1`)

  `A` is a number. (`num/1`)

is_clkey/1: PREDICATE

Usage: `is_clkey(Id)`

Succeeds if `Id` is a valid clause identifier. The predicate name must not contain '/' characters.

– *Call and exit should be compatible with:*

`Id` is an atom. (`atm/1`)

is_litkey/1: PREDICATE

Usage: `is_litkey(Id)`

Succeeds if `Id` is a valid atom identifier. The predicate name must not contain '/' characters.

– *Call and exit should be compatible with:*

`Id` is an atom. (`atm/1`)

is_entrykey/1: PREDICATE

Usage: `is_entrykey(Id)`

Succeeds if `Id` is a valid *entry point* identifier (predkey for exported predicates, clkey for entry assertions)

– *Call and exit should be compatible with:*

`Id` is an atom. (`atm/1`)

is_directive/3: PREDICATE

No further documentation available for this predicate.

is_clause/4: PREDICATE

No further documentation available for this predicate.

orig_clause_id/2: PREDICATE

`orig_clause_id(ClId,Orig_ClId)`

`Orig_ClId` is the clause identifier resulting from replacing the name of a newly generated predicate with the name of its corresponding predicate in the original program.

lit_ppkey/3: PREDICATE

Usage:

– *Call and exit should be compatible with:*

`literal_ppkey(Arg1)` (`literal_ppkey/1`)

`Arg2` is a term which represents a goal, i.e., an atom or a structure. (`cgoal/1`)

`Arg3` is an atom that uniquely identifies a program point of a program clause body literal. (`litkey/1`)

clid_of_atomid/2: PREDICATE

Usage: `clid_of_atomid(AtomId,ClId)`

Returns in `ClId` the clause identifier corresponding to `AtomId` goal identifier.

- *Call and exit should be compatible with:*
  `AtomId` is an atom. (`atm/1`)
  `ClId` is an atom. (`atm/1`)
- *The following properties should hold at call time:*
  `AtomId` is currently a term which is not a free variable. (`nonvar/1`)
  `ClId` is a free variable. (`var/1`)

clause/1: REGTYPE

Defined as:

```
clause(clause(Head,Body):Clid) :-
    cgoal(Head),
    clausebody(Body),
    clkey(Clid).
clause(directive(Body):Clid) :-
    clausebody(Body),
    drid(Clid).
```

clausebody/1: PREDICATE

`clausebody(Body)`

`Body` is a conjunction of simple goals; if a goal is a meta-predicate, its meta-arguments are terms of the form `'$'(Term,Body,Type)`, where `Term` is the original goal term, `Type` the type of meta-term as in the meta_predicate directive, and, if `Type` corresponds to an executable form of meta-term, `Body` is the `clausebody` that corresponds to `Term`.

```
clausebody(L) :-
    literal_ppkey(L).
clausebody((L,B)) :-
    literal_ppkey(L),
    clausebody(B).
```

literal_ppkey/1: REGTYPE

A regular type, defined as follows:

```
literal_ppkey(Lit:PPKey) :-
    cgoal(Lit),
    litkey(PPKey).
literal_ppkey(builtin(B):noinfo) :-
    cgoal(B).
literal_ppkey(!).
```

litkey/1: REGTYPE

Usage: `litkey(Key)`

`Key` is an atom that uniquely identifies a program point of a program clause body literal.

clkey/1: REGTYPE

Usage: `clkey(Id)`

`Id` is an atom that uniquely identifies a program clause.

predkey/1: REGTYPE

Usage: `predkey(Id)`

`Id` is an atom that uniquely identifies a program predicate.

counter/3: PREDICATE

No further documentation available for this predicate. The predicate is of type *data*.

get_predkey/3: PREDICATE

Usage: `get_predkey(F,A,Key)`

`Key` is the key corresponding to predicate `F/A`

- *The following properties should hold at call time:*
  - `F` is an atom. (`atm/1`)
  - `A` is an atom. (`atm/1`)
  - `Key` is a free variable. (`var/1`)
- *The following properties should hold upon exit:*
  - `Key` is an atom that uniquely identifies a program predicate. (`predkey/1`)

get_predkeys/2: PREDICATE

Usage: `get_predkeys(Preds,Keys)`

`Keys` is the list of atoms obtained from `Preds`.

- *The following properties should hold at call time:*
  - `Preds` is a list. (`list/1`)
- *The following properties should hold upon exit:*
  - `Keys` is a list of `predkey`s. (`list/2`)

predkey_from_sg/2: PREDICATE

Usage: `predkey_from_sg(Sg,Key)`

`Key` is the key corresponding to functor `Sg`

- *The following properties should hold at call time:*
  - `Sg` is any term. (`term/1`)
  - `Key` is a free variable. (`var/1`)
- *The following properties should hold upon exit:*
  - `Key` is an atom that uniquely identifies a program predicate. (`predkey/1`)

get_clkey/4: PREDICATE

Usage: `get_clkey(F,A,Cl,ClKey)`

`ClKey` is the key corresponding to clause `F/A/Cl`

- *The following properties should hold at call time:*
  - `F` is an atom. (`atm/1`)
  - `A` is an atom. (`atm/1`)
  - `Cl` is an atom. (`atm/1`)
  - `ClKey` is a free variable. (`var/1`)
- *The following properties should hold upon exit:*
  - `ClKey` is an atom that uniquely identifies a program clause. (`clkey/1`)

get_litkey/5: PREDICATE

Usage: `get_litkey(F,A,Cl,Lit,ClKey)`

`ClKey` is the key corresponding to clause `F/A/Cl/Lit`

- *The following properties should hold at call time:*
  - `F` is an atom. (`atm/1`)
  - `A` is an atom. (`atm/1`)
  - `Cl` is an atom. (`atm/1`)
  - `Lit` is an atom. (`atm/1`)
  - `ClKey` is a free variable. (`var/1`)
- *The following properties should hold upon exit:*
  - `ClKey` is an atom that uniquely identifies a program point of a program clause body literal. (`litkey/1`)

94.3 Documentation on internals

body/1: PREDICATE

`body(Body)`

`Body` is a conjunction of simple goals:

```
body((G,B)) :-
    cgoal(G),
    body(B).
body(G) :-
    cgoal(G).
```

drid/1: REGTYPE

Usage: `drid(Id)`

`Id` is a number that uniquely identifies a program directive.

head/1: REGTYPE

Defined as

```
head(0).
head(1).
head(X) :-
    cgoal(X).
```

Usage: `head(Head)`

`Head` is a clause head identifier.

94.4 Documentation on imports

This module has the following direct dependencies:

- *System library modules:*
  `datafacts_rt`, `lists`, `prednames`, `p_unit_basic`.
- *Internal (engine) modules:*
  `term_basic`, `arithmetic`, `atomic_basic`, `basiccontrol`, `exceptions`, `term_compare`, `term_typing`, `debugger_support`, `basic_props`, `hiord_rt`, `messages_basic`.
- *Packages:*
  `prelude`, `initial`, `condcomp`, `assertions`, `assertions/assertions_basic`, `basicmodes`, `regtypes`, `datafacts`.

95 tr_syntax (library)

95.1 Usage and interface

- **Library usage:**
  `:- use_module(library(compiler/p_unit/tr_syntax)).`
- **Exports:**
  – *Predicates:*
    `cleanup_tr_syntax/0`, `traverse_clauses/4`.

95.2 Documentation on exports

cleanup_tr_syntax/0: PREDICATE

`cleanup_tr_syntax`

Cleanups internal database.

traverse_clauses/4: PREDICATE

Usage: `traverse_clauses(Cls,Ds,NewCls,NewDs)`

It traverses program clauses `Cls` suitably changing the format for the following step (analysis, annotation, etc). In general, it will have to remove disjunctions, if-then-elses, remote cuts, etc. In doing this, the old clauses will be transformed and new clauses (and thus new dictionaries) can be created.

– *The following properties should hold at call time:*

`Cls` is currently a term which is not a free variable. (`nonvar/1`)

`Ds` is currently a term which is not a free variable. (`nonvar/1`)

`NewCls` is a free variable. (`var/1`)

`NewDs` is a free variable. (`var/1`)

95.3 Documentation on imports

This module has the following direct dependencies:

– *System library modules:*
  `datafacts_rt`, `program_keys`, `native`, `p_unit`, `p_unit_db`, `lists`, `sets`, `terms_vars`, `vndict`.
– *Internal (engine) modules:*
  `term_basic`, `arithmetic`, `atomic_basic`, `basiccontrol`, `exceptions`, `term_compare`, `term_typing`, `debugger_support`, `basic_props`, `hiord_rt`.
– *Packages:*
  `prelude`, `initial`, `condcomp`, `assertions`, `assertions/assertions_basic`, `basicmodes`, `datafacts`.

96 The Global Information Server

This component is in charge of serving the rest of the precompiler all known semantic information on the current module, either from any analysis or from the assertions.

The (basic) organization of the code is as follows:

Module `infer` is in charge of interfacing with the rest of CiaoPP. Module `modes` implements a mode analysis based on sharing information. Module `vartypes` implements a moded type analysis based on freeness and regular type information. Module `infer_db` is the centralized repository for all analysis information. Module `typeslib` implements a library for manipulation of regular types.

97 Analysis Information Server

This module provides the other parts of the compiler with the information they require on properties of the program, either inferred or from the assertions.

97.1 Usage and interface

- **Library usage:**
  `:- use_module(ciaopp(infer)).`
- **Exports:**
  - *Predicates:*
    `get_info/5`, `get_absint/4`, `type2measure/3`, `type_holds/3`, `type_fails/3`, `get_memo_lub/5`, `get_completes_lub/6`, `get_completes/4`, `kind_of_info/7`.

97.2 Documentation on exports

get_info/5: PREDICATE

No further documentation available for this predicate.

get_absint/4: PREDICATE

No further documentation available for this predicate.

type2measure/3: PREDICATE

No further documentation available for this predicate.

type_holds/3: PREDICATE

No further documentation available for this predicate.

type_fails/3: PREDICATE

No further documentation available for this predicate.

get_memo_lub/5: PREDICATE

Usage: `get_memo_lub(Key,Vars,AbsInt,Lub,ASub)`

Obtains program point info inferred by domain `AbsInt` at point `Key`.

- *The following properties should hold at call time:*
  - `Key` is ground. (`gnd/1`)
  - `Vars` is any term. (`term/1`)
  - `AbsInt` is an atom. (`atm/1`)
  - `Lub` is a free variable. (`var/1`)
  - `ASub` is a free variable. (`var/1`)

- *The following properties should hold upon exit:*
  `Key` is ground. (`gnd/1`)
  `Vars` is a list of `vars`. (`list/2`)
  `AbsInt` is ground. (`gnd/1`)
  `Lub` is currently a term which is not a free variable. (`nonvar/1`)
  `ASub` is currently a term which is not a free variable. (`nonvar/1`)

get_completes_lub/6: PREDICATE

No further documentation available for this predicate.

get_completes/4: PREDICATE

No further documentation available for this predicate.

kind_of_info/7: PREDICATE

No further documentation available for this predicate.

97.3 Documentation on imports

This module has the following direct dependencies:

- *Application modules:*
  `infer_db`, `infer_dom`, `inferseff`, `vartypes`, `domains`, `plai_db`, `pool`, `preprocess_flags`, `nfdet_common`.
- *System library modules:*
  `datafacts_rt`, `typeslib`, `p_unit`, `program_keys`, `aggregates`, `lists`, `sort`, `terms_check`, `terms_vars`.
- *Internal (engine) modules:*
  `term_basic`, `arithmetic`, `atomic_basic`, `basiccontrol`, `exceptions`, `term_compare`, `term_typing`, `debugger_support`, `basic_props`.
- *Packages:*
  `prelude`, `initial`, `condcomp`, `assertions`, `assertions/assertions_basic`, `datafacts`, `ciaopp(ciaopp_options)`.

98 The Abstract Interpretation Component -PLAI

This component is in charge of analysing the program with techniques based on abstract interpretation. See Chapter 7 [Available abstract domains], page 31 for the available abstract domains.

99 PLAI- Abstract Interpretation-based Analysis

Abstract interpretation based analysis in PLAI is implemented in a parametric fashion: there are four available fixpoint computation algorithms for the analysis which can be used with several abstract domains.

The (basic) organization of the code is as follows:

Module `plai` is in charge of interfacing with the rest of CiaoPP. Modules `tarjan`, `normalize`, and `transform` support the preprocessing needed for analysis: identify strongly connected components, normalize the program (if required), and assert it in a format suitable for analysis, respectively. Module `plai_db` stores the final results of analysis. Module `trust` handles the assertions used during analysis and module `domains` interfaces to the domain-dependent operations.

Module `fixpo_plai` implements the classical top-down fixpoint algorithm of PLAI. In the figure, it represents several other fixpoint algorithms available, namely: `fixpo_del` for delay analysis, and `fixpo_dd` and `fixpo_di` for incremental analysis. Module `re_analysis` supports the recovery of analysis information after transformation of the source program.

Modules `trace_fixp` and `view_fixp` implement tracing of the analysis procedure, including a graphic visualization of the and-or graph being constructed during analysis.

100 Program Analysis (PLAI)

Stability: [**devel**] Currently the subject of active development and/or research. Functionality may be limited and API and/or functionality may change without warning or deprecation period. Not recommended yet for use in production.

100.1 Usage and interface

- **Library usage:**
  `:- use_module(ciaopp(plai)).`
- **Exports:**
  - *Predicates:*
    `plai/5`, `cleanup_plai/1`, `mod_plai/5`, `is_checker/1`, `analyze/7`, `init_fixpoint/1`, `entry_point/5`, `generate_trans_clauses/4`, `transform_clauses_/5`, `topdown_analysis/3`, `mod_topdown_analysis/3`.

100.2 Documentation on exports

plai/5: PREDICATE

`plai(Cls,Ds,Fixp,AbsInt,Info)`

Performs the analysis of the clauses in `Cls` (dictionaries of variables in `Ds`) with fixpoint algorithm `Fixp` and analysis domain `AbsInt`. The analysis results are left in a permanent database (see `plai_db`). `Info` is a list of properties and informations. One possible value is `[time(Total,[(subtask1,T1),...,(subtaskN,TN)])]`.

cleanup_plai/1: PREDICATE

`cleanup_plai(AbsInt)`

Cleanups the database of analysis of all permanent information regarding abstract domain `AbsInt`.

mod_plai/5: PREDICATE

`mod_plai(Cls,Ds,Fixp,AbsInt,Time)`

Performs the analysis of the clauses in `Cls` (dictionaries of variables in `Ds`) with fixpoint algorithm `Fixp` and analysis domain `AbsInt`. The analysis results are left in a permanent database (see `plai_db`).

The analysis uses the entry information provided by the entry policy selected in entry_policy preprocessing flag.

is_checker/1: PREDICATE

No further documentation available for this predicate.

analyze/7: PREDICATE

No further documentation available for this predicate.

init_fixpoint/1: PREDICATE

No further documentation available for this predicate.

entry_point/5: PREDICATE

No further documentation available for this predicate.

generate_trans_clauses/4: PREDICATE

No further documentation available for this predicate.

transform_clauses_/5: PREDICATE

No further documentation available for this predicate.

topdown_analysis/3: PREDICATE

No further documentation available for this predicate.

mod_topdown_analysis/3: PREDICATE

No further documentation available for this predicate.

100.3 Documentation on imports

This module has the following direct dependencies:

- *Application modules:*
  `analysis_stats`, `ciaopp_log`, `sp_clauses`, `unfold_times`, `frontend_driver`, `intermod_entry`, `preprocess_flags`, `mem_usage`, `domains`, `normalize_args`, `plai_errors`, `fixpo_plai`, `fixpo_dd`, `fixpo_di`, `fixpo_check_di`, `fixpo_check_di2`, `fixpo_check_di3`, `fixpo_check_di4`, `fixpo_check_di5`, `fixpo_plai_gfp`, `fixpo_check_reduced_di`, `fixpo_bu`, `tarjan`, `transform`, `fixpo_ops`, `trace_fixp`.
- *System library modules:*
  `aggregates`, `counters`, `lists`, `terms_vars`, `vndict`, `port_reify`, `messages`, `p_unit`, `program_keys`, `p_unit_db`.
- *Internal (engine) modules:*
  `term_basic`, `arithmetic`, `atomic_basic`, `basiccontrol`, `exceptions`, `term_compare`, `term_typing`, `debugger_support`, `basic_props`.
- *Packages:*
  `prelude`, `initial`, `condcomp`, `assertions`, `assertions/assertions_basic`.

101 fixpo_plai (library)

Author(s): Kalyan Muthukumar, Maria Garcia de la Banda, Francisco Bueno.

Stability: [**alpha**] A good part of the functionality is there but has not been the subject of significant testing and/or verification.

This module implements the top-down fixpoint algorithm of PLAI, both in its mono-variant and multi-variant on successes versions. It is always multi-variant on calls. The algorithm is parametric on the particular analysis domain.

101.1 Usage and interface

- **Library usage:**

  `:- use_module(ciaopp(plai/fixpo_plai)).`
- **Exports:**
  - *Predicates:*

    `query/8`, `init_fixpoint/0`, `cleanup_fixpoint/1`, `entry_to_exit/9`, `approx/6`, `fixpoint/6`, `$depend_list/3`.

101.2 Documentation on exports

query/8: PREDICATE

Usage: `query(AbsInt,QKey,Query,Qv,RFlag,N,Call,Succ)`

The success pattern of `Query` with `Call` is `Succ` in the analysis domain `AbsInt`. The predicate called is identified by `QKey`, and `RFlag` says if it is recursive or not. The goal `Query` has variables `Qv`, and the call pattern is uniquely identified by `N`.

- *The following properties should hold at call time:*

  `AbsInt` is currently a term which is not a free variable. (`nonvar/1`)

  `QKey` is currently a term which is not a free variable. (`nonvar/1`)

  `Query` is currently a term which is not a free variable. (`nonvar/1`)

  `Qv` is currently a term which is not a free variable. (`nonvar/1`)

  `RFlag` is currently a term which is not a free variable. (`nonvar/1`)

  `Call` is currently a term which is not a free variable. (`nonvar/1`)

  `Succ` is a free variable. (`var/1`)

init_fixpoint/0: PREDICATE

Usage:

Cleanups the database of analysis of temporary information.

cleanup_fixpoint/1: PREDICATE

Usage: `cleanup_fixpoint(AbsInt)`

Cleans up the database of analysis, of both temporary as well as permanent information regarding abstract domain `AbsInt`.

entry_to_exit/9: PREDICATE

No further documentation available for this predicate.

approx/6: PREDICATE

No further documentation available for this predicate. The predicate is of type *data*.

fixpoint/6: PREDICATE

No further documentation available for this predicate. The predicate is of type *data*.

$depend_list/3: PREDICATE

No further documentation available for this predicate. The predicate is of type *data*.

101.3 Documentation on imports

This module has the following direct dependencies:

- *Application modules:*
  `preprocess_flags`, `fixpo_ops`, `domains`, `plai_db`, `psets`, `transform`, `apply_assertions_old`.
- *System library modules:*
  `native_props`, `datafacts_rt`, `aggregates`, `lists`, `terms_vars`, `sets`, `sort`, `messages`.
- *Internal (engine) modules:*
  `term_basic`, `arithmetic`, `atomic_basic`, `basiccontrol`, `exceptions`, `term_compare`, `term_typing`, `debugger_support`, `basic_props`, `hiord_rt`.
- *Packages:*
  `prelude`, `initial`, `condcomp`, `assertions`, `assertions/assertions_basic`, `basicmodes`, `nativeprops`, `datafacts`, `ciaopp(ciaopp_options)`, `.(notrace)`, `spec(no_debug)`.

102 Available type domains and widenings

102.1 Available widenings

All widenings are briefly described in the eterms domain paper [VB02]

Widening	Implemented	Domain	Typeslib	Using 'names'	Tested	Working
Type Jungle	yes	evalterms	yes	yes (why?)	no	not connected to plai
Shortening	yes	termsd	yes	no	barely	apparently
Topological Clash	yes	ptypes	yes	no	barely	no
Structural	yes	eterms	yes	yes	yes	yes (slow)
Depth	no					
Functor	no					

Restricted shortening is also implemented according to comments in termsd.

102.2 Other type domains

Domain	widening needed?	Typeslib	Using 'names'	Tested	Working
svterms	yes (structural)	yes	yes (structural)	barely	apparently (slow)
deftypes	no	yes	no	barely	apparently
depthk	no	no	no	barely	?

102.3 Some experiments

Some very simple experiments FixFail means that analyzing with plai the message "Something has failed" appeared.

Benchmarks are located in `ciaopp/tests/benchs/widening/` list_of_lists and sorted were used because they are used as examples in the eterms paper. `minimal_lists` and `minimal_functors` were designed to capture the complexity problem that currently eterms has.

Widening	list_of_lists	sorted	minimal_lists	minimal_functors
Shortening	2.532	30.195	26.668	9.28
Topological Clash	FixFail	FixFail	FixFail	FixFail
Structural	2.388	2.855	23161.302	9.822
svterms	3.183	3.443	37948.465	8.553
deftypes	1.647	2.348	2.603	8.803
depthk	8.527	5.725	4.879	7.11

For structural widening, e.g. using domain eterms or svterms, analyzing `minimal_lists` creates 1500 types.

103 plai_db (library)

103.1 Usage and interface

- **Library usage:**
  `:- use_module(ciaopp(plai/plai_db)).`
- **Exports:**
  - *Predicates:*
    `complete/7`, `lub_complete/6`, `memo_call/5`, `memo_table/6`, `memo_lub/5`, `pragma/5`, `complete_parent/2`, `cleanup_plai_db/1`, `invalid_call/6`, `raw_success/6`, `get_parent_key/4`, `get_complete/8`, `get_memo_table/7`, `get_raw_success/7`, `add_parent_complete/5`, `del_parent_complete/6`, `patch_parents/6`, `merge_parents/3`, `del_parent/4`, `update_complete_id/4`, `delete_complete/3`, `remove_extra_info_complete/3`,
    `delete_plai_db_one_clause/4`, `complete_key_from_id/3`, `erase_previous_memo_tables_and_parents/4`, `erase_last_memo_table/3`, `erase_previous_memo_lubs/2`, `erase_previous_parents_info/5`, `init_rev_idx/1`, `clean_rev_idx/1`, `complete_id_key/3`, `memo_table_id_key/3`, `add_invalid_call/6`, `store_raw_success/6`, `get_raw_success/6`.
  - *Regular Types:*
    `plai_db_id/1`.

103.2 Documentation on exports

complete/7: PREDICATE

`complete(SgKey,AbsInt,Sg,Proj,Prime,Id,Parents)`

The predicate `SgKey` has a variant success pattern `(Sg,Proj,Prime)` on the domain `AbsInt`. The and-or graph node is `Id`, and is called from the program points in list `Parents`.

Usage: `complete(SgKey,AbsInt,Sg,Proj,Prime,Id,Parents)`

- *The following properties should hold upon exit:*
  - `SgKey` is an atom that uniquely identifies a program predicate. (`predkey/1`)
  - `AbsInt` is an atom. (`atm/1`)
  - `Sg` is any term. (`term/1`)
  - `Proj` is any term. (`term/1`)
  - `Prime` is a list of `terms`. (`list/2`)
  - `Id` is an integer. (`int/1`)
  - `parents_complete(Parents)` (`parents_complete/1`)

The predicate is of type *data*.

lub_complete/6: PREDICATE

`lub_complete(SgKey,AbsInt,Lub,Sg,Proj,Prime)`

The predicate `SgKey` has in all variants success pattern `(Sg,Proj,Prime)` on the domain `AbsInt`. If `Lub` is yes, `Proj` and `Prime` are substitutions, if `Lub` is no, they are lists of substitutions. The predicate is of type *data*.

memo_call/5: PREDICATE

No further documentation available for this predicate. The predicate is of type *data*.

memo_table/6: PREDICATE

`memo_table(LitKey,AbsInt,Id,Child,Vars_u,Call)`

Before calling the goal at program point `PointKey`, there is a variant in which `Call` on the domain `AbsInt` holds upon the program clause variables `Vars_u`. These variables need be sorted conveniently so that `Call` makes sense. The and-or graph node that causes this is `Id` and the call originated to the goal at this program point generates and-or graph node `Child`.

Usage: `memo_table(LitKey,AbsInt,Id,Child,Vars_u,Call)`

– *The following properties should hold upon exit:*

`LitKey` is an atom that uniquely identifies a program point of a program clause body literal. (`litkey/1`)

`AbsInt` is an atom. (`atm/1`)

`Id` is an integer. (`int/1`)

`Child` is an integer. (`int/1`)

`Vars_u` is any term. (`term/1`)

`Call` is a list of `terms`. (`list/2`)

The predicate is of type *data*.

memo_lub/5: PREDICATE

`memo_lub(PointKey,AbsInt,Lub,Vars_u,Call)`

Before calling the goal at program point `PointKey`, in all variants `Call` on the domain `AbsInt` holds upon the program clause variables `Vars_u`. If `Lub` is yes, `Call` is a substitution, if If `Lub` is no, `Call` is a list of substitutions. The predicate is of type *data*.

pragma/5: PREDICATE

No further documentation available for this predicate. The predicate is of type *data*.

complete_parent/2: PREDICATE

`complete_parent(Id,Parents)`

Used to keep the trace of the parents in the and-or graph for the dd/di fixpoints. The parents are used to choose previous aproximations to apply the widening operators on calls The and-or graph node is `Id`, and `Parents` is a list of couples of graph node and program points. The predicate is of type *data*.

cleanup_plai_db/1: PREDICATE

`cleanup_plai_db(AbsInt)`

Erases all data in this module for the analysis domain `AbsInt`.

invalid_call/6: PREDICATE

No further documentation available for this predicate. The predicate is of type *data*.

raw_success/6: PREDICATE

No further documentation available for this predicate. The predicate is of type *data*.

get_parent_key/4: PREDICATE

Usage: `get_parent_key(LitKey,Id,AbsInt,CKey)`

`CKey` is the key that corresponds to `Id`, it may not correspong to the decoding LitKey because of how meta_calls are indexed in the fixpoints

- *The following properties should hold upon exit:*
  - `LitKey` is an atom that uniquely identifies a program point of a program clause body literal. (`litkey/1`)
  - `Id` is an integer. (`int/1`)
  - `AbsInt` is an atom. (`atm/1`)
  - `CKey` is an atom that uniquely identifies a program predicate. (`predkey/1`)

get_complete/8: PREDICATE

Usage: `get_complete(SgKey,AbsInt,Sg,Proj,Prime,Id,Ps,Ref)`

- *The following properties should hold at call time:*
  - `Id` is currently a term which is not a free variable. (`nonvar/1`)
  - `SgKey` is an atom that uniquely identifies a program predicate. (`predkey/1`)
  - `AbsInt` is an atom. (`atm/1`)
  - `plai_db_id(Id)` (`plai_db_id/1`)
- *The following properties should hold globally:*
  - All calls of the form `get_complete(SgKey,AbsInt,Sg,Proj,Prime,Id,Ps,Ref)` are deterministic. (`is_det/1`)

get_memo_table/7: PREDICATE

Usage: `get_memo_table(LitKey,AbsInt,Id,Child,Vars,Call,Ref)`

- *The following properties should hold at call time:*
  - `LitKey` is currently a term which is not a free variable. (`nonvar/1`)
  - `AbsInt` is currently a term which is not a free variable. (`nonvar/1`)
  - `Id` is currently a term which is not a free variable. (`nonvar/1`)
  - `LitKey` is an atom that uniquely identifies a program point of a program clause body literal. (`litkey/1`)
  - `AbsInt` is an atom. (`atm/1`)
  - `plai_db_id(Id)` (`plai_db_id/1`)

get_raw_success/7: PREDICATE

Usage: `get_raw_success(ClKey,AbsInt,Id,Sg,Proj,Prime,Ref)`

- *The following properties should hold at call time:*
  - `ClKey` is currently a term which is not a free variable. (`nonvar/1`)
  - `AbsInt` is currently a term which is not a free variable. (`nonvar/1`)
  - `Id` is currently a term which is not a free variable. (`nonvar/1`)
  - `ClKey` is an atom that uniquely identifies a program point of a program clause body literal. (`litkey/1`)
  - `AbsInt` is an atom. (`atm/1`)
  - `plai_db_id(Id)` (`plai_db_id/1`)
- *The following properties should hold upon exit:*
  - `Sg` is currently a term which is not a free variable. (`nonvar/1`)
  - `Proj` is currently a term which is not a free variable. (`nonvar/1`)
  - `Prime` is currently a term which is not a free variable. (`nonvar/1`)
  - `Ref` is currently a term which is not a free variable. (`nonvar/1`)

add_parent_complete/5: PREDICATE

Usage: `add_parent_complete(SgKey,AbsInt,Id,PId,LitKey)`

- *The following properties should hold at call time:*
  - `SgKey` is currently a term which is not a free variable. (`nonvar/1`)
  - `AbsInt` is currently a term which is not a free variable. (`nonvar/1`)
  - `Id` is currently a term which is not a free variable. (`nonvar/1`)
  - `PId` is currently a term which is not a free variable. (`nonvar/1`)
  - `LitKey` is currently a term which is not a free variable. (`nonvar/1`)

del_parent_complete/6: PREDICATE

Usage: `del_parent_complete(SgKey,AbsInt,Id,PId,LitKey,NPs)`

- *The following properties should hold at call time:*
  - `SgKey` is currently a term which is not a free variable. (`nonvar/1`)
  - `AbsInt` is currently a term which is not a free variable. (`nonvar/1`)
  - `Id` is currently a term which is not a free variable. (`nonvar/1`)
  - `PId` is currently a term which is not a free variable. (`nonvar/1`)
  - `LitKey` is currently a term which is not a free variable. (`nonvar/1`)
  - `NPs` is a free variable. (`var/1`)

patch_parents/6: PREDICATE

Usage: `patch_parents(Ref,Memo,Key,Id,NewFs,Fs)`

- *The following properties should hold at call time:*
  - `Ref` is currently a term which is not a free variable. (`nonvar/1`)
  - `Memo` is currently a term which is not a free variable. (`nonvar/1`)
  - `Key` is currently a term which is not a free variable. (`nonvar/1`)
  - `Id` is currently a term which is not a free variable. (`nonvar/1`)
  - `Fs` is currently a term which is not a free variable. (`nonvar/1`)
  - `Fs` is a list. (`list/1`)

- *The following properties should hold globally:*
  All the calls of the form `patch_parents(Ref,Memo,Key,Id,NewFs,Fs)` do not fail. (`not_fails/1`)

Meta-predicate with arguments: `patch_parents(?,fact,?,?,?,?)`.

merge_parents/3: PREDICATE

No further documentation available for this predicate.

del_parent/4: PREDICATE

No further documentation available for this predicate.

update_complete_id/4: PREDICATE

No further documentation available for this predicate.

delete_complete/3: PREDICATE

Usage: `delete_complete(SgKey,AbsInt,Id)`

Deletes the information of the complete with `Id`. This predicate does not delete recursively following its parents.

- *The following properties should hold at call time:*
  `SgKey` is currently a term which is not a free variable. (`nonvar/1`)
  `AbsInt` is currently a term which is not a free variable. (`nonvar/1`)
  `Id` is currently a term which is not a free variable. (`nonvar/1`)
  `SgKey` is an atom. (`atm/1`)
  `AbsInt` is an atom. (`atm/1`)
  `plai_db_id(Id)` (`plai_db_id/1`)

remove_extra_info_complete/3: PREDICATE

No further documentation available for this predicate.

delete_plai_db_one_clause/4: PREDICATE

Usage: `delete_plai_db_one_clause(PredKey,ClKey,Id,AbsInt)`

Removes from plai_db the information relative to the clause `ClKey` of complete `Id` for the domain `AbsInt`

- *The following properties should hold at call time:*
  `PredKey` is currently a term which is not a free variable. (`nonvar/1`)
  `ClKey` is currently a term which is not a free variable. (`nonvar/1`)
  `Id` is currently a term which is not a free variable. (`nonvar/1`)
  `AbsInt` is currently a term which is not a free variable. (`nonvar/1`)
  `PredKey` is an atom. (`atm/1`)
  `ClKey` is an atom. (`atm/1`)
  `plai_db_id(Id)` (`plai_db_id/1`)
  `AbsInt` is an atom. (`atm/1`)

complete_key_from_id/3: PREDICATE

No further documentation available for this predicate.

erase_previous_memo_tables_and_parents/4: PREDICATE

Usage: `erase_previous_memo_tables_and_parents(Body,AbsInt,ClKey,Id)`

We use `Body` to erase all the memo_tables and all the pointers in the dependency graph that are not longer valid

- *The following properties should hold at call time:*

  `Body` is currently a term which is not a free variable. (`nonvar/1`)

  `AbsInt` is currently a term which is not a free variable. (`nonvar/1`)

  `ClKey` is currently a term which is not a free variable. (`nonvar/1`)

  `Id` is currently a term which is not a free variable. (`nonvar/1`)

erase_last_memo_table/3: PREDICATE

Usage: `erase_last_memo_table(AbsInt,ClKey,Id)`

- *The following properties should hold at call time:*

  `AbsInt` is currently a term which is not a free variable. (`nonvar/1`)

  `ClKey` is currently a term which is not a free variable. (`nonvar/1`)

  `Id` is currently a term which is not a free variable. (`nonvar/1`)

- *The following properties should hold globally:*

  All the calls of the form `erase_last_memo_table(AbsInt,ClKey,Id)` do not fail. (`not_fails/1`)

erase_previous_memo_lubs/2: PREDICATE

Usage: `erase_previous_memo_lubs(Body,ClKey)`

This predicate traverses `Body` and removes the existing memo_lubs. When `Body` is finished, we also remove the memo-lub for the end of the clause using `ClKey`.

- *The following properties should hold at call time:*

  `Body` is currently a term which is not a free variable. (`nonvar/1`)

  `ClKey` is currently a term which is not a free variable. (`nonvar/1`)

erase_previous_parents_info/5: PREDICATE

Usage: `erase_previous_parents_info(Id,Goal,AbsInt,Key,NewN)`

`Id` is the node identifier of the complete that has this `Goal` in its list of parents. If Id is no then nothing needs to be done (it is a builtin)

- *The following properties should hold at call time:*

  `Id` is currently a term which is not a free variable. (`nonvar/1`)

  `Goal` is currently a term which is not a free variable. (`nonvar/1`)

  `AbsInt` is currently a term which is not a free variable. (`nonvar/1`)

  `Key` is currently a term which is not a free variable. (`nonvar/1`)

  `NewN` is currently a term which is not a free variable. (`nonvar/1`)

init_rev_idx/1: PREDICATE

No further documentation available for this predicate.

clean_rev_idx/1: PREDICATE

No further documentation available for this predicate.

complete_id_key/3: PREDICATE

No further documentation available for this predicate.

memo_table_id_key/3: PREDICATE

No further documentation available for this predicate.

add_invalid_call/6: PREDICATE

Usage: `add_invalid_call(SgKey,AbsInt,LitKey,Id,Sg,Proj)`

– *The following properties should hold at call time:*

`SgKey` is currently a term which is not a free variable. (`nonvar/1`)
`AbsInt` is currently a term which is not a free variable. (`nonvar/1`)
`LitKey` is currently a term which is not a free variable. (`nonvar/1`)
`Id` is currently a term which is not a free variable. (`nonvar/1`)
`Sg` is currently a term which is not a free variable. (`nonvar/1`)
`Proj` is currently a term which is not a free variable. (`nonvar/1`)

store_raw_success/6: PREDICATE

Usage: `store_raw_success(ClKey,AbsInt,Id,Sg,Proj,LPrime)`

– *The following properties should hold at call time:*

`ClKey` is currently a term which is not a free variable. (`nonvar/1`)
`AbsInt` is currently a term which is not a free variable. (`nonvar/1`)
`Id` is currently a term which is not a free variable. (`nonvar/1`)
`Sg` is currently a term which is not a free variable. (`nonvar/1`)
`Proj` is currently a term which is not a free variable. (`nonvar/1`)
`LPrime` is currently a term which is not a free variable. (`nonvar/1`)

get_raw_success/6: PREDICATE

Usage: `get_raw_success(ClKey,AbsInt,Id,Sg,Proj,LPrime)`

– *The following properties should hold at call time:*

`ClKey` is currently a term which is not a free variable. (`nonvar/1`)
`AbsInt` is currently a term which is not a free variable. (`nonvar/1`)
`Id` is currently a term which is not a free variable. (`nonvar/1`)

– *The following properties should hold upon exit:*

`LPrime` is a list. (`list/1`)

plai_db_id/1: REGTYPE

A regular type, defined as follows:

```
plai_db_id(no).
plai_db_id(X) :-
    int(X).
```

103.3 Documentation on imports

This module has the following direct dependencies:

- *Application modules:*
  transform.
- *System library modules:*
  datafacts_rt, native_props, program_keys.
- *Internal (engine) modules:*
  term_basic, arithmetic, atomic_basic, basiccontrol, exceptions, term_compare, term_typing, debugger_support, basic_props, hiord_rt.
- *Packages:*
  prelude, initial, condcomp, assertions, assertions/assertions_basic, datafacts, regtypes, isomodes, nativeprops.

104 Improving Analysis Using Trust Assertions

This section describes the use of trust assertions for improving (abstract interpretation based) top-down analyses. Trust assertions are (user given) assertions that do hold of the program. Therefore, analysis can take advantage of them to improve the information inferred.

A trust assertion for a given predicate will be denoted `C` ⇒`S`, where `C` is a call pattern and `S` a success pattern. Trust assertions for a given predicate are considered closed, that is, the set of call patterns covers all possible calls to the predicate.[1]

In the following, `L` ⇒`L'` will denote the abstract pattern inferred for a given predicate call by the analysis which is going to be improved with trust assertions. In the absence of the predicate, the success pattern cannot be inferred: however, it can be approximated a priori by the downwards closed information in the call pattern. Thus, if this is the case, `L' = L*`, where `L*` denotes the downwards closure restriction of `L` (i. e., the restriction of `L` to only its downwards closed information).

104.1 Complementation of Trust Assertions

Programmers are lazy writing trust assertions. This is not to blame them: it is just a matter of fact. They try to write as less as possible, and therefore make certain (justified) assumptions when writing assertions. Because of this, trust assertions need to be complemented in order to have as much information as possible available to the analysis.

One thing that programmers usually do is to write things like:

`ground(X)` ⇒`var(Y)`

where it is obvious that `ground(X)` does also hold on success. In order to deal with this case, trust assertions of the form `C` ⇒`S` are replaced by `C` ⇒`S'`, where:

`S' = C* meet S`

Another usual assumption in writing assertions is to say things once and for all. For example, programmers will probable write:

`true` ⇒`ground(X)`

`var(X)` ⇒`var(Y)`

where it is obvious that `ground(X)` does also hold on success in the second case. In order to deal with this case, trust assertions of the form `C`⇒`S'` are replaced by `C`⇒`S''` where:

`S'' = S' meet (meet S'_i)`

being `C_i`⇒`S'_i` a trust assertion such that `C_i >= C`.

Now we are in a position to take full advantage of all the information given in trust assertions.

104.2 Using Trust Assertions

Given abstract pattern `L`⇒`L'`, the success pattern can be improved by using the available trust assertions. The resulting success pattern is denoted `L_trust`. For better precision the most concrete applicable trust assertion should be used. Consider the set:

`St = {S''_i | C_i >= L and not exists j =/= i . C_j =< C_i }`

then:

`L_trust = L' meet (union St)`

However, `St` might be empty. In this case, no trust assertion is guaranteed to hold. But the call pattern inferred might be an over-approximation of the call patterns of some trust assertions.

[1] Reconsider this: It is not true for trust assertions inferred during inter-modular analysis. It may not be convenient, either, when having true/trust and check assertions intermixed.

And one of them must hold (since the trust assertions are supposed to be closed). Thus, in this case `St` is defined as:

```
St = {S''_i | C_i =< L and not exists j =/= i . C_j >= C_i | }
```

Nonetheless, it might also be the case that this set is empty. Then, since the trust assertions are supposed to be closed, this is an error: the inferred call pattern does not cover the possible calls.

But, in order to guarantee this error, all trust assertions should be applicable to the analysis domain (all call patterns should be fully abstractable in that domain). If this is not the case, a default success pattern can be used by defining:[2]

```
St = {S''_i | exists i}
```

[2] In contrast to the above, here `St` must include all trust assertions, whether or not applicable to the domain. This implies that the second `St` above might be incorrect!

105 Debugging the fixpoint operations

This component is in charge of tracing the fixpoint operations to ease debugging.

Make sure that the fixpoint that you want to trace is not importing package notrace (If this is the case, nothing will be shown).

The fixpoint can be traced in several forms: graphical (with daVinci graph viewer) or ascii, using `trace_fixp/1`.

105.1 Example of visualization

Note that daVinci executable has to be accesible in the path for Ciao.

```
?- trace_fixp(view).      % This will open a uDraw(Graph) window

?- module(PathToModule). % load the module to analyze (located in PathToModule)

?- analyze(AbsInt).       % Analyze for abstract domain AbsInt
% The process will appear in the uDraw(Graph) window.

?- trace_fixp(no).        % Stop the tracing
```

The implementation may not support all features of CiaoPP.

E.g. in the actions: `?- analyze(AbsInt), analyze(AbsInt).` the second analysis will fail (only if the tracing is on).

105.2 uDraw(Graph) extended features

A menu is generated in uDraw(Graph) (Edit) that allows to execute the analysis in a syncronous and asynchronous way (see Edit->Mode) to swich options.

If the synchronous option is on, once the analysis has started, the visualization (and the fixpoint) will do one operation each step. The next step will be performed only if command Edit->'One step ahead' is executed.

106 Fixpoint tracer

Stability: [**devel**] Currently the subject of active development and/or research. Functionality may be limited and API and/or functionality may change without warning or deprecation period. Not recommended yet for use in production.

This module traces the operations of fixpoints by explicitely calling `fixpoint_trace/7` during computation.

To deactivate tracing (it can slowdown the analysis), package `ciaopp(plai/notrace)` has to be imported in the used fixpoint.

106.1 Usage and interface

- **Library usage:**
  `:- use_module(ciaopp(plai/trace_fixp)).`
- **Exports:**
  - *Predicates:*
    `fixpoint_trace/7`, `trace_fixp/1`, `memotable_trace/3`, `show_spypoint_op_count/0`, `show_updated_memotable/3`, `trace_init/0`, `trace_end/0`, `trace_reset/0`.
  - *Regular Types:*
    `trace_option/1`.

106.2 Documentation on exports

fixpoint_trace/7: PREDICATE

No further documentation available for this predicate.

trace_fixp/1: PREDICATE

`trace_fixp(X)`

Toggle a trace of the fixpoint computation during analysis. The (X=op_count) option counts steps of fixpoint operations, show certain relevant spy points during analysis (X=trace), and/or display the analysis graph which is being constructed (X=view). The information added up can be seen at end of analysis with `show_fixpoint_op_count`.

Usage 1: `trace_fixp(X)`

Mode for setting the current flag to a single value.

- *The following properties should hold at call time:*
  `trace_option(X)` (`trace_option/1`)
- *The following properties should hold globally:*
  `X` is not further instantiated. (`not_further_inst/2`)

Usage 2: `trace_fixp(X)`

Mode for setting the current flag to several values.

- *The following properties should hold at call time:*
  X is a list of trace_options. (list/2)
- *The following properties should hold globally:*
  X is not further instantiated. (not_further_inst/2)

memotable_trace/3: PREDICATE

No further documentation available for this predicate.

show_spypoint_op_count/0: PREDICATE

No further documentation available for this predicate.

show_updated_memotable/3: PREDICATE

No further documentation available for this predicate.

trace_init/0: PREDICATE

No further documentation available for this predicate.

trace_end/0: PREDICATE

No further documentation available for this predicate.

trace_reset/0: PREDICATE

No further documentation available for this predicate.

trace_option/1: REGTYPE

A regular type, defined as follows:

```
trace_option(no).
trace_option(op_count).
trace_option(trace).
trace_option(view).
```

106.3 Documentation on imports

This module has the following direct dependencies:

– *Application modules:*
  analysis_stats, view_fixp, preprocess_flags, fixpo_plai, plai_db.
– *System library modules:*
  datafacts_rt, aggregates, format, write, p_unit_db, pathnames, terms, terms_vars.
– *Internal (engine) modules:*
  term_basic, arithmetic, atomic_basic, basiccontrol, exceptions, term_compare, term_typing, debugger_support, basic_props, messages_basic, stream_basic, io_basic.
– *Packages:*
  prelude, initial, condcomp, assertions, assertions/assertions_basic, regtypes, datafacts.

107 view_fixp (library)

107.1 Usage and interface

- **Library usage:**
  `:- use_module(ciaopp(plai/view_fixp)).`
- **Exports:**
  - *Predicates:*
    `start_view/0`, `end_view/0`, `clean_fixpoint_graph/0`, `view_fixpoint/7`.

107.2 Documentation on exports

start_view/0: PREDICATE
No further documentation available for this predicate.

end_view/0: PREDICATE
No further documentation available for this predicate.

clean_fixpoint_graph/0: PREDICATE
No further documentation available for this predicate.

view_fixpoint/7: PREDICATE
No further documentation available for this predicate.

107.3 Documentation on imports

This module has the following direct dependencies:

- *System library modules:*
  `datafacts_rt`.
- *Internal (engine) modules:*
  `term_basic`, `arithmetic`, `atomic_basic`, `basiccontrol`, `exceptions`, `term_compare`, `term_typing`, `debugger_support`, `basic_props`.
- *Packages:*
  `prelude`, `initial`, `condcomp`, `assertions`, `assertions/assertions_basic`, `datafacts`, `ciaopp(ciaopp_options)`.

108 Batch analysis client

Author(s): Isabel Garcia-Contreras.

Stability: [**devel**] Currently the subject of active development and/or research. Functionality may be limited and API and/or functionality may change without warning or deprecation period. Not recommended yet for use in production.

This module contains predicates for managing the batch analysis of sets of modules.

Note: this code is not performing multi-modular analysis. Some of its components can be generalized for other processes (compilation, program transformation) and be integrated into the build system, compiler, analyzer, etc.

108.1 Analysis options

- **no_incremental** : By default, analysis is made only for modules in Paths which were not analyzed before. With this option, analysis will be runned from scratch.
- **timeout(**`T`**)** : `T` is the time limit for the analysis of each module independently.
- **analysis(**`ListAbsInt`**)** : `ListAbsInt` is a list of abstract domains available in ciaopp.
- **workers(**`N`**)** : `N` is the number of threads use to perform the analysis of independent modules. Currently only 1 worker is supported.

Example:

```
? - use_module(ciaopp_batch(ciaopp_batch)).
? - analysis_start(['~/ciao-devel/core/lib/'], [timeout(40), analysis([eterms, gr])]).
```

Fast load through caching of assertions from libraries is available in `ciaopp_worker`.
To **create** and **manually update** this cache run command: `update_lib_cache`

Configuring ciaopp flags is not available yet.

108.2 Description of the implementation

108.2.1 Prolog files:

- 'ciaopp_batch.pl': driver for the per-module CiaoPP analysis
- 'ciaopp_master.pl': performs control of ciaopp_workers, i.e., launchs them and kills them.
- 'ciaopp_worker.pl': does the actual analysis.
- 'ciaopp_batch_aux.pl': file for auxiliary predicates.
- 'statistics_collecter.pl': collects statistics of the analyzed files (time and memory).

108.2.2 Other directories:

- 'Attic/': placeholder for not used files that may be useful at some point

108.3 Data files

The status of the analysis is stored in the ‘data/‘ directory:

- ‘data/task_status.pl‘: status of the modules (analyzed, errors, etc)
- ‘last_analyzed_file.pl‘: last analyzed predicate
- ‘analysis.log‘: date and time when each module started being analyzed

108.4 Usage and interface

- **Library usage:**
  `:- use_module(ciaopp_batch(ciaopp_batch)).`
- **Exports:**
  - *Predicates:*
    `analysis_start/2`.

108.5 Documentation on exports

analysis_start/2: PREDICATE

Usage: `analysis_start(Paths,Opts)`

Analyzes modules in some `Paths` and dumps their analysis information to disk, given some `Opts`.

- *The following properties should hold at call time:*
  `Paths` is a list. (`list/1`)
  `Opts` is a list. (`list/1`)

update_tasks_analysis_status/0: (UNDOC_REEXPORT)

Imported from `db_analysis` (see the corresponding documentation for details).

update_tasks_analysis_status/0: (UNDOC_REEXPORT)

Imported from `db_analysis` (see the corresponding documentation for details).

108.6 Documentation on internals

analysis_opt/1: PROPERTY

Usage: `analysis_opt(Opt)`

Option of ciaopp batch (not ciaopp flags)

108.7 Documentation on imports

This module has the following direct dependencies:

- *Application modules:*

  db_analysis.
- *System library modules:*

  datafacts_rt, process, lists, system, config_common.
- *Internal (engine) modules:*

  term_basic, arithmetic, atomic_basic, basiccontrol, exceptions, term_compare, term_typing, debugger_support, basic_props, hiord_rt, io_basic, stream_basic.
- *Packages:*

  prelude, initial, condcomp, assertions, assertions/assertions_basic, isomodes, regtypes, datafacts.

109 ciaopp_master (application)

This programs analyzes modules read in the form of terms from the standard input.

This program should not be used directly, it should be used through `ciaopp_batch`.

Workflow: This program uses workers to launch ciaopp analysis. It controls the analysis time of each worker with a timeout.

110 Worker for CiaoPP batch processing

Author(s): Isabel Garcia-Contreras.

This module implements an execution mode for CiaoPP as a worker for `ciaopp-batch` (using `ciaopp_batch(db_analysis)` for communication).

Note that this tool has to be used with an external timeout because the analysis of some modules could require too much memory or time to be performed. It must be used with ⟨undefined⟩ [Analysis management predicates], page ⟨undefined⟩.

110.1 Generated data

- Analysis information: For each module abstract analysis information is stored in a `.dump` file generated in the same location. This information and can be restored later.
- Run-time statistics: Statistics of time and memory used are stored in as a term in a `.err` file in the same location as the original module. This file contains also the output of the analyzer.
- Status of the analysis: For each module, information of load and analysis success is stored in `data/task_status.pl`.

  This allows also the script to be incremental, i.e., it does not repeat ciaopp analysis for a module if it has already been done.

  If the the user wants the tool to redo an analysis for all files, `task_status.pl` has to be removed before starting.
- `last_analyzed_file.pl`: This file contains the file that is being analyzed.

110.2 Usage and interface

- **Library usage:**

  `:- use_module(ciaopp_batch(ciaopp_worker)).`
- **Exports:**
  - *Predicates:*

    `start_worker/1`, `logged_once_port_reify/4`.

110.3 Documentation on exports

start_worker/1: PREDICATE

No further documentation available for this predicate.

logged_once_port_reify/4: PREDICATE

Usage: `logged_once_port_reify(Goal,Port,OutString,ErrString)`

Executes `Goal`, `Port` is the state when the predicate finishes (true, fail). Its stdout and stderr are stored in `OutString` and `ErrString` respectively.

- *The following properties should hold upon exit:*

  `OutString` is a string (a list of character codes). (`string/1`)

  `ErrString` is a string (a list of character codes). (`string/1`)

Meta-predicate with arguments: `logged_once_port_reify(goal,?,?,?)`.

110.4 Documentation on internals

task_status/1: REGTYPE

Usage: `task_status(Status)`

Type of analysis status.

110.5 Documentation on imports

This module has the following direct dependencies:

- *Application modules:*
  `frontend_driver`, `analyze_driver`, `preprocess_flags`, `p_dump`, `ciaopp_batch_aux`, `tasks_db`, `db_analysis`.
- *System library modules:*
  `datafacts_rt`, `lists`, `system`, `system_extra`, `port_reify`, `io_alias_redirection`, `stream_utils`, `read`.
- *Internal (engine) modules:*
  `term_basic`, `arithmetic`, `atomic_basic`, `basiccontrol`, `exceptions`, `term_compare`, `term_typing`, `debugger_support`, `basic_props`, `hiord_rt`, `stream_basic`, `io_basic`.
- *Packages:*
  `prelude`, `initial`, `condcomp`, `assertions`, `assertions/assertions_basic`, `modes`, `regtypes`, `datafacts`.

111 Auxiliary predicates

Auxiliary predicates for ciaopp batch

111.1 Usage and interface

- **Library usage:**
  `:- use_module(ciaopp_batch(ciaopp_batch_aux)).`
- **Exports:**
  - *Predicates:*
    `dump_file/4`, `string_contained/2`, `get_mod_name_from_file/2`, `newer/2`, `make_dir_nofail/1`.

111.2 Documentation on exports

dump_file/4: PREDICATE

Usage: `dump_file(FilePath,Module,AbsInt,DumpFile)`

Given a `FilePath` and `Module` name, generates its associated `DumpFile`

- *The following properties should hold at call time:*
  - `FilePath` is an atom. (`atm/1`)
  - `Module` is an atom. (`atm/1`)
  - `AbsInt` is an atom. (`atm/1`)
  - `DumpFile` is a free variable. (`var/1`)

string_contained/2: PREDICATE

Usage: `string_contained(L1,L2)`

`L1` is a sublist of `L2`.

- *The following properties should hold at call time:*
  - `L1` is a string (a list of character codes). (`string/1`)
  - `L2` is a string (a list of character codes). (`string/1`)

get_mod_name_from_file/2: PREDICATE

Usage: `get_mod_name_from_file(FilePath,ModName)`

`ModName` is the name of the module located in `FilePath`.

newer/2: PREDICATE

Usage: `newer(FileA,FileB)`

`FileA` was modified later than `FileB`.

- *The following properties should hold at call time:*
  - `FileA` is currently instantiated to an atom. (`atom/1`)
  - `FileB` is currently instantiated to an atom. (`atom/1`)

make_dir_nofail/1: PREDICATE

Usage: `make_dir_nofail(X)`

Creates `X` if it does not exist already

– *The following properties should hold at call time:*

`X` is an atom. (`atm/1`)

111.3 Documentation on imports

This module has the following direct dependencies:

– *System library modules:*
`pathnames`, `system`, `lists`.

– *Internal (engine) modules:*
`term_basic`, `arithmetic`, `atomic_basic`, `basiccontrol`, `exceptions`, `term_compare`, `term_typing`, `debugger_support`, `basic_props`.

– *Packages:*
`prelude`, `initial`, `condcomp`, `assertions`, `assertions/assertions_basic`.

112 Database for ciaopp_batch process

Author(s): Isabel Garcia-Contreras.

This module manages the analysis database for ciaopp_batch.

112.1 Usage and interface

- **Library usage:**
  `:- use_module(ciaopp_batch(db_analysis)).`
- **Exports:**
  - *Predicates:*
    `add_task_status/5`, `task_status/5`, `update_task_status/0`, `last_file_log/2`, `init_analysis_db/0`, `init_tmp_db/1`, `clean_db/0`, `clean_tmp_db/0`, `set_current_task/2`, `current_task_proc/2`, `set_last_error/3`, `update_tasks_analysis_status/0`, `report_mod_info/3`, `db_data_dir/1`.
  - *Multifiles:*
    `$is_persistent/2`, `persistent_dir/2`, `persistent_dir/4`.

112.2 Documentation on exports

add_task_status/5: PREDICATE

Usage: `add_task_status(ModName,ModPath,Action,Flags,Status)`

Adds ciaopp action status for `ModName` located in `ModPath`

- *The following properties should hold at call time:*
  - `ModName` is an atom. (`atm/1`)
  - `ModPath` is an atom. (`atm/1`)
  - `Action` is an atom. (`atm/1`)
  - `Flags` is any term. (`term/1`)
  - `Status` is an atom. (`atm/1`)

task_status/5: PREDICATE

Usage: `task_status(ModName,ModPath,Action,Flags,Status)`

`Status` is the information of how `ModName` located in `ModPath` was analyzed

- *Call and exit should be compatible with:*
  - `ModName` is an atom. (`atm/1`)
  - `ModPath` is an atom. (`atm/1`)
  - `Action` is an atom. (`atm/1`)
  - `Flags` is any term. (`term/1`)
  - `Status` is an atom. (`atm/1`)

update_task_status/0: PREDICATE

Usage:

Updates the data with the last analysis information

last_file_log/2: PREDICATE

Usage: `last_file_log(ID,F)`

`F` is the associated file log to `ID`

- *The following properties should hold at call time:*
  `ID` is an atom. (`atm/1`)
- *The following properties should hold upon exit:*
  `F` is an atom. (`atm/1`)

init_analysis_db/0: PREDICATE

Usage:

Initializes analysis database

init_tmp_db/1: PREDICATE

Usage: `init_tmp_db(ID)`

Initializes a temporal DB for `ID`.

- *The following properties should hold at call time:*
  `ID` is an atom. (`atm/1`)

clean_db/0: PREDICATE

Usage:

Clears all analysis data.

clean_tmp_db/0: PREDICATE

Usage:

Removes all temporal databases.

set_current_task/2: PREDICATE

Usage: `set_current_task(ID,Task)`

Writes in a file that `Module` is currently being analyzed.

current_task_proc/2: PREDICATE

Usage: `current_task_proc(ID,Task)`

`Module` is the module being analyzed by `batch_analyze` currenlty

- *The following properties should hold at call time:*
  `Task` is a free variable. (`var/1`)

set_last_error/3: PREDICATE

Usage: `set_last_error(ModName,FilePath,AbsInt)`

Appends to the error file of `FilePath` that the analysis for domain `AbsInt` of the last module (`Module`) had a timeout.

update_tasks_analysis_status/0: PREDICATE

Usage:

Updates the task_status db to remove modules which are not updated, i.e., their `.dump` file is older than their `.pl` file.

For modules which could not be analyze, no information is updated

report_mod_info/3: PREDICATE

Usage: `report_mod_info(FilePath,ModName,LDiag)`

Stores in a file with extension `.err` `Info` about `ModName`

db_data_dir/1: PREDICATE

Usage: `db_data_dir(F)`

`F` is the location of data in this bundle

112.3 Documentation on multifiles

$is_persistent/2: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

The predicate is of type *data*.

persistent_dir/2: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

The predicate is of type *data*.

persistent_dir/4: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

The predicate is of type *data*.

112.4 Documentation on imports

This module has the following direct dependencies:

- *Application modules:*
  ciaopp_batch_aux.
- *System library modules:*
  datafacts_rt, persdb_rt, system, terms_io, system_extra, read, pathnames, write, bundle_paths, datadir, format.
- *Internal (engine) modules:*
  term_basic, arithmetic, atomic_basic, basiccontrol, exceptions, term_compare, term_typing, debugger_support, basic_props, hiord_rt, stream_basic.
- *Packages:*
  prelude, initial, condcomp, assertions, assertions/assertions_basic, isomodes, persdb, datafacts, persdb/persdb_decl.

113 tasks_db (library)

This handles the data base of modules to be analyzed. Currently it is implemented with persistent predicates

113.1 Usage and interface

- **Library usage:**
  `:- use_module(ciaopp_batch(tasks_db)).`
- **Exports:**
  - *Predicates:*
    `add_task/1`, `add_tasks/1`, `pop_task/4`, `analysis_task/4`, `clean_db/0`, `init_tasks_db/0`.
  - *Multifiles:*
    `$is_persistent/2`, `persistent_dir/2`, `persistent_dir/4`.

113.2 Documentation on exports

add_task/1: PREDICATE

Usage: `add_task(Mod)`

Add a task to analyze with abtract domain `AbsInt` and CiaoPP configuration `Flags`

add_tasks/1: PREDICATE

Usage: `add_tasks(Tasks)`

The same as `add_task/3` but for a list of tasks (for efficiency)

- *The following properties should hold at call time:*
  `Tasks` is a list. (`list/1`)

pop_task/4: PREDICATE

Usage: `pop_task(File,Mod,AbsInt,Flags)`

Remove a task to analyze

analysis_task/4: PREDICATE

Usage: `analysis_task(File,Mod,AbsInt,Flags)`

Checks a task to analyze, this predicate does not modify the DB

clean_db/0: PREDICATE

Usage 1:

Empties the database.

Usage 2:

Initializes the database.

init_tasks_db/0: PREDICATE

No further documentation available for this predicate.

113.3 Documentation on multifiles

$is_persistent/2: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

The predicate is of type *data*.

persistent_dir/2: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

The predicate is of type *data*.

persistent_dir/4: PREDICATE

No further documentation available for this predicate. The predicate is *multifile*.

The predicate is of type *data*.

113.4 Documentation on imports

This module has the following direct dependencies:

- *Application modules:*
  `db_analysis`.
- *System library modules:*
  `datafacts_rt`, `persdb_rt`, `format`, `bundle_paths`.
- *Internal (engine) modules:*
  `term_basic`, `arithmetic`, `atomic_basic`, `basiccontrol`, `exceptions`, `term_compare`, `term_typing`, `debugger_support`, `basic_props`.
- *Packages:*
  `prelude`, `initial`, `condcomp`, `assertions`, `assertions/assertions_basic`, `persdb`, `datafacts`, `persdb/persdb_decl`.

References

[APG06] E. Albert, G. Puebla, and J. Gallagher.
Non-Leftmost Unfolding in Partial Evaluation of Logic Programs with Impure Predicates.
In *15th International Symposium on Logic-based Program Synthesis and Transformation (LOPSTR'05)*, number 3901 in LNCS, pages 115–132. Springer-Verlag, April 2006.

[BCC04] F. Bueno, D. Cabeza, M. Carro, M. V. Hermenegildo, P. López-García, and G. Puebla-(Eds.).
The Ciao System. Reference Manual (v1.10).
The Ciao System Documentation Series–TR CLIP3/97.1.10(04), School of Computer Science, Technical University of Madrid (UPM), Facultad de Informática, 28660 Boadilla del Monte, Madrid, Spain, August 2004.
System and on-line version of the manual available at `http://ciao-lang.org`.

[BCHP96] F. Bueno, D. Cabeza, M. V. Hermenegildo, and G. Puebla.
Global Analysis of Standard Prolog Programs.
In *European Symposium on Programming*, number 1058 in LNCS, pages 108–124, Sweden, April 1996. Springer-Verlag.

[BGH94] F. Bueno, M. García de la Banda, and M. V. Hermenegildo.
The PLAI Abstract Interpretation System.
Technical Report CLIP2/94.0, Computer Science Dept., Technical U. of Madrid (UPM), Facultad Informatica UPM, 28660-Boadilla del Monte, Madrid-Spain, February 1994.

[BLGH04] F. Bueno, P. López-García, and M. V. Hermenegildo.
Multivariant Non-Failure Analysis via Standard Abstract Interpretation.
In *7th International Symposium on Functional and Logic Programming (FLOPS 2004)*, number 2998 in LNCS, pages 100–116, Heidelberg, Germany, April 2004. Springer-Verlag.

[CH94] D. Cabeza and M. Hermenegildo.
Extracting Non-strict Independent And-parallelism Using Sharing and Freeness Information.
In *1994 International Static Analysis Symposium*, number 864 in LNCS, pages 297–313, Namur, Belgium, September 1994. Springer-Verlag.

[CMB93] M. Codish, A. Mulkers, M. Bruynooghe, M. García de la Banda, and M. Hermenegildo.
Improving Abstract Interpretations by Combining Domains.
In *Proc. ACM SIGPLAN Symposium on Partial Evaluation and Semantics Based Program Manipulation*, pages 194–206. ACM, June 1993.

[COS96] The COSYTEC Team.
CHIP System Documentation, April 1996.

[DEDC96] P. Deransart, A. Ed-Dbali, and L. Cervoni.
Prolog: The Standard.
Springer-Verlag, 1996.

[DL93] S. K. Debray and N. W. Lin.
Cost Analysis of Logic Programs.
ACM Transactions on Programming Languages and Systems, 15(5):826–875, November 1993.

[DLGH97] S.K. Debray, P. López-García, and M. V. Hermenegildo.
Non-Failure Analysis for Logic Programs.
In *1997 International Conference on Logic Programming*, pages 48–62, Cambridge, MA, June 1997. MIT Press, Cambridge, MA.

[DLGHL97]
S. K. Debray, P. López-García, M. V. Hermenegildo, and N.-W. Lin.
Lower Bound Cost Estimation for Logic Programs.
In *1997 International Logic Programming Symposium*, pages 291–305. MIT Press, Cambridge, MA, October 1997.

[Dum94] Veroniek Dumortier.
Freeness and Related Analyses of Constraint Logic Programs Using Abstract Interpretation.
PhD thesis, K.U.Leuven, Dept. of Computer Science, October 1994.

[DZ92] P.W. Dart and J. Zobel.
A Regular Type Language for Logic Programs.
In *Types in Logic Programming*, pages 157–187. MIT Press, 1992.

[GdW94] J.P. Gallagher and D.A. de Waal.
Fast and Precise Regular Approximations of Logic Programs.
In Pascal Van Hentenryck, editor, *Proc. of the 11th International Conference on Logic Programming (ICLP'94)*, pages 599–613. MIT Press, 1994.

[GHM00] M. García de la Banda, M. V. Hermenegildo, and K. Marriott.
Independence in CLP Languages.
ACM Transactions on Programming Languages and Systems, 22(2):269–339, March 2000.

[HBC12] M. V. Hermenegildo, F. Bueno, M. Carro, P. López, E. Mera, J.F. Morales, and G. Puebla.
An Overview of Ciao and its Design Philosophy.
Theory and Practice of Logic Programming, 12(1–2):219–252, January 2012.
http://arxiv.org/abs/1102.5497.

[Her99] M. V. Hermenegildo.
A Documentation Generator for Logic Programming Systems.
Technical Report CLIP10/99.0, Facultad de Informática, UPM, September 1999.

[HPB99] M. V. Hermenegildo, G. Puebla, and F. Bueno.
Using Global Analysis, Partial Specifications, and an Extensible Assertion Language for Program Validation and Debugging.
In K. R. Apt, V. Marek, M. Truszczynski, and D. S. Warren, editors, *The Logic Programming Paradigm: a 25–Year Perspective*, pages 161–192. Springer-Verlag, July 1999.

[HPBLG05]
M. V. Hermenegildo, G. Puebla, F. Bueno, and P. Lopez-Garcia.
Integrated Program Debugging, Verification, and Optimization Using Abstract Interpretation (and The Ciao System Preprocessor).
Science of Computer Programming, 58(1–2):115–140, October 2005.

[HPMS00] M. V. Hermenegildo, G. Puebla, K. Marriott, and P. Stuckey.
Incremental Analysis of Constraint Logic Programs.
ACM Transactions on Programming Languages and Systems, 22(2):187–223, March 2000.

[HR95] M. Hermenegildo and F. Rossi.
Strict and Non-Strict Independent And-Parallelism in Logic Programs: Correctness,

Efficiency, and Compile-Time Conditions.
Journal of Logic Programming, 22(1):1–45, 1995.

[JB92] G. Janssens and M. Bruynooghe.
Deriving Descriptions of Possible Values of Program Variables by means of Abstract Interpretation.
Journal of Logic Programming, 13(2 and 3):205–258, July 1992.

[JL88] D. Jacobs and A. Langen.
Compilation of Logic Programs for Restricted And-Parallelism.
In *European Symposium on Programming*, pages 284–297, 1988.

[Knu84] D. Knuth.
Literate programming.
Computer Journal, 27:97–111, 1984.

[Leu98] M. Leuschel.
On the Power of Homeomorphic Embedding for Online Termination.
In Giorgio Levi, editor, *Proceedings of SAS'98*, volume 1503 of *LNCS*, pages 230–245, Pisa, Italy, September 1998. Springer-Verlag.

[LGBH05] P. López-García, F. Bueno, and M. V. Hermenegildo.
Determinacy Analysis for Logic Programs Using Mode and Type Information.
In *Proceedings of the 14th International Symposium on Logic-based Program Synthesis and Transformation (LOPSTR'04)*, number 3573 in LNCS, pages 19–35. Springer-Verlag, August 2005.

[LGBH10] P. López-García, F. Bueno, and M. V. Hermenegildo.
Automatic Inference of Determinacy and Mutual Exclusion for Logic Programs Using Mode and Type Information.
New Generation Computing, 28(2):117–206, 2010.

[LGHD96] P. López-García, M. V. Hermenegildo, and S. K. Debray.
A Methodology for Granularity Based Control of Parallelism in Logic Programs.
Journal of Symbolic Computation, Special Issue on Parallel Symbolic Computation, 21(4–6):715–734, 1996.

[MBdlBH99]
K. Muthukumar, F. Bueno, M. García de la Banda, and M. Hermenegildo.
Automatic Compile-time Parallelization of Logic Programs for Restricted, Goal-level, Independent And-parallelism.
Journal of Logic Programming, 38(2):165–218, February 1999.

[MH89a] K. Muthukumar and M. Hermenegildo.
Determination of Variable Dependence Information at Compile-Time Through Abstract Interpretation.
Technical Report ACA-ST-232-89, Microelectronics and Computer Technology Corporation (MCC), Austin, TX 78759, March 1989.

[MH89b] K. Muthukumar and M. Hermenegildo.
Determination of Variable Dependence Information at Compile-Time Through Abstract Interpretation.
In *1989 North American Conference on Logic Programming*, pages 166–189. MIT Press, October 1989.

[MH91] K. Muthukumar and M. Hermenegildo.
Combined Determination of Sharing and Freeness of Program Variables Through Abstract Interpretation.
In *International Conference on Logic Programming (ICLP 1991)*, pages 49–63. MIT Press, June 1991.

[MH92] K. Muthukumar and M. Hermenegildo.
Compile-time Derivation of Variable Dependency Using Abstract Interpretation.
Journal of Logic Programming, 13(2/3):315–347, July 1992.

[MS94] K. Marriott and P. Stuckey.
Approximating Interaction Between Linear Arithmetic Constraints.
In *1994 International Symposium on Logic Programming*, pages 571–585. MIT Press, 1994.

[Mye86] Eugene W Myers.
An O(ND) difference algorithm and its variations.
Algorithmica, 1(1-4):251–266, 1986.

[PAH04] G. Puebla, E. Albert, and M. V. Hermenegildo.
Abstract Interpretation with Specialized Definitions.
Technical Report CLIP12/2004.0, Technical University of Madrid, School of Computer Science, UPM, September 2004.

[PBH00] G. Puebla, F. Bueno, and M. V. Hermenegildo.
An Assertion Language for Constraint Logic Programs.
In P. Deransart, M. V. Hermenegildo, and J. Maluszynski, editors, *Analysis and Visualization Tools for Constraint Programming*, number 1870 in LNCS, pages 23–61. Springer-Verlag, September 2000.

[PCH03] G. Puebla, J. Correas, and M. V. Hermenegildo.
The design of a generic framework for context-sensitive analysis of modular programs.
Technical Report CLIP7/2003.0, Technical University of Madrid, School of Computer Science, UPM, November 2003.

[PH96] G. Puebla and M. V. Hermenegildo.
Optimized Algorithms for the Incremental Analysis of Logic Programs.
In *International Static Analysis Symposium (SAS 1996)*, number 1145 in Lecture Notes in Computer Science, pages 270–284. Springer-Verlag, September 1996.

[PRO] The PROLOG IV Team.
PROLOG IV Manual.

[SG94] H. Saglam and J. Gallagher.
Approximating Logic Programs Using Types and Regular Descriptions.
Technical Report CSTR-94-19, Department of Computer Science, University of Bristol, Bristol BS8 1TR, 1994.

[Son86] H. Søndergaard.
An Application of Abstract Interpretation of Logic Programs: Occur Check Reduction.
In *European Symposium on Programming, LNCS 123*, pages 327–338. Springer-Verlag, 1986.

[VB02] C. Vaucheret and F. Bueno.
More Precise yet Efficient Type Inference for Logic Programs.
In *9th International Static Analysis Symposium (SAS'02)*, volume 2477 of *Lecture Notes in Computer Science*, pages 102–116. Springer-Verlag, September 2002.

[VHCL95] P. Van Hentenryck, A. Cortesi, and B. Le Charlier.
Type analysis of prolog using type graphs.
Journal of Logic Programming, 22(3):179–209, 1995.

Library/Module Index

R

S

T

V

Predicate Index

$

A

B

C

D

F

G

T

U

V

W

Property Index

A

B

C

D

E

F

H

I

L

M

N

P

R

S

T

V

Regular Type Index

S

T

V

Declaration Index

C

D

E

M

P

R

S

T

Concept Index

Author Index

Global Index

This is a global index containing pointers to places where concepts, predicates, modes, properties, types, applications, authors, etc., are referred to in the text of the document.

$

,

(

*

-

.

/

:

=

+

A

B

C

D

E

F

G

I

M

N

O

P

Q

R

S

T

U

V

W

Y