# argmat-clpb

0.1

Generated by Doxygen 1.8.11

# Contents

**1 argumatrix** 1

**2 Hierarchical Index** 6

2.1 Class Hierarchy . . . . . . 6

**3 Class Index** 7

3.1 Class List . . . . . . 7

**4 File Index** 8

4.1 File List . . . . . . 8

**5 Class Documentation** 9

5.1 argumatrix::AdmissiblePlReasoner Class Reference . . . . . . 9

5.1.1 Detailed Description . . . . . . 12

5.1.2 Member Function Documentation . . . . . . 12

5.1.3 Member Data Documentation . . . . . . 27

5.2 argumatrix::ArgumatrixPlEngine Class Reference . . . . . . 28

5.2.1 Detailed Description . . . . . . 28

5.2.2 Constructor & Destructor Documentation . . . . . . 29

5.3 argumatrix::ArgumentProperty Struct Reference . . . . . . 31

5.3.1 Detailed Description . . . . . . 31

5.4 argumatrix::AttackProperty Class Reference . . . . . . 32

5.4.1 Detailed Description . . . . . . 32

5.5 argumatrix::bitmatrix Class Reference . . . . . . 32

5.5.1 Detailed Description . . . . . . 32

5.5.2 Member Function Documentation . . . . . . 33

5.6 argumatrix::bitvector Class Reference . . . . . . 33

5.6.1 Detailed Description . . . . . . 33

5.6.2 Member Function Documentation . . . . . . 33

5.7 argumatrix::ClpbProblem Struct Reference . . . . . . 35

5.7.1 Detailed Description . . . . . . 35

5.7.2 Member Function Documentation . . . . . 35
5.8 argumatrix::CompletePlReasoner Class Reference . . . . . 36
5.8.1 Detailed Description . . . . . 39
5.8.2 Member Function Documentation . . . . . 39
5.8.3 Member Data Documentation . . . . . 53
5.9 argumatrix::ConflictfreePlReasoner Class Reference . . . . . 54
5.9.1 Detailed Description . . . . . 57
5.9.2 Member Function Documentation . . . . . 57
5.9.3 Member Data Documentation . . . . . 72
5.10 argumatrix::DungAF Class Reference . . . . . 74
5.10.1 Detailed Description . . . . . 74
5.10.2 Member Function Documentation . . . . . 74
5.11 argumatrix::GroundedReasoner Class Reference . . . . . 76
5.11.1 Detailed Description . . . . . 78
5.11.2 Member Function Documentation . . . . . 78
5.11.3 Member Data Documentation . . . . . 87
5.12 boost::graph::internal_vertex_constructor< argumatrix::ArgumentProperty > Struct Template Reference . . . . . 88
5.12.1 Detailed Description . . . . . 88
5.13 boost::graph::internal_vertex_name< argumatrix::ArgumentProperty > Struct Template Reference 89
5.13.1 Detailed Description . . . . . 89
5.14 argumatrix::parser Class Reference . . . . . 89
5.14.1 Detailed Description . . . . . 89
5.14.2 Member Function Documentation . . . . . 89
5.15 argumatrix::PlReasoner Class Reference . . . . . 90
5.15.1 Detailed Description . . . . . 93
5.15.2 Member Function Documentation . . . . . 93
5.15.3 Member Data Documentation . . . . . 106
5.16 argumatrix::PlTimeQry Class Reference . . . . . 107
5.16.1 Detailed Description . . . . . 108
5.16.2 Constructor & Destructor Documentation . . . . . 108

5.16.3 Member Function Documentation . . . . . 110
5.17 argumatrix::PreferredReasoner Class Reference . . . . . 110
5.17.1 Detailed Description . . . . . 113
5.17.2 Member Function Documentation . . . . . 113
5.17.3 Member Data Documentation . . . . . 122
5.18 argumatrix::Reasoner Class Reference . . . . . 123
5.18.1 Detailed Description . . . . . 125
5.18.2 Member Function Documentation . . . . . 125
5.18.3 Member Data Documentation . . . . . 134
5.19 argumatrix::StablePlReasoner Class Reference . . . . . 135
5.19.1 Detailed Description . . . . . 138
5.19.2 Member Function Documentation . . . . . 138
5.19.3 Member Data Documentation . . . . . 152
**6 File Documentation 153**
6.1 F:/Codespace/BoostProject/argumatrix/argmat-clpb/ClpbProblem.hpp File Reference . . . . . 153
6.1.1 Detailed Description . . . . . 154
**Index 155**

# 1 argumatrix

AUTHOR: Fuan Pu, Pu.Fuan@gmail.com, School of Software, Tsinghua University

## Introduction

Argumatrix is a set of C/C++ codes to intended to assist a researcher in studying argumentation algorithms based on matrix theory.

## Build

The argumatrix can be build both on Window and Linux.

### Build on Windows

#### 1. Install dependences

The argumatrix depends on two open source libraries: boost and SWI-Prolog.

- Build Boost on Windows:

  Most Boost libraries are header-only: they consist entirely of header files containing templates and inline functions, and require no separately-compiled library binaries or special treatment when linking. However, some libraries must be built separately (see Header-Only Libraries). The following Boost libraries are used in argumatrix:

[x] Boost.Graph is used to construct and manage an argument graph.

- [x] Boost.dynamic_bitset is used to implement the bitvector and bitmatrix modules.
- [x] Boost.Regex is used in module **parser** to read aspartix files.

  in which the Boost.Regex library needs to be built separately. The simplest way to build boost libraries by Visual C++ can refer to Build Boost libraries by Visual C++. More details see Boost - Getting Started on Windows
- Install SWI-Prolog

  The stable release sources and excutables of SWI-Prolog can be downloaded from http://www.↩
swi-prolog.org/download/stable.

#### 2. Build argumatrix on windows

The argumatrix can be easily build on Windows, and can be debugged and run on Microsoft Visual Studio 2012. It is compatible with x64 machines in particular (however you need to make the x64 adjustments in VS2012 yourself).

- Download the source code package from github.
- Open *argumatrix.sln* in Visual Studio 2012. You will see that the argumatrix consists of several projects. Each project is a separate module and owns independent function. I provide each project with a main function, which can be used for the unit testing of the project, and in particular provides instructions to use the source codes in each module. I have configured these projects successfully, and if you have the different setting with mine, they may not be compiled successfully. Then, you should reconfigure them as the following steps.

```
argumatrix/ .................................... The "argumatrix" root directory
    argumatrix.sln ........................................ The solution file
    Project/ .........................................The project (module) folder
        XXX.hpp ................................. The source codes of each module
        XXX_main.cpp .................... The cpp code where the main function is
        Project.vcxproj..................The main project file for VC++ projects
        Project.vcxproj.filters...............The filters file for VC++ projects
        Makefile.................The makefile for compiling the project on linux
```

- Link Boost within the Visual Studio IDE

  For header-only libraries, you should specify where the boost header files are.

  - Right-click the project in the *Solution Explorer* pane and select *Properties* from the resulting pop-up menu.
  - In *Configuration Properties* > *C/C++∗* > ∗*General* > *Additional Include Directories*, enter the path to the Boost root directory, e.g., "D:/Boost/boost_1_57_0".

For separately-compiled libraries, you should also do the following steps:

– In *Configuration Properties* > *Linker* > *Additional Library Directories*, enter the path to the Boost binaries, e.g. D:/Boost/boost_1_57_0/stage/lib/.

Link SWI-Prolog within the Visual Studio IDE

- Add include folder: In *Configuration Properties* > *C/C++** > **General* > *Additional Include Directories*, enter the path to the SWI-Prolog root directory, e.g., "D:/swipl/include/".
- Add swipl library: In *Configuration Properties* > *Linker* > *Input* > *Additional dependences*, enter the path to the SWI-Prolog library, e.g. "D:/swipl/lib/libswipl.dll.a".

Build the solution in Debug and Release mode.

### Build on Linux

To build the argumatrix on linux may be quite easy since Boost and SWI-Prolog can be installed on linux. Of course, you can install them by compiling the source codes (see `Install SWI-Prolog on Linux from source codes`).

#### 1. Install Boost and SWI-Prolog on Linux

- Install Boost

  On Ubuntu, you can use the following command to install boost libraries

```
sudo apt-get install libboost-all-dev
```

As I use the Boost.Regex in module parser, you can compile the parser project by the following command

```
g++ -std=gnu++11 parser_main.cpp -lboost_regex -I ../
```

- Install SWI-Prolog

```
sudo apt-add-repository ppa:swi-prolog/stable
sudo apt-get update
sudo apt-get install swi-prolog
```

If you need a fairly recent version, I advice to compile the sources of the SWI-Prolog (see `Install SWI-Prolog on Linux from source codes`).

#### 2. Build argumatrix on linux

The argumatrix are organized by several projects, representing independent function module. Each project has a main function entry, which can be used for unit testing, and provides instructions to show how to use these codes. I provide each project with a *Makefile*, you can build each project easily with it.

```
make                             % default compile and link
make clean                       % clean all intermediate files
make rebuild                     % clean and build
make compile                     % only compile
```

You can also compile each project with one command, e.g., to build the argmat-clpb project, you can use:

```
g++ -std=gnu++11 argmat-clpb-main.cpp -L. -lswipl -lboost_regex -I ../ -I /usr/lib/swi-prolog/include/ -o
      argmat-clpb.a
```

## Modules and Usages

### dung_theory

The dung_theory module is the core module, and contains the basic concepts and operations of Dung's abstract argumentation. It also provides an abstract reasoner, which contains some basic operations about Dung's acceptability semantics, and implements the grounded reasoner. This module is essential to all modules.

**[Usage:]** The following codes present a basic way to construct an abstract argumentation framework:

```
{C++}
 // #include "DungAF.hpp"
 DungAF daf;  // define an abstract argumentation framework

 Argument a = daf.addArgument("A");  // add an argument with label "A"
 Argument b = daf.addArgument("B");
 Argument c = daf.addArgument("C");
 Argument d = daf.addArgument("D");
 Argument e = daf.addArgument("E");
 Argument f = daf.addArgument("F");

 // add attacks
 daf.addAttack(a, b);
 daf.addAttack(b, c);
 daf.addAttack(a, d);
 daf.addAttack(d, e);

 // or add attacks by labels
 daf.addAttack("D", "F");

 // print the argumentation framework
 cout << daf.toString() << endl;

 // Each argument does not only have an unique label, but also has
 // an unique index. The following codes can be used to obtain the
 // index of an argument when given the label or argument
 size_type idx1 = daf.getArgumentIdx("A");   // Given the label
 size_type idx2 = daf.getArgumentIdx(a);     // Given the Argument

 // The next code is to obtains all labels of arguments in an
 // argumentation framework, and store them into a vector of strings.
 vector<string> labels = daf.getArgumentLabels();

 // Then, we can achieve the label of an argument by its unique index, e.g.,
 string _arg_label = labels[idx2];
```

### bitmatrix

The bitmatrix module defines two data structures, *bitvector* and *bitmatrix*, that are used to represent an abstract argumentation framework. The *bitvector* can be used to represent a set of arguments as there is a one-to-one correspondence between a set and a Boolean vector. For example, let the universal set $\mathcal{X} = \{x_1, x_2, \cdots, x_5\}$ and the set $S = \{x_1, x_2, x_5\}$, then bitvector of $S$ can be represented as $bv = 11001$, in contrary, given a bitvector, we can easily obtain the set with respect to $\mathcal{X}$. The set operations, such as union, intersection and negate operations, can be easily implemented by the operators of *bitvector* . The *bitmatrix* is an array of bitvectors. It can be used to represent the attack graph of an argumentation framework, or a set of extensions.

**[Usage:]** The bitvector is implemented based on boost librarirs: <boost/dynamic_bitset/dynamic_bitset.hpp>.

```
{C++}
// #include "bitvector.hpp"
// #include "bitmatrix.hpp"

// Create a bitvector with a string
bitvector bv1("11001001");
cout << bv1 << endl;    // standard output

// Create a bitvector of length 8 with all bits are 0s
bitvector bv2(8, 0);

// intersection
bitvector bv3 = bv1 & bv2;
```

```

// union
bitvector bv3 = bv1 | bv2;

 // negate
bitvector bv4 = ~bv1;

// intersects with ..
bool is_in = bv1.intersects(bv2);
// or
bool is_in2 = (bv1 * bv2);

// is subset of
bool is_sub = bv1.is_subset_of(bv2);

// is equal
bool is_eq = (bv1 == bv2);


// Create an 8x8 bitmatrix
bitmatrix bm1(8);
bm1[1][4] = 1;      // set a bit

// Get the 5-th row of bm1, which is a bitvector
bitvector bv5 = bm1[5];

// The multiplication of a bitmatrix and a bitvector
bitvector bv6 = bm1 * bv1;

// Dung's abstract argumentation framework is a directed graph,
// which can be represented by an adjacent matrix with only 0s and 1s.
bitmatrix bm2 = daf.getAttackMatrix().
```

### parser

The parser module is provided to read abstract argumentation frameworks from files. Now, it supports the formats *aspartix* and *tgf*.

**[Usage:]** This module provides two static function to read *aspartix* and *tgf* files respectively.

```
{C++}
std::string inputFile = "../datasets/kleinberg/9/Kle_n3e0.9_01.dl";

DungAF daf;

// read aspartix
parser::Aspartix2DungAF(inputFile, daf);

// read tgf
parser::TrivialGraph2DungAF(inputFile, daf);
```

### PlReasoner

The PlReasoner module implements some acceptability semantics based on the SWI-Prolog engine using the prolog module Constraint Logic Programming over Boolean variables. It currently contains ConflictfreeReasoner, StableReasoner, AdmissibleReasoner and CompleteReasoner.

**[Usage:]**

```
{C++}
Reasoner* rsner = new AdmissiblePlReasoner(daf, "AD2");
cout << "EE:" << endl;
rsner->task_EE();
```

#### argmat-clpb

The argmat-clpb module provides a command line interface to excute argumentation reasoning tasks based on PlReasoner. The command line interface follows the *probo* interface in the `rules of ICCMA15`.

- Command line interface
  - ./argmat-clpb
    To display version information and usages.
  - ./argmat-clpb –help
    To print usages.
  - ./argmat-clpb –problem
    To print the supported acceptability semantics and reasoning tasks.
  - ./argmat-clpb –format
    To print the supported file format.
  - ./argmat-clpb -p <problem> -f <file> -fo <fileformat> [-a <additional_parameter>] [-o <file>]
    - -p Specify the problems,
    - -f Specify the file which stores the argumentation framework
    - -fo Specify the file format
    - -a Specify the additional argument(s), if specify a set of arguments, it must be divided by a comma with no space. For example, "-a arg1,arg2,arg3"
    - -o Specify the output file where the result are stored. This parameter is optional, if it is not specified, then the standard output is used.

## 2 Hierarchical Index

### 2.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

**argumatrix::ArgumatrixPlEngine** 28
**argumatrix::ArgumentProperty** 31
**argumatrix::AttackProperty** 32
**argumatrix::bitmatrix** 32
**argumatrix::ClpbProblem** 35
**argumatrix::DungAF** 74
dynamic_bitset
    **argumatrix::bitvector** 33
**boost::graph::internal_vertex_constructor< argumatrix::ArgumentProperty >** 88
**boost::graph::internal_vertex_name< argumatrix::ArgumentProperty >** 89
**argumatrix::parser** 89
**argumatrix::PlTimeQry** 107

argumatrix::Reasoner 123
  argumatrix::GroundedReasoner 76
  argumatrix::PlReasoner 90
    argumatrix::AdmissiblePlReasoner 9
    argumatrix::CompletePlReasoner 36
    argumatrix::ConflictfreePlReasoner 54
    argumatrix::StablePlReasoner 135
  argumatrix::PreferredReasoner 110

# 3 Class Index

## 3.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

argumatrix::AdmissiblePlReasoner ??

argumatrix::ArgumatrixPlEngine
We rewrite the PlEngine since we do not want the PlEngine to print the welcome and version information. These information can not be printed if we use command "-g true" or "--quiet" or "-q". Note that both PlEngine and ArgumatrixPlEngine must be defined as a static instance, since multiple engines exist may lead to crash when one of the engins is closed (i.e., invoking PL_cleanup) then all other running engines are closed. Define PlEngine as a static instance can solve this problem since static instances are global and stored in static variable area. They are initialized before the main function and released when the program exits ??

argumatrix::ArgumentProperty ??

argumatrix::AttackProperty ??

argumatrix::bitmatrix ??

argumatrix::bitvector ??

argumatrix::ClpbProblem ??

argumatrix::CompletePlReasoner
Tackle computational problems for complete semantics. TODO: long description ??

argumatrix::ConflictfreePlReasoner ??

argumatrix::DungAF ??

argumatrix::GroundedReasoner ??

boost::graph::internal_vertex_constructor< argumatrix::ArgumentProperty > ??

boost::graph::internal_vertex_name< argumatrix::ArgumentProperty >
Use the label of ArgumentProperty as a key for indexing Arguments in a graph ??

argumatrix::parser ??

argumatrix::PIReasoner ??

argumatrix::PITimeQry
Create a *time* query which can output the time information ??

argumatrix::PreferredReasoner ??

argumatrix::Reasoner ??

argumatrix::StablePIReasoner ??

## 4 File Index

### 4.1 File List

Here is a list of all documented files with brief descriptions:

F:/Codespace/BoostProject/argumatrix/argmat-clpb/argmat-clpb-main.cpp ??

F:/Codespace/BoostProject/argumatrix/argmat-clpb/ClpbProblem.hpp
TODO: long description ??

F:/Codespace/BoostProject/argumatrix/bitmatrix/bitmatrix.hpp ??

F:/Codespace/BoostProject/argumatrix/bitmatrix/bitmatrix_main.cpp ??

F:/Codespace/BoostProject/argumatrix/bitmatrix/bitvector.hpp ??

F:/Codespace/BoostProject/argumatrix/config/config.hpp ??

F:/Codespace/BoostProject/argumatrix/config/config_main.cpp ??

F:/Codespace/BoostProject/argumatrix/config/cptimer.hpp ??

F:/Codespace/BoostProject/argumatrix/dung_theory/ArgumentProperty.hpp ??

F:/Codespace/BoostProject/argumatrix/dung_theory/AttackProperty.hpp ??

F:/Codespace/BoostProject/argumatrix/dung_theory/dung_main.cpp ??

F:/Codespace/BoostProject/argumatrix/dung_theory/DungAF.hpp ??

F:/Codespace/BoostProject/argumatrix/dung_theory/GroundedReasoner.hpp ??

F:/Codespace/BoostProject/argumatrix/dung_theory/PreferredReasoner.hpp ??

F:/Codespace/BoostProject/argumatrix/dung_theory/Reasoner.hpp ??

F:/Codespace/BoostProject/argumatrix/parser/parser.hpp ??

F:/Codespace/BoostProject/argumatrix/parser/parser_main.cpp ??

F:/Codespace/BoostProject/argumatrix/PIReasoner/AdmissiblePIReasoner.hpp ??

F:/Codespace/BoostProject/argumatrix/PIReasoner/argmat-clpb.pl ??

F:/Codespace/BoostProject/argumatrix/PIReasoner/CompletePIReasoner.hpp ??

F:/Codespace/BoostProject/argumatrix/PIReasoner/ConflictfreePIReasoner.hpp ??

**F:/Codespace/BoostProject/argumatrix/PlReasoner/pl_main.cpp** ??

**F:/Codespace/BoostProject/argumatrix/PlReasoner/PlReasoner.hpp** ??

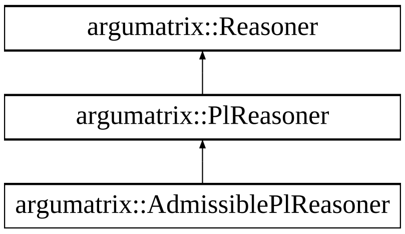
**F:/Codespace/BoostProject/argumatrix/PlReasoner/StablePlReasoner.hpp** ??

**F:/Codespace/BoostProject/argumatrix/PlReasoner/swipl-util.hpp** ??

# 5 Class Documentation

## 5.1 argumatrix::AdmissiblePlReasoner Class Reference

Inheritance diagram for argumatrix::AdmissiblePlReasoner:



### Public Member Functions

- **AdmissiblePlReasoner** (const DungAF &daf, const string &sm_task="AD", streambuf ∗osbuff=std::cout.↩ rdbuf())
- void task_EE ()

  *Print all extensions (with a vector of integers {0,1,2})*
- void **task_EX** ()
- void task_EC (const std::set< string > &argset)

  *Given an ${AF}=< {X}, {R}>$ and an argument $s {X}$ (respectively, a set of arguments $S {X}$), enumerate all sets $E {X}$ such that $E {E}_(AF)$ and $s E$ (respectively, $S E$).*
- void task_SC (const std::set< string > &argset)

  *Given an ${AF}=< {X}, {R}>$ and an argument $s {X}$ (respectively, a set of arguments $S {X}$), enumerate some set $E {X}$ such that $E {E}_(AF)$ and $s E$ (respectively, $S E$).*
- void task_SE ()

  *Given an ${AF}=< {X}, {R}>$, enumerate some set $E {X}$ that are in ${E}_(AF)$.*
- void findAllExts ()
- void task_DE (const std::set< string > &argset)

  *Given an $AF = <X,R>$ and a set of arguments $S X$. Decide whether S is a -extension of AF, i.e., S E_(AF).*
- void task_DN ()

  *Given an $AF = <X,R>$ and a set of arguments $S X$. Decide whether there exist a non-empty -extension for AF.*
- void task_DC (const std::set< string > &argset)

  *Given an $AF = <X,R>$ and an argument s X (respectively, a set of arguments $S X$). Decide whether s contained (respectively, S included) in some E E_(AF) (i.e., credulously justified).*
- void task_DS (const std::set< string > &argset)

  *Given an $AF = <X,R>$ and an argument s X (respectively, a set of arguments $S X$). Decide whether s contained (respectively, S included) in each E E_(AF) (i.e., skeptically justified).*
- void consultPlFile (std::string plFile=ARG_PROLOG_FILE)

  *Loading the Prolog source file [argmat-clpb].*

- void **Test_time** ()
- void printAllExts (const std::string &predct)

  *Print all extensions with respect to semantic predct.*
- void printAllExts (const std::string &predct, const std::set< string > &argset)

  *Given an ${AF}=< {X}, {R}>$ and an argument $s {X}$ (respectively, a set of arguments $S {X}$), enumerate all sets $E {X}$ such that $E {E}_(AF)$ and $s E$ (respectively, $S E$).*
- void **printAllExts** (const std::string &predct, const std::vector< int > &vecii)
- void **printAllExts2** (const std::string &predct)
- void fetchAllExts (const std::string &predct)
- void printSomeExt (const std::string &predct, const std::set< string > &argset)

  *Given an ${AF}=< {X}, {R}>$ and an argument $s {X}$ (respectively, a set of arguments $S {X}$), enumerate some set $E {X}$ such that $E {E}_(AF)$ and $s E$ (respectively, $S E$).*
- void **printSomeExt** (const std::string &predct, const vector< int > &vecii)
- bool verifyNonemptyExt (const std::string &predct)

  *Given an $AF = <X,R>$ and a set of arguments $S X$. Decide whether there exist a non-empty $$-extension for AF.*
- bool verifyExtension (const std::string &predct, std::set< string > &argset)

  *Given an $AF = <X,R>$ and a set of arguments $S X$. Decide whether S is a $$-extension of AF, i.e., $S E_(AF)$.*
- bool **verifyExtension** (const std::string &predct, bitvector &bvecii)
- bool isCredulouslyJustified (const std::string &predct, const std::set< string > &argset)

  *Given an $AF = <X,R>$ and an argument s X (respectively, a set of arguments $S X$). Decide whether s contained (respectively, S included) in some E E_(AF) (i.e., credulously justified).*
- bool isSkepticallyJustified (const std::string &predct, const std::set< string > &argset)

  *Given an $AF = <X,R>$ and an argument s X (respectively, a set of arguments $S X$). Decide whether s contained (respectively, S included) in each E E_(AF) (i.e., skeptically justified).*
- bitvector getAttacked (const bitvector &_bv)

  *Get the attacked arguments by arguments in _bv,* $R^+(S)$ $R^+(S) = x|xisattackedbyS$. $R^+(S_bv) = D * S_bv$.
- bitvector characteristic (const bitvector &_bv)

  *The characteristic function of an abstract argumentation framework:* $F_{AF}(S) = A|Aisacceptablewrt.S$. *F_AF(↩ S_bv) = not{ R^+[ not( R^+(S_bv) ) ] }.*
- bitvector characteristic ()

  *The characteristic function of an abstract argumentation framework with the initialization of empty set.*
- bitvector neutrality (const bitvector &_bv)

  *The neutrality function of an abstract argumentation framework: N_AF(S) = {A| All arguments that not attacked by S}. N_AF(S_bv) = not( R^+(S_bv) )*
- bool is_conflict_free (const bitvector &_bv)

  *Argument set S is conflict-free? S is said to be conflict-free iff for any two arguments a,b in S such that a does not attack b.*
- bool **is_conflict_free** (const set< string > &argset)
- bool is_acceptable (const bitvector &S, const bitvector &A)

  *Argument set A is acceptable w.r.t S? Alternately, S defends A? A can be an argument or an argument set. S defends argument (set) A iff for any argument x in X, if attacks A (or a in A) then there is an argument y in S such that y attacks x.*
- bool is_self_attacking (size_type idx)

  *Argument a is self-attacking iff it attacks itself.*
- bool is_admissible (const bitvector &_bv)

  *To decide whether a set of arguments (with the form of bitvector) is an admissible extension. $S$ is an admissible extension iff $S F(S) N(S)$.*
- bool is_complete (const bitvector &_bv)

  *To decide whether a set of arguments (with the form of bitvector) is a complete extension. $S$ is a complete extension iff $S == F(S) N(S)$.*
- bool is_stable (const bitvector &_bv)

  *To decide whether a set of arguments (with the form of bitvector) is a stable extension. $S$ is a stable extension iff $S == N(S)$.*

- bool is_grounded (const bitvector &_bv)

  *To decide whether a set of arguments (with the form of bitvector) is a grounded extension. $S$ is a grounded extension iff it is the least fixed point of the characteristic function F.*
- bitvector getSelfAttackingArguments ()

  *Get all arguments which are self-attacking. Obviously, if an argument i attacks itself, the entry[i][i] of the attack matrix must be 1 (true). Therefore to get the set of self-attacking arguments is to get the diagonal elements of the attack matrix.*
- const set< bitvector > & getBvExtensions ()

  *Get all extensions given a specific semantics in format of bitvector. Each extension is a set of arguments which can "survive the conflict together". Here, we use a bitvector to represent an extension. Therefore, all extensions are formed an set of bit vectors, i.e., set<bitvector>. Each bitvector in set is an extension w.r.t. a given semantics. The computed extensions are stored in m_extensions, therefore, to get all extensions, it must first invoke the function computeExtension()*
- vector< int > getGroundedIntVector ()

  *Get a vector of integers {0, 1, 2}: 2 – Unknown, 1 – in grounded extension, and 0 – attacked by the grounded extension.*
- streambuf ∗ setOutput (streambuf ∗strbuf)

  *Redirect the output stream to streambuf∗ strbuf or ostream& os. If strbuf = cout.rdbuf(), then the output is standard output. It can also redirect the output to a file by the following codes:*
  ```
  ofstream ofile("output.txt");
  streambuf* oldsb = xxx.setOutput(ofile.rdbuf());
  ```
  *or*
  ```
  streambuf* oldsb = xxx.setOutput(ofile);
  ```
  .
- streambuf ∗ setOutput (ostream &os=std::cout)
- void printLabSet (const bitvector &bv_ext)

  *Print an extension with the form of bitvector. Assume the bitvector is [0, 1, 0, 1], the arguments corresponding to entry 1 will be print. The member m_output will determine where to print.*
- void printLabSet (const std::set< string > &labset)

  *Print a set of arguments. The member m_output will determine where to print.*
- void printBvExts ()

  *Output all extensions given a specific semantics with an ostream. Each extension is a set of arguments which can "survive the conflict together". Here, we return a string to represent all extensions. For example, if there are two extensions, [a,b] and [b, d], for some problem w.r.t. a given semantics, then we return string "[[a,b],[d,c]]". If extensions is not existing, the string "[ ]" will return.*
- bitvector getGroundedExtension ()

  *Get the grounded extension.*
- void printGroundedExt ()

  *Print the grounded extension.*
- vector< int > labelSet2IntVector (const std::set< string > &label_set)

  *Convert a set of argument (string) labels to an integer vector of {0,1,2}, All arguments in these set, the indices is 1; otherwise 2 (representing unknown)*

### Protected Member Functions

- void findAllExts (const std::string &predct)

  *Find all extensions and store them in m_extensions with the form of bitvector.*
- void createPlAttackMatrix ()

  *Create the attack matrix with the form of PlTerm, which is an input of SWI-Prolog.*
- void printLableExtByBlistTerm (const PlTerm &plt)

  *Print an extension with the form of the bool list term. Assume a bool list term is [0, 1, 0, 1], the arguments corresponding to entry 1 will be print. The member m_output will determine where to print.*
- void printLableExtByBmatrixTerm (const PlTerm &pmtx)

  *Print an extension with the form of the bool matrix term. Assume a bool list term is [[0, 1, 0, 1], [0, 1, 0, 1]], the arguments corresponding to entry 1 will be print. The member m_output will determine where to print.*

- void addBlTermToBvExts (const PlTerm &plt)

  *Convert a bool list term into a bitvertor, and add it to extension.*
- bool verifyInclusion (const std::string &predct, const vector< int > &vecii)

  *Given an $AF = <X,R>$ and a set of arguments $S X$. Decide whether there exist a -extension that includes S.*
- bool verifyExclusion (const std::string &predct, const bitvector &vecB)

  *Given an $AF = <X,R>$ and a set of arguments $S X$. Decide whether there exist a -extension that excludes S.*

### Protected Attributes

- string **m_predicate**
- PlTerm **m_PlAtkMtx**
- bitmatrix m_BmAtkMtx
- size_type m_argNum
- const DungAF & m_daf
- set< bitvector > **m_extensions**
- vector< std::string > m_argLabels
- std::ostream m_output

### 5.1.1 Detailed Description

Definition at line 37 of file AdmissiblePlReasoner.hpp.

### 5.1.2 Member Function Documentation

#### 5.1.2.1 void argumatrix::PlReasoner::addBlTermToBvExts ( const PlTerm & *plt* ) [protected], [inherited]

Convert a bool list term into a bitvertor, and add it to extension.

**Parameters**

| | |
|---|---|
| *PlTerm&* | plt : The PlTerm is a bool list term, which represents an extension. |

**Returns**

no return. The results are added into m_extensions.

Definition at line 303 of file PlReasoner.hpp.

#### 5.1.2.2 __inline argumatrix::bitvector argumatrix::Reasoner::characteristic ( const bitvector & *_bv* ) [inherited]

The characteristic function of an abstract argumentation framework: $F_{AF}(S) = A|Aisacceptablewrt.S.$ F_AF(↩
S_bv) = not{ R^+[ not( R^+(S_bv) ) ] }.

**Parameters**

| | |
|---|---|
| *extension* | an extension (a set of arguments), the default is empty set. |

**Returns**

a set of arguments in bitvector form.

Definition at line 368 of file Reasoner.hpp.

#### 5.1.2.3 __inline argumatrix::bitvector argumatrix::Reasoner::characteristic ( ) [inherited]

The characteristic function of an abstract argumentation framework with the initialization of empty set.

**Returns**

a set of arguments in bitvector form.

Definition at line 375 of file Reasoner.hpp.

Referenced by argumatrix::Reasoner::getGroundedExtension(), and argumatrix::Reasoner::is_acceptable().

#### 5.1.2.4 void argumatrix::PlReasoner::consultPlFile ( std::string *plFile* = `ARG_PROLOG_FILE` ) [inherited]

Loading the Prolog source file [argmat-clpb].

Method: consultPlFile

**Parameters**

| | |
|---|---|
| *std::string* | plFile |

**Note**

SWI-Prolog supports compilation of individual or multiple Prolog source files into ‘Quick Load Files’. A ‘Quick Load File’ (.qlf file) stores the contents of the file in a precompiled format. These files load considerably faster than source files and are normally more compact. They are machine-independent and may thus be loaded on any implementation of SWI-Prolog. Note, however, that clauses are stored as virtual machine instructions. Changes to the compiler will generally make old compiled files unusable. Quick Load Files are created using qcompile/1. They are loaded using consult/1 or one of the other file-loading predicates described in section 4.3. If consult/1 is given an explicit .pl file, it will load the Prolog source. When given a .qlf file, it will load the file. When no extension is specified, it will load the .qlf file when present and the .pl file otherwise.

Definition at line 253 of file PlReasoner.hpp.

#### 5.1.2.5 void argumatrix::PlReasoner::createPlAttackMatrix ( ) [protected], [inherited]

Create the attack matrix with the form of PlTerm, which is an input of SWI-Prolog.

**Returns**

no return. The result is stored in member variable *m_PlAtkMtx*.

Definition at line 243 of file PlReasoner.hpp.

#### 5.1.2.6 void argumatrix::PlReasoner::fetchAllExts ( const std::string & *predct* ) [inherited]

Method: fetchAllExts FullName: public argumatrix::PlReasoner::fetchAllExts

**Parameters**

| | |
|---|---|
| *const* | std::string & predct |

**Returns**

void

**Return values**

| | |
|---|---|
| | |

Definition at line 701 of file PlReasoner.hpp.

#### 5.1.2.7 __inline void argumatrix::AdmissiblePlReasoner::findAllExts ( )

Find all extensions and store them in *m_extensions* with the form of bitvector.

**Parameters**

| | |
|---|---|
| *no* | argument. |

**Returns**

no return. The results are added into *m_extensions*.

Definition at line 177 of file AdmissiblePlReasoner.hpp.

#### 5.1.2.8 void argumatrix::PlReasoner::findAllExts ( const std::string & *predct* ) [protected], [inherited]

Find all extensions and store them in *m_extensions* with the form of bitvector.

**Parameters**

| | |
|---|---|
| *string&* | predct: The predicate. |

**Returns**

no return. The results are added into *m_extensions*.

Definition at line 669 of file PlReasoner.hpp.

Referenced by findAllExts(), argumatrix::StablePlReasoner::findAllExts(), argumatrix::ConflictfreePlReasoner↩
::findAllExts(), and argumatrix::CompletePlReasoner::findAllExts().

#### 5.1.2.9 __inline argumatrix::bitvector argumatrix::Reasoner::getAttacked ( const bitvector & *_bv* ) [inherited]

Get the attacked arguments by arguments in _bv, $R^+(S)$ $R^+(S) = x|xisattackedbyS$. $R^+(S_bv) = D * S_bv$.

**Parameters**

| | |
|---|---|
| *_bv* | the bitvector with respect to the set $S$. |

**Returns**

a set of arguments in bitvector form.

Definition at line 360 of file Reasoner.hpp.

Referenced by argumatrix::Reasoner::getGroundedIntVector(), argumatrix::Reasoner::is_conflict_free(), and argumatrix::Reasoner::neutrality().

#### 5.1.2.10 __inline const set< bitvector > & argumatrix::Reasoner::getBvExtensions ( ) [inherited]

Get all extensions given a specific semantics in format of bitvector. Each extension is a set of arguments which can "survive the conflict together". Here, we use a bitvector to represent an extension. Therefore, all extensions are formed an set of bit vectors, i.e., set<bitvector>. Each bitvector in set is an extension w.r.t. a given semantics. The computed extensions are stored in m_extensions, therefore, to get all extensions, it must first invoke the function computeExtension()

**Returns**

a set of bitvector, i.e., set<bitvector>.

Definition at line 400 of file Reasoner.hpp.

#### 5.1.2.11 argumatrix::bitvector argumatrix::Reasoner::getGroundedExtension ( ) [inherited]

Get the grounded extension.

**Returns**

a set of arguments in bitvector form.

Definition at line 484 of file Reasoner.hpp.

Referenced by argumatrix::Reasoner::is_grounded(), and argumatrix::Reasoner::printGroundedExt().

#### 5.1.2.12 vector< int > argumatrix::Reasoner::getGroundedIntVector ( ) [inherited]

Get a vector of integers {0, 1, 2}: 2 – Unknown, 1 – in grounded extension, and 0 – attacked by the grounded extension.

**Returns**

a vector of integers {0, 1, 2}.

Definition at line 499 of file Reasoner.hpp.

#### 5.1.2.13 __inline argumatrix::bitvector argumatrix::Reasoner::getSelfAttackingArguments ( ) [inherited]

Get all arguments which are self-attacking. Obviously, if an argument i attacks itself, the entry[i][i] of the attack matrix must be 1 (true). Therefore to get the set of self-attacking arguments is to get the diagonal elements of the attack matrix.

**Returns**

a set of arguments in bitvector form, if bitvector[i] = true representing the argument (with index) i is a self-attacking argument, otherwise is not.

Definition at line 414 of file Reasoner.hpp.

#### 5.1.2.14 __inline bool argumatrix::Reasoner::is_acceptable ( const bitvector & *S*, const bitvector & *A* ) [inherited]

Argument set A is acceptable w.r.t S? Alternately, S defends A? A can be an argument or an argument set. S defends argument (set) A iff for any argument x in X, if attacks A (or a in A) then there is an argument y in S such that y attacks x.

**Parameters**

| | |
|---|---|
| *S* | a set of arguments. |
| *A* | an argument or an argument set |

**Returns**

true if S defends A.

Definition at line 394 of file Reasoner.hpp.

#### 5.1.2.15 __inline bool argumatrix::Reasoner::is_admissible ( const bitvector & *_bv* ) [inherited]

To decide whether a set of arguments (with the form of bitvector) is an admissible extension. $S$ is an admissible extension iff $S F(S) N(S)$.

**Returns**

true if _bv is admissible; otherwise returns false.

Definition at line 553 of file Reasoner.hpp.

#### 5.1.2.16 __inline bool argumatrix::Reasoner::is_complete ( const bitvector & *_bv* ) [inherited]

To decide whether a set of arguments (with the form of bitvector) is a complete extension. $S$ is a complete extension iff $S == F(S) N(S)$.

**Returns**

true if _bv is complete; otherwise returns false.

Definition at line 564 of file Reasoner.hpp.

#### 5.1.2.17 __inline bool argumatrix::Reasoner::is_conflict_free ( const bitvector & *_bv* ) [inherited]

Argument set S is conflict-free? S is said to be conflict-free iff for any two arguments a,b in S such that a does not attack b.

**Parameters**

| | |
|---|---|
| *extension* | an extension (a set of arguments). |

**Returns**

true if S is conflict-free.

Definition at line 381 of file Reasoner.hpp.

#### 5.1.2.18 __inline bool argumatrix::Reasoner::is_grounded ( const bitvector & *_bv* ) [inherited]

To decide whether a set of arguments (with the form of bitvector) is a grounded extension. $S$ is a grounded extension iff it is the least fixed point of the characteristic function F.

**Returns**

true if _bv is grounded; otherwise returns false.

Definition at line 584 of file Reasoner.hpp.

#### 5.1.2.19 __inline bool argumatrix::Reasoner::is_self_attacking ( size_type *idx* ) [inherited]

Argument a is self-attacking iff it attacks itself.

**Parameters**

| | |
|---|---|
| *idx* | the index of the argument w.r.t the attack matrix |

**Returns**

true if S defends A.

Definition at line 406 of file Reasoner.hpp.

#### 5.1.2.20 __inline bool argumatrix::Reasoner::is_stable ( const bitvector & *_bv* ) [inherited]

To decide whether a set of arguments (with the form of bitvector) is a stable extension. $S$ is a stable extension iff $S == N(S)$.

**Returns**

true if _bv is stable; otherwise returns false.

Definition at line 575 of file Reasoner.hpp.

#### 5.1.2.21 bool argumatrix::PlReasoner::isCredulouslyJustified ( const std::string & *predct,* const std::set< string > & *argset* ) `[inherited]`

Given an $AF = <X,R>$ and an argument s X (respectively, a set of arguments $S X$). Decide whether s contained (respectively, S included) in some E E_(AF) (i.e., credulously justified).

Problem [DC-$$]

Definition at line 642 of file PlReasoner.hpp.

#### 5.1.2.22 bool argumatrix::PlReasoner::isSkepticallyJustified ( const std::string & *predct,* const std::set< string > & *argset* ) `[inherited]`

Given an $AF = <X,R>$ and an argument s X (respectively, a set of arguments $S X$). Decide whether s contained (respectively, S included) in each E E_(AF) (i.e., skeptically justified).

Problem [DS-$$]

Definition at line 654 of file PlReasoner.hpp.

#### 5.1.2.23 vector< int > argumatrix::Reasoner::labelSet2IntVector ( const std::set< string > & *label_set* ) `[inherited]`

Convert a set of argument (string) labels to an integer vector of {0,1,2}, All arguments in these set, the indices is 1; otherwise 2 (representing unknown)

**Returns**

no return.

Definition at line 526 of file Reasoner.hpp.

#### 5.1.2.24 __inline argumatrix::bitvector argumatrix::Reasoner::neutrality ( const bitvector & *_bv* ) `[inherited]`

The neutrality function of an abstract argumentation framework: N_AF(S) = {A| All arguments that not attacked by S}. N_AF(S_bv) = not( R^+(S_bv) )

**Parameters**

| | |
|---|---|
| *extension* | an extension (a set of arguments). |

**Returns**

a set of arguments in bitvector form.

Definition at line 547 of file Reasoner.hpp.

Referenced by argumatrix::Reasoner::characteristic(), argumatrix::Reasoner::getGroundedIntVector(), argumatrix↩ ::Reasoner::is_admissible(), argumatrix::Reasoner::is_complete(), and argumatrix::Reasoner::is_stable().

#### 5.1.2.25 void argumatrix::PlReasoner::printAllExts ( const std::string & *predct* ) `[inherited]`

Print all extensions with respect to semantic predct.

Problem [EE-$$]

**Parameters**

| | |
|---|---|
| *no* | argument |

**Returns**

no return.

Definition at line 430 of file PIReasoner.hpp.

Referenced by argumatrix::PIReasoner::printAllExts(), argumatrix::StablePIReasoner::task_EC(), task_EC(), argumatrix::ConflictfreePIReasoner::task_EC(), argumatrix::CompletePIReasoner::task_EC(), argumatrix::↩ ConflictfreePIReasoner::task_EE(), task_EE(), argumatrix::StablePIReasoner::task_EE(), and argumatrix::↩ CompletePIReasoner::task_EE().

#### 5.1.2.26 __inline void argumatrix::PIReasoner::printAllExts ( const std::string & *predct*, const std::set< string > & *argset* ) [inherited]

Given an ${AF}=< {X}, {R}>$ and an argument $s {X}$ (respectively, a set of arguments $S {X}$), enumerate all sets $E {X}$ such that $E {E}_(AF)$ and $s E$ (respectively, $S E$).

Problem [EC-$$]

**Parameters**

| | |
|---|---|
| *set<string>,or* | a Boolean vector: a set of argument |

**Returns**

no return.

Definition at line 386 of file PIReasoner.hpp.

#### 5.1.2.27 __inline void argumatrix::Reasoner::printBvExts ( ) [inherited]

Output all extensions given a specific semantics with an ostream. Each extension is a set of arguments which can "survive the conflict together". Here, we return a string to represent all extensions. For example, if there are two extensions, [a,b] and [b, d], for some problem w.r.t. a given semantics, then we return string "[[a,b],[d,c]]". If extensions is not existing, the string "[ ]" will return.

**Parameters**

| | |
|---|---|
| *ostream&* | os = std::cout, output the resluts into the ostream os. |

**Returns**

a set of bitvector, i.e., set<bitvector>.

Definition at line 428 of file Reasoner.hpp.

#### 5.1.2.28 __inline void argumatrix::Reasoner::printGroundedExt ( ) [inherited]

Print the grounded extension.

**Returns**

no return.

Definition at line 539 of file Reasoner.hpp.

#### 5.1.2.29 void argumatrix::PlReasoner::printLableExtByBlistTerm ( const PlTerm & *plt* ) [protected], [inherited]

Print an extension with the form of the bool list term. Assume a bool list term is [0, 1, 0, 1], the arguments corresponding to entry 1 will be print. The member *m_output* will determine where to print.

**Parameters**

| | |
|---|---|
| *PlTerm&* | plt : The PlTerm is a bool list term, which represents an extension. |

**Returns**

no return.

Definition at line 271 of file PlReasoner.hpp.

#### 5.1.2.30 void argumatrix::PlReasoner::printLableExtByBmatrixTerm ( const PlTerm & *pmtx* ) [protected], [inherited]

Print an extension with the form of the bool matrix term. Assume a bool list term is [[0, 1, 0, 1], [0, 1, 0, 1]], the arguments corresponding to entry 1 will be print. The member *m_output* will determine where to print.

**Parameters**

| | |
|---|---|
| *PlTerm&* | plt : The PlTerm is a bool list term, which represents an extension. |

**Returns**

no return.

Definition at line 725 of file PlReasoner.hpp.

#### 5.1.2.31 void argumatrix::Reasoner::printLabSet ( const bitvector & *bv_ext* ) [inherited]

Print an extension with the form of bitvector. Assume the bitvector is [0, 1, 0, 1], the arguments corresponding to entry 1 will be print. The member *m_output* will determine where to print.

**Parameters**

| | |
|---|---|
| *bitvector&* | bv_ext : The bv_ext is a bitvector, which represents an extension. |

**Returns**

no return.

**See also**

[setOutput]

Definition at line 433 of file Reasoner.hpp.

Referenced by argumatrix::Reasoner::printGroundedExt().

#### 5.1.2.32 void argumatrix::Reasoner::printLabSet ( const std::set< string > & *labset* ) [inherited]

Print a set of arguments. The member *m_output* will determine where to print.

**Parameters**

| | |
|---|---|
| *const* | std::set<string>& labset |

**Returns**

no return.

**See also**

setOutput(streambuf∗ strbuf = std::cout.rdbuf());

Definition at line 454 of file Reasoner.hpp.

#### 5.1.2.33 __inline void argumatrix::PlReasoner::printSomeExt ( const std::string & *predct,* const std::set< string > & *argset* ) [inherited]

Given an ${AF}=< {X}, {R}>$ and an argument $s {X}$ (respectively, a set of arguments $S {X}$), enumerate some set $E {X}$ such that $E {E}_(AF)$ and $s E$ (respectively, $S E$).

Problem [SC-$$]

**Parameters**

| | |
|---|---|
| *a* | set of arguments. |

**Returns**

no return.

Definition at line 468 of file PlReasoner.hpp.

Referenced by task_SC(), argumatrix::StablePlReasoner::task_SC(), argumatrix::CompletePlReasoner::task_SC(), argumatrix::StablePlReasoner::task_SE(), and argumatrix::CompletePlReasoner::task_SE().

#### 5.1.2.34 __inline streambuf ∗ argumatrix::Reasoner::setOutput ( streambuf ∗ *strbuf* ) [inherited]

Redirect the output stream to streambuf∗ strbuf or ostream& os. If strbuf = cout.rdbuf(), then the output is standard output. It can also redirect the output to a file by the following codes:

```
ofstream ofile("output.txt");
streambuf* oldsb = xxx.setOutput(ofile.rdbuf());
```

or

```
streambuf* oldsb = xxx.setOutput(ofile);
```

.

**Parameters**

| | |
|---|---|
| *strbuf* | is a streambuf pointer. |

**Returns**

return the old streambuf, which can be used to redirect the old streambuf.

Definition at line 473 of file Reasoner.hpp.

#### 5.1.2.35 __inline streambuf ∗ argumatrix::Reasoner::setOutput ( ostream & *os* = std::cout ) [inherited]

Method: setOutput FullName: public argumatrix::Reasoner::setOutput

**See also**

streambuf∗ setOutput(streambuf∗ strbuf = std::cout.rdbuf());

**Parameters**

| | |
|---|---|
| *ostream* | & os |

**Returns**

streambuf∗

Definition at line 479 of file Reasoner.hpp.

#### 5.1.2.36 __inline void argumatrix::AdmissiblePlReasoner::task_DC ( const std::set< string > & *argset* ) [virtual]

Given an $AF = <X,R>$ and an argument s X (respectively, a set of arguments $S X$). Decide whether s contained (respectively, S included) in some E E_(AF) (i.e., credulously justified).

Problem [DC-$$]

Reimplemented from argumatrix::Reasoner.

Definition at line 228 of file AdmissiblePlReasoner.hpp.

#### 5.1.2.37 __inline void argumatrix::AdmissiblePlReasoner::task_DE ( const std::set< string > & *argset* ) [virtual]

Given an $AF = <X,R>$ and a set of arguments $S X$. Decide whether S is a -extension of AF, i.e., S E_(AF).

Problem [DE-$$]

**Parameters**

| | |
|---|---|
| *a* | set of arguments. |

**Returns**

true if argset is a -extension; otherwise return false.

Reimplemented from argumatrix::Reasoner.

Definition at line 183 of file AdmissiblePlReasoner.hpp.

#### 5.1.2.38 void argumatrix::AdmissiblePlReasoner::task_DN ( ) [virtual]

Given an $AF = <X,R>$ and a set of arguments $S X$. Decide whether there exist a non-empty -extension for AF.

Problem [DN-$$]

**Returns**

true if there exist a non-empty -extension; otherwise return false.

Reimplemented from argumatrix::Reasoner.

Definition at line 199 of file AdmissiblePlReasoner.hpp.

#### 5.1.2.39 __inline void argumatrix::AdmissiblePlReasoner::task_DS ( const std::set< string > & *argset* ) [virtual]

Given an $AF = <X,R>$ and an argument s X (respectively, a set of arguments $S X$). Decide whether s contained (respectively, S included) in each E E_(AF) (i.e., skeptically justified).

Problem [DS-$$]

Reimplemented from argumatrix::Reasoner.

Definition at line 243 of file AdmissiblePlReasoner.hpp.

#### 5.1.2.40 __inline void argumatrix::AdmissiblePlReasoner::task_EC ( const std::set< string > & *argset* ) [virtual]

Given an ${AF}=< {X}, {R}>$ and an argument $s {X}$ (respectively, a set of arguments $S {X}$), enumerate all sets $E {X}$ such that $E {E}_(AF)$ and $s E$ (respectively, $S E$).

Problem [EC-$$]

**Parameters**

| | |
|---|---|
| *set<string>,or* | a Boolean vector: a set of argument |

**Returns**

no return.

Reimplemented from argumatrix::Reasoner.

Definition at line 166 of file AdmissiblePIReasoner.hpp.

#### 5.1.2.41 __inline void argumatrix::AdmissiblePIReasoner::task_EE ( ) `[virtual]`

Print all extensions (with a vector of integers {0,1,2})

Problem [EE-$$]

**Parameters**

| | |
|---|---|
| *no* | argument |

**Returns**

no return.

Reimplemented from argumatrix::Reasoner.

Definition at line 160 of file AdmissiblePIReasoner.hpp.

#### 5.1.2.42 __inline void argumatrix::AdmissiblePIReasoner::task_SC ( const std::set< string > & *argset* ) `[virtual]`

Given an ${AF}=< {X}, {R}>$ and an argument $s {X}$ (respectively, a set of arguments $S {X}$), enumerate some set $E {X}$ such that $E {E}_(AF)$ and $s E$ (respectively, $S E$).

Problem [SC-$$]

**Parameters**

| | |
|---|---|
| *a* | set of arguments. |

**Returns**

no return.

Reimplemented from argumatrix::Reasoner.

Definition at line 209 of file AdmissiblePIReasoner.hpp.

#### 5.1.2.43 __inline void argumatrix::AdmissiblePIReasoner::task_SE ( ) [virtual]

Given an ${AF}=< {X}, {R}>$, enumerate some set $E {X}$ that are in ${E}_(AF)$.

Problem [SE-$$]

**Parameters**

| | |
|---|---|
| *a* | set of arguments. |

**Returns**

no return.

Reimplemented from argumatrix::Reasoner.

Definition at line 220 of file AdmissiblePIReasoner.hpp.

#### 5.1.2.44 bool argumatrix::PIReasoner::verifyExclusion ( const std::string & *predct*, const bitvector & *vecB* ) [protected], [inherited]

Given an $AF = <X,R>$ and a set of arguments $S X$. Decide whether there exist a -extension that excludes S.

**Parameters**

| | |
|---|---|
| *a* | set of arguments. |

**Returns**

true if there exist a -extension that excludes S; otherwise return false.

Definition at line 576 of file PIReasoner.hpp.

#### 5.1.2.45 __inline bool argumatrix::PIReasoner::verifyExtension ( const std::string & *predct*, std::set< string > & *argset* ) [inherited]

Given an $AF = <X,R>$ and a set of arguments $S X$. Decide whether S is a $$-extension of AF, i.e., $S E_(AF)$.

Problem [DE-$$]

**Parameters**

| | |
|---|---|
| *a* | set of arguments. |

**Returns**

true if argset is a -extension; otherwise return false.

Definition at line 609 of file PIReasoner.hpp.

#### 5.1.2.46 bool argumatrix::PIReasoner::verifyInclusion ( const std::string & *predct*, const vector< int > & *vecii* ) [protected], [inherited]

Given an $AF = <X,R>$ and a set of arguments $S X$. Decide whether there exist a -extension that includes S.

**Parameters**

| | |
|---|---|
| *a* | set of arguments. |

**Returns**

true if there exist a -extension that includes S; otherwise return false.

Definition at line 543 of file PlReasoner.hpp.

#### 5.1.2.47 bool argumatrix::PlReasoner::verifyNonemptyExt ( const std::string & *predct* ) `[inherited]`

Given an $AF = <X,R>$ and a set of arguments $S X$. Decide whether there exist a non-empty $$-extension for AF.

Problem [DN-$$]

**Returns**

true if there exist a non-empty -extension; otherwise return false.

Definition at line 510 of file PlReasoner.hpp.

### 5.1.3 Member Data Documentation

#### 5.1.3.1 vector< std::string > argumatrix::Reasoner::m_argLabels `[protected]`, `[inherited]`

Dung's abstract argumentation framework

Definition at line 346 of file Reasoner.hpp.

Referenced by argumatrix::Reasoner::printLabSet().

#### 5.1.3.2 size_type argumatrix::Reasoner::m_argNum `[protected]`, `[inherited]`

The number of arguments

Definition at line 339 of file Reasoner.hpp.

Referenced by argumatrix::Reasoner::characteristic(), argumatrix::Reasoner::getGroundedExtension(), argumatrix↩ ::Reasoner::getGroundedIntVector(), and argumatrix::Reasoner::labelSet2IntVector().

#### 5.1.3.3 bitmatrix argumatrix::Reasoner::m_BmAtkMtx `[protected]`, `[inherited]`

The bitmatrix of the Dung Abstract argumentation framework. We can access all attackers of an argument. The attackers of the argument with index i is m_BmAtkMtx[i].

Definition at line 334 of file Reasoner.hpp.

Referenced by argumatrix::Reasoner::getAttacked(), argumatrix::Reasoner::getSelfAttackingArguments(), and argumatrix::Reasoner::is_self_attacking().

#### 5.1.3.4 const DungAF& argumatrix::Reasoner::m_daf `[protected]`, `[inherited]`

Dung's abstract argumentation framework

Definition at line 342 of file Reasoner.hpp.

Referenced by argumatrix::Reasoner::is_conflict_free(), argumatrix::Reasoner::labelSet2IntVector(), and argumatrix::Reasoner::printBvExts().

#### 5.1.3.5 std::ostream argumatrix::Reasoner::m_output `[protected]`, `[inherited]`

Where to output

Definition at line 348 of file Reasoner.hpp.

Referenced by argumatrix::Reasoner::printBvExts(), argumatrix::Reasoner::printGroundedExt(), argumatrix::↩
Reasoner::printLabSet(), and argumatrix::Reasoner::setOutput().

The documentation for this class was generated from the following file:

- F:/Codespace/BoostProject/argumatrix/PlReasoner/AdmissiblePlReasoner.hpp

## 5.2 argumatrix::ArgumatrixPlEngine Class Reference

We rewrite the PlEngine since we do not want the PlEngine to print the welcome and version information. These information can not be printed if we use command "-g true" or "--quiet" or "-q". Note that both PlEngine and ArgumatrixPlEngine must be defined as a static instance, since multiple engines exist may lead to crash when one of the engins is closed (i.e., invoking PL_cleanup) then all other running engines are closed. Define PlEngine as a static instance can solve this problem since static instances are global and stored in static variable area. They are initialized before the main function and released when the program exits.

```
#include <swipl-util.hpp>
```

#### Public Member Functions

- ArgumatrixPlEngine (int argc, char $**$argv)

  *Create the Prolog engine.*
- ArgumatrixPlEngine (char $*$program=(char $*$)"ArgumatrixPlEngine")

  *Initialize the Prolog engine with some default settings.*

### 5.2.1 Detailed Description

We rewrite the PlEngine since we do not want the PlEngine to print the welcome and version information. These information can not be printed if we use command "-g true" or "--quiet" or "-q". Note that both PlEngine and ArgumatrixPlEngine must be defined as a static instance, since multiple engines exist may lead to crash when one of the engins is closed (i.e., invoking PL_cleanup) then all other running engines are closed. Define PlEngine as a static instance can solve this problem since static instances are global and stored in static variable area. They are initialized before the main function and released when the program exits.

class: ArgumatrixPlEngine

#### See also

The definition of ArgumatrixPlEngine is at the beginning of [PlReasoner.hpp].

Definition at line 53 of file swipl-util.hpp.

### 5.2.2 Constructor & Destructor Documentation

#### 5.2.2.1 argumatrix::ArgumatrixPlEngine::ArgumatrixPlEngine ( int *argc*, char ** *argv* ) `[inline]`

Create the Prolog engine.

Method: ArgumatrixPlEngine FullName: public argumatrix::ArgumatrixPlEngine::ArgumatrixPlEngine

**Parameters**

| | |
|---|---|
| *plArgc* | the number of arguments |
| *plArgv* | a list of options. Options: <br>• -x state Start from state (must be first)<br>• -[LGT]size[KMG] Specify {Local,Global,Trail} limits<br>• -t toplevel Toplevel goal<br>• -g goal Initialisation goal<br>• -f file User initialisation file<br>• -F file System initialisation file<br>• -l file Script source file<br>• -s file Script source file<br>• -p alias=path Define file search path 'alias'<br>• [+/-]tty Allow tty control<br>• -O Optimised compilation<br>• –nosignals Do not modify any signal handling<br>• –nodebug Omit generation of debug info<br>• –quiet Quiet operation (also -q)<br>• –traditional Disable extensions of version 7<br>• –home=DIR Use DIR as SWI-Prolog home<br>• –pldoc[=port] Start PlDoc server [at port]<br>• –win_app Behave as Windows application |

**Note**

We implement it by a cross-platform and robust solution, which can handle the ERROR: **[FATAL ERROR: Could not find system resources]** when invokes the function *PL_initialise*, or when Prolog is embedded in a C/C++/Java/... application.

Definition at line 86 of file swipl-util.hpp.

#### 5.2.2.2 argumatrix::ArgumatrixPlEngine::ArgumatrixPlEngine ( char ∗ *program* = `(char*) "ArgumatrixPlEngine"` ) `[inline]`

Initialize the Prolog engine with some default settings.

Method: Constructor: ArgumatrixPlEngine

**Parameters**

| | |
|---|---|
| *char* | ∗ program |

#### Returns

Two variables used to initialize the SWI-Prolog Engine.

#### Parameters

| | |
|---|---|
| *plArgc* | the number of arguments |
| *plArgv* | a list of arguments Options: -x state Start from state (must be first) -[LGT]size[KMG] Specify {Local,Global,Trail} limits -t toplevel Toplevel goal -g goal Initialisation goal -f file User initialisation file -F file System initialisation file -l file Script source file -s file Script source file -p alias=path Define file search path 'alias' [+/-]tty Allow tty control -O Optimised compilation –nosignals Do not modify any signal handling –nodebug Omit generation of debug info –quiet Quiet operation (also -q) –traditional Disable extensions of version 7 –home=DIR Use DIR as SWI-Prolog home –pldoc[=port] Start PlDoc server [at port] –win_app Behave as Windows application |

Definition at line 119 of file swipl-util.hpp.

The documentation for this class was generated from the following file:

- F:/Codespace/BoostProject/argumatrix/PlReasoner/swipl-util.hpp

## 5.3 argumatrix::ArgumentProperty Struct Reference

#### Public Member Functions

- **ArgumentProperty** (string _uniq_label, string _title="", string _description="")
- **ArgumentProperty** (const ArgumentProperty &_ap)
- std::string **getTitle** () const
- void **setTitle** (std::string val)

#### Public Attributes

- std::string **label**
- std::string **title**
- std::string **description**

### 5.3.1 Detailed Description

Definition at line 18 of file ArgumentProperty.hpp.

The documentation for this struct was generated from the following file:

- F:/Codespace/BoostProject/argumatrix/dung_theory/ArgumentProperty.hpp

## 5.4 argumatrix::AttackProperty Class Reference

#### Public Member Functions

- **AttackProperty** (string _l="no_label", weight_type _w=0.0)
- std::string **getLabel** () const
- void **setLabel** (std::string val)
- weight_type **getWeight** () const
- void **setWeight** (weight_type val)

### 5.4.1 Detailed Description

Definition at line 17 of file AttackProperty.hpp.

The documentation for this class was generated from the following file:

- F:/Codespace/BoostProject/argumatrix/dung_theory/AttackProperty.hpp

## 5.5 argumatrix::bitmatrix Class Reference

#### Public Member Functions

- **bitmatrix** (size_type num_size)
- **bitmatrix** (size_type num_rows, size_type num_columns)
- **bitmatrix** (const bitmatrix &_bm)
- void **show** ()
- bitvector & **operator[ ]** (size_type pos)
- const bitvector & **operator[ ]** (size_type pos) const
- bitvector **operator∗** (const bitvector &_bv)
- bitmatrix **operator∗** (const bitmatrix &_bm)
- bitmatrix **operator=** (const bitmatrix &_bm)
- void **setBitvector** (const bitvector &_bv, size_type pos)
- bitvector **getBitvector** (size_type pos)
- bitmatrix **transpose** () const
- size_type **sizeR** () const
- size_type **sizeC** () const
- void **push_back** (bitvector &_bv)
- bitvector diag ()

### 5.5.1 Detailed Description

Definition at line 20 of file bitmatrix.hpp.

### 5.5.2 Member Function Documentation

#### 5.5.2.1 bitvector argumatrix::bitmatrix::diag ( )

Get the diagonal elements of a bitmatrix

**Returns**

the bitvector of the diagonal elements

Definition at line 179 of file bitmatrix.hpp.

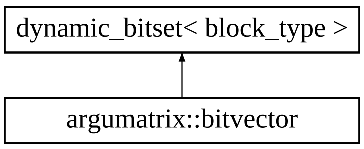
Referenced by argumatrix::Reasoner::getSelfAttackingArguments().

The documentation for this class was generated from the following file:

- F:/Codespace/BoostProject/argumatrix/bitmatrix/bitmatrix.hpp

## 5.6 argumatrix::bitvector Class Reference

Inheritance diagram for argumatrix::bitvector:



**Public Member Functions**

- **bitvector** (size_type _sz, unsigned long value=0)
- **bitvector** (const std::string &_s)
- **bitvector** (const dynamic_bitset< block_type > &_db)
- bool operator∗ (const bitvector &_bv)
- bool is_emptyset ()

  *To decide whether the bitvector of an argument set is empty-set? If all entries of this bitvector are 0's, then it is an empty set, and this function return true, else return false. we must discriminate this function from the function empty(), which is used to determine the size of the bitvector is zero. This function is the same as the function none().*
- bool is_universal ()

  *To decide whether the bitvector of an argument set is universal? If all entries of this bitvector are 1's, then it is an universal set, and this function return true, else return false.*
- bool Increase ()

**Static Public Member Functions**

- static bitvector EmptySet (size_type _sz)

  *Create an empty set (all 0's) or a universal set (all 1's) under a given size.*
- static bitvector **UniversalSet** (size_type _sz)

### 5.6.1 Detailed Description

Definition at line 60 of file bitvector.hpp.

### 5.6.2 Member Function Documentation

#### 5.6.2.1 __inline argumatrix::bitvector argumatrix::bitvector::EmptySet ( size_type *_sz* ) [static]

Create an empty set (all 0's) or a universal set (all 1's) under a given size.

**Parameters**

| | |
|---|---|
| *_sz* | the size of the empty set or the universal set |

**Returns**

the bitvector of the empty set or the universal set.

Definition at line 126 of file bitvector.hpp.

Referenced by argumatrix::Reasoner::characteristic(), argumatrix::PreferredReasoner::computeExtensions(), argumatrix::Reasoner::getGroundedExtension(), and argumatrix::Reasoner::getGroundedIntVector().

#### 5.6.2.2 bool argumatrix::bitvector::Increase ( )

increaser the bit vector with 1.

**Parameters**

| | |
|---|---|
| *_sz* | the size of the empty set or the universal set |

**Returns**

the bitvector of the empty set or the universal set.

Definition at line 141 of file bitvector.hpp.

#### 5.6.2.3 bool argumatrix::bitvector::is_emptyset ( ) `[inline]`

To decide whether the bitvector of an argument set is empty-set? If all entries of this bitvector are 0's, then it is an empty set, and this function return true, else return false. we must discriminate this function from the function empty(), which is used to determine the size of the bitvector is zero. This function is the same as the function none().

**Returns**

true if all entries of the bitvector are 0's, otherwise return false.

Definition at line 120 of file bitvector.hpp.

Referenced by argumatrix::PreferredReasoner::computeExtensions(), and argumatrix::GroundedReasoner::task←
_DN().

#### 5.6.2.4 __inline bool argumatrix::bitvector::operator∗ ( const bitvector & *_bv* )

The multiplication of two bitvector. Assume A=[a_1, a_2, ..., a_n] and B=[b_1, b_2, ..., b_n], the multiplication of A∗B = (a_1∗b_1)+(a_2∗b_2)+...+(a_n∗b_n). This operation is equivalent in function to determine whether A intersects with B.

**Parameters**

| | |
|---|---|
| *two* | bitvector a and b |

**Returns**

the bool value of A∗B.

Definition at line 114 of file bitvector.hpp.

The documentation for this class was generated from the following file:

- F:/Codespace/BoostProject/argumatrix/bitmatrix/bitvector.hpp

## 5.7 argumatrix::ClpbProblem Struct Reference

### Public Member Functions

- void **parseParams** (int argc, char ∗argv[ ])
- void **solve** ()
- void **loadDungAF** ()
- void **reasoning** (Reasoner ∗rsner, const string &task)
- void parseAddtionalParams (const string &oargs)

  *interpret additional arguments into argument set. The format of input is arg1[,arg2,arg3,...], e.g., a1,a2. The interpretted results are stored in the member variable set<string> otherArgs.*

### Public Attributes

- string **m_problemType**
- string **m_fileFmt**
- string **m_inputFile**
- string **m_outputFile**
- set< string > **m_otherArgs**
- bool **m_timeFlag**
- DungAF **daf**

### Friends

- ostream & **operator**<< (ostream &out, ClpbProblem &problem)

### 5.7.1 Detailed Description

Definition at line 57 of file ClpbProblem.hpp.

### 5.7.2 Member Function Documentation

#### 5.7.2.1 void argumatrix::ClpbProblem::parseAddtionalParams ( const string & *oargs* )

interpret additional arguments into argument set. The format of input is arg1[,arg2,arg3,...], e.g., a1,a2. The interpretted results are stored in the member variable *set<string> otherArgs*.

Method: parseAddtionalParams FullName: public argumatrix::ClpbProblem::parseAddtionalParams

**Parameters**

| | |
|---|---|
| *const* | string & oargs |

**See also**

Definition at line 286 of file ClpbProblem.hpp.

The documentation for this struct was generated from the following file:

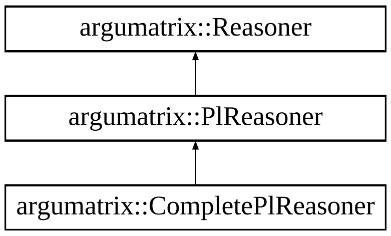
- F:/Codespace/BoostProject/argumatrix/argmat-clpb/ClpbProblem.hpp

## 5.8 argumatrix::CompletePlReasoner Class Reference

Tackle computational problems for complete semantics. TODO: long description.

```
#include <CompletePlReasoner.hpp>
```

Inheritance diagram for argumatrix::CompletePlReasoner:



**Public Member Functions**

- **CompletePlReasoner** (const DungAF &daf, const string &sm_task="CO", streambuf ∗osbuff=std::cout.↩ rdbuf())
- void task_EE ()

  *Print all extensions (with a vector of integers {0,1,2})*
- void **task_EX** ()
- void task_EC (const std::set< string > &argset)

  *Given an ${AF}=< {X}, {R}>$ and an argument $s {X}$ (respectively, a set of arguments $S {X}$), enumerate all sets $E {X}$ such that $E {E}_(AF)$ and $s E$ (respectively, $S E$).*
- void task_SC (const std::set< string > &argset)

  *Given an ${AF}=< {X}, {R}>$ and an argument $s {X}$ (respectively, a set of arguments $S {X}$), enumerate some set $E {X}$ such that $E {E}_(AF)$ and $s E$ (respectively, $S E$).*
- void task_SE ()

  *Given an ${AF}=< {X}, {R}>$, enumerate some set $E {X}$ that are in ${E}_(AF)$.*
- void findAllExts ()
- void task_DE (const std::set< string > &argset)

  *Given an $AF = <X,R>$ and a set of arguments $S X$. Decide whether S is a -extension of AF, i.e., S E_(AF).*
- void task_DN ()

  *Given an $AF = <X,R>$ and a set of arguments $S X$. Decide whether there exist a non-empty -extension for AF.*

- void task_DC (const std::set< string > &argset)

  *Given an $AF = <X,R>$ and an argument s X (respectively, a set of arguments $S X$). Decide whether s contained (respectively, S included) in some E E_(AF) (i.e., credulously justified).*
- void task_DS (const std::set< string > &argset)

  *Given an $AF = <X,R>$ and an argument s X (respectively, a set of arguments $S X$). Decide whether s contained (respectively, S included) in each E E_(AF) (i.e., skeptically justified).*
- void consultPlFile (std::string plFile=ARG_PROLOG_FILE)

  *Loading the Prolog source file [argmat-clpb].*
- void **Test_time** ()
- void printAllExts (const std::string &predct)

  *Print all extensions with respect to semantic predct.*
- void printAllExts (const std::string &predct, const std::set< string > &argset)

  *Given an ${AF}=< {X}, {R}>$ and an argument $s {X}$ (respectively, a set of arguments $S {X}$), enumerate all sets $E {X}$ such that $E {E}_(AF)$ and $s E$ (respectively, $S E$).*
- void **printAllExts** (const std::string &predct, const std::vector< int > &vecii)
- void **printAllExts2** (const std::string &predct)
- void fetchAllExts (const std::string &predct)
- void printSomeExt (const std::string &predct, const std::set< string > &argset)

  *Given an ${AF}=< {X}, {R}>$ and an argument $s {X}$ (respectively, a set of arguments $S {X}$), enumerate some set $E {X}$ such that $E {E}_(AF)$ and $s E$ (respectively, $S E$).*
- void **printSomeExt** (const std::string &predct, const vector< int > &vecii)
- bool verifyNonemptyExt (const std::string &predct)

  *Given an $AF = <X,R>$ and a set of arguments $S X$. Decide whether there exist a non-empty $$-extension for AF.*
- bool verifyExtension (const std::string &predct, std::set< string > &argset)

  *Given an $AF = <X,R>$ and a set of arguments $S X$. Decide whether S is a $$-extension of AF, i.e., $S E_(AF)$.*
- bool **verifyExtension** (const std::string &predct, bitvector &bvecii)
- bool isCredulouslyJustified (const std::string &predct, const std::set< string > &argset)

  *Given an $AF = <X,R>$ and an argument s X (respectively, a set of arguments $S X$). Decide whether s contained (respectively, S included) in some E E_(AF) (i.e., credulously justified).*
- bool isSkepticallyJustified (const std::string &predct, const std::set< string > &argset)

  *Given an $AF = <X,R>$ and an argument s X (respectively, a set of arguments $S X$). Decide whether s contained (respectively, S included) in each E E_(AF) (i.e., skeptically justified).*
- bitvector getAttacked (const bitvector &_bv)

  *Get the attacked arguments by arguments in _bv, $R^+(S)$ $R^+(S) = x | x is attacked by S$. $R^+(S_bv) = D * S_bv$.*
- bitvector characteristic (const bitvector &_bv)

  *The characteristic function of an abstract argumentation framework: $F_{AF}(S) = A | A is acceptable wrt. S$. F_AF(↩ S_bv) = not{ R^+[ not( R^+(S_bv) ) ] }.*
- bitvector characteristic ()

  *The characteristic function of an abstract argumentation framework with the initialization of empty set.*
- bitvector neutrality (const bitvector &_bv)

  *The neutrality function of an abstract argumentation framework: N_AF(S) = {A| All arguments that not attacked by S}. N_AF(S_bv) = not( R^+(S_bv) )*
- bool is_conflict_free (const bitvector &_bv)

  *Argument set S is conflict-free? S is said to be conflict-free iff for any two arguments a,b in S such that a does not attack b.*
- bool **is_conflict_free** (const set< string > &argset)
- bool is_acceptable (const bitvector &S, const bitvector &A)

  *Argument set A is acceptable w.r.t S? Alternately, S defends A? A can be an argument or an argument set. S defends argument (set) A iff for any argument x in X, if attacks A (or a in A) then there is an argument y in S such that y attacks x.*
- bool is_self_attacking (size_type idx)

  *Argument a is self-attacking iff it attacks itself.*
- bool is_admissible (const bitvector &_bv)

*To decide whether a set of arguments (with the form of bitvector) is an admissible extension. $S$ is an admissible extension iff $S F(S) N(S)$.*

- bool is_complete (const bitvector &_bv)

  *To decide whether a set of arguments (with the form of bitvector) is a complete extension. $S$ is a complete extension iff $S == F(S) N(S)$.*

- bool is_stable (const bitvector &_bv)

  *To decide whether a set of arguments (with the form of bitvector) is a stable extension. $S$ is a stable extension iff $S == N(S)$.*

- bool is_grounded (const bitvector &_bv)

  *To decide whether a set of arguments (with the form of bitvector) is a grounded extension. $S$ is a grounded extension iff it is the least fixed point of the characteristic function F.*

- bitvector getSelfAttackingArguments ()

  *Get all arguments which are self-attacking. Obviously, if an argument i attacks itself, the entry[i][i] of the attack matrix must be 1 (true). Therefore to get the set of self-attacking arguments is to get the diagonal elements of the attack matrix.*

- const set< bitvector > & getBvExtensions ()

  *Get all extensions given a specific semantics in format of bitvector. Each extension is a set of arguments which can "survive the conflict together". Here, we use a bitvector to represent an extension. Therefore, all extensions are formed an set of bit vectors, i.e., set<bitvector>. Each bitvector in set is an extension w.r.t. a given semantics. The computed extensions are stored in m_extensions, therefore, to get all extensions, it must first invoke the function computeExtension()*

- vector< int > getGroundedIntVector ()

  *Get a vector of integers {0, 1, 2}: 2 – Unknown, 1 – in grounded extension, and 0 – attacked by the grounded extension.*

- streambuf * setOutput (streambuf *strbuf)

  *Redirect the output stream to streambuf* strbuf or ostream& os. If strbuf = cout.rdbuf(), then the output is standard output. It can also redirect the output to a file by the following codes:*

  ```
  ofstream ofile("output.txt");
  streambuf* oldsb = xxx.setOutput(ofile.rdbuf());
  ```
  *or*
  ```
  streambuf* oldsb = xxx.setOutput(ofile);
  ```
  .

- streambuf * setOutput (ostream &os=std::cout)
- void printLabSet (const bitvector &bv_ext)

  *Print an extension with the form of bitvector. Assume the bitvector is [0, 1, 0, 1], the arguments corresponding to entry 1 will be print. The member m_output will determine where to print.*

- void printLabSet (const std::set< string > &labset)

  *Print a set of arguments. The member m_output will determine where to print.*

- void printBvExts ()

  *Output all extensions given a specific semantics with an ostream. Each extension is a set of arguments which can "survive the conflict together". Here, we return a string to represent all extensions. For example, if there are two extensions, [a,b] and [b, d], for some problem w.r.t. a given semantics, then we return string "[[a,b],[d,c]]". If extensions is not existing, the string "[ ]" will return.*

- bitvector getGroundedExtension ()

  *Get the grounded extension.*

- void printGroundedExt ()

  *Print the grounded extension.*

- vector< int > labelSet2IntVector (const std::set< string > &label_set)

  *Convert a set of argument (string) labels to an integer vector of {0,1,2}, All arguments in these set, the indices is 1; otherwise 2 (representing unknown)*

## Protected Member Functions

- void findAllExts (const std::string &predct)

  *Find all extensions and store them in m_extensions with the form of bitvector.*

- void createPlAttackMatrix ()

  *Create the attack matrix with the form of PlTerm, which is an input of SWI-Prolog.*
- void printLableExtByBlistTerm (const PlTerm &plt)

  *Print an extension with the form of the bool list term. Assume a bool list term is [0, 1, 0, 1], the arguments corresponding to entry 1 will be print. The member m_output will determine where to print.*
- void printLableExtByBmatrixTerm (const PlTerm &pmtx)

  *Print an extension with the form of the bool matrix term. Assume a bool list term is [[0, 1, 0, 1], [0, 1, 0, 1]], the arguments corresponding to entry 1 will be print. The member m_output will determine where to print.*
- void addBlTermToBvExts (const PlTerm &plt)

  *Convert a bool list term into a bitvertor, and add it to extension.*
- bool verifyInclusion (const std::string &predct, const vector< int > &vecii)

  *Given an $AF = <X,R>$ and a set of arguments $S X$. Decide whether there exist a -extension that includes S.*
- bool verifyExclusion (const std::string &predct, const bitvector &vecB)

  *Given an $AF = <X,R>$ and a set of arguments $S X$. Decide whether there exist a -extension that excludes S.*

#### Protected Attributes

- string **m_predicate**
- PlTerm **m_PlAtkMtx**
- bitmatrix m_BmAtkMtx
- size_type m_argNum
- const DungAF & m_daf
- set< bitvector > **m_extensions**
- vector< std::string > m_argLabels
- std::ostream m_output

### 5.8.1 Detailed Description

Tackle computational problems for complete semantics. TODO: long description.

**See also**

[dung_theory/Reasoner.hpp]

**Author**

Fuan Pu

**Version**

1.0

**Date**

May 2016

Contact: `Pu.Fuan@gmail.com`

Definition at line 51 of file CompletePlReasoner.hpp.

### 5.8.2 Member Function Documentation

#### 5.8.2.1 void argumatrix::PlReasoner::addBlTermToBvExts ( const PlTerm & *plt* ) `[protected]`, `[inherited]`

Convert a bool list term into a bitvertor, and add it to extension.

**Parameters**

| | |
|---|---|
| *PlTerm&* | plt : The PlTerm is a bool list term, which represents an extension. |

**Returns**

no return. The results are added into m_extensions.

Definition at line 303 of file PlReasoner.hpp.

#### 5.8.2.2 __inline **argumatrix::bitvector** argumatrix::Reasoner::characteristic ( const **bitvector** & *_bv* ) [inherited]

The characteristic function of an abstract argumentation framework: $F_{AF}(S) = A|Aisacceptablewrt.S$. F_AF(↩ S_bv) = not{ R^+[ not( R^+(S_bv) ) ] }.

**Parameters**

| | |
|---|---|
| *extension* | an extension (a set of arguments), the default is empty set. |

**Returns**

a set of arguments in bitvector form.

Definition at line 368 of file Reasoner.hpp.

#### 5.8.2.3 __inline **argumatrix::bitvector** argumatrix::Reasoner::characteristic ( ) [inherited]

The characteristic function of an abstract argumentation framework with the initialization of empty set.

**Returns**

a set of arguments in bitvector form.

Definition at line 375 of file Reasoner.hpp.

Referenced by argumatrix::Reasoner::getGroundedExtension(), and argumatrix::Reasoner::is_acceptable().

#### 5.8.2.4 void argumatrix::PlReasoner::consultPlFile ( std::string *plFile* = ARG_PROLOG_FILE ) [inherited]

Loading the Prolog source file [argmat-clpb].

Method: consultPlFile

**Parameters**

| | |
|---|---|
| *std::string* | plFile |

**Note**

SWI-Prolog supports compilation of individual or multiple Prolog source files into 'Quick Load Files'. A 'Quick Load File' (.qlf file) stores the contents of the file in a precompiled format. These files load considerably faster than source files and are normally more compact. They are machine-independent and may thus be loaded on any implementation of SWI-Prolog. Note, however, that clauses are stored as virtual machine instructions. Changes to the compiler will generally make old compiled files unusable. Quick Load Files are created using qcompile/1. They are loaded using consult/1 or one of the other file-loading predicates described in section 4.3. If consult/1 is given an explicit .pl file, it will load the Prolog source. When given a .qlf file, it will load the file. When no extension is specified, it will load the .qlf file when present and the .pl file otherwise.

Definition at line 253 of file PlReasoner.hpp.

#### 5.8.2.5 void argumatrix::PlReasoner::createPlAttackMatrix ( ) `[protected]`, `[inherited]`

Create the attack matrix with the form of PlTerm, which is an input of SWI-Prolog.

**Returns**

no return. The result is stored in member variable *m_PlAtkMtx*.

Definition at line 243 of file PlReasoner.hpp.

#### 5.8.2.6 void argumatrix::PlReasoner::fetchAllExts ( const std::string & *predct* ) `[inherited]`

Method: fetchAllExts FullName: public argumatrix::PlReasoner::fetchAllExts

**Parameters**

| | |
|---|---|
| *const* | std::string & predct |

**Returns**

void

**Return values**

| | |
|---|---|
| | |

Definition at line 701 of file PlReasoner.hpp.

#### 5.8.2.7 __inline void argumatrix::CompletePlReasoner::findAllExts ( )

Find all extensions and store them in *m_extensions* with the form of bitvector.

**Parameters**

| | |
|---|---|
| *no* | argument. |

**Returns**

no return. The results are added into *m_extensions*.

Definition at line 193 of file CompletePlReasoner.hpp.

### 5.8.2.8 void argumatrix::PlReasoner::findAllExts ( const std::string & *predct* ) [protected], [inherited]

Find all extensions and store them in *m_extensions* with the form of bitvector.

**Parameters**

| | |
|---|---|
| *string&* | predct: The predicate. |

**Returns**

no return. The results are added into *m_extensions*.

Definition at line 669 of file PlReasoner.hpp.

Referenced by argumatrix::AdmissiblePlReasoner::findAllExts(), argumatrix::StablePlReasoner::findAllExts(), argumatrix::ConflictfreePlReasoner::findAllExts(), and findAllExts().

### 5.8.2.9 __inline argumatrix::bitvector argumatrix::Reasoner::getAttacked ( const bitvector & *_bv* ) [inherited]

Get the attacked arguments by arguments in _bv, $R^+(S)$ $R^+(S) = x|xisattackedbyS$. $R^+(S_bv) = D * S_bv$.

**Parameters**

| | |
|---|---|
| *_bv* | the bitvector with respect to the set $S$. |

**Returns**

a set of arguments in bitvector form.

Definition at line 360 of file Reasoner.hpp.

Referenced by argumatrix::Reasoner::getGroundedIntVector(), argumatrix::Reasoner::is_conflict_free(), and argumatrix::Reasoner::neutrality().

### 5.8.2.10 __inline const set< bitvector > & argumatrix::Reasoner::getBvExtensions ( ) [inherited]

Get all extensions given a specific semantics in format of bitvector. Each extension is a set of arguments which can "survive the conflict together". Here, we use a bitvector to represent an extension. Therefore, all extensions are formed an set of bit vectors, i.e., set<bitvector>. Each bitvector in set is an extension w.r.t. a given semantics. The computed extensions are stored in m_extensions, therefore, to get all extensions, it must first invoke the function computeExtension()

**Returns**

a set of bitvector, i.e., set<bitvector>.

Definition at line 400 of file Reasoner.hpp.

#### 5.8.2.11 argumatrix::bitvector argumatrix::Reasoner::getGroundedExtension ( ) [inherited]

Get the grounded extension.

**Returns**

a set of arguments in bitvector form.

Definition at line 484 of file Reasoner.hpp.

Referenced by argumatrix::Reasoner::is_grounded(), and argumatrix::Reasoner::printGroundedExt().

#### 5.8.2.12 vector< int > argumatrix::Reasoner::getGroundedIntVector ( ) [inherited]

Get a vector of integers {0, 1, 2}: 2 – Unknown, 1 – in grounded extension, and 0 – attacked by the grounded extension.

**Returns**

a vector of integers {0, 1, 2}.

Definition at line 499 of file Reasoner.hpp.

#### 5.8.2.13 __inline argumatrix::bitvector argumatrix::Reasoner::getSelfAttackingArguments ( ) [inherited]

Get all arguments which are self-attacking. Obviously, if an argument i attacks itself, the entry[i][i] of the attack matrix must be 1 (true). Therefore to get the set of self-attacking arguments is to get the diagonal elements of the attack matrix.

**Returns**

a set of arguments in bitvector form, if bitvector[i] = true representing the argument (with index) i is a self-attacking argument, otherwise is not.

Definition at line 414 of file Reasoner.hpp.

#### 5.8.2.14 __inline bool argumatrix::Reasoner::is_acceptable ( const bitvector & *S*, const bitvector & *A* ) [inherited]

Argument set A is acceptable w.r.t S? Alternately, S defends A? A can be an argument or an argument set. S defends argument (set) A iff for any argument x in X, if attacks A (or a in A) then there is an argument y in S such that y attacks x.

**Parameters**

| | |
|---|---|
| *S* | a set of arguments. |
| *A* | an argument or an argument set |

**Returns**

true if S defends A.

Definition at line 394 of file Reasoner.hpp.

#### 5.8.2.15 __inline bool argumatrix::Reasoner::is_admissible ( const bitvector & *_bv* ) `[inherited]`

To decide whether a set of arguments (with the form of bitvector) is an admissible extension. $S$ is an admissible extension iff $S F(S) N(S)$.

**Returns**

true if _bv is admissible; otherwise returns false.

Definition at line 553 of file Reasoner.hpp.

#### 5.8.2.16 __inline bool argumatrix::Reasoner::is_complete ( const bitvector & *_bv* ) `[inherited]`

To decide whether a set of arguments (with the form of bitvector) is a complete extension. $S$ is a complete extension iff $S == F(S) N(S)$.

**Returns**

true if _bv is complete; otherwise returns false.

Definition at line 564 of file Reasoner.hpp.

#### 5.8.2.17 __inline bool argumatrix::Reasoner::is_conflict_free ( const bitvector & *_bv* ) `[inherited]`

Argument set S is conflict-free? S is said to be conflict-free iff for any two arguments a,b in S such that a does not attack b.

**Parameters**

| | |
|---|---|
| *extension* | an extension (a set of arguments). |

**Returns**

true if S is conflict-free.

Definition at line 381 of file Reasoner.hpp.

#### 5.8.2.18 __inline bool argumatrix::Reasoner::is_grounded ( const bitvector & *_bv* ) `[inherited]`

To decide whether a set of arguments (with the form of bitvector) is a grounded extension. $S$ is a grounded extension iff it is the least fixed point of the characteristic function F.

**Returns**

true if _bv is grounded; otherwise returns false.

Definition at line 584 of file Reasoner.hpp.

#### 5.8.2.19 __inline bool argumatrix::Reasoner::is_self_attacking ( size_type *idx* ) `[inherited]`

Argument a is self-attacking iff it attacks itself.

**Parameters**

|  |  |
|---|---|
| *idx* | the index of the argument w.r.t the attack matrix |

**Returns**

true if S defends A.

Definition at line 406 of file Reasoner.hpp.

#### 5.8.2.20 __inline bool argumatrix::Reasoner::is_stable ( const bitvector & *_bv* ) `[inherited]`

To decide whether a set of arguments (with the form of bitvector) is a stable extension. $S$ is a stable extension iff $S == N(S)$.

**Returns**

true if _bv is stable; otherwise returns false.

Definition at line 575 of file Reasoner.hpp.

#### 5.8.2.21 bool argumatrix::PlReasoner::isCredulouslyJustified ( const std::string & *predct,* const std::set< string > & *argset* ) `[inherited]`

Given an $AF = <X,R>$ and an argument s X (respectively, a set of arguments $S X$). Decide whether s contained (respectively, S included) in some E E_(AF) (i.e., credulously justified).

Problem [DC-$$]

Definition at line 642 of file PlReasoner.hpp.

#### 5.8.2.22 bool argumatrix::PlReasoner::isSkepticallyJustified ( const std::string & *predct,* const std::set< string > & *argset* ) `[inherited]`

Given an $AF = <X,R>$ and an argument s X (respectively, a set of arguments $S X$). Decide whether s contained (respectively, S included) in each E E_(AF) (i.e., skeptically justified).

Problem [DS-$$]

Definition at line 654 of file PlReasoner.hpp.

#### 5.8.2.23 vector< int > argumatrix::Reasoner::labelSet2IntVector ( const std::set< string > & *label_set* ) `[inherited]`

Convert a set of argument (string) labels to an integer vector of {0,1,2}, All arguments in these set, the indices is 1; otherwise 2 (representing unknown)

**Returns**

no return.

Definition at line 526 of file Reasoner.hpp.

#### 5.8.2.24 __inline argumatrix::bitvector argumatrix::Reasoner::neutrality ( const bitvector & *_bv* ) `[inherited]`

The neutrality function of an abstract argumentation framework: N_AF(S) = {A| All arguments that not attacked by S}. N_AF(S_bv) = not( R^+(S_bv) )

**Parameters**

| | |
|---|---|
| *extension* | an extension (a set of arguments). |

**Returns**

a set of arguments in bitvector form.

Definition at line 547 of file Reasoner.hpp.

Referenced by argumatrix::Reasoner::characteristic(), argumatrix::Reasoner::getGroundedIntVector(), argumatrix↩ ::Reasoner::is_admissible(), argumatrix::Reasoner::is_complete(), and argumatrix::Reasoner::is_stable().

#### 5.8.2.25 void argumatrix::PlReasoner::printAllExts ( const std::string & *predct* ) [inherited]

Print all extensions with respect to semantic predct.

Problem [EE-$$]

**Parameters**

| | |
|---|---|
| *no* | argument |

**Returns**

no return.

Definition at line 430 of file PlReasoner.hpp.

Referenced by argumatrix::PlReasoner::printAllExts(), argumatrix::StablePlReasoner::task_EC(), argumatrix↩ ::ConflictfreePlReasoner::task_EC(), argumatrix::AdmissiblePlReasoner::task_EC(), task_EC(), argumatrix↩ ::ConflictfreePlReasoner::task_EE(), argumatrix::StablePlReasoner::task_EE(), argumatrix::AdmissiblePl↩ Reasoner::task_EE(), and task_EE().

#### 5.8.2.26 __inline void argumatrix::PlReasoner::printAllExts ( const std::string & *predct,* const std::set< string > & *argset* ) [inherited]

Given an ${AF}=< {X}, {R}>$ and an argument $s {X}$ (respectively, a set of arguments $S {X}$), enumerate all sets $E {X}$ such that $E {E}_(AF)$ and $s E$ (respectively, $S E$).

Problem [EC-$$]

**Parameters**

| | |
|---|---|
| *set<string>,or* | a Boolean vector: a set of argument |

**Returns**

no return.

Definition at line 386 of file PlReasoner.hpp.

#### 5.8.2.27 __inline void argumatrix::Reasoner::printBvExts ( ) [inherited]

Output all extensions given a specific semantics with an ostream. Each extension is a set of arguments which can "survive the conflict together". Here, we return a string to represent all extensions. For example, if there are two extensions, [a,b] and [b, d], for some problem w.r.t. a given semantics, then we return string "[[a,b],[d,c]]". If extensions is not existing, the string "[ ]" will return.

**Parameters**

| | |
|---|---|
| *ostream&* | os = std::cout, output the resluts into the ostream os. |

**Returns**

a set of bitvector, i.e., set<bitvector>.

Definition at line 428 of file Reasoner.hpp.

#### 5.8.2.28 __inline void argumatrix::Reasoner::printGroundedExt ( ) [inherited]

Print the grounded extension.

**Returns**

no return.

Definition at line 539 of file Reasoner.hpp.

#### 5.8.2.29 void argumatrix::PlReasoner::printLableExtByBlistTerm ( const PlTerm & *plt* ) [protected], [inherited]

Print an extension with the form of the bool list term. Assume a bool list term is [0, 1, 0, 1], the arguments corresponding to entry 1 will be print. The member *m_output* will determine where to print.

**Parameters**

| | |
|---|---|
| *PlTerm&* | plt : The PlTerm is a bool list term, which represents an extension. |

**Returns**

no return.

Definition at line 271 of file PlReasoner.hpp.

#### 5.8.2.30 void argumatrix::PlReasoner::printLableExtByBmatrixTerm ( const PlTerm & *pmtx* ) [protected], [inherited]

Print an extension with the form of the bool matrix term. Assume a bool list term is [[0, 1, 0, 1], [0, 1, 0, 1]], the arguments corresponding to entry 1 will be print. The member *m_output* will determine where to print.

**Parameters**

| | |
|---|---|
| *PlTerm&* | plt : The PlTerm is a bool list term, which represents an extension. |

**Returns**

no return.

Definition at line 725 of file PlReasoner.hpp.

#### 5.8.2.31 void argumatrix::Reasoner::printLabSet ( const bitvector & *bv_ext* ) [inherited]

Print an extension with the form of bitvector. Assume the bitvector is [0, 1, 0, 1], the arguments corresponding to entry 1 will be print. The member *m_output* will determine where to print.

**Parameters**

| | |
|---|---|
| *bitvector&* | bv_ext : The bv_ext is a bitvector, which represents an extension. |

**Returns**

no return.

**See also**

[setOutput]

Definition at line 433 of file Reasoner.hpp.

Referenced by argumatrix::Reasoner::printGroundedExt().

#### 5.8.2.32 void argumatrix::Reasoner::printLabSet ( const std::set< string > & *labset* ) [inherited]

Print a set of arguments. The member *m_output* will determine where to print.

**Parameters**

| | |
|---|---|
| *const* | std::set<string>& labset |

**Returns**

no return.

**See also**

setOutput(streambuf∗ strbuf = std::cout.rdbuf());

Definition at line 454 of file Reasoner.hpp.

#### 5.8.2.33 __inline void argumatrix::PlReasoner::printSomeExt ( const std::string & *predct*, const std::set< string > & *argset* ) [inherited]

Given an ${AF}=< {X}, {R}>$ and an argument $s {X}$ (respectively, a set of arguments $S {X}$), enumerate some set $E {X}$ such that $E {E}_(AF)$ and $s E$ (respectively, $S E$).

Problem [SC-$$]

**Parameters**

| | |
|---|---|
| *a* | set of arguments. |

**Returns**

no return.

Definition at line 468 of file PlReasoner.hpp.

Referenced by argumatrix::AdmissiblePlReasoner::task_SC(), argumatrix::StablePlReasoner::task_SC(), task_S↩C(), argumatrix::StablePlReasoner::task_SE(), and task_SE().

#### 5.8.2.34 __inline streambuf ∗ argumatrix::Reasoner::setOutput ( streambuf ∗ *strbuf* ) [inherited]

Redirect the output stream to streambuf∗ strbuf or ostream& os. If strbuf = cout.rdbuf(), then the output is standard output. It can also redirect the output to a file by the following codes:

```
ofstream ofile("output.txt");
streambuf* oldsb = xxx.setOutput(ofile.rdbuf());
```

or

```
streambuf* oldsb = xxx.setOutput(ofile);
```

.

**Parameters**

| | |
|---|---|
| *strbuf* | is a streambuf pointer. |

**Returns**

return the old streambuf, which can be used to redirect the old streambuf.

Definition at line 473 of file Reasoner.hpp.

#### 5.8.2.35 __inline streambuf ∗ argumatrix::Reasoner::setOutput ( ostream & *os* = std::cout ) [inherited]

Method: setOutput FullName: public argumatrix::Reasoner::setOutput

**See also**

streambuf∗ setOutput(streambuf∗ strbuf = std::cout.rdbuf());

**Parameters**

| | |
|---|---|
| *ostream* | & os |

**Returns**

streambuf∗

Definition at line 479 of file Reasoner.hpp.

#### 5.8.2.36 __inline void argumatrix::CompletePlReasoner::task_DC ( const std::set< string > & *argset* ) [virtual]

Given an $AF = <X,R>$ and an argument s X (respectively, a set of arguments $S X$). Decide whether s contained (respectively, S included) in some E E_(AF) (i.e., credulously justified).

Problem [DC-$$]

Reimplemented from argumatrix::Reasoner.

Definition at line 243 of file CompletePlReasoner.hpp.

#### 5.8.2.37 __inline void argumatrix::CompletePlReasoner::task_DE ( const std::set< string > & *argset* ) [virtual]

Given an $AF = <X,R>$ and a set of arguments $S X$. Decide whether S is a -extension of AF, i.e., S E_(AF).

Problem [DE-$$]

**Parameters**

| | |
|---|---|
| *a* | set of arguments. |

**Returns**

true if argset is a -extension; otherwise return false.

Reimplemented from argumatrix::Reasoner.

Definition at line 199 of file CompletePlReasoner.hpp.

#### 5.8.2.38 void argumatrix::CompletePlReasoner::task_DN ( ) [virtual]

Given an $AF = <X,R>$ and a set of arguments $S X$. Decide whether there exist a non-empty -extension for AF.

Problem [DN-$$]

**Returns**

true if there exist a non-empty -extension; otherwise return false.

Reimplemented from argumatrix::Reasoner.

Definition at line 215 of file CompletePlReasoner.hpp.

#### 5.8.2.39 __inline void argumatrix::CompletePlReasoner::task_DS ( const std::set< string > & *argset* ) [virtual]

Given an $AF = <X,R>$ and an argument s X (respectively, a set of arguments $S X$). Decide whether s contained (respectively, S included) in each E E_(AF) (i.e., skeptically justified).

Problem [DS-$$]

Reimplemented from argumatrix::Reasoner.

Definition at line 258 of file CompletePlReasoner.hpp.

#### 5.8.2.40 __inline void argumatrix::CompletePlReasoner::task_EC ( const std::set< string > & *argset* ) [virtual]

Given an ${AF}=< {X}, {R}>$ and an argument $s {X}$ (respectively, a set of arguments $S {X}$), enumerate all sets $E {X}$ such that $E {E}_(AF)$ and $s E$ (respectively, $S E$).

Problem [EC-$$]

**Parameters**

| | |
|---|---|
| *set<string>,or* | a Boolean vector: a set of argument |

**Returns**

no return.

Reimplemented from argumatrix::Reasoner.

Definition at line 182 of file CompletePlReasoner.hpp.

#### 5.8.2.41 __inline void argumatrix::CompletePlReasoner::task_EE ( ) [virtual]

Print all extensions (with a vector of integers {0,1,2})

Problem [EE-$$]

**Parameters**

| | |
|---|---|
| *no* | argument |

**Returns**

no return.

Reimplemented from argumatrix::Reasoner.

Definition at line 175 of file CompletePlReasoner.hpp.

#### 5.8.2.42 __inline void argumatrix::CompletePlReasoner::task_SC ( const std::set< string > & *argset* ) [virtual]

Given an ${AF}=< {X}, {R}>$ and an argument $s {X}$ (respectively, a set of arguments $S {X}$), enumerate some set $E {X}$ such that $E {E}_(AF)$ and $s E$ (respectively, $S E$).

Problem [SC-$$]

**Parameters**

| | |
|---|---|
| *a* | set of arguments. |

**Returns**

no return.

Reimplemented from argumatrix::Reasoner.

Definition at line 226 of file CompletePlReasoner.hpp.

#### 5.8.2.43 __inline void argumatrix::CompletePlReasoner::task_SE ( ) [virtual]

Given an ${AF}=< {X}, {R}>$, enumerate some set $E {X}$ that are in ${E}_(AF)$.

Problem [SE-$$]

**Parameters**

| | |
|---|---|
| *a* | set of arguments. |

**Returns**

no return.

Reimplemented from argumatrix::Reasoner.

Definition at line 237 of file CompletePlReasoner.hpp.

#### 5.8.2.44 bool argumatrix::PlReasoner::verifyExclusion ( const std::string & *predct*, const **bitvector** & *vecB* ) [protected], [inherited]

Given an $AF = <X,R>$ and a set of arguments $S X$. Decide whether there exist a -extension that excludes S.

**Parameters**

| | |
|---|---|
| *a* | set of arguments. |

**Returns**

true if there exist a -extension that excludes S; otherwise return false.

Definition at line 576 of file PlReasoner.hpp.

#### 5.8.2.45 __inline bool argumatrix::PlReasoner::verifyExtension ( const std::string & *predct*, std::set< string > & *argset* ) [inherited]

Given an $AF = <X,R>$ and a set of arguments $S X$. Decide whether S is a $$-extension of AF, i.e., $S E_(AF)$.

Problem [DE-$$]

**Parameters**

|  |  |
|---|---|
| *a* | set of arguments. |

**Returns**

true if argset is a -extension; otherwise return false.

Definition at line 609 of file PIReasoner.hpp.

#### 5.8.2.46 bool argumatrix::PIReasoner::verifyInclusion ( const std::string & *predct,* const vector< int > & *vecii* ) `[protected]`, `[inherited]`

Given an $AF = <X,R>$ and a set of arguments $S X$. Decide whether there exist a -extension that includes S.

**Parameters**

|  |  |
|---|---|
| *a* | set of arguments. |

**Returns**

true if there exist a -extension that includes S; otherwise return false.

Definition at line 543 of file PIReasoner.hpp.

#### 5.8.2.47 bool argumatrix::PIReasoner::verifyNonemptyExt ( const std::string & *predct* ) `[inherited]`

Given an $AF = <X,R>$ and a set of arguments $S X$. Decide whether there exist a non-empty $$-extension for AF.

Problem [DN-$$]

**Returns**

true if there exist a non-empty -extension; otherwise return false.

Definition at line 510 of file PIReasoner.hpp.

### 5.8.3 Member Data Documentation

#### 5.8.3.1 vector< std::string > argumatrix::Reasoner::m_argLabels `[protected]`, `[inherited]`

Dung's abstract argumentation framework

Definition at line 346 of file Reasoner.hpp.

Referenced by argumatrix::Reasoner::printLabSet().

#### 5.8.3.2 size_type argumatrix::Reasoner::m_argNum [protected], [inherited]

The number of arguments

Definition at line 339 of file Reasoner.hpp.

Referenced by argumatrix::Reasoner::characteristic(), argumatrix::Reasoner::getGroundedExtension(), argumatrix↩ ::Reasoner::getGroundedIntVector(), and argumatrix::Reasoner::labelSet2IntVector().

#### 5.8.3.3 bitmatrix argumatrix::Reasoner::m_BmAtkMtx [protected], [inherited]

The bitmatrix of the Dung Abstract argumentation framework. We can access all attackers of an argument. The attackers of the argument with index i is m_BmAtkMtx[i].

Definition at line 334 of file Reasoner.hpp.

Referenced by argumatrix::Reasoner::getAttacked(), argumatrix::Reasoner::getSelfAttackingArguments(), and argumatrix::Reasoner::is_self_attacking().

#### 5.8.3.4 const DungAF& argumatrix::Reasoner::m_daf [protected], [inherited]

Dung's abstract argumentation framework

Definition at line 342 of file Reasoner.hpp.

Referenced by argumatrix::Reasoner::is_conflict_free(), argumatrix::Reasoner::labelSet2IntVector(), and argumatrix::Reasoner::printBvExts().

#### 5.8.3.5 std::ostream argumatrix::Reasoner::m_output [protected], [inherited]

Where to output

Definition at line 348 of file Reasoner.hpp.

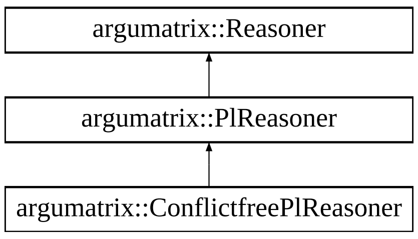
Referenced by argumatrix::Reasoner::printBvExts(), argumatrix::Reasoner::printGroundedExt(), argumatrix::↩ Reasoner::printLabSet(), and argumatrix::Reasoner::setOutput().

The documentation for this class was generated from the following file:

- F:/Codespace/BoostProject/argumatrix/PlReasoner/CompletePlReasoner.hpp

## 5.9 argumatrix::ConflictfreePlReasoner Class Reference

Inheritance diagram for argumatrix::ConflictfreePlReasoner:



#### Public Member Functions

- **ConflictfreePlReasoner** (const DungAF &daf, const string &sm_task="CF", streambuf ∗osbuff=std::cout.↩ rdbuf())
- void task_EE ()
- void **task_EX** ()
- void task_EC (const std::set< string > &argset)

  *Given an ${AF}=< {X}, {R}>$ and an argument $s {X}$ (respectively, a set of arguments $S {X}$), enumerate all sets $E {X}$ such that $E {E}_(AF)$ and $s E$ (respectively, $S E$).*
- void task_SC (const std::set< string > &argset)

  *Given an ${AF}=< {X}, {R}>$ and an argument $s {X}$ (respectively, a set of arguments $S {X}$), enumerate some set $E {X}$ such that $E {E}_(AF)$ and $s E$ (respectively, $S E$).*
- void task_SE ()

  *Given an ${AF}=< {X}, {R}>$, enumerate some set $E {X}$ that are in ${E}_(AF)$.*
- void findAllExts ()
- void task_DE (const std::set< string > &argset)

  *Given an $AF = <X,R>$ and a set of arguments $S X$. Decide whether S is a -extension of AF, i.e., S E_(AF).*
- void task_DN ()

  *Given an $AF = <X,R>$ and a set of arguments $S X$. Decide whether there exist a non-empty -extension for AF.*
- void task_DC (const std::set< string > &argset)

  *Given an $AF = <X,R>$ and an argument s X (respectively, a set of arguments $S X$). Decide whether s contained (respectively, S included) in some E E_(AF) (i.e., credulously justified).*
- void task_DS (const std::set< string > &argset)

  *Given an $AF = <X,R>$ and an argument s X (respectively, a set of arguments $S X$). Decide whether s contained (respectively, S included) in each E E_(AF) (i.e., skeptically justified).*
- void consultPlFile (std::string plFile=ARG_PROLOG_FILE)

  *Loading the Prolog source file [argmat-clpb].*
- void **Test_time** ()
- void printAllExts (const std::string &predct)

  *Print all extensions with respect to semantic predct.*
- void printAllExts (const std::string &predct, const std::set< string > &argset)

  *Given an ${AF}=< {X}, {R}>$ and an argument $s {X}$ (respectively, a set of arguments $S {X}$), enumerate all sets $E {X}$ such that $E {E}_(AF)$ and $s E$ (respectively, $S E$).*
- void **printAllExts** (const std::string &predct, const std::vector< int > &vecii)
- void **printAllExts2** (const std::string &predct)
- void fetchAllExts (const std::string &predct)
- void printSomeExt (const std::string &predct, const std::set< string > &argset)

  *Given an ${AF}=< {X}, {R}>$ and an argument $s {X}$ (respectively, a set of arguments $S {X}$), enumerate some set $E {X}$ such that $E {E}_(AF)$ and $s E$ (respectively, $S E$).*
- void **printSomeExt** (const std::string &predct, const vector< int > &vecii)
- bool verifyNonemptyExt (const std::string &predct)

  *Given an $AF = <X,R>$ and a set of arguments $S X$. Decide whether there exist a non-empty $$-extension for AF.*
- bool verifyExtension (const std::string &predct, std::set< string > &argset)

  *Given an $AF = <X,R>$ and a set of arguments $S X$. Decide whether S is a $$-extension of AF, i.e., $S E_(AF)$.*
- bool **verifyExtension** (const std::string &predct, bitvector &bvecii)
- bool isCredulouslyJustified (const std::string &predct, const std::set< string > &argset)

  *Given an $AF = <X,R>$ and an argument s X (respectively, a set of arguments $S X$). Decide whether s contained (respectively, S included) in some E E_(AF) (i.e., credulously justified).*
- bool isSkepticallyJustified (const std::string &predct, const std::set< string > &argset)

  *Given an $AF = <X,R>$ and an argument s X (respectively, a set of arguments $S X$). Decide whether s contained (respectively, S included) in each E E_(AF) (i.e., skeptically justified).*
- bitvector getAttacked (const bitvector &_bv)

  *Get the attacked arguments by arguments in _bv,* $R^+(S)$ $R^+(S) = x|xisattackedbyS$. $R^+(S_bv) = D * S_bv$.

- bitvector characteristic (const bitvector &_bv)

  *The characteristic function of an abstract argumentation framework: $F_{AF}(S) = A|Aisacceptablewrt.S.$ F_AF(↩ S_bv) = not{ R^+[ not( R^+(S_bv) ) ] }.*
- bitvector characteristic ()

  *The characteristic function of an abstract argumentation framework with the initialization of empty set.*
- bitvector neutrality (const bitvector &_bv)

  *The neutrality function of an abstract argumentation framework: N_AF(S) = {A| All arguments that not attacked by S}. N_AF(S_bv) = not( R^+(S_bv) )*
- bool is_conflict_free (const bitvector &_bv)

  *Argument set S is conflict-free? S is said to be conflict-free iff for any two arguments a,b in S such that a does not attack b.*
- bool **is_conflict_free** (const set< string > &argset)
- bool is_acceptable (const bitvector &S, const bitvector &A)

  *Argument set A is acceptable w.r.t S? Alternately, S defends A? A can be an argument or an argument set. S defends argument (set) A iff for any argument x in X, if attacks A (or a in A) then there is an argument y in S such that y attacks x.*
- bool is_self_attacking (size_type idx)

  *Argument a is self-attacking iff it attacks itself.*
- bool is_admissible (const bitvector &_bv)

  *To decide whether a set of arguments (with the form of bitvector) is an admissible extension. $S$ is an admissible extension iff $S F(S) N(S)$.*
- bool is_complete (const bitvector &_bv)

  *To decide whether a set of arguments (with the form of bitvector) is a complete extension. $S$ is a complete extension iff $S == F(S) N(S)$.*
- bool is_stable (const bitvector &_bv)

  *To decide whether a set of arguments (with the form of bitvector) is a stable extension. $S$ is a stable extension iff $S == N(S)$.*
- bool is_grounded (const bitvector &_bv)

  *To decide whether a set of arguments (with the form of bitvector) is a grounded extension. $S$ is a grounded extension iff it is the least fixed point of the characteristic function F.*
- bitvector getSelfAttackingArguments ()

  *Get all arguments which are self-attacking. Obviously, if an argument i attacks itself, the entry[i][i] of the attack matrix must be 1 (true). Therefore to get the set of self-attacking arguments is to get the diagonal elements of the attack matrix.*
- const set< bitvector > & getBvExtensions ()

  *Get all extensions given a specific semantics in format of bitvector. Each extension is a set of arguments which can "survive the conflict together". Here, we use a bitvector to represent an extension. Therefore, all extensions are formed an set of bit vectors, i.e., set<bitvector>. Each bitvector in set is an extension w.r.t. a given semantics. The computed extensions are stored in m_extensions, therefore, to get all extensions, it must first invoke the function computeExtension()*
- vector< int > getGroundedIntVector ()

  *Get a vector of integers {0, 1, 2}: 2 – Unknown, 1 – in grounded extension, and 0 – attacked by the grounded extension.*
- streambuf ∗ setOutput (streambuf ∗strbuf)

  *Redirect the output stream to streambuf∗ strbuf or ostream& os. If strbuf = cout.rdbuf(), then the output is standard output. It can also redirect the output to a file by the following codes:*
  ```
  ofstream ofile("output.txt");
  streambuf* oldsb = xxx.setOutput(ofile.rdbuf());
  ```
  *or*
  ```
  streambuf* oldsb = xxx.setOutput(ofile);
  ```
  .
- streambuf ∗ setOutput (ostream &os=std::cout)
- void printLabSet (const bitvector &bv_ext)

  *Print an extension with the form of bitvector. Assume the bitvector is [0, 1, 0, 1], the arguments corresponding to entry 1 will be print. The member m_output will determine where to print.*
- void printLabSet (const std::set< string > &labset)

*Print a set of arguments. The member m_output will determine where to print.*
- void printBvExts ()

  *Output all extensions given a specific semantics with an ostream. Each extension is a set of arguments which can "survive the conflict together". Here, we return a string to represent all extensions. For example, if there are two extensions, [a,b] and [b, d], for some problem w.r.t. a given semantics, then we return string "[[a,b],[d,c]]". If extensions is not existing, the string "[ ]" will return.*
- bitvector getGroundedExtension ()

  *Get the grounded extension.*
- void printGroundedExt ()

  *Print the grounded extension.*
- vector< int > labelSet2IntVector (const std::set< string > &label_set)

  *Convert a set of argument (string) labels to an integer vector of {0,1,2}, All arguments in these set, the indices is 1; otherwise 2 (representing unknown)*

### Protected Member Functions

- void findAllExts (const std::string &predct)

  *Find all extensions and store them in m_extensions with the form of bitvector.*
- void createPlAttackMatrix ()

  *Create the attack matrix with the form of PlTerm, which is an input of SWI-Prolog.*
- void printLableExtByBlistTerm (const PlTerm &plt)

  *Print an extension with the form of the bool list term. Assume a bool list term is [0, 1, 0, 1], the arguments corresponding to entry 1 will be print. The member m_output will determine where to print.*
- void printLableExtByBmatrixTerm (const PlTerm &pmtx)

  *Print an extension with the form of the bool matrix term. Assume a bool list term is [[0, 1, 0, 1], [0, 1, 0, 1]], the arguments corresponding to entry 1 will be print. The member m_output will determine where to print.*
- void addBlTermToBvExts (const PlTerm &plt)

  *Convert a bool list term into a bitvertor, and add it to extension.*
- bool verifyInclusion (const std::string &predct, const vector< int > &vecii)

  *Given an $AF = <X,R>$ and a set of arguments $S X$. Decide whether there exist a -extension that includes S.*
- bool verifyExclusion (const std::string &predct, const bitvector &vecB)

  *Given an $AF = <X,R>$ and a set of arguments $S X$. Decide whether there exist a -extension that excludes S.*

### Protected Attributes

- string **m_predicate**
- PlTerm **m_PlAtkMtx**
- bitmatrix m_BmAtkMtx
- size_type m_argNum
- const DungAF & m_daf
- set< bitvector > **m_extensions**
- vector< std::string > m_argLabels
- std::ostream m_output

### 5.9.1 Detailed Description

Definition at line 37 of file ConflictfreePlReasoner.hpp.

### 5.9.2 Member Function Documentation

#### 5.9.2.1 void argumatrix::PlReasoner::addBlTermToBvExts ( const PlTerm & *plt* ) [protected], [inherited]

Convert a bool list term into a bitvertor, and add it to extension.

**Parameters**

| | |
|---|---|
| *PlTerm&* | plt : The PlTerm is a bool list term, which represents an extension. |

**Returns**

no return. The results are added into m_extensions.

Definition at line 303 of file PlReasoner.hpp.

#### 5.9.2.2 __inline argumatrix::bitvector argumatrix::Reasoner::characteristic ( const bitvector & *_bv* ) [inherited]

The characteristic function of an abstract argumentation framework: $F_{AF}(S) = A|Aisacceptablewrt.S$. F_AF(↩
S_bv) = not{ R^+[ not( R^+(S_bv) ) ] }.

**Parameters**

| | |
|---|---|
| *extension* | an extension (a set of arguments), the default is empty set. |

**Returns**

a set of arguments in bitvector form.

Definition at line 368 of file Reasoner.hpp.

#### 5.9.2.3 __inline argumatrix::bitvector argumatrix::Reasoner::characteristic ( ) [inherited]

The characteristic function of an abstract argumentation framework with the initialization of empty set.

**Returns**

a set of arguments in bitvector form.

Definition at line 375 of file Reasoner.hpp.

Referenced by argumatrix::Reasoner::getGroundedExtension(), and argumatrix::Reasoner::is_acceptable().

#### 5.9.2.4 void argumatrix::PlReasoner::consultPlFile ( std::string *plFile* = ARG_PROLOG_FILE ) [inherited]

Loading the Prolog source file [argmat-clpb].

Method: consultPlFile

**Parameters**

| | |
|---|---|
| *std::string* | plFile |

**Note**

SWI-Prolog supports compilation of individual or multiple Prolog source files into ‘Quick Load Files’. A ‘Quick Load File’ (.qlf file) stores the contents of the file in a precompiled format. These files load considerably faster than source files and are normally more compact. They are machine-independent and may thus be loaded on any implementation of SWI-Prolog. Note, however, that clauses are stored as virtual machine instructions. Changes to the compiler will generally make old compiled files unusable. Quick Load Files are created using qcompile/1. They are loaded using consult/1 or one of the other file-loading predicates described in section 4.3. If consult/1 is given an explicit .pl file, it will load the Prolog source. When given a .qlf file, it will load the file. When no extension is specified, it will load the .qlf file when present and the .pl file otherwise.

Definition at line 253 of file PlReasoner.hpp.

#### 5.9.2.5 void argumatrix::PlReasoner::createPlAttackMatrix ( ) [protected], [inherited]

Create the attack matrix with the form of PlTerm, which is an input of SWI-Prolog.

**Returns**

no return. The result is stored in member variable *m_PlAtkMtx*.

Definition at line 243 of file PlReasoner.hpp.

#### 5.9.2.6 void argumatrix::PlReasoner::fetchAllExts ( const std::string & *predct* ) [inherited]

Method: fetchAllExts FullName: public argumatrix::PlReasoner::fetchAllExts

**Parameters**

| | |
|---|---|
| *const* | std::string & predct |

**Returns**

void

**Return values**

| | |
|---|---|
| | |

Definition at line 701 of file PlReasoner.hpp.

#### 5.9.2.7 __inline void argumatrix::ConflictfreePlReasoner::findAllExts ( )

Find all extensions and store them in *m_extensions* with the form of bitvector.

**Parameters**

| | |
|---|---|
| *no* | argument. |

**Returns**

no return. The results are added into *m_extensions*.

Definition at line 176 of file ConflictfreePlReasoner.hpp.

#### 5.9.2.8 void argumatrix::PlReasoner::findAllExts ( const std::string & *predct* ) `[protected], [inherited]`

Find all extensions and store them in *m_extensions* with the form of bitvector.

**Parameters**

| | |
|---|---|
| *string&* | predct: The predicate. |

**Returns**

no return. The results are added into *m_extensions*.

Definition at line 669 of file PlReasoner.hpp.

Referenced by argumatrix::AdmissiblePlReasoner::findAllExts(), argumatrix::StablePlReasoner::findAllExts(), findAllExts(), and argumatrix::CompletePlReasoner::findAllExts().

#### 5.9.2.9 __inline argumatrix::bitvector argumatrix::Reasoner::getAttacked ( const bitvector & *_bv* ) `[inherited]`

Get the attacked arguments by arguments in _bv, $R^+(S)$ $R^+(S) = x|xisattackedbyS$. $R^+(S_bv) = D * S_bv$.

**Parameters**

| | |
|---|---|
| *_bv* | the bitvector with respect to the set $S$. |

**Returns**

a set of arguments in bitvector form.

Definition at line 360 of file Reasoner.hpp.

Referenced by argumatrix::Reasoner::getGroundedIntVector(), argumatrix::Reasoner::is_conflict_free(), and argumatrix::Reasoner::neutrality().

#### 5.9.2.10 __inline const set< bitvector > & argumatrix::Reasoner::getBvExtensions ( ) `[inherited]`

Get all extensions given a specific semantics in format of bitvector. Each extension is a set of arguments which can "survive the conflict together". Here, we use a bitvector to represent an extension. Therefore, all extensions are formed an set of bit vectors, i.e., set<bitvector>. Each bitvector in set is an extension w.r.t. a given semantics. The computed extensions are stored in m_extensions, therefore, to get all extensions, it must first invoke the function computeExtension()

**Returns**

a set of bitvector, i.e., set<bitvector>.

Definition at line 400 of file Reasoner.hpp.

#### 5.9.2.11 argumatrix::bitvector argumatrix::Reasoner::getGroundedExtension ( ) [inherited]

Get the grounded extension.

**Returns**

a set of arguments in bitvector form.

Definition at line 484 of file Reasoner.hpp.

Referenced by argumatrix::Reasoner::is_grounded(), and argumatrix::Reasoner::printGroundedExt().

#### 5.9.2.12 vector< int > argumatrix::Reasoner::getGroundedIntVector ( ) [inherited]

Get a vector of integers {0, 1, 2}: 2 – Unknown, 1 – in grounded extension, and 0 – attacked by the grounded extension.

**Returns**

a vector of integers {0, 1, 2}.

Definition at line 499 of file Reasoner.hpp.

#### 5.9.2.13 __inline argumatrix::bitvector argumatrix::Reasoner::getSelfAttackingArguments ( ) [inherited]

Get all arguments which are self-attacking. Obviously, if an argument i attacks itself, the entry[i][i] of the attack matrix must be 1 (true). Therefore to get the set of self-attacking arguments is to get the diagonal elements of the attack matrix.

**Returns**

a set of arguments in bitvector form, if bitvector[i] = true representing the argument (with index) i is a self-attacking argument, otherwise is not.

Definition at line 414 of file Reasoner.hpp.

#### 5.9.2.14 __inline bool argumatrix::Reasoner::is_acceptable ( const bitvector & *S*, const bitvector & *A* ) [inherited]

Argument set A is acceptable w.r.t S? Alternately, S defends A? A can be an argument or an argument set. S defends argument (set) A iff for any argument x in X, if attacks A (or a in A) then there is an argument y in S such that y attacks x.

**Parameters**

| | |
|---|---|
| *S* | a set of arguments. |
| *A* | an argument or an argument set |

**Returns**

true if S defends A.

Definition at line 394 of file Reasoner.hpp.

#### 5.9.2.15 __inline bool argumatrix::Reasoner::is_admissible ( const bitvector & *_bv* ) [inherited]

To decide whether a set of arguments (with the form of bitvector) is an admissible extension. $S$ is an admissible extension iff $S F(S) N(S)$.

**Returns**

true if _bv is admissible; otherwise returns false.

Definition at line 553 of file Reasoner.hpp.

#### 5.9.2.16 __inline bool argumatrix::Reasoner::is_complete ( const bitvector & *_bv* ) [inherited]

To decide whether a set of arguments (with the form of bitvector) is a complete extension. $S$ is a complete extension iff $S == F(S) N(S)$.

**Returns**

true if _bv is complete; otherwise returns false.

Definition at line 564 of file Reasoner.hpp.

#### 5.9.2.17 __inline bool argumatrix::Reasoner::is_conflict_free ( const bitvector & *_bv* ) [inherited]

Argument set S is conflict-free? S is said to be conflict-free iff for any two arguments a,b in S such that a does not attack b.

**Parameters**

| | |
|---|---|
| *extension* | an extension (a set of arguments). |

**Returns**

true if S is conflict-free.

Definition at line 381 of file Reasoner.hpp.

#### 5.9.2.18 __inline bool argumatrix::Reasoner::is_grounded ( const bitvector & *_bv* ) [inherited]

To decide whether a set of arguments (with the form of bitvector) is a grounded extension. $S$ is a grounded extension iff it is the least fixed point of the characteristic function F.

**Returns**

true if _bv is grounded; otherwise returns false.

Definition at line 584 of file Reasoner.hpp.

#### 5.9.2.19 __inline bool argumatrix::Reasoner::is_self_attacking ( size_type *idx* ) [inherited]

Argument a is self-attacking iff it attacks itself.

**Parameters**

| | |
|---|---|
| *idx* | the index of the argument w.r.t the attack matrix |

**Returns**

true if S defends A.

Definition at line 406 of file Reasoner.hpp.

#### 5.9.2.20 __inline bool argumatrix::Reasoner::is_stable ( const bitvector & *_bv* ) [inherited]

To decide whether a set of arguments (with the form of bitvector) is a stable extension. $S$ is a stable extension iff $S == N(S)$.

**Returns**

true if _bv is stable; otherwise returns false.

Definition at line 575 of file Reasoner.hpp.

#### 5.9.2.21 bool argumatrix::PlReasoner::isCredulouslyJustified ( const std::string & *predct,* const std::set< string > & *argset* ) [inherited]

Given an $AF = <X,R>$ and an argument s X (respectively, a set of arguments $S X$). Decide whether s contained (respectively, S included) in some E E_(AF) (i.e., credulously justified).

Problem [DC-$$]

Definition at line 642 of file PlReasoner.hpp.

#### 5.9.2.22 bool argumatrix::PlReasoner::isSkepticallyJustified ( const std::string & *predct,* const std::set< string > & *argset* ) [inherited]

Given an $AF = <X,R>$ and an argument s X (respectively, a set of arguments $S X$). Decide whether s contained (respectively, S included) in each E E_(AF) (i.e., skeptically justified).

Problem [DS-$$]

Definition at line 654 of file PlReasoner.hpp.

#### 5.9.2.23 vector< int > argumatrix::Reasoner::labelSet2IntVector ( const std::set< string > & *label_set* ) [inherited]

Convert a set of argument (string) labels to an integer vector of {0,1,2}, All arguments in these set, the indices is 1; otherwise 2 (representing unknown)

**Returns**

no return.

Definition at line 526 of file Reasoner.hpp.

#### 5.9.2.24 __inline argumatrix::bitvector argumatrix::Reasoner::neutrality ( const bitvector & *_bv* ) [inherited]

The neutrality function of an abstract argumentation framework: N_AF(S) = {A| All arguments that not attacked by S}. N_AF(S_bv) = not( R^+(S_bv) )

**Parameters**

| | |
|---|---|
| *extension* | an extension (a set of arguments). |

**Returns**

a set of arguments in bitvector form.

Definition at line 547 of file Reasoner.hpp.

Referenced by argumatrix::Reasoner::characteristic(), argumatrix::Reasoner::getGroundedIntVector(), argumatrix↩ ::Reasoner::is_admissible(), argumatrix::Reasoner::is_complete(), and argumatrix::Reasoner::is_stable().

#### 5.9.2.25 void argumatrix::PlReasoner::printAllExts ( const std::string & *predct* ) `[inherited]`

Print all extensions with respect to semantic predct.

Problem [EE-$$]

**Parameters**

| | |
|---|---|
| *no* | argument |

**Returns**

no return.

Definition at line 430 of file PlReasoner.hpp.

Referenced by argumatrix::PlReasoner::printAllExts(), argumatrix::StablePlReasoner::task_EC(), argumatrix↩ ::AdmissiblePlReasoner::task_EC(), task_EC(), argumatrix::CompletePlReasoner::task_EC(), task_EE(), argumatrix::AdmissiblePlReasoner::task_EE(), argumatrix::StablePlReasoner::task_EE(), and argumatrix::↩ CompletePlReasoner::task_EE().

#### 5.9.2.26 __inline void argumatrix::PlReasoner::printAllExts ( const std::string & *predct,* const std::set< string > & *argset* ) `[inherited]`

Given an ${AF}=< {X}, {R}>$ and an argument $s {X}$ (respectively, a set of arguments $S {X}$), enumerate all sets $E {X}$ such that $E {E}_(AF)$ and $s E$ (respectively, $S E$).

Problem [EC-$$]

**Parameters**

| | |
|---|---|
| *set<string>,or* | a Boolean vector: a set of argument |

**Returns**

no return.

Definition at line 386 of file PlReasoner.hpp.

#### 5.9.2.27 __inline void argumatrix::Reasoner::printBvExts ( ) [inherited]

Output all extensions given a specific semantics with an ostream. Each extension is a set of arguments which can "survive the conflict together". Here, we return a string to represent all extensions. For example, if there are two extensions, [a,b] and [b, d], for some problem w.r.t. a given semantics, then we return string "[[a,b],[d,c]]". If extensions is not existing, the string "[ ]" will return.

**Parameters**

| | |
|---|---|
| *ostream&* | os = std::cout, output the resluts into the ostream os. |

**Returns**

a set of bitvector, i.e., set$<$bitvector$>$.

Definition at line 428 of file Reasoner.hpp.

#### 5.9.2.28 __inline void argumatrix::Reasoner::printGroundedExt ( ) [inherited]

Print the grounded extension.

**Returns**

no return.

Definition at line 539 of file Reasoner.hpp.

#### 5.9.2.29 void argumatrix::PlReasoner::printLableExtByBlistTerm ( const PlTerm & *plt* ) [protected], [inherited]

Print an extension with the form of the bool list term. Assume a bool list term is [0, 1, 0, 1], the arguments corresponding to entry 1 will be print. The member *m_output* will determine where to print.

**Parameters**

| | |
|---|---|
| *PlTerm&* | plt : The PlTerm is a bool list term, which represents an extension. |

**Returns**

no return.

Definition at line 271 of file PlReasoner.hpp.

#### 5.9.2.30 void argumatrix::PlReasoner::printLableExtByBmatrixTerm ( const PlTerm & *pmtx* ) [protected], [inherited]

Print an extension with the form of the bool matrix term. Assume a bool list term is [[0, 1, 0, 1], [0, 1, 0, 1]], the arguments corresponding to entry 1 will be print. The member *m_output* will determine where to print.

**Parameters**

| | |
|---|---|
| *PlTerm&* | plt : The PlTerm is a bool list term, which represents an extension. |

**Returns**

no return.

Definition at line 725 of file PlReasoner.hpp.

#### 5.9.2.31 void argumatrix::Reasoner::printLabSet ( const bitvector & *bv_ext* ) [inherited]

Print an extension with the form of bitvector. Assume the bitvector is [0, 1, 0, 1], the arguments corresponding to entry 1 will be print. The member *m_output* will determine where to print.

**Parameters**

| | |
|---|---|
| *bitvector&* | bv_ext : The bv_ext is a bitvector, which represents an extension. |

**Returns**

no return.

**See also**

[setOutput]

Definition at line 433 of file Reasoner.hpp.

Referenced by argumatrix::Reasoner::printGroundedExt().

#### 5.9.2.32 void argumatrix::Reasoner::printLabSet ( const std::set< string > & *labset* ) [inherited]

Print a set of arguments. The member *m_output* will determine where to print.

**Parameters**

| | |
|---|---|
| *const* | std::set<string>& labset |

**Returns**

no return.

**See also**

setOutput(streambuf∗ strbuf = std::cout.rdbuf());

Definition at line 454 of file Reasoner.hpp.

#### 5.9.2.33 __inline void argumatrix::PlReasoner::printSomeExt ( const std::string & *predct,* const std::set< string > & *argset* ) [inherited]

Given an ${AF}=< {X}, {R}>$ and an argument $s {X}$ (respectively, a set of arguments $S {X}$), enumerate some set $E {X}$ such that $E {E}_(AF)$ and $s E$ (respectively, $S E$).

Problem [SC-$$]

**Parameters**

| | |
|---|---|
| *a* | set of arguments. |

**Returns**

no return.

Definition at line 468 of file PlReasoner.hpp.

Referenced by argumatrix::AdmissiblePlReasoner::task_SC(), argumatrix::StablePlReasoner::task_SC(), argumatrix::CompletePlReasoner::task_SC(), argumatrix::StablePlReasoner::task_SE(), and argumatrix::CompletePlReasoner::task_SE().

#### 5.9.2.34 __inline streambuf ∗ argumatrix::Reasoner::setOutput ( streambuf ∗ *strbuf* ) [inherited]

Redirect the output stream to streambuf∗ strbuf or ostream& os. If strbuf = cout.rdbuf(), then the output is standard output. It can also redirect the output to a file by the following codes:

```
ofstream ofile("output.txt");
streambuf* oldsb = xxx.setOutput(ofile.rdbuf());
```

or

```
streambuf* oldsb = xxx.setOutput(ofile);
```

.

**Parameters**

| | |
|---|---|
| *strbuf* | is a streambuf pointer. |

**Returns**

return the old streambuf, which can be used to redirect the old streambuf.

Definition at line 473 of file Reasoner.hpp.

#### 5.9.2.35 __inline streambuf ∗ argumatrix::Reasoner::setOutput ( ostream & *os* = std::cout ) [inherited]

Method: setOutput FullName: public argumatrix::Reasoner::setOutput

**See also**

streambuf∗ setOutput(streambuf∗ strbuf = std::cout.rdbuf());

**Parameters**

| | |
|---|---|
| *ostream* | & os |

**Returns**

streambuf∗

Definition at line 479 of file Reasoner.hpp.

#### 5.9.2.36 __inline void argumatrix::ConflictfreePlReasoner::task_DC ( const std::set< string > & *argset* ) [virtual]

Given an $AF = <X,R>$ and an argument s X (respectively, a set of arguments $S X$). Decide whether s contained (respectively, S included) in some E E_(AF) (i.e., credulously justified).

Problem [DC-$$]

Reimplemented from argumatrix::Reasoner.

Definition at line 239 of file ConflictfreePlReasoner.hpp.

#### 5.9.2.37 __inline void argumatrix::ConflictfreePlReasoner::task_DE ( const std::set< string > & *argset* ) [virtual]

Given an $AF = <X,R>$ and a set of arguments $S X$. Decide whether S is a -extension of AF, i.e., S E_(AF).

Problem [DE-$$]

**Parameters**

| | |
|---|---|
| *a* | set of arguments. |

**Returns**

true if argset is a -extension; otherwise return false.

Reimplemented from argumatrix::Reasoner.

Definition at line 182 of file ConflictfreePlReasoner.hpp.

#### 5.9.2.38 void argumatrix::ConflictfreePlReasoner::task_DN ( ) [virtual]

Given an $AF = <X,R>$ and a set of arguments $S X$. Decide whether there exist a non-empty -extension for AF.

Problem [DN-$$]

**Returns**

true if there exist a non-empty -extension; otherwise return false.

**Note**

If an AF does not exist non-empty conflict-free extension iff all its arguments are self-attacking.

Reimplemented from argumatrix::Reasoner.

Definition at line 197 of file ConflictfreePlReasoner.hpp.

#### 5.9.2.39 __inline void argumatrix::ConflictfreePlReasoner::task_DS ( const std::set< string > & *argset* ) [virtual]

Given an $AF = <X,R>$ and an argument s X (respectively, a set of arguments $S X$). Decide whether s contained (respectively, S included) in each E E_(AF) (i.e., skeptically justified).

Problem [DS-$$]

Reimplemented from argumatrix::Reasoner.

Definition at line 254 of file ConflictfreePlReasoner.hpp.

#### 5.9.2.40 __inline void argumatrix::ConflictfreePlReasoner::task_EC ( const std::set< string > & *argset* ) [virtual]

Given an ${AF}=< {X}, {R}>$ and an argument $s {X}$ (respectively, a set of arguments $S {X}$), enumerate all sets $E {X}$ such that $E {E}_(AF)$ and $s E$ (respectively, $S E$).

Problem [EC-$$]

**Parameters**

| | |
|---|---|
| *set<string>,or* | a Boolean vector: a set of argument |

**Returns**

no return.

Reimplemented from argumatrix::Reasoner.

Definition at line 165 of file ConflictfreePlReasoner.hpp.

#### 5.9.2.41 __inline void argumatrix::ConflictfreePlReasoner::task_EE ( ) [virtual]

Problem [EE-$$] Print all extensions (with a vector of integers {0,1,2})

**Parameters**

| | |
|---|---|
| *no* | argument |

**Returns**

no return.

Reimplemented from argumatrix::Reasoner.

Definition at line 157 of file ConflictfreePlReasoner.hpp.

#### 5.9.2.42 __inline void argumatrix::ConflictfreePlReasoner::task_SC ( const std::set< string > & *argset* ) [virtual]

Given an ${AF}=< {X}, {R}>$ and an argument $s {X}$ (respectively, a set of arguments $S {X}$), enumerate some set $E {X}$ such that $E {E}_(AF)$ and $s E$ (respectively, $S E$).

Problem [SC-$$]

**Parameters**

| | |
|---|---|
| *a* | set of arguments. |

**Returns**

no return.

Reimplemented from argumatrix::Reasoner.

Definition at line 216 of file ConflictfreePIReasoner.hpp.

#### 5.9.2.43 __inline void argumatrix::ConflictfreePIReasoner::task_SE ( ) [virtual]

Given an ${AF}=< {X}, {R}>$, enumerate some set $E {X}$ that are in ${E}_(AF)$.

Problem [SE-$$]

**Parameters**

| | |
|---|---|
| *a* | set of arguments. |

**Returns**

no return.

Reimplemented from argumatrix::Reasoner.

Definition at line 232 of file ConflictfreePIReasoner.hpp.

#### 5.9.2.44 bool argumatrix::PIReasoner::verifyExclusion ( const std::string & *predct,* const **bitvector** & *vecB* ) [protected], [inherited]

Given an $AF = <X,R>$ and a set of arguments $S X$. Decide whether there exist a -extension that excludes S.

**Parameters**

| | |
|---|---|
| *a* | set of arguments. |

**Returns**

true if there exist a -extension that excludes S; otherwise return false.

Definition at line 576 of file PIReasoner.hpp.

#### 5.9.2.45 __inline bool argumatrix::PIReasoner::verifyExtension ( const std::string & *predct,* std::set< string > & *argset* ) [inherited]

Given an $AF = <X,R>$ and a set of arguments $S X$. Decide whether S is a $$-extension of AF, i.e., $S E_(AF)$.

Problem [DE-$$]

**Parameters**

| | |
|---|---|
| *a* | set of arguments. |

**Returns**

true if argset is a -extension; otherwise return false.

Definition at line 609 of file PIReasoner.hpp.

#### 5.9.2.46 bool argumatrix::PIReasoner::verifyInclusion ( const std::string & *predct,* const vector< int > & *vecii* ) [protected], [inherited]

Given an $AF = <X,R>$ and a set of arguments $S X$. Decide whether there exist a -extension that includes S.

**Parameters**

| | |
|---|---|
| *a* | set of arguments. |

**Returns**

true if there exist a -extension that includes S; otherwise return false.

Definition at line 543 of file PIReasoner.hpp.

#### 5.9.2.47 bool argumatrix::PIReasoner::verifyNonemptyExt ( const std::string & *predct* ) [inherited]

Given an $AF = <X,R>$ and a set of arguments $S X$. Decide whether there exist a non-empty $$-extension for AF.

Problem [DN-$$]

**Returns**

true if there exist a non-empty -extension; otherwise return false.

Definition at line 510 of file PIReasoner.hpp.

### 5.9.3 Member Data Documentation

#### 5.9.3.1 vector< std::string > argumatrix::Reasoner::m_argLabels [protected], [inherited]

Dung's abstract argumentation framework

Definition at line 346 of file Reasoner.hpp.

Referenced by argumatrix::Reasoner::printLabSet().

#### 5.9.3.2 size_type argumatrix::Reasoner::m_argNum [protected], [inherited]

The number of arguments

Definition at line 339 of file Reasoner.hpp.

Referenced by argumatrix::Reasoner::characteristic(), argumatrix::Reasoner::getGroundedExtension(), argumatrix↩ ::Reasoner::getGroundedIntVector(), and argumatrix::Reasoner::labelSet2IntVector().

#### 5.9.3.3 bitmatrix argumatrix::Reasoner::m_BmAtkMtx [protected], [inherited]

The bitmatrix of the Dung Abstract argumentation framework. We can access all attackers of an argument. The attackers of the argument with index i is m_BmAtkMtx[i].

Definition at line 334 of file Reasoner.hpp.

Referenced by argumatrix::Reasoner::getAttacked(), argumatrix::Reasoner::getSelfAttackingArguments(), and argumatrix::Reasoner::is_self_attacking().

#### 5.9.3.4 const DungAF& argumatrix::Reasoner::m_daf [protected], [inherited]

Dung's abstract argumentation framework

Definition at line 342 of file Reasoner.hpp.

Referenced by argumatrix::Reasoner::is_conflict_free(), argumatrix::Reasoner::labelSet2IntVector(), and argumatrix::Reasoner::printBvExts().

#### 5.9.3.5 std::ostream argumatrix::Reasoner::m_output [protected], [inherited]

Where to output

Definition at line 348 of file Reasoner.hpp.

Referenced by argumatrix::Reasoner::printBvExts(), argumatrix::Reasoner::printGroundedExt(), argumatrix::↩ Reasoner::printLabSet(), and argumatrix::Reasoner::setOutput().

The documentation for this class was generated from the following file:

- F:/Codespace/BoostProject/argumatrix/PlReasoner/ConflictfreePlReasoner.hpp

## 5.10 argumatrix::DungAF Class Reference

### Public Member Functions

- Argument **addArgument** (const ArgumentProperty &_argProp)
- Argument **addArgument** (string _label, string _title="", string _description="")
- Attack **addAttack** (const std::string &_arg1_label, const std::string &_arg2_label, const AttackProperty &_↩ attProp=AttackProperty())
- Attack **addAttack** (const Argument &_arg1, const std::string &_arg2_label, const AttackProperty &_att↩ Prop=AttackProperty())
- Attack **addAttack** (const std::string &_arg1_label, const Argument &_arg2, const AttackProperty &_att↩ Prop=AttackProperty())
- Attack **addAttack** (const Argument &_arg1, const Argument &_arg2, const AttackProperty &_att↩ Prop=AttackProperty())
- Attack **addAttack** (const std::pair< Argument &, Argument & > &_edge, const AttackProperty &_att↩ Prop=AttackProperty())
- const ArgumentProperty & getArgumentProperty (const Argument &_arg) const
- const ArgumentProperty & **getArgumentProperty** (const std::string &_arg_label) const
- size_type **getArgumentIdx** (const std::string &_arg_label) const
- size_type **getArgumentIdx** (const Argument &_arg) const
- const AttackProperty & **getAttackProperty** (const Attack &_atk) const
- std::vector< std::string > **getArgumentLabels** () const
- void **clear** ()
- bitvector set2bv (const set< Argument > &sa) const
- set< Argument > **bv2set** (const bitvector &bv) const
- set< string > **bv2label_set** (const bitvector &bv) const
- bitvector **labelSet2bv** (const set< string > &_ls) const
- set< bitvector > **label_ext2bv_ext** (const set< set< string > > &_le) const
- std::string **toString** () const
- bitmatrix **getAttackMatrix** () const
- size_type **getNumberOfArguments** () const
- void **showSet** (const set< Argument > &as, std::ostream &os=std::cout) const
- void outputBv (const bitvector &_bv, std::ostream &os=std::cout) const
- void outputBvSet (const std::set< bitvector > &_bv_set, std::ostream &os=std::cout) const

### 5.10.1 Detailed Description

Definition at line 97 of file DungAF.hpp.

### 5.10.2 Member Function Documentation

#### 5.10.2.1 const ArgumentProperty & argumatrix::DungAF::getArgumentProperty ( const Argument & *_arg* ) const [inline]

Get the Property of the Argument or the Attack Note that the return value must be const since all external changes on properties are not safe. Let the label of argument a is "A" and assume the returned property is not constant, for instance, if an argument b, whose label is 'B', then we can modify the label of argument b as "A", which may contradict with the restriction that each argument has a unique label. The return value of constant may avoid this issue. We also provide a series of methods to change the property of some argument or attack with safety-check.

**Parameters**

| | |
|---|---|
| *Argument* | or Attack, or the unique argument label |

**Returns**

no return. The results are stored in m_extensions.

Definition at line 304 of file DungAF.hpp.

#### 5.10.2.2 void argumatrix::DungAF::outputBv ( const **bitvector** & *_bv,* std::ostream & *os* = `std::cout` ) const

Output a bitvector. A bitvector represent a set of arguments in bool vector form. This function will output a bitvector in string, but not the string with 0 and 1.

**Parameters**

| | |
|---|---|
| *bitvector* | _bv: A bitvector. |
| *ostream&* | os: The out put stream. The default value is cout. |

**Returns**

no return. The results are stored in m_extensions.

Definition at line 426 of file DungAF.hpp.

#### 5.10.2.3 void argumatrix::DungAF::outputBvSet ( const std::set< **bitvector** > & *_bv_set,* std::ostream & *os* = `std::cout` ) const

Output a set of bitvector set<bitvector>. We know that one bitvector represent a set of arguments in bool vector form. Therefore, a set of bitvector may represent a set of a set of arguments. This function will output a set of bitvector in string.

**Parameters**

| | |
|---|---|
| *set<* | bitvector >& _bv_set: A set of bitvector. |
| *ostream&* | os: The out put stream. The default value is cout. |

**Returns**

no return. The results are stored in m_extensions.

Definition at line 445 of file DungAF.hpp.

Referenced by argumatrix::Reasoner::printBvExts().

#### 5.10.2.4 **bitvector** argumatrix::DungAF::set2bv ( const set< Argument > & *sa* ) const

Method: set2bv FullName: argumatrix::DungAF::set2bv Access: public

#### Parameters

| | |
|---|---|
| *const* | set<Argument> & sa |

#### Returns

argumatrix::bitvector

Definition at line 321 of file DungAF.hpp.

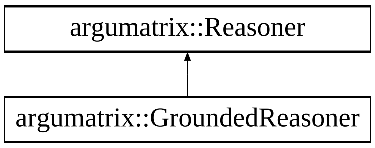
The documentation for this class was generated from the following file:

- F:/Codespace/BoostProject/argumatrix/dung_theory/DungAF.hpp

## 5.11 argumatrix::GroundedReasoner Class Reference

```
#include <GroundedReasoner.hpp>
```

Inheritance diagram for argumatrix::GroundedReasoner:



#### Public Member Functions

- **GroundedReasoner** (const DungAF &daf, streambuf $*$osbuff=std::cout.rdbuf())
- void task_EE ()
- void task_EC (const std::set< string > &argset)

  *Given an AF $= \langle \mathcal{X}, \mathcal{R} \rangle$ and an argument $s \in \mathcal{X}$ (respectively, a set of arguments $S \subseteq \mathcal{X}$), enumerate all sets $E \subseteq \mathcal{X}$ such that $E \in \mathcal{E}_{GR}(AF)$ and $s \in E$ (respectively, $S \subseteq E$).*
- void task_SC (const std::set< string > &argset)

  *Given an AF $= \langle \mathcal{X}, \mathcal{R} \rangle$ and an argument $s \in \mathcal{X}$ (respectively, a set of arguments $S \subseteq \mathcal{X}$), enumerate some set $E \subseteq \mathcal{X}$ such that $E \in \mathcal{E}_{GR}(AF)$ and $s \in E$ (respectively, $S \subseteq E$).*
- void task_SE ()

  *Given an AF $= \langle \mathcal{X}, \mathcal{R} \rangle$, enumerate some set $E \subseteq \mathcal{X}$ that are in $\mathcal{E}_{GR}(AF)$.*
- void task_DE (const std::set< string > &argset)

  *Given an $AF = \langle X, R \rangle$ and a set of arguments $S \subseteq X$. Decide whether S is a {GR}-extension of AF, i.e., S E_{GR}(AF).*
- void task_DN ()

  *Given an $AF = \langle X, R \rangle$ and a set of arguments $S \subseteq X$. Decide whether there exist a non-empty {GR}-extension for AF.*
- void task_DC (const std::set< string > &argset)

  *Given an $AF = \langle X, R \rangle$ and an argument s X (respectively, a set of arguments $S \subseteq X$). Decide whether s contained (respectively, S included) in some E E_{GR}(AF) (i.e., credulously justified).*
- void task_DS (const std::set< string > &argset)

  *Given an $AF = \langle X, R \rangle$ and an argument s X (respectively, a set of arguments $S \subseteq X$). Decide whether s contained (respectively, S included) in each E E_{GR}(AF) (i.e., skeptically justified).*

- bitvector getAttacked (const bitvector &_bv)

  *Get the attacked arguments by arguments in _bv,* $R^{+}(S)$ $R^{+}(S) = x|xisattackedbyS.$ $R^{+}(S_bv) = D * S_bv.$

- bitvector characteristic (const bitvector &_bv)

  *The characteristic function of an abstract argumentation framework:* $F_{AF}(S) = A|Aisacceptablewrt.S.$ *F_AF(↩ S_bv) = not{ R^+[ not( R^+(S_bv) ) ] }.*

- bitvector characteristic ()

  *The characteristic function of an abstract argumentation framework with the initialization of empty set.*

- bitvector neutrality (const bitvector &_bv)

  *The neutrality function of an abstract argumentation framework: N_AF(S) = {A| All arguments that not attacked by S}. N_AF(S_bv) = not( R^+(S_bv) )*

- bool is_conflict_free (const bitvector &_bv)

  *Argument set S is conflict-free? S is said to be conflict-free iff for any two arguments a,b in S such that a does not attack b.*

- bool **is_conflict_free** (const set< string > &argset)
- bool is_acceptable (const bitvector &S, const bitvector &A)

  *Argument set A is acceptable w.r.t S? Alternately, S defends A? A can be an argument or an argument set. S defends argument (set) A iff for any argument x in X, if attacks A (or a in A) then there is an argument y in S such that y attacks x.*

- bool is_self_attacking (size_type idx)

  *Argument a is self-attacking iff it attacks itself.*

- bool is_admissible (const bitvector &_bv)

  *To decide whether a set of arguments (with the form of bitvector) is an admissible extension. $S$ is an admissible extension iff $S F(S) N(S)$.*

- bool is_complete (const bitvector &_bv)

  *To decide whether a set of arguments (with the form of bitvector) is a complete extension. $S$ is a complete extension iff $S == F(S) N(S)$.*

- bool is_stable (const bitvector &_bv)

  *To decide whether a set of arguments (with the form of bitvector) is a stable extension. $S$ is a stable extension iff $S == N(S)$.*

- bool is_grounded (const bitvector &_bv)

  *To decide whether a set of arguments (with the form of bitvector) is a grounded extension. $S$ is a grounded extension iff it is the least fixed point of the characteristic function F.*

- bitvector getSelfAttackingArguments ()

  *Get all arguments which are self-attacking. Obviously, if an argument i attacks itself, the entry[i][i] of the attack matrix must be 1 (true). Therefore to get the set of self-attacking arguments is to get the diagonal elements of the attack matrix.*

- const set< bitvector > & getBvExtensions ()

  *Get all extensions given a specific semantics in format of bitvector. Each extension is a set of arguments which can "survive the conflict together". Here, we use a bitvector to represent an extension. Therefore, all extensions are formed an set of bit vectors, i.e., set<bitvector>. Each bitvector in set is an extension w.r.t. a given semantics. The computed extensions are stored in m_extensions, therefore, to get all extensions, it must first invoke the function computeExtension()*

- vector< int > getGroundedIntVector ()

  *Get a vector of integers {0, 1, 2}: 2 – Unknown, 1 – in grounded extension, and 0 – attacked by the grounded extension.*

- streambuf ∗ setOutput (streambuf ∗strbuf)

  *Redirect the output stream to streambuf∗ strbuf or ostream& os. If strbuf = cout.rdbuf(), then the output is standard output. It can also redirect the output to a file by the following codes:*
  ```
  ofstream ofile("output.txt");
  streambuf* oldsb = xxx.setOutput(ofile.rdbuf());
  ```
  *or*
  ```
  streambuf* oldsb = xxx.setOutput(ofile);
  ```
  *.*

- streambuf ∗ setOutput (ostream &os=std::cout)
- void printLabSet (const bitvector &bv_ext)

*Print an extension with the form of bitvector. Assume the bitvector is [0, 1, 0, 1], the arguments corresponding to entry 1 will be print. The member m_output will determine where to print.*

- void printLabSet (const std::set< string > &labset)

  *Print a set of arguments. The member m_output will determine where to print.*
- void printBvExts ()

  *Output all extensions given a specific semantics with an ostream. Each extension is a set of arguments which can "survive the conflict together". Here, we return a string to represent all extensions. For example, if there are two extensions, [a,b] and [b, d], for some problem w.r.t. a given semantics, then we return string "[[a,b],[d,c]]". If extensions is not existing, the string "[ ]" will return.*
- bitvector getGroundedExtension ()

  *Get the grounded extension.*
- void printGroundedExt ()

  *Print the grounded extension.*
- vector< int > labelSet2IntVector (const std::set< string > &label_set)

  *Convert a set of argument (string) labels to an integer vector of {0,1,2}, All arguments in these set, the indices is 1; otherwise 2 (representing unknown)*
- virtual void **task_EX** ()

#### Protected Attributes

- bitmatrix m_BmAtkMtx
- size_type m_argNum
- const DungAF & m_daf
- set< bitvector > **m_extensions**
- vector< std::string > m_argLabels
- std::ostream m_output

### 5.11.1 Detailed Description

This reasoner for Dung theories performs inference on the grounded extension. Computes the (unique) grounded extension, i.e., the least fixpoint of the characteristic function.

Definition at line 33 of file GroundedReasoner.hpp.

### 5.11.2 Member Function Documentation

#### 5.11.2.1 __inline **argumatrix::bitvector** argumatrix::Reasoner::characteristic ( const **bitvector** & *_bv* ) [inherited]

The characteristic function of an abstract argumentation framework: $F_{AF}(S) = A|Aisacceptablewrt.S.$ F_AF(↩ S_bv) = not{ R^+[ not( R^+(S_bv) ) ] }.

**Parameters**

| | |
|---|---|
| *extension* | an extension (a set of arguments), the default is empty set. |

**Returns**

a set of arguments in bitvector form.

Definition at line 368 of file Reasoner.hpp.

#### 5.11.2.2 __inline argumatrix::bitvector argumatrix::Reasoner::characteristic ( ) [inherited]

The characteristic function of an abstract argumentation framework with the initialization of empty set.

**Returns**

a set of arguments in bitvector form.

Definition at line 375 of file Reasoner.hpp.

Referenced by argumatrix::Reasoner::getGroundedExtension(), and argumatrix::Reasoner::is_acceptable().

#### 5.11.2.3 __inline argumatrix::bitvector argumatrix::Reasoner::getAttacked ( const bitvector & _bv ) [inherited]

Get the attacked arguments by arguments in _bv, $R^{+}(S)$ $R^{+}(S) = x | xisattackedbyS$. $R^{+}(S_bv) = D * S_bv$.

**Parameters**

| | |
|---|---|
| *_bv* | the bitvector with respect to the set $S$. |

**Returns**

a set of arguments in bitvector form.

Definition at line 360 of file Reasoner.hpp.

Referenced by argumatrix::Reasoner::getGroundedIntVector(), argumatrix::Reasoner::is_conflict_free(), and argumatrix::Reasoner::neutrality().

#### 5.11.2.4 __inline const set< bitvector > & argumatrix::Reasoner::getBvExtensions ( ) [inherited]

Get all extensions given a specific semantics in format of bitvector. Each extension is a set of arguments which can "survive the conflict together". Here, we use a bitvector to represent an extension. Therefore, all extensions are formed an set of bit vectors, i.e., set<bitvector>. Each bitvector in set is an extension w.r.t. a given semantics. The computed extensions are stored in m_extensions, therefore, to get all extensions, it must first invoke the function computeExtension()

**Returns**

a set of bitvector, i.e., set<bitvector>.

Definition at line 400 of file Reasoner.hpp.

#### 5.11.2.5 argumatrix::bitvector argumatrix::Reasoner::getGroundedExtension ( ) [inherited]

Get the grounded extension.

**Returns**

a set of arguments in bitvector form.

Definition at line 484 of file Reasoner.hpp.

Referenced by argumatrix::Reasoner::is_grounded(), and argumatrix::Reasoner::printGroundedExt().

#### 5.11.2.6 vector< int > argumatrix::Reasoner::getGroundedIntVector ( ) [inherited]

Get a vector of integers {0, 1, 2}: 2 – Unknown, 1 – in grounded extension, and 0 – attacked by the grounded extension.

**Returns**

a vector of integers {0, 1, 2}.

Definition at line 499 of file Reasoner.hpp.

#### 5.11.2.7 __inline argumatrix::bitvector argumatrix::Reasoner::getSelfAttackingArguments ( ) [inherited]

Get all arguments which are self-attacking. Obviously, if an argument i attacks itself, the entry[i][i] of the attack matrix must be 1 (true). Therefore to get the set of self-attacking arguments is to get the diagonal elements of the attack matrix.

**Returns**

a set of arguments in bitvector form, if bitvector[i] = true representing the argument (with index) i is a self-attacking argument, otherwise is not.

Definition at line 414 of file Reasoner.hpp.

#### 5.11.2.8 __inline bool argumatrix::Reasoner::is_acceptable ( const bitvector & *S*, const bitvector & *A* ) [inherited]

Argument set A is acceptable w.r.t S? Alternately, S defends A? A can be an argument or an argument set. S defends argument (set) A iff for any argument x in X, if attacks A (or a in A) then there is an argument y in S such that y attacks x.

**Parameters**

| | |
|---|---|
| *S* | a set of arguments. |
| *A* | an argument or an argument set |

**Returns**

true if S defends A.

Definition at line 394 of file Reasoner.hpp.

#### 5.11.2.9 __inline bool argumatrix::Reasoner::is_admissible ( const bitvector & *_bv* ) [inherited]

To decide whether a set of arguments (with the form of bitvector) is an admissible extension. $S$ is an admissible extension iff $S F(S) N(S)$.

**Returns**

true if _bv is admissible; otherwise returns false.

Definition at line 553 of file Reasoner.hpp.

#### 5.11.2.10 __inline bool argumatrix::Reasoner::is_complete ( const bitvector & *_bv* ) `[inherited]`

To decide whether a set of arguments (with the form of bitvector) is a complete extension. $S$ is a complete extension iff $S == F(S) N(S)$.

**Returns**

true if _bv is complete; otherwise returns false.

Definition at line 564 of file Reasoner.hpp.

#### 5.11.2.11 __inline bool argumatrix::Reasoner::is_conflict_free ( const bitvector & *_bv* ) `[inherited]`

Argument set S is conflict-free? S is said to be conflict-free iff for any two arguments a,b in S such that a does not attack b.

**Parameters**

| | |
|---|---|
| *extension* | an extension (a set of arguments). |

**Returns**

true if S is conflict-free.

Definition at line 381 of file Reasoner.hpp.

#### 5.11.2.12 __inline bool argumatrix::Reasoner::is_grounded ( const bitvector & *_bv* ) `[inherited]`

To decide whether a set of arguments (with the form of bitvector) is a grounded extension. $S$ is a grounded extension iff it is the least fixed point of the characteristic function F.

**Returns**

true if _bv is grounded; otherwise returns false.

Definition at line 584 of file Reasoner.hpp.

#### 5.11.2.13 __inline bool argumatrix::Reasoner::is_self_attacking ( size_type *idx* ) `[inherited]`

Argument a is self-attacking iff it attacks itself.

**Parameters**

| | |
|---|---|
| *idx* | the index of the argument w.r.t the attack matrix |

**Returns**

true if S defends A.

Definition at line 406 of file Reasoner.hpp.

#### 5.11.2.14 __inline bool argumatrix::Reasoner::is_stable ( const bitvector & *_bv* ) [inherited]

To decide whether a set of arguments (with the form of bitvector) is a stable extension. $S$ is a stable extension iff $S == N(S)$.

**Returns**

true if _bv is stable; otherwise returns false.

Definition at line 575 of file Reasoner.hpp.

#### 5.11.2.15 vector< int > argumatrix::Reasoner::labelSet2IntVector ( const std::set< string > & *label_set* ) [inherited]

Convert a set of argument (string) labels to an integer vector of {0,1,2}, All arguments in these set, the indices is 1; otherwise 2 (representing unknown)

**Returns**

no return.

Definition at line 526 of file Reasoner.hpp.

#### 5.11.2.16 __inline argumatrix::bitvector argumatrix::Reasoner::neutrality ( const bitvector & *_bv* ) [inherited]

The neutrality function of an abstract argumentation framework: N_AF(S) = {A| All arguments that not attacked by S}. N_AF(S_bv) = not( R^+(S_bv) )

**Parameters**

| | |
|---|---|
| *extension* | an extension (a set of arguments). |

**Returns**

a set of arguments in bitvector form.

Definition at line 547 of file Reasoner.hpp.

Referenced by argumatrix::Reasoner::characteristic(), argumatrix::Reasoner::getGroundedIntVector(), argumatrix↩ ::Reasoner::is_admissible(), argumatrix::Reasoner::is_complete(), and argumatrix::Reasoner::is_stable().

#### 5.11.2.17 __inline void argumatrix::Reasoner::printBvExts ( ) [inherited]

Output all extensions given a specific semantics with an ostream. Each extension is a set of arguments which can "survive the conflict together". Here, we return a string to represent all extensions. For example, if there are two extensions, [a,b] and [b, d], for some problem w.r.t. a given semantics, then we return string "[[a,b],[d,c]]". If extensions is not existing, the string "[ ]" will return.

**Parameters**

| | |
|---|---|
| *ostream&* | os = std::cout, output the resluts into the ostream os. |

**Returns**

a set of bitvector, i.e., set<bitvector>.

Definition at line 428 of file Reasoner.hpp.

#### 5.11.2.18 __inline void argumatrix::Reasoner::printGroundedExt ( ) [inherited]

Print the grounded extension.

**Returns**

no return.

Definition at line 539 of file Reasoner.hpp.

#### 5.11.2.19 void argumatrix::Reasoner::printLabSet ( const bitvector & *bv_ext* ) [inherited]

Print an extension with the form of bitvector. Assume the bitvector is [0, 1, 0, 1], the arguments corresponding to entry 1 will be print. The member *m_output* will determine where to print.

**Parameters**

| | |
|---|---|
| *bitvector&* | bv_ext : The bv_ext is a bitvector, which represents an extension. |

**Returns**

no return.

**See also**

[setOutput]

Definition at line 433 of file Reasoner.hpp.

Referenced by argumatrix::Reasoner::printGroundedExt().

#### 5.11.2.20 void argumatrix::Reasoner::printLabSet ( const std::set< string > & *labset* ) [inherited]

Print a set of arguments. The member *m_output* will determine where to print.

**Parameters**

| | |
|---|---|
| *const* | std::set<string>& labset |

**Returns**

no return.

**See also**

setOutput(streambuf∗ strbuf = std::cout.rdbuf());

Definition at line 454 of file Reasoner.hpp.

#### 5.11.2.21 __inline streambuf ∗ argumatrix::Reasoner::setOutput ( streambuf ∗ *strbuf* ) [inherited]

Redirect the output stream to streambuf∗ strbuf or ostream& os. If strbuf = cout.rdbuf(), then the output is standard output. It can also redirect the output to a file by the following codes:

```
ofstream ofile("output.txt");
streambuf* oldsb = xxx.setOutput(ofile.rdbuf());
```

or

```
streambuf* oldsb = xxx.setOutput(ofile);
```

.

**Parameters**

| | |
|---|---|
| *strbuf* | is a streambuf pointer. |

**Returns**

return the old streambuf, which can be used to redirect the old streambuf.

Definition at line 473 of file Reasoner.hpp.

#### 5.11.2.22 __inline streambuf ∗ argumatrix::Reasoner::setOutput ( ostream & *os* = std::cout ) [inherited]

Method: setOutput FullName: public argumatrix::Reasoner::setOutput

**See also**

streambuf∗ setOutput(streambuf∗ strbuf = std::cout.rdbuf());

**Parameters**

| | |
|---|---|
| *ostream* | & os |

**Returns**

streambuf∗

Definition at line 479 of file Reasoner.hpp.

#### 5.11.2.23 __inline void argumatrix::GroundedReasoner::task_DC ( const std::set< string > & *argset* ) [virtual]

Given an $AF = \langle X, R \rangle$ and an argument s X (respectively, a set of arguments $S \subseteq X$). Decide whether s contained (respectively, S included) in some E E_{GR}(AF) (i.e., credulously justified).

Problem [DC- GR]

Reimplemented from argumatrix::Reasoner.

Definition at line 219 of file GroundedReasoner.hpp.

#### 5.11.2.24 __inline void argumatrix::GroundedReasoner::task_DE ( const std::set< string > & *argset* ) [virtual]

Given an $AF = \langle X, R \rangle$ and a set of arguments $S \subseteq X$. Decide whether S is a {GR}-extension of AF, i.e., S E_{GR}(AF).

Problem [DE- GR]

**Parameters**

| | |
|---|---|
| *a* | set of arguments. |

**Returns**

true if argset is a {GR}-extension; otherwise return false.

**Note**

For this task, the *set<string>& argset* can be empty, which means to decide whether the empty set is a GR-extension of AF. This indicates that the option -a is not necessary, then the default *set<string>& argset* is empty.

Reimplemented from argumatrix::Reasoner.

Definition at line 156 of file GroundedReasoner.hpp.

#### 5.11.2.25 __inline void argumatrix::GroundedReasoner::task_DN ( ) [virtual]

Given an $AF = \langle X, R \rangle$ and a set of arguments $S \subseteq X$. Decide whether there exist a non-empty {GR}-extension for AF.

Problem [DN- GR]

**Returns**

true if there exist a non-empty {GR}-extension; otherwise return false.

Reimplemented from argumatrix::Reasoner.

Definition at line 174 of file GroundedReasoner.hpp.

#### 5.11.2.26 __inline void argumatrix::GroundedReasoner::task_DS ( const std::set< string > & *argset* ) [virtual]

Given an $AF = \langle X, R \rangle$ and an argument s X (respectively, a set of arguments $S \subseteq X$). Decide whether s contained (respectively, S included) in each E E_{GR}(AF) (i.e., skeptically justified).

Problem [DS- GR]

Reimplemented from argumatrix::Reasoner.

Definition at line 240 of file GroundedReasoner.hpp.

#### 5.11.2.27 __inline void argumatrix::GroundedReasoner::task_EC ( const std::set< string > & *argset* ) [virtual]

Given an $AF = \langle \mathcal{X}, \mathcal{R} \rangle$ and an argument $s \in \mathcal{X}$ (respectively, a set of arguments $S \subseteq \mathcal{X}$), enumerate all sets $E \subseteq \mathcal{X}$ such that $E \in \mathcal{E}_{\text{GR}}(AF)$ and $s \in E$ (respectively, $S \subseteq E$).

Problem [EC- GR]

**Parameters**

| | |
|---|---|
| *set<string>,or* | a Boolean vector: a set of argument |

**Returns**

no return.

Reimplemented from argumatrix::Reasoner.

Definition at line 130 of file GroundedReasoner.hpp.

#### 5.11.2.28 __inline void argumatrix::GroundedReasoner::task_EE ( ) [virtual]

Problem [EE- GR] Print all extensions

**Parameters**

| | |
|---|---|
| *no* | argument |

**Returns**

no return.

Reimplemented from argumatrix::Reasoner.

Definition at line 124 of file GroundedReasoner.hpp.

#### 5.11.2.29 __inline void argumatrix::GroundedReasoner::task_SC ( const std::set< string > & *argset* ) [virtual]

Given an $AF = \langle \mathcal{X}, \mathcal{R} \rangle$ and an argument $s \in \mathcal{X}$ (respectively, a set of arguments $S \subseteq \mathcal{X}$), enumerate some set $E \subseteq \mathcal{X}$ such that $E \in \mathcal{E}_{\text{GR}}(AF)$ and $s \in E$ (respectively, $S \subseteq E$).

Problem [SC- GR]

**Parameters**

|  |  |
|---|---|
| *a* | set of arguments. |

**Returns**

no return.

Reimplemented from argumatrix::Reasoner.

Definition at line 188 of file GroundedReasoner.hpp.

#### 5.11.2.30 __inline void argumatrix::GroundedReasoner::task_SE ( ) [virtual]

Given an $AF = \langle \mathcal{X}, \mathcal{R} \rangle$, enumerate some set $E \subseteq \mathcal{X}$ that are in $\mathcal{E}_{\mathrm{GR}}(AF)$.

Problem [SE- GR]

**Parameters**

|  |  |
|---|---|
| *a* | set of arguments. |

**Returns**

no return.

Reimplemented from argumatrix::Reasoner.

Definition at line 213 of file GroundedReasoner.hpp.

### 5.11.3 Member Data Documentation

#### 5.11.3.1 vector< std::string > argumatrix::Reasoner::m_argLabels [protected], [inherited]

Dung's abstract argumentation framework

Definition at line 346 of file Reasoner.hpp.

Referenced by argumatrix::Reasoner::printLabSet().

#### 5.11.3.2 size_type argumatrix::Reasoner::m_argNum [protected], [inherited]

The number of arguments

Definition at line 339 of file Reasoner.hpp.

Referenced by argumatrix::Reasoner::characteristic(), argumatrix::Reasoner::getGroundedExtension(), argumatrix↵ ::Reasoner::getGroundedIntVector(), and argumatrix::Reasoner::labelSet2IntVector().

#### 5.11.3.3 bitmatrix argumatrix::Reasoner::m_BmAtkMtx [protected], [inherited]

The bitmatrix of the Dung Abstract argumentation framework. We can access all attackers of an argument. The attackers of the argument with index i is m_BmAtkMtx[i].

Definition at line 334 of file Reasoner.hpp.

Referenced by argumatrix::Reasoner::getAttacked(), argumatrix::Reasoner::getSelfAttackingArguments(), and argumatrix::Reasoner::is_self_attacking().

#### 5.11.3.4 const DungAF& argumatrix::Reasoner::m_daf [protected], [inherited]

Dung's abstract argumentation framework

Definition at line 342 of file Reasoner.hpp.

Referenced by argumatrix::Reasoner::is_conflict_free(), argumatrix::Reasoner::labelSet2IntVector(), and argumatrix::Reasoner::printBvExts().

#### 5.11.3.5 std::ostream argumatrix::Reasoner::m_output [protected], [inherited]

Where to output

Definition at line 348 of file Reasoner.hpp.

Referenced by argumatrix::Reasoner::printBvExts(), argumatrix::Reasoner::printGroundedExt(), argumatrix::↩
Reasoner::printLabSet(), and argumatrix::Reasoner::setOutput().

The documentation for this class was generated from the following file:

- F:/Codespace/BoostProject/argumatrix/dung_theory/GroundedReasoner.hpp

## 5.12 boost::graph::internal_vertex_constructor< argumatrix::ArgumentProperty > Struct Template Reference

```
#include <DungAF.hpp>
```

### Public Types

- typedef vertex_from_name< argumatrix::ArgumentProperty > **type**

### 5.12.1 Detailed Description

**template<>**
**struct boost::graph::internal_vertex_constructor< argumatrix::ArgumentProperty >**

Allow the graph to build Arguments given only their labels (filling in the defaults for fields).

Definition at line 51 of file DungAF.hpp.

The documentation for this struct was generated from the following file:

- F:/Codespace/BoostProject/argumatrix/dung_theory/DungAF.hpp

## 5.13 boost::graph::internal_vertex_name< argumatrix::ArgumentProperty > Struct Template Reference

Use the label of ArgumentProperty as a key for indexing Arguments in a graph.

`#include <DungAF.hpp>`

### Public Types

- typedef multi_index::member< argumatrix::ArgumentProperty, std::string,&argumatrix::ArgumentProperty↩ ::label > **type**

### 5.13.1 Detailed Description

**template<>**
**struct boost::graph::internal_vertex_name< argumatrix::ArgumentProperty >**

Use the label of ArgumentProperty as a key for indexing Arguments in a graph.

Definition at line 42 of file DungAF.hpp.

The documentation for this struct was generated from the following file:

- F:/Codespace/BoostProject/argumatrix/dung_theory/DungAF.hpp

## 5.14 argumatrix::parser Class Reference

### Static Public Member Functions

- static bool Aspartix2DungAF (const std::string &filePath, DungAF &daf)

  *Read Aspartix format from a file to DungAF. We have implemented three approaches to read aspartix format. stl↩ ::regex, boost::regex and the approach from ArgSemSAT.1.0rc3. By testing, we choose boost::regex since it is more stable than stl::regex, and slightly faster than ArgSemSAT.*
- static bool TrivialGraph2DungAF (const std::string &filePath, DungAF &daf)

### 5.14.1 Detailed Description

Definition at line 27 of file parser.hpp.

### 5.14.2 Member Function Documentation

#### 5.14.2.1 bool argumatrix::parser::Aspartix2DungAF ( const std::string & *filePath*, DungAF & *daf* ) `[static]`

Read Aspartix format from a file to DungAF. We have implemented three approaches to read aspartix format. stl↩ ::regex, boost::regex and the approach from ArgSemSAT.1.0rc3. By testing, we choose boost::regex since it is more stable than stl::regex, and slightly faster than ArgSemSAT.

**Parameters**

| | |
|---|---|
| *std::string* | filePath – the file path of the Aspartix format file. |
| *DungAF&* | daf – the output abstract argumentation framework |

**Returns**

bool. If the translation is successful return true, else return false.

Definition at line 113 of file parser.hpp.

#### 5.14.2.2 bool argumatrix::parser::TrivialGraph2DungAF ( const std::string & *filePath,* DungAF & *daf* ) [static]

Read Trivial Graph Format from a file to DungAF

**Parameters**

| | |
|---|---|
| *std::string* | filePath – the file path of the Aspartix format file. |
| *DungAF&* | daf – the output abstract argumentation framework |

**Returns**

bool. If the translation is successful return true, else return false.

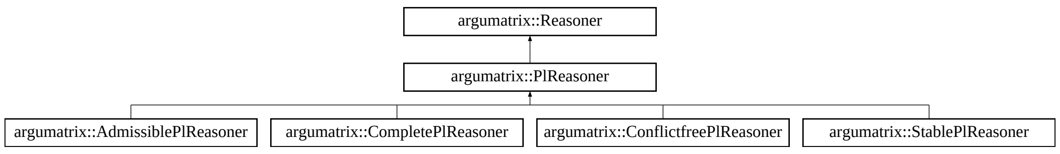
Definition at line 236 of file parser.hpp.

The documentation for this class was generated from the following file:

- F:/Codespace/BoostProject/argumatrix/parser/parser.hpp

## 5.15 argumatrix::PlReasoner Class Reference

Inheritance diagram for argumatrix::PlReasoner:


argumatrix::Reasoner
argumatrix::PlReasoner
argumatrix::AdmissiblePlReasoner
argumatrix::CompletePlReasoner
argumatrix::ConflictfreePlReasoner
argumatrix::StablePlReasoner

**Public Member Functions**

- **PlReasoner** (const DungAF &daf, streambuf ∗osbuff=std::cout.rdbuf())
- void consultPlFile (std::string plFile=ARG_PROLOG_FILE)

  *Loading the Prolog source file [argmat-clpb].*
- void **Test_time** ()
- void printAllExts (const std::string &predct)

  *Print all extensions with respect to semantic predct.*

- void **printAllExts2** (const std::string &predct)
- void fetchAllExts (const std::string &predct)
- void printAllExts (const std::string &predct, const std::set< string > &argset)

  *Given an ${AF}=< {X}, {R}>$ and an argument $s {X}$ (respectively, a set of arguments $S {X}$), enumerate all sets $E {X}$ such that $E {E}_(AF)$ and $s E$ (respectively, $S E$).*
- void **printAllExts** (const std::string &predct, const std::vector< int > &vecii)
- void printSomeExt (const std::string &predct, const std::set< string > &argset)

  *Given an ${AF}=< {X}, {R}>$ and an argument $s {X}$ (respectively, a set of arguments $S {X}$), enumerate some set $E {X}$ such that $E {E}_(AF)$ and $s E$ (respectively, $S E$).*
- void **printSomeExt** (const std::string &predct, const vector< int > &vecii)
- bool verifyNonemptyExt (const std::string &predct)

  *Given an $AF = <X,R>$ and a set of arguments $S X$. Decide whether there exist a non-empty $$-extension for AF.*
- bool verifyExtension (const std::string &predct, std::set< string > &argset)

  *Given an $AF = <X,R>$ and a set of arguments $S X$. Decide whether S is a $$-extension of AF, i.e., $S E_(AF)$.*
- bool **verifyExtension** (const std::string &predct, bitvector &bvecii)
- bool isCredulouslyJustified (const std::string &predct, const std::set< string > &argset)

  *Given an $AF = <X,R>$ and an argument s X (respectively, a set of arguments $S X$). Decide whether s contained (respectively, S included) in some E E_(AF) (i.e., credulously justified).*
- bool isSkepticallyJustified (const std::string &predct, const std::set< string > &argset)

  *Given an $AF = <X,R>$ and an argument s X (respectively, a set of arguments $S X$). Decide whether s contained (respectively, S included) in each E E_(AF) (i.e., skeptically justified).*
- bitvector getAttacked (const bitvector &_bv)

  *Get the attacked arguments by arguments in _bv,* $R^+(S)$ $R^+(S) = x|xisattackedbyS.$ $R^+(S_bv) = D * S_bv.$
- bitvector characteristic (const bitvector &_bv)

  *The characteristic function of an abstract argumentation framework:* $F_{AF}(S) = A|Aisacceptablewrt.S.$ *F_AF(↩ S_bv) = not{ R^+[ not( R^+(S_bv) ) ] }.*
- bitvector characteristic ()

  *The characteristic function of an abstract argumentation framework with the initialization of empty set.*
- bitvector neutrality (const bitvector &_bv)

  *The neutrality function of an abstract argumentation framework: N_AF(S) = {A| All arguments that not attacked by S}. N_AF(S_bv) = not( R^+(S_bv) )*
- bool is_conflict_free (const bitvector &_bv)

  *Argument set S is conflict-free? S is said to be conflict-free iff for any two arguments a,b in S such that a does not attack b.*
- bool **is_conflict_free** (const set< string > &argset)
- bool is_acceptable (const bitvector &S, const bitvector &A)

  *Argument set A is acceptable w.r.t S? Alternately, S defends A? A can be an argument or an argument set. S defends argument (set) A iff for any argument x in X, if attacks A (or a in A) then there is an argument y in S such that y attacks x.*
- bool is_self_attacking (size_type idx)

  *Argument a is self-attacking iff it attacks itself.*
- bool is_admissible (const bitvector &_bv)

  *To decide whether a set of arguments (with the form of bitvector) is an admissible extension. $S$ is an admissible extension iff $S F(S) N(S)$.*
- bool is_complete (const bitvector &_bv)

  *To decide whether a set of arguments (with the form of bitvector) is a complete extension. $S$ is a complete extension iff $S == F(S) N(S)$.*
- bool is_stable (const bitvector &_bv)

  *To decide whether a set of arguments (with the form of bitvector) is a stable extension. $S$ is a stable extension iff $S == N(S)$.*
- bool is_grounded (const bitvector &_bv)

  *To decide whether a set of arguments (with the form of bitvector) is a grounded extension. $S$ is a grounded extension iff it is the least fixed point of the characteristic function F.*

- bitvector getSelfAttackingArguments ()

  *Get all arguments which are self-attacking. Obviously, if an argument i attacks itself, the entry[i][i] of the attack matrix must be 1 (true). Therefore to get the set of self-attacking arguments is to get the diagonal elements of the attack matrix.*
- const set< bitvector > & getBvExtensions ()

  *Get all extensions given a specific semantics in format of bitvector. Each extension is a set of arguments which can "survive the conflict together". Here, we use a bitvector to represent an extension. Therefore, all extensions are formed an set of bit vectors, i.e., set<bitvector>. Each bitvector in set is an extension w.r.t. a given semantics. The computed extensions are stored in m_extensions, therefore, to get all extensions, it must first invoke the function computeExtension()*
- vector< int > getGroundedIntVector ()

  *Get a vector of integers {0, 1, 2}: 2 – Unknown, 1 – in grounded extension, and 0 – attacked by the grounded extension.*
- streambuf ∗ setOutput (streambuf ∗strbuf)

  *Redirect the output stream to streambuf∗ strbuf or ostream& os. If strbuf = cout.rdbuf(), then the output is standard output. It can also redirect the output to a file by the following codes:*
  ```
  ofstream ofile("output.txt");
  streambuf* oldsb = xxx.setOutput(ofile.rdbuf());
  ```
  *or*
  ```
  streambuf* oldsb = xxx.setOutput(ofile);
  ```
  .
- streambuf ∗ setOutput (ostream &os=std::cout)
- void printLabSet (const bitvector &bv_ext)

  *Print an extension with the form of bitvector. Assume the bitvector is [0, 1, 0, 1], the arguments corresponding to entry 1 will be print. The member m_output will determine where to print.*
- void printLabSet (const std::set< string > &labset)

  *Print a set of arguments. The member m_output will determine where to print.*
- void printBvExts ()

  *Output all extensions given a specific semantics with an ostream. Each extension is a set of arguments which can "survive the conflict together". Here, we return a string to represent all extensions. For example, if there are two extensions, [a,b] and [b, d], for some problem w.r.t. a given semantics, then we return string "[[a,b],[d,c]]". If extensions is not existing, the string "[ ]" will return.*
- bitvector getGroundedExtension ()

  *Get the grounded extension.*
- void printGroundedExt ()

  *Print the grounded extension.*
- vector< int > labelSet2IntVector (const std::set< string > &label_set)

  *Convert a set of argument (string) labels to an integer vector of {0,1,2}, All arguments in these set, the indices is 1; otherwise 2 (representing unknown)*
- virtual void task_EE ()

  *Problem [EE-$$] Print all extensions.*
- virtual void **task_EX** ()
- virtual void task_EC (const std::set< string > &argset)

  *Problem [EC-$$] Given an ${AF}=< {X}, {R}>$ and an argument $s {X}$ (respectively, a set of arguments $S {X}$), enumerate all sets $E {X}$ such that $E {E}_(AF)$ and $s E$ (respectively, $S E$).*
- virtual void task_SC (const std::set< string > &argset)

  *Problem [SC-$$] Given an ${AF}=< {X}, {R}>$ and an argument $s {X}$ (respectively, a set of arguments $S {X}$), enumerate some set $E {X}$ such that $E {E}_(AF)$ and $s E$ (respectively, $S E$).*
- virtual void task_SE ()

  *Problem [SE-$$] Given an ${AF}=< {X}, {R}>$, enumerate some set $E {X}$ that are in ${E}_(AF)$.*
- virtual void task_DE (const std::set< string > &argset)

  *Problem [DE-$$] Given an $AF = <X,R>$ and a set of arguments $S X$. Decide whether S is a -extension of AF, i.e., S E_(AF).*
- virtual void task_DN ()

  *Problem [DN-$$] Given an $AF = <X,R>$ and a set of arguments $S X$. Decide whether there exist a non-empty -extension for AF.*

- virtual void task_DC (const std::set< string > &argset)

  *Problem [DC-$$] Given an $AF = <X,R>$ and an argument s X (respectively, a set of arguments $S X$). Decide whether s contained (respectively, S included) in some E E_(AF) (i.e., credulously justified).*
- virtual void task_DS (const std::set< string > &argset)

  *Problem [DS-$$] Given an $AF = <X,R>$ and an argument s X (respectively, a set of arguments $S X$). Decide whether s contained (respectively, S included) in each E E_(AF) (i.e., skeptically justified).*

### Protected Member Functions

- void findAllExts (const std::string &predct)

  *Find all extensions and store them in m_extensions with the form of bitvector.*
- void createPlAttackMatrix ()

  *Create the attack matrix with the form of PlTerm, which is an input of SWI-Prolog.*
- void printLableExtByBlistTerm (const PlTerm &plt)

  *Print an extension with the form of the bool list term. Assume a bool list term is [0, 1, 0, 1], the arguments corresponding to entry 1 will be print. The member m_output will determine where to print.*
- void printLableExtByBmatrixTerm (const PlTerm &pmtx)

  *Print an extension with the form of the bool matrix term. Assume a bool list term is [[0, 1, 0, 1], [0, 1, 0, 1]], the arguments corresponding to entry 1 will be print. The member m_output will determine where to print.*
- void addBlTermToBvExts (const PlTerm &plt)

  *Convert a bool list term into a bitvertor, and add it to extension.*
- bool verifyInclusion (const std::string &predct, const vector< int > &vecii)

  *Given an $AF = <X,R>$ and a set of arguments $S X$. Decide whether there exist a -extension that includes S.*
- bool verifyExclusion (const std::string &predct, const bitvector &vecB)

  *Given an $AF = <X,R>$ and a set of arguments $S X$. Decide whether there exist a -extension that excludes S.*

### Protected Attributes

- PlTerm **m_PlAtkMtx**
- bitmatrix m_BmAtkMtx
- size_type m_argNum
- const DungAF & m_daf
- set< bitvector > **m_extensions**
- vector< std::string > m_argLabels
- std::ostream m_output

### 5.15.1 Detailed Description

Definition at line 55 of file PlReasoner.hpp.

### 5.15.2 Member Function Documentation

#### 5.15.2.1 void argumatrix::PlReasoner::addBlTermToBvExts ( const PlTerm & *plt* ) [protected]

Convert a bool list term into a bitvertor, and add it to extension.

**Parameters**

| | |
|---|---|
| *PlTerm&* | plt : The PlTerm is a bool list term, which represents an extension. |

**Returns**

no return. The results are added into m_extensions.

Definition at line 303 of file PlReasoner.hpp.

#### 5.15.2.2 __inline **argumatrix::bitvector** argumatrix::Reasoner::characteristic ( const **bitvector** & *_bv* ) `[inherited]`

The characteristic function of an abstract argumentation framework: $F_{AF}(S) = A|Aisacceptablewrt.S$. F_AF(↩
S_bv) = not{ R^+[ not( R^+(S_bv) ) ] }.

**Parameters**

| | |
|---|---|
| *extension* | an extension (a set of arguments), the default is empty set. |

**Returns**

a set of arguments in bitvector form.

Definition at line 368 of file Reasoner.hpp.

#### 5.15.2.3 __inline **argumatrix::bitvector** argumatrix::Reasoner::characteristic ( ) `[inherited]`

The characteristic function of an abstract argumentation framework with the initialization of empty set.

**Returns**

a set of arguments in bitvector form.

Definition at line 375 of file Reasoner.hpp.

Referenced by argumatrix::Reasoner::getGroundedExtension(), and argumatrix::Reasoner::is_acceptable().

#### 5.15.2.4 void argumatrix::PlReasoner::consultPlFile ( std::string *plFile =* `ARG_PROLOG_FILE` )

Loading the Prolog source file [argmat-clpb].

Method: consultPlFile

**Parameters**

| | |
|---|---|
| *std::string* | plFile |

**Note**

SWI-Prolog supports compilation of individual or multiple Prolog source files into 'Quick Load Files'. A 'Quick Load File' (.qlf file) stores the contents of the file in a precompiled format. These files load considerably faster than source files and are normally more compact. They are machine-independent and may thus be loaded on any implementation of SWI-Prolog. Note, however, that clauses are stored as virtual machine instructions. Changes to the compiler will generally make old compiled files unusable. Quick Load Files are created using qcompile/1. They are loaded using consult/1 or one of the other file-loading predicates described in section 4.3. If consult/1 is given an explicit .pl file, it will load the Prolog source. When given a .qlf file, it will load the file. When no extension is specified, it will load the .qlf file when present and the .pl file otherwise.

Definition at line 253 of file PlReasoner.hpp.

#### 5.15.2.5 void argumatrix::PlReasoner::createPlAttackMatrix ( ) [protected]

Create the attack matrix with the form of PlTerm, which is an input of SWI-Prolog.

**Returns**

no return. The result is stored in member variable *m_PlAtkMtx*.

Definition at line 243 of file PlReasoner.hpp.

#### 5.15.2.6 void argumatrix::PlReasoner::fetchAllExts ( const std::string & *predct* )

Method: fetchAllExts FullName: public argumatrix::PlReasoner::fetchAllExts

**Parameters**

| | |
|---|---|
| *const* | std::string & predct |

**Returns**

void

**Return values**

| | |
|---|---|
| | |

Definition at line 701 of file PlReasoner.hpp.

#### 5.15.2.7 void argumatrix::PlReasoner::findAllExts ( const std::string & *predct* ) [protected]

Find all extensions and store them in *m_extensions* with the form of bitvector.

**Parameters**

| | |
|---|---|
| *string&* | predct: The predicate. |

**Returns**

no return. The results are added into *m_extensions*.

Definition at line 669 of file PlReasoner.hpp.

Referenced by argumatrix::AdmissiblePlReasoner::findAllExts(), argumatrix::StablePlReasoner::findAllExts(), argumatrix::ConflictfreePlReasoner::findAllExts(), and argumatrix::CompletePlReasoner::findAllExts().

#### 5.15.2.8 __inline argumatrix::bitvector argumatrix::Reasoner::getAttacked ( const bitvector & _bv ) [inherited]

Get the attacked arguments by arguments in _bv, $R^{+}(S)$ $R^{+}(S) = x|xisattackedbyS$. $R^{+}(S_bv) = D * S_bv$.

**Parameters**

| | |
|---|---|
| *_bv* | the bitvector with respect to the set $S$. |

**Returns**

a set of arguments in bitvector form.

Definition at line 360 of file Reasoner.hpp.

Referenced by argumatrix::Reasoner::getGroundedIntVector(), argumatrix::Reasoner::is_conflict_free(), and argumatrix::Reasoner::neutrality().

#### 5.15.2.9 __inline const set< bitvector > & argumatrix::Reasoner::getBvExtensions ( ) [inherited]

Get all extensions given a specific semantics in format of bitvector. Each extension is a set of arguments which can "survive the conflict together". Here, we use a bitvector to represent an extension. Therefore, all extensions are formed an set of bit vectors, i.e., set<bitvector>. Each bitvector in set is an extension w.r.t. a given semantics. The computed extensions are stored in m_extensions, therefore, to get all extensions, it must first invoke the function computeExtension()

**Returns**

a set of bitvector, i.e., set<bitvector>.

Definition at line 400 of file Reasoner.hpp.

#### 5.15.2.10 argumatrix::bitvector argumatrix::Reasoner::getGroundedExtension ( ) [inherited]

Get the grounded extension.

**Returns**

a set of arguments in bitvector form.

Definition at line 484 of file Reasoner.hpp.

Referenced by argumatrix::Reasoner::is_grounded(), and argumatrix::Reasoner::printGroundedExt().

#### 5.15.2.11 vector< int > argumatrix::Reasoner::getGroundedIntVector ( ) [inherited]

Get a vector of integers {0, 1, 2}: 2 – Unknown, 1 – in grounded extension, and 0 – attacked by the grounded extension.

**Returns**

a vector of integers {0, 1, 2}.

Definition at line 499 of file Reasoner.hpp.

#### 5.15.2.12 __inline argumatrix::bitvector argumatrix::Reasoner::getSelfAttackingArguments ( ) [inherited]

Get all arguments which are self-attacking. Obviously, if an argument i attacks itself, the entry[i][i] of the attack matrix must be 1 (true). Therefore to get the set of self-attacking arguments is to get the diagonal elements of the attack matrix.

**Returns**

a set of arguments in bitvector form, if bitvector[i] = true representing the argument (with index) i is a self-attacking argument, otherwise is not.

Definition at line 414 of file Reasoner.hpp.

#### 5.15.2.13 __inline bool argumatrix::Reasoner::is_acceptable ( const bitvector & *S*, const bitvector & *A* ) [inherited]

Argument set A is acceptable w.r.t S? Alternately, S defends A? A can be an argument or an argument set. S defends argument (set) A iff for any argument x in X, if attacks A (or a in A) then there is an argument y in S such that y attacks x.

**Parameters**

| | |
|---|---|
| *S* | a set of arguments. |
| *A* | an argument or an argument set |

**Returns**

true if S defends A.

Definition at line 394 of file Reasoner.hpp.

#### 5.15.2.14 __inline bool argumatrix::Reasoner::is_admissible ( const bitvector & *_bv* ) [inherited]

To decide whether a set of arguments (with the form of bitvector) is an admissible extension. $S$ is an admissible extension iff $S F(S) N(S)$.

**Returns**

true if _bv is admissible; otherwise returns false.

Definition at line 553 of file Reasoner.hpp.

#### 5.15.2.15 __inline bool argumatrix::Reasoner::is_complete ( const bitvector & *_bv* ) [inherited]

To decide whether a set of arguments (with the form of bitvector) is a complete extension. $S$ is a complete extension iff $S == F(S) N(S)$.

**Returns**

true if _bv is complete; otherwise returns false.

Definition at line 564 of file Reasoner.hpp.

#### 5.15.2.16 __inline bool argumatrix::Reasoner::is_conflict_free ( const bitvector & *_bv* ) [inherited]

Argument set S is conflict-free? S is said to be conflict-free iff for any two arguments a,b in S such that a does not attack b.

**Parameters**

| | |
|---|---|
| *extension* | an extension (a set of arguments). |

**Returns**

true if S is conflict-free.

Definition at line 381 of file Reasoner.hpp.

#### 5.15.2.17 __inline bool argumatrix::Reasoner::is_grounded ( const bitvector & *_bv* ) [inherited]

To decide whether a set of arguments (with the form of bitvector) is a grounded extension. $S$ is a grounded extension iff it is the least fixed point of the characteristic function F.

**Returns**

true if _bv is grounded; otherwise returns false.

Definition at line 584 of file Reasoner.hpp.

#### 5.15.2.18 __inline bool argumatrix::Reasoner::is_self_attacking ( size_type *idx* ) [inherited]

Argument a is self-attacking iff it attacks itself.

**Parameters**

| | |
|---|---|
| *idx* | the index of the argument w.r.t the attack matrix |

**Returns**

true if S defends A.

Definition at line 406 of file Reasoner.hpp.

#### 5.15.2.19 __inline bool argumatrix::Reasoner::is_stable ( const bitvector & *_bv* ) [inherited]

To decide whether a set of arguments (with the form of bitvector) is a stable extension. $S$ is a stable extension iff $S == N(S)$.

**Returns**

true if _bv is stable; otherwise returns false.

Definition at line 575 of file Reasoner.hpp.

#### 5.15.2.20 bool argumatrix::PIReasoner::isCredulouslyJustified ( const std::string & *predct,* const std::set< string > & *argset* )

Given an $AF = <X,R>$ and an argument s X (respectively, a set of arguments $S X$). Decide whether s contained (respectively, S included) in some E E_(AF) (i.e., credulously justified).

Problem [DC-$$]

Definition at line 642 of file PIReasoner.hpp.

#### 5.15.2.21 bool argumatrix::PlReasoner::isSkepticallyJustified ( const std::string & *predct,* const std::set< string > & *argset* )

Given an $AF = <X,R>$ and an argument s X (respectively, a set of arguments $S X$). Decide whether s contained (respectively, S included) in each E E_(AF) (i.e., skeptically justified).

Problem [DS-$$]

Definition at line 654 of file PlReasoner.hpp.

#### 5.15.2.22 vector< int > argumatrix::Reasoner::labelSet2IntVector ( const std::set< string > & *label_set* ) [inherited]

Convert a set of argument (string) labels to an integer vector of {0,1,2}, All arguments in these set, the indices is 1; otherwise 2 (representing unknown)

**Returns**

no return.

Definition at line 526 of file Reasoner.hpp.

#### 5.15.2.23 __inline argumatrix::bitvector argumatrix::Reasoner::neutrality ( const bitvector & *_bv* ) [inherited]

The neutrality function of an abstract argumentation framework: N_AF(S) = {A| All arguments that not attacked by S}. N_AF(S_bv) = not( R^+(S_bv) )

**Parameters**

| | |
|---|---|
| *extension* | an extension (a set of arguments). |

**Returns**

a set of arguments in bitvector form.

Definition at line 547 of file Reasoner.hpp.

Referenced by argumatrix::Reasoner::characteristic(), argumatrix::Reasoner::getGroundedIntVector(), argumatrix↩ ::Reasoner::is_admissible(), argumatrix::Reasoner::is_complete(), and argumatrix::Reasoner::is_stable().

#### 5.15.2.24 void argumatrix::PlReasoner::printAllExts ( const std::string & *predct* )

Print all extensions with respect to semantic predct.

Problem [EE-$$]

**Parameters**

| | |
|---|---|
| *no* | argument |

**Returns**

no return.

Definition at line 430 of file PlReasoner.hpp.

Referenced by printAllExts(), argumatrix::StablePlReasoner::task_EC(), argumatrix::ConflictfreePlReasoner↩︎::task_EC(), argumatrix::AdmissiblePlReasoner::task_EC(), argumatrix::CompletePlReasoner::task_EC(), argumatrix::ConflictfreePlReasoner::task_EE(), argumatrix::StablePlReasoner::task_EE(), argumatrix::Admissible↩︎PlReasoner::task_EE(), and argumatrix::CompletePlReasoner::task_EE().

#### 5.15.2.25 __inline void argumatrix::PlReasoner::printAllExts ( const std::string & *predct,* const std::set< string > & *argset* )

Given an ${AF}=< {X}, {R}>$ and an argument $s {X}$ (respectively, a set of arguments $S {X}$), enumerate all sets $E {X}$ such that $E {E}_(AF)$ and $s E$ (respectively, $S E$).

Problem [EC-$$]

**Parameters**

| | |
|---|---|
| *set<string>,or* | a Boolean vector: a set of argument |

**Returns**

no return.

Definition at line 386 of file PlReasoner.hpp.

#### 5.15.2.26 __inline void argumatrix::Reasoner::printBvExts ( ) [inherited]

Output all extensions given a specific semantics with an ostream. Each extension is a set of arguments which can "survive the conflict together". Here, we return a string to represent all extensions. For example, if there are two extensions, [a,b] and [b, d], for some problem w.r.t. a given semantics, then we return string "[[a,b],[d,c]]". If extensions is not existing, the string "[ ]" will return.

**Parameters**

| | |
|---|---|
| *ostream&* | os = std::cout, output the resluts into the ostream os. |

**Returns**

a set of bitvector, i.e., set<bitvector>.

Definition at line 428 of file Reasoner.hpp.

#### 5.15.2.27 __inline void argumatrix::Reasoner::printGroundedExt ( ) [inherited]

Print the grounded extension.

**Returns**

no return.

Definition at line 539 of file Reasoner.hpp.

#### 5.15.2.28 void argumatrix::PlReasoner::printLableExtByBlistTerm ( const PlTerm & *plt* ) [protected]

Print an extension with the form of the bool list term. Assume a bool list term is [0, 1, 0, 1], the arguments corresponding to entry 1 will be print. The member *m_output* will determine where to print.

**Parameters**

| | |
|---|---|
| *PlTerm&* | plt : The PlTerm is a bool list term, which represents an extension. |

**Returns**

no return.

Definition at line 271 of file PlReasoner.hpp.

#### 5.15.2.29 void argumatrix::PlReasoner::printLableExtByBmatrixTerm ( const PlTerm & *pmtx* ) [protected]

Print an extension with the form of the bool matrix term. Assume a bool list term is [[0, 1, 0, 1], [0, 1, 0, 1]], the arguments corresponding to entry 1 will be print. The member *m_output* will determine where to print.

**Parameters**

| | |
|---|---|
| *PlTerm&* | plt : The PlTerm is a bool list term, which represents an extension. |

**Returns**

no return.

Definition at line 725 of file PlReasoner.hpp.

#### 5.15.2.30 void argumatrix::Reasoner::printLabSet ( const bitvector & *bv_ext* ) [inherited]

Print an extension with the form of bitvector. Assume the bitvector is [0, 1, 0, 1], the arguments corresponding to entry 1 will be print. The member *m_output* will determine where to print.

**Parameters**

| | |
|---|---|
| *bitvector&* | bv_ext : The bv_ext is a bitvector, which represents an extension. |

**Returns**

no return.

**See also**

[setOutput]

Definition at line 433 of file Reasoner.hpp.

Referenced by argumatrix::Reasoner::printGroundedExt().

#### 5.15.2.31 void argumatrix::Reasoner::printLabSet ( const std::set< string > & *labset* ) [inherited]

Print a set of arguments. The member *m_output* will determine where to print.

**Parameters**

| | |
|---|---|
| *const* | std::set<string>& labset |

**Returns**

no return.

**See also**

setOutput(streambuf∗ strbuf = std::cout.rdbuf());

Definition at line 454 of file Reasoner.hpp.

#### 5.15.2.32 __inline void argumatrix::PlReasoner::printSomeExt ( const std::string & *predct,* const std::set< string > & *argset* )

Given an ${AF}=< {X}, {R}>$ and an argument $s {X}$ (respectively, a set of arguments $S {X}$), enumerate some set $E {X}$ such that $E {E}_(AF)$ and $s E$ (respectively, $S E$).

Problem [SC-$$]

**Parameters**

| | |
|---|---|
| *a* | set of arguments. |

**Returns**

no return.

Definition at line 468 of file PlReasoner.hpp.

Referenced by argumatrix::AdmissiblePlReasoner::task_SC(), argumatrix::StablePlReasoner::task_SC(), argumatrix↩ ::CompletePlReasoner::task_SC(), argumatrix::StablePlReasoner::task_SE(), and argumatrix::CompletePl↩ Reasoner::task_SE().

#### 5.15.2.33 __inline streambuf ∗ argumatrix::Reasoner::setOutput ( streambuf ∗ *strbuf* ) [inherited]

Redirect the output stream to streambuf∗ strbuf or ostream& os. If strbuf = cout.rdbuf(), then the output is standard output. It can also redirect the output to a file by the following codes:

```
ofstream ofile("output.txt");
streambuf* oldsb = xxx.setOutput(ofile.rdbuf());
```

or

```
streambuf* oldsb = xxx.setOutput(ofile);
```

.

**Parameters**

|  |  |
|---|---|
| *strbuf* | is a streambuf pointer. |

**Returns**

return the old streambuf, which can be used to redirect the old streambuf.

Definition at line 473 of file Reasoner.hpp.

#### 5.15.2.34 __inline streambuf ∗ argumatrix::Reasoner::setOutput ( ostream & *os* = `std::cout` ) `[inherited]`

Method: setOutput FullName: public argumatrix::Reasoner::setOutput

**See also**

streambuf∗ setOutput(streambuf∗ strbuf = std::cout.rdbuf());

**Parameters**

|  |  |
|---|---|
| *ostream* | & os |

**Returns**

streambuf∗

Definition at line 479 of file Reasoner.hpp.

#### 5.15.2.35 virtual void argumatrix::Reasoner::task_DE ( const std::set< string > & *argset* ) `[inline]`, `[virtual]`, `[inherited]`

Problem [DE-$$] Given an $AF = <X,R>$ and a set of arguments $S X$. Decide whether S is a -extension of AF, i.e., S E_(AF).

**Parameters**

|  |  |
|---|---|
| *a* | set of arguments. |

**Returns**

true if argset is a -extension; otherwise return false.

**Note**

For this task, the *set<string>& argset* can be empty, which means to decide whether the empty set is a $$-extension of AF. This indicates that the option -a is not necessary, then the default *set<string>& argset* is empty.

Reimplemented in argumatrix::CompletePlReasoner, argumatrix::ConflictfreePlReasoner, argumatrix::↵ AdmissiblePlReasoner, argumatrix::StablePlReasoner, and argumatrix::GroundedReasoner.

Definition at line 303 of file Reasoner.hpp.

#### 5.15.2.36 virtual void argumatrix::Reasoner::task_DN ( ) [inline], [virtual], [inherited]

Problem [DN-$$] Given an $AF = <X,R>$ and a set of arguments $S X$. Decide whether there exist a non-empty -extension for AF.

**Returns**

true if there exist a non-empty -extension; otherwise return false.

Reimplemented in argumatrix::CompletePlReasoner, argumatrix::ConflictfreePlReasoner, argumatrix::↩ AdmissiblePlReasoner, argumatrix::StablePlReasoner, and argumatrix::GroundedReasoner.

Definition at line 313 of file Reasoner.hpp.

#### 5.15.2.37 virtual void argumatrix::Reasoner::task_EC ( const std::set< string > & *argset* ) [inline], [virtual], [inherited]

Problem [EC-$$] Given an ${AF}=< {X}, {R}>$ and an argument $s {X}$ (respectively, a set of arguments $S {X}$), enumerate all sets $E {X}$ such that $E {E}_(AF)$ and $s E$ (respectively, $S E$).

**Parameters**

| | |
|---|---|
| *set<string>,or* | a Boolean vector: a set of argument |

**Returns**

no return.

Reimplemented in argumatrix::CompletePlReasoner, argumatrix::AdmissiblePlReasoner, argumatrix::↩ ConflictfreePlReasoner, argumatrix::StablePlReasoner, and argumatrix::GroundedReasoner.

Definition at line 267 of file Reasoner.hpp.

#### 5.15.2.38 virtual void argumatrix::Reasoner::task_EE ( ) [inline], [virtual], [inherited]

Problem [EE-$$] Print all extensions.

**Parameters**

| | |
|---|---|
| *no* | argument |

**Returns**

no return.

Reimplemented in argumatrix::CompletePlReasoner, argumatrix::AdmissiblePlReasoner, argumatrix::↩ ConflictfreePlReasoner, argumatrix::StablePlReasoner, and argumatrix::GroundedReasoner.

Definition at line 254 of file Reasoner.hpp.

#### 5.15.2.39 virtual void argumatrix::Reasoner::task_SC ( const std::set< string > & *argset* ) [inline], [virtual], [inherited]

Problem [SC-$$] Given an ${AF}=< {X}, {R}>$ and an argument $s {X}$ (respectively, a set of arguments $S {X}$), enumerate some set $E {X}$ such that $E {E}_(AF)$ and $s E$ (respectively, $S E$).

**Parameters**

| | |
|---|---|
| *a* | set of arguments. |

**Returns**

no return.

Reimplemented in argumatrix::CompletePlReasoner, argumatrix::AdmissiblePlReasoner, argumatrix::↩ ConflictfreePlReasoner, argumatrix::StablePlReasoner, and argumatrix::GroundedReasoner.

Definition at line 280 of file Reasoner.hpp.

#### 5.15.2.40 virtual void argumatrix::Reasoner::task_SE ( ) [inline], [virtual], [inherited]

Problem [SE-$$] Given an ${AF}=< {X}, {R}>$, enumerate some set $E {X}$ that are in ${E}_(AF)$.

**Parameters**

| | |
|---|---|
| *a* | set of arguments. |

**Returns**

no return.

Reimplemented in argumatrix::CompletePlReasoner, argumatrix::AdmissiblePlReasoner, argumatrix::↩ ConflictfreePlReasoner, argumatrix::StablePlReasoner, and argumatrix::GroundedReasoner.

Definition at line 290 of file Reasoner.hpp.

#### 5.15.2.41 bool argumatrix::PlReasoner::verifyExclusion ( const std::string & *predct,* const bitvector & *vecB* ) [protected]

Given an $AF = <X,R>$ and a set of arguments $S X$. Decide whether there exist a -extension that excludes S.

**Parameters**

| | |
|---|---|
| *a* | set of arguments. |

**Returns**

true if there exist a -extension that excludes S; otherwise return false.

Definition at line 576 of file PlReasoner.hpp.

#### 5.15.2.42 __inline bool argumatrix::PlReasoner::verifyExtension ( const std::string & *predct,* std::set< string > & *argset* )

Given an $AF = <X,R>$ and a set of arguments $S X$. Decide whether S is a $$-extension of AF, i.e., $S E_(AF)$.

Problem [DE-$$]

**Parameters**

| | |
|---|---|
| *a* | set of arguments. |

**Returns**

true if argset is a -extension; otherwise return false.

Definition at line 609 of file PlReasoner.hpp.

#### 5.15.2.43 bool argumatrix::PlReasoner::verifyInclusion ( const std::string & *predct,* const vector< int > & *vecii* ) [protected]

Given an $AF = <X,R>$ and a set of arguments $S X$. Decide whether there exist a -extension that includes S.

**Parameters**

| | |
|---|---|
| *a* | set of arguments. |

**Returns**

true if there exist a -extension that includes S; otherwise return false.

Definition at line 543 of file PlReasoner.hpp.

#### 5.15.2.44 bool argumatrix::PlReasoner::verifyNonemptyExt ( const std::string & *predct* )

Given an $AF = <X,R>$ and a set of arguments $S X$. Decide whether there exist a non-empty $$-extension for AF.

Problem [DN-$$]

**Returns**

true if there exist a non-empty -extension; otherwise return false.

Definition at line 510 of file PlReasoner.hpp.

### 5.15.3 Member Data Documentation

#### 5.15.3.1 vector< std::string > argumatrix::Reasoner::m_argLabels [protected], [inherited]

Dung's abstract argumentation framework

Definition at line 346 of file Reasoner.hpp.

Referenced by argumatrix::Reasoner::printLabSet().

#### 5.15.3.2 size_type argumatrix::Reasoner::m_argNum [protected], [inherited]

The number of arguments

Definition at line 339 of file Reasoner.hpp.

Referenced by argumatrix::Reasoner::characteristic(), argumatrix::Reasoner::getGroundedExtension(), argumatrix↩ ::Reasoner::getGroundedIntVector(), and argumatrix::Reasoner::labelSet2IntVector().

#### 5.15.3.3 bitmatrix argumatrix::Reasoner::m_BmAtkMtx [protected], [inherited]

The bitmatrix of the Dung Abstract argumentation framework. We can access all attackers of an argument. The attackers of the argument with index i is m_BmAtkMtx[i].

Definition at line 334 of file Reasoner.hpp.

Referenced by argumatrix::Reasoner::getAttacked(), argumatrix::Reasoner::getSelfAttackingArguments(), and argumatrix::Reasoner::is_self_attacking().

#### 5.15.3.4 const DungAF& argumatrix::Reasoner::m_daf [protected], [inherited]

Dung's abstract argumentation framework

Definition at line 342 of file Reasoner.hpp.

Referenced by argumatrix::Reasoner::is_conflict_free(), argumatrix::Reasoner::labelSet2IntVector(), and argumatrix::Reasoner::printBvExts().

#### 5.15.3.5 std::ostream argumatrix::Reasoner::m_output [protected], [inherited]

Where to output

Definition at line 348 of file Reasoner.hpp.

Referenced by argumatrix::Reasoner::printBvExts(), argumatrix::Reasoner::printGroundedExt(), argumatrix::↩ Reasoner::printLabSet(), and argumatrix::Reasoner::setOutput().

The documentation for this class was generated from the following file:

- F:/Codespace/BoostProject/argumatrix/PlReasoner/PlReasoner.hpp

## 5.16 argumatrix::PlTimeQry Class Reference

Create a *time* query which can output the time information.

```
#include <swipl-util.hpp>
```

#### Public Member Functions

- PlTimeQry (const char ∗name, const PlTermv &av)

  *Create a time query where name defines the name of the predicate and av the argument vector. The arity is deduced from av.size The predicate is located in the Prolog module user.*
- PlTimeQry (const char ∗module, const char ∗name, const PlTermv &av)

  *Same, but performs the time query lookup in the indicated module.*
- int next_solution ()

  *Generate the first (next) solution for the given query. The return value is TRUE if a solution was found, or FALSE to indicate the query could not be proven. This function may be called repeatedly until it fails to generate all solutions to the query.*

#### Public Attributes

- qid_t **qid**

### 5.16.1 Detailed Description

Create a *time* query which can output the time information.

#### Date

May 2016

Definition at line 180 of file swipl-util.hpp.

### 5.16.2 Constructor & Destructor Documentation

#### 5.16.2.1 argumatrix::PlTimeQry::PlTimeQry ( const char ∗ *name,* const PlTermv & *av* ) [inline]

Create a *time* query where name defines the name of the predicate and av the argument vector. The arity is deduced from av.size The predicate is located in the Prolog module user.

Method: Constructor: PlTimeQry

#### Parameters

| | |
|---|---|
| *const* | char ∗ name |
| *const* | PlTermv & av |

#### Note

A Time Query is a query to measure the time of this query. It is equivalence to the query: time(Goal), in which the Goal is the real body of the query and is a compound term PlCompound(name, av).

Definition at line 196 of file swipl-util.hpp.

#### 5.16.2.2 argumatrix::PlTimeQry::PlTimeQry ( const char ∗ *module*, const char ∗ *name*, const PlTermv & *av* ) [inline]

Same, but performs the time query lookup in the indicated module.

Method: Constructor: PlTimeQry

**Parameters**

| | |
|---|---|
| *const* | char ∗ module |
| *const* | char ∗ name |
| *const* | PlTermv & av |

Definition at line 210 of file swipl-util.hpp.

### 5.16.3 Member Function Documentation

#### 5.16.3.1 __inline int argumatrix::PlTimeQry::next_solution ( )

Generate the first (next) solution for the given query. The return value is TRUE if a solution was found, or FALSE to indicate the query could not be proven. This function may be called repeatedly until it fails to generate all solutions to the query.

Method: next_solution FullName: public argumatrix::PlTimeQry::next_solution

**Returns**

int

Definition at line 261 of file swipl-util.hpp.

Referenced by argumatrix::PlReasoner::addBlTermToBvExts(), argumatrix::PlReasoner::fetchAllExts(), argumatrix↩ ::PlReasoner::findAllExts(), argumatrix::PlReasoner::printAllExts(), argumatrix::PlReasoner::printSomeExt(), argumatrix::PlReasoner::verifyExclusion(), argumatrix::PlReasoner::verifyInclusion(), and argumatrix::Pl↩ Reasoner::verifyNonemptyExt().

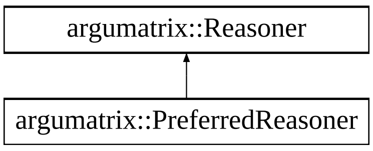
The documentation for this class was generated from the following file:

- F:/Codespace/BoostProject/argumatrix/PlReasoner/swipl-util.hpp

## 5.17 argumatrix::PreferredReasoner Class Reference

```
#include <PreferredReasoner.hpp>
```

Inheritance diagram for argumatrix::PreferredReasoner:



## Public Member Functions

- **PreferredReasoner** (const DungAF &daf, streambuf ∗osbuff=std::cout.rdbuf())
- void computeExtensions ()
- bitvector getAttackers (const bitvector &_bv)
- bitvector getAttacked (const bitvector &_bv)

  *Get the attacked arguments by arguments in _bv,* $R^{+}(S)$ $R^{+}(S) = x|xisattackedbyS.$ $R^{+}(S_bv) = D * S_bv.$
- bitvector characteristic (const bitvector &_bv)

  *The characteristic function of an abstract argumentation framework:* $F_{AF}(S) = A|Aisacceptablewrt.S.$ *F_AF(*←↩ *S_bv) = not{ R^+[ not( R^+(S_bv) ) ] }.*
- bitvector characteristic ()

  *The characteristic function of an abstract argumentation framework with the initialization of empty set.*
- bitvector neutrality (const bitvector &_bv)

  *The neutrality function of an abstract argumentation framework: N_AF(S) = {A| All arguments that not attacked by S}. N_AF(S_bv) = not( R^+(S_bv) )*
- bool is_conflict_free (const bitvector &_bv)

  *Argument set S is conflict-free? S is said to be conflict-free iff for any two arguments a,b in S such that a does not attack b.*
- bool **is_conflict_free** (const set< string > &argset)
- bool is_acceptable (const bitvector &S, const bitvector &A)

  *Argument set A is acceptable w.r.t S? Alternately, S defends A? A can be an argument or an argument set. S defends argument (set) A iff for any argument x in X, if attacks A (or a in A) then there is an argument y in S such that y attacks x.*
- bool is_self_attacking (size_type idx)

  *Argument a is self-attacking iff it attacks itself.*
- bool is_admissible (const bitvector &_bv)

  *To decide whether a set of arguments (with the form of bitvector) is an admissible extension. $S$ is an admissible extension iff $S F(S) N(S)$.*
- bool is_complete (const bitvector &_bv)

  *To decide whether a set of arguments (with the form of bitvector) is a complete extension. $S$ is a complete extension iff $S == F(S) N(S)$.*
- bool is_stable (const bitvector &_bv)

  *To decide whether a set of arguments (with the form of bitvector) is a stable extension. $S$ is a stable extension iff $S == N(S)$.*
- bool is_grounded (const bitvector &_bv)

  *To decide whether a set of arguments (with the form of bitvector) is a grounded extension. $S$ is a grounded extension iff it is the least fixed point of the characteristic function F.*
- bitvector getSelfAttackingArguments ()

  *Get all arguments which are self-attacking. Obviously, if an argument i attacks itself, the entry[i][i] of the attack matrix must be 1 (true). Therefore to get the set of self-attacking arguments is to get the diagonal elements of the attack matrix.*
- const set< bitvector > & getBvExtensions ()

  *Get all extensions given a specific semantics in format of bitvector. Each extension is a set of arguments which can "survive the conflict together". Here, we use a bitvector to represent an extension. Therefore, all extensions are formed an set of bit vectors, i.e., set<bitvector>. Each bitvector in set is an extension w.r.t. a given semantics. The computed extensions are stored in m_extensions, therefore, to get all extensions, it must first invoke the function computeExtension()*
- vector< int > getGroundedIntVector ()

  *Get a vector of integers {0, 1, 2}: 2 – Unknown, 1 – in grounded extension, and 0 – attacked by the grounded extension.*
- streambuf ∗ setOutput (streambuf ∗strbuf)

  *Redirect the output stream to streambuf∗ strbuf or ostream& os. If strbuf = cout.rdbuf(), then the output is standard output. It can also redirect the output to a file by the following codes:*
  ```
  ofstream ofile("output.txt");
  streambuf* oldsb = xxx.setOutput(ofile.rdbuf());
  ```
  *or*

```
streambuf* oldsb = xxx.setOutput(ofile);
```

.

- streambuf $*$ setOutput (ostream &os=std::cout)
- void printLabSet (const bitvector &bv_ext)

  *Print an extension with the form of bitvector. Assume the bitvector is [0, 1, 0, 1], the arguments corresponding to entry 1 will be print. The member m_output will determine where to print.*
- void printLabSet (const std::set$<$ string $>$ &labset)

  *Print a set of arguments. The member m_output will determine where to print.*
- void printBvExts ()

  *Output all extensions given a specific semantics with an ostream. Each extension is a set of arguments which can "survive the conflict together". Here, we return a string to represent all extensions. For example, if there are two extensions, [a,b] and [b, d], for some problem w.r.t. a given semantics, then we return string "[[a,b],[d,c]]". If extensions is not existing, the string "[ ]" will return.*
- bitvector getGroundedExtension ()

  *Get the grounded extension.*
- void printGroundedExt ()

  *Print the grounded extension.*
- vector$<$ int $>$ labelSet2IntVector (const std::set$<$ string $>$ &label_set)

  *Convert a set of argument (string) labels to an integer vector of {0,1,2}, All arguments in these set, the indices is 1; otherwise 2 (representing unknown)*
- virtual void task_EE ()

  *Problem [EE-$$] Print all extensions.*
- virtual void **task_EX** ()
- virtual void task_EC (const std::set$<$ string $>$ &argset)

  *Problem [EC-$$] Given an ${AF}=< {X}, {R}>$ and an argument $s {X}$ (respectively, a set of arguments $S {X}$), enumerate all sets $E {X}$ such that $E {E}_(AF)$ and $s E$ (respectively, $S E$).*
- virtual void task_SC (const std::set$<$ string $>$ &argset)

  *Problem [SC-$$] Given an ${AF}=< {X}, {R}>$ and an argument $s {X}$ (respectively, a set of arguments $S {X}$), enumerate some set $E {X}$ such that $E {E}_(AF)$ and $s E$ (respectively, $S E$).*
- virtual void task_SE ()

  *Problem [SE-$$] Given an ${AF}=< {X}, {R}>$, enumerate some set $E {X}$ that are in ${E}_(AF)$.*
- virtual void task_DE (const std::set$<$ string $>$ &argset)

  *Problem [DE-$$] Given an $AF = <X,R>$ and a set of arguments $S X$. Decide whether S is a -extension of AF, i.e., S E_(AF).*
- virtual void task_DN ()

  *Problem [DN-$$] Given an $AF = <X,R>$ and a set of arguments $S X$. Decide whether there exist a non-empty -extension for AF.*
- virtual void task_DC (const std::set$<$ string $>$ &argset)

  *Problem [DC-$$] Given an $AF = <X,R>$ and an argument s X (respectively, a set of arguments $S X$). Decide whether s contained (respectively, S included) in some E E_(AF) (i.e., credulously justified).*
- virtual void task_DS (const std::set$<$ string $>$ &argset)

  *Problem [DS-$$] Given an $AF = <X,R>$ and an argument s X (respectively, a set of arguments $S X$). Decide whether s contained (respectively, S included) in each E E_(AF) (i.e., skeptically justified).*

## Protected Attributes

- bitmatrix m_BmAtkMtx
- size_type m_argNum
- const DungAF & m_daf
- set$<$ bitvector $>$ **m_extensions**
- vector$<$ std::string $>$ m_argLabels
- std::ostream m_output

### 5.17.1 Detailed Description

This reasoner for Dung theories performs inference on the preferred extension. Computes the (unique) grounded extension, i.e., the least fixpoint of the characteristic function.

Definition at line 38 of file PreferredReasoner.hpp.

### 5.17.2 Member Function Documentation

#### 5.17.2.1 __inline **argumatrix::bitvector** argumatrix::Reasoner::characteristic ( const **bitvector** & *_bv* ) [inherited]

The characteristic function of an abstract argumentation framework: $F_{AF}(S) = A|Aisacceptablewrt.S.$ F_AF(↩ S_bv) = not{ R^+[ not( R^+(S_bv) ) ] }.

**Parameters**

| | |
|---|---|
| *extension* | an extension (a set of arguments), the default is empty set. |

**Returns**

a set of arguments in bitvector form.

Definition at line 368 of file Reasoner.hpp.

#### 5.17.2.2 __inline **argumatrix::bitvector** argumatrix::Reasoner::characteristic ( ) [inherited]

The characteristic function of an abstract argumentation framework with the initialization of empty set.

**Returns**

a set of arguments in bitvector form.

Definition at line 375 of file Reasoner.hpp.

Referenced by argumatrix::Reasoner::getGroundedExtension(), and argumatrix::Reasoner::is_acceptable().

#### 5.17.2.3 void argumatrix::PreferredReasoner::computeExtensions ( )

Compute all extensions given a specific semantics. Each extension is a set of arguments. Here, we use a bitvector to represent an extension. The results are stored in m_extensions. When a new extension bv is computed, it can be added to m_extensions by m_extensions.push_back(bv).

**Returns**

no return. The results are stored in m_extensions.

Definition at line 97 of file PreferredReasoner.hpp.

#### 5.17.2.4 __inline **argumatrix::bitvector** argumatrix::Reasoner::getAttacked ( const **bitvector** & *_bv* ) [inherited]

Get the attacked arguments by arguments in _bv, $R^+(S)$ $R^+(S) = x|xisattackedbyS.$ $R^+(S_bv) = D * S_bv.$

**Parameters**

| | |
|---|---|
| *_bv* | the bitvector with respect to the set $S$. |

**Returns**

a set of arguments in bitvector form.

Definition at line 360 of file Reasoner.hpp.

Referenced by argumatrix::Reasoner::getGroundedIntVector(), argumatrix::Reasoner::is_conflict_free(), and argumatrix::Reasoner::neutrality().

#### 5.17.2.5 bitvector argumatrix::PreferredReasoner::getAttackers ( const bitvector & *_bv* )

Get attackers of arguments in _bv, R^-(S) R^-(S) = {x|x attacks some argument in S}. R^-(S_bv) = D^T∗S_bv

**Parameters**

| | |
|---|---|
| *_bv* | the bitvector of a set of arguments S. |

**Returns**

a set of arguments in bitvector form.

Definition at line 246 of file PreferredReasoner.hpp.

#### 5.17.2.6 __inline const set< bitvector > & argumatrix::Reasoner::getBvExtensions ( ) [inherited]

Get all extensions given a specific semantics in format of bitvector. Each extension is a set of arguments which can "survive the conflict together". Here, we use a bitvector to represent an extension. Therefore, all extensions are formed an set of bit vectors, i.e., set<bitvector>. Each bitvector in set is an extension w.r.t. a given semantics. The computed extensions are stored in m_extensions, therefore, to get all extensions, it must first invoke the function computeExtension()

**Returns**

a set of bitvector, i.e., set<bitvector>.

Definition at line 400 of file Reasoner.hpp.

#### 5.17.2.7 argumatrix::bitvector argumatrix::Reasoner::getGroundedExtension ( ) [inherited]

Get the grounded extension.

**Returns**

a set of arguments in bitvector form.

Definition at line 484 of file Reasoner.hpp.

Referenced by argumatrix::Reasoner::is_grounded(), and argumatrix::Reasoner::printGroundedExt().

#### 5.17.2.8 vector< int > argumatrix::Reasoner::getGroundedIntVector ( ) [inherited]

Get a vector of integers {0, 1, 2}: 2 – Unknown, 1 – in grounded extension, and 0 – attacked by the grounded extension.

**Returns**

a vector of integers {0, 1, 2}.

Definition at line 499 of file Reasoner.hpp.

#### 5.17.2.9 __inline argumatrix::bitvector argumatrix::Reasoner::getSelfAttackingArguments ( ) [inherited]

Get all arguments which are self-attacking. Obviously, if an argument i attacks itself, the entry[i][i] of the attack matrix must be 1 (true). Therefore to get the set of self-attacking arguments is to get the diagonal elements of the attack matrix.

**Returns**

a set of arguments in bitvector form, if bitvector[i] = true representing the argument (with index) i is a self-attacking argument, otherwise is not.

Definition at line 414 of file Reasoner.hpp.

#### 5.17.2.10 __inline bool argumatrix::Reasoner::is_acceptable ( const bitvector & *S*, const bitvector & *A* ) [inherited]

Argument set A is acceptable w.r.t S? Alternately, S defends A? A can be an argument or an argument set. S defends argument (set) A iff for any argument x in X, if attacks A (or a in A) then there is an argument y in S such that y attacks x.

**Parameters**

| | |
|---|---|
| *S* | a set of arguments. |
| *A* | an argument or an argument set |

**Returns**

true if S defends A.

Definition at line 394 of file Reasoner.hpp.

#### 5.17.2.11 __inline bool argumatrix::Reasoner::is_admissible ( const bitvector & *_bv* ) [inherited]

To decide whether a set of arguments (with the form of bitvector) is an admissible extension. $S$ is an admissible extension iff $S F(S) N(S)$.

**Returns**

true if _bv is admissible; otherwise returns false.

Definition at line 553 of file Reasoner.hpp.

#### 5.17.2.12 __inline bool argumatrix::Reasoner::is_complete ( const bitvector & *_bv* ) [inherited]

To decide whether a set of arguments (with the form of bitvector) is a complete extension. $S$ is a complete extension iff $S == F(S) N(S)$.

**Returns**

true if _bv is complete; otherwise returns false.

Definition at line 564 of file Reasoner.hpp.

#### 5.17.2.13 __inline bool argumatrix::Reasoner::is_conflict_free ( const bitvector & *_bv* ) [inherited]

Argument set S is conflict-free? S is said to be conflict-free iff for any two arguments a,b in S such that a does not attack b.

**Parameters**

| | |
|---|---|
| *extension* | an extension (a set of arguments). |

**Returns**

true if S is conflict-free.

Definition at line 381 of file Reasoner.hpp.

#### 5.17.2.14 __inline bool argumatrix::Reasoner::is_grounded ( const bitvector & *_bv* ) [inherited]

To decide whether a set of arguments (with the form of bitvector) is a grounded extension. $S$ is a grounded extension iff it is the least fixed point of the characteristic function F.

**Returns**

true if _bv is grounded; otherwise returns false.

Definition at line 584 of file Reasoner.hpp.

#### 5.17.2.15 __inline bool argumatrix::Reasoner::is_self_attacking ( size_type *idx* ) [inherited]

Argument a is self-attacking iff it attacks itself.

**Parameters**

| | |
|---|---|
| *idx* | the index of the argument w.r.t the attack matrix |

**Returns**

true if S defends A.

Definition at line 406 of file Reasoner.hpp.

#### 5.17.2.16 __inline bool argumatrix::Reasoner::is_stable ( const bitvector & *_bv* ) [inherited]

To decide whether a set of arguments (with the form of bitvector) is a stable extension. $S$ is a stable extension iff $S == N(S)$.

**Returns**

true if _bv is stable; otherwise returns false.

Definition at line 575 of file Reasoner.hpp.

#### 5.17.2.17 vector< int > argumatrix::Reasoner::labelSet2IntVector ( const std::set< string > & *label_set* ) [inherited]

Convert a set of argument (string) labels to an integer vector of {0,1,2}, All arguments in these set, the indices is 1; otherwise 2 (representing unknown)

**Returns**

no return.

Definition at line 526 of file Reasoner.hpp.

#### 5.17.2.18 __inline argumatrix::bitvector argumatrix::Reasoner::neutrality ( const bitvector & *_bv* ) [inherited]

The neutrality function of an abstract argumentation framework: N_AF(S) = {A| All arguments that not attacked by S}. N_AF(S_bv) = not( R^+(S_bv) )

**Parameters**

| | |
|---|---|
| *extension* | an extension (a set of arguments). |

**Returns**

a set of arguments in bitvector form.

Definition at line 547 of file Reasoner.hpp.

Referenced by argumatrix::Reasoner::characteristic(), argumatrix::Reasoner::getGroundedIntVector(), argumatrix↩ ::Reasoner::is_admissible(), argumatrix::Reasoner::is_complete(), and argumatrix::Reasoner::is_stable().

#### 5.17.2.19 __inline void argumatrix::Reasoner::printBvExts ( ) [inherited]

Output all extensions given a specific semantics with an ostream. Each extension is a set of arguments which can "survive the conflict together". Here, we return a string to represent all extensions. For example, if there are two extensions, [a,b] and [b, d], for some problem w.r.t. a given semantics, then we return string "[[a,b],[d,c]]". If extensions is not existing, the string "[ ]" will return.

**Parameters**

| | |
|---|---|
| *ostream&* | os = std::cout, output the resluts into the ostream os. |

**Returns**

a set of bitvector, i.e., set<bitvector>.

Definition at line 428 of file Reasoner.hpp.

#### 5.17.2.20 __inline void argumatrix::Reasoner::printGroundedExt ( ) [inherited]

Print the grounded extension.

**Returns**

no return.

Definition at line 539 of file Reasoner.hpp.

#### 5.17.2.21 void argumatrix::Reasoner::printLabSet ( const bitvector & *bv_ext* ) [inherited]

Print an extension with the form of bitvector. Assume the bitvector is [0, 1, 0, 1], the arguments corresponding to entry 1 will be print. The member *m_output* will determine where to print.

**Parameters**

| | |
|---|---|
| *bitvector&* | bv_ext : The bv_ext is a bitvector, which represents an extension. |

**Returns**

no return.

**See also**

[setOutput]

Definition at line 433 of file Reasoner.hpp.

Referenced by argumatrix::Reasoner::printGroundedExt().

#### 5.17.2.22 void argumatrix::Reasoner::printLabSet ( const std::set< string > & *labset* ) [inherited]

Print a set of arguments. The member *m_output* will determine where to print.

**Parameters**

| | |
|---|---|
| *const* | std::set<string>& labset |

**Returns**

no return.

**See also**

setOutput(streambuf∗ strbuf = std::cout.rdbuf());

Definition at line 454 of file Reasoner.hpp.

#### 5.17.2.23 __inline streambuf ∗ argumatrix::Reasoner::setOutput ( streambuf ∗ *strbuf* ) [inherited]

Redirect the output stream to streambuf∗ strbuf or ostream& os. If strbuf = cout.rdbuf(), then the output is standard output. It can also redirect the output to a file by the following codes:

```
ofstream ofile("output.txt");
streambuf* oldsb = xxx.setOutput(ofile.rdbuf());
```

or

```
streambuf* oldsb = xxx.setOutput(ofile);
```

.

**Parameters**

| | |
|---|---|
| *strbuf* | is a streambuf pointer. |

**Returns**

return the old streambuf, which can be used to redirect the old streambuf.

Definition at line 473 of file Reasoner.hpp.

#### 5.17.2.24 __inline streambuf ∗ argumatrix::Reasoner::setOutput ( ostream & *os* = std::cout ) [inherited]

Method: setOutput FullName: public argumatrix::Reasoner::setOutput

**See also**

streambuf∗ setOutput(streambuf∗ strbuf = std::cout.rdbuf());

**Parameters**

| | |
|---|---|
| *ostream* | & os |

**Returns**

streambuf∗

Definition at line 479 of file Reasoner.hpp.

#### 5.17.2.25 virtual void argumatrix::Reasoner::task_DE ( const std::set< string > & *argset* ) [inline], [virtual], [inherited]

Problem [DE-$$] Given an $AF = <X,R>$ and a set of arguments $S X$. Decide whether S is a -extension of AF, i.e., S E_(AF).

**Parameters**

| | |
|---|---|
| *a* | set of arguments. |

**Returns**

true if argset is a -extension; otherwise return false.

**Note**

For this task, the *set<string>& argset* can be empty, which means to decide whether the empty set is a $$-extension of AF. This indicates that the option -a is not necessary, then the default *set<string>& argset* is empty.

Reimplemented in argumatrix::CompletePlReasoner, argumatrix::ConflictfreePlReasoner, argumatrix::↩ AdmissiblePlReasoner, argumatrix::StablePlReasoner, and argumatrix::GroundedReasoner.

Definition at line 303 of file Reasoner.hpp.

#### 5.17.2.26 virtual void argumatrix::Reasoner::task_DN ( ) [inline], [virtual], [inherited]

Problem [DN-$$] Given an $AF = <X,R>$ and a set of arguments $S X$. Decide whether there exist a non-empty -extension for AF.

**Returns**

true if there exist a non-empty -extension; otherwise return false.

Reimplemented in argumatrix::CompletePlReasoner, argumatrix::ConflictfreePlReasoner, argumatrix::↩ AdmissiblePlReasoner, argumatrix::StablePlReasoner, and argumatrix::GroundedReasoner.

Definition at line 313 of file Reasoner.hpp.

#### 5.17.2.27 virtual void argumatrix::Reasoner::task_EC ( const std::set< string > & *argset* ) [inline], [virtual], [inherited]

Problem [EC-$$] Given an ${AF}=< {X}, {R}>$ and an argument $s {X}$ (respectively, a set of arguments $S {X}$), enumerate all sets $E {X}$ such that $E {E}_(AF)$ and $s E$ (respectively, $S E$).

**Parameters**

| | |
|---|---|
| *set<string>,or* | a Boolean vector: a set of argument |

**Returns**

no return.

Reimplemented in argumatrix::CompletePlReasoner, argumatrix::AdmissiblePlReasoner, argumatrix::↩ ConflictfreePlReasoner, argumatrix::StablePlReasoner, and argumatrix::GroundedReasoner.

Definition at line 267 of file Reasoner.hpp.

#### 5.17.2.28 virtual void argumatrix::Reasoner::task_EE ( ) `[inline]`, `[virtual]`, `[inherited]`

Problem [EE-$$] Print all extensions.

**Parameters**

| | |
|---|---|
| *no* | argument |

**Returns**

no return.

Reimplemented in argumatrix::CompletePlReasoner, argumatrix::AdmissiblePlReasoner, argumatrix::↩ ConflictfreePlReasoner, argumatrix::StablePlReasoner, and argumatrix::GroundedReasoner.

Definition at line 254 of file Reasoner.hpp.

#### 5.17.2.29 virtual void argumatrix::Reasoner::task_SC ( const std::set< string > & *argset* ) `[inline]`, `[virtual]`, `[inherited]`

Problem [SC-$$] Given an ${AF}=< {X}, {R}>$ and an argument $s {X}$ (respectively, a set of arguments $S {X}$), enumerate some set $E {X}$ such that $E {E}_(AF)$ and $s E$ (respectively, $S E$).

**Parameters**

| | |
|---|---|
| *a* | set of arguments. |

**Returns**

no return.

Reimplemented in argumatrix::CompletePlReasoner, argumatrix::AdmissiblePlReasoner, argumatrix::↩ ConflictfreePlReasoner, argumatrix::StablePlReasoner, and argumatrix::GroundedReasoner.

Definition at line 280 of file Reasoner.hpp.

#### 5.17.2.30 virtual void argumatrix::Reasoner::task_SE ( ) `[inline]`, `[virtual]`, `[inherited]`

Problem [SE-$$] Given an ${AF}=< {X}, {R}>$, enumerate some set $E {X}$ that are in ${E}_(AF)$.

**Parameters**

|   |   |
|---|---|
| *a* | set of arguments. |

**Returns**

no return.

Reimplemented in argumatrix::CompletePlReasoner, argumatrix::AdmissiblePlReasoner, argumatrix::↩ ConflictfreePlReasoner, argumatrix::StablePlReasoner, and argumatrix::GroundedReasoner.

Definition at line 290 of file Reasoner.hpp.

### 5.17.3 Member Data Documentation

#### 5.17.3.1 vector< std::string > argumatrix::Reasoner::m_argLabels [protected], [inherited]

Dung's abstract argumentation framework

Definition at line 346 of file Reasoner.hpp.

Referenced by argumatrix::Reasoner::printLabSet().

#### 5.17.3.2 size_type argumatrix::Reasoner::m_argNum [protected], [inherited]

The number of arguments

Definition at line 339 of file Reasoner.hpp.

Referenced by argumatrix::Reasoner::characteristic(), argumatrix::Reasoner::getGroundedExtension(), argumatrix↩ ::Reasoner::getGroundedIntVector(), and argumatrix::Reasoner::labelSet2IntVector().

#### 5.17.3.3 bitmatrix argumatrix::Reasoner::m_BmAtkMtx [protected], [inherited]

The bitmatrix of the Dung Abstract argumentation framework. We can access all attackers of an argument. The attackers of the argument with index i is m_BmAtkMtx[i].

Definition at line 334 of file Reasoner.hpp.

Referenced by argumatrix::Reasoner::getAttacked(), argumatrix::Reasoner::getSelfAttackingArguments(), and argumatrix::Reasoner::is_self_attacking().

#### 5.17.3.4 const DungAF& argumatrix::Reasoner::m_daf [protected], [inherited]

Dung's abstract argumentation framework

Definition at line 342 of file Reasoner.hpp.

Referenced by argumatrix::Reasoner::is_conflict_free(), argumatrix::Reasoner::labelSet2IntVector(), and argumatrix::Reasoner::printBvExts().

#### 5.17.3.5 std::ostream argumatrix::Reasoner::m_output [protected], [inherited]

Where to output

Definition at line 348 of file Reasoner.hpp.

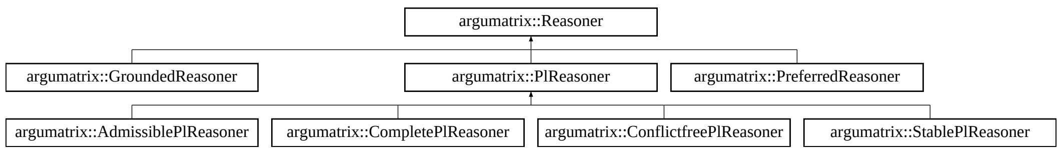
Referenced by argumatrix::Reasoner::printBvExts(), argumatrix::Reasoner::printGroundedExt(), argumatrix::↩ Reasoner::printLabSet(), and argumatrix::Reasoner::setOutput().

The documentation for this class was generated from the following file:

- F:/Codespace/BoostProject/argumatrix/dung_theory/PreferredReasoner.hpp

## 5.18 argumatrix::Reasoner Class Reference

Inheritance diagram for argumatrix::Reasoner:


argumatrix::Reasoner
argumatrix::GroundedReasoner
argumatrix::PlReasoner
argumatrix::PreferredReasoner
argumatrix::AdmissiblePlReasoner
argumatrix::CompletePlReasoner
argumatrix::ConflictfreePlReasoner
argumatrix::StablePlReasoner

### Public Member Functions

- **Reasoner** (const DungAF &daf, streambuf $*$osbuff=std::cout.rdbuf())
- bitvector getAttacked (const bitvector &_bv)

  *Get the attacked arguments by arguments in _bv,* $R^+(S)$ $R^+(S) = x|xisattackedbyS$. $R^+(S_bv) = D * S_bv$.
- bitvector characteristic (const bitvector &_bv)

  *The characteristic function of an abstract argumentation framework:* $F_{AF}(S) = A|Aisacceptablewrt.S$. *F_AF(↩ S_bv) = not{ R^+[ not( R^+(S_bv) ) ] }.*
- bitvector characteristic ()

  *The characteristic function of an abstract argumentation framework with the initialization of empty set.*
- bitvector neutrality (const bitvector &_bv)

  *The neutrality function of an abstract argumentation framework: N_AF(S) = {A| All arguments that not attacked by S}. N_AF(S_bv) = not( R^+(S_bv) )*
- bool is_conflict_free (const bitvector &_bv)

  *Argument set S is conflict-free? S is said to be conflict-free iff for any two arguments a,b in S such that a does not attack b.*
- bool **is_conflict_free** (const set< string > &argset)
- bool is_acceptable (const bitvector &S, const bitvector &A)

  *Argument set A is acceptable w.r.t S? Alternately, S defends A? A can be an argument or an argument set. S defends argument (set) A iff for any argument x in X, if attacks A (or a in A) then there is an argument y in S such that y attacks x.*
- bool is_self_attacking (size_type idx)

  *Argument a is self-attacking iff it attacks itself.*
- bool is_admissible (const bitvector &_bv)

  *To decide whether a set of arguments (with the form of bitvector) is an admissible extension. $S$ is an admissible extension iff $S F(S) N(S)$.*
- bool is_complete (const bitvector &_bv)

*To decide whether a set of arguments (with the form of bitvector) is a complete extension. $S$ is a complete extension iff $S == F(S) N(S)$.*

- bool is_stable (const bitvector &_bv)

  *To decide whether a set of arguments (with the form of bitvector) is a stable extension. $S$ is a stable extension iff $S == N(S)$.*
- bool is_grounded (const bitvector &_bv)

  *To decide whether a set of arguments (with the form of bitvector) is a grounded extension. $S$ is a grounded extension iff it is the least fixed point of the characteristic function F.*
- bitvector getSelfAttackingArguments ()

  *Get all arguments which are self-attacking. Obviously, if an argument i attacks itself, the entry[i][i] of the attack matrix must be 1 (true). Therefore to get the set of self-attacking arguments is to get the diagonal elements of the attack matrix.*
- const set< bitvector > & getBvExtensions ()

  *Get all extensions given a specific semantics in format of bitvector. Each extension is a set of arguments which can "survive the conflict together". Here, we use a bitvector to represent an extension. Therefore, all extensions are formed an set of bit vectors, i.e., set<bitvector>. Each bitvector in set is an extension w.r.t. a given semantics. The computed extensions are stored in m_extensions, therefore, to get all extensions, it must first invoke the function computeExtension()*
- vector< int > getGroundedIntVector ()

  *Get a vector of integers {0, 1, 2}: 2 – Unknown, 1 – in grounded extension, and 0 – attacked by the grounded extension.*
- streambuf * setOutput (streambuf *strbuf)

  *Redirect the output stream to streambuf* strbuf or ostream& os. If strbuf = cout.rdbuf(), then the output is standard output. It can also redirect the output to a file by the following codes:*
  ```
  ofstream ofile("output.txt");
  streambuf* oldsb = xxx.setOutput(ofile.rdbuf());
  ```
  *or*
  ```
  streambuf* oldsb = xxx.setOutput(ofile);
  ```
  .
- streambuf * setOutput (ostream &os=std::cout)
- void printLabSet (const bitvector &bv_ext)

  *Print an extension with the form of bitvector. Assume the bitvector is [0, 1, 0, 1], the arguments corresponding to entry 1 will be print. The member m_output will determine where to print.*
- void printLabSet (const std::set< string > &labset)

  *Print a set of arguments. The member m_output will determine where to print.*
- void printBvExts ()

  *Output all extensions given a specific semantics with an ostream. Each extension is a set of arguments which can "survive the conflict together". Here, we return a string to represent all extensions. For example, if there are two extensions, [a,b] and [b, d], for some problem w.r.t. a given semantics, then we return string "[[a,b],[d,c]]". If extensions is not existing, the string "[ ]" will return.*
- bitvector getGroundedExtension ()

  *Get the grounded extension.*
- void printGroundedExt ()

  *Print the grounded extension.*
- vector< int > labelSet2IntVector (const std::set< string > &label_set)

  *Convert a set of argument (string) labels to an integer vector of {0,1,2}, All arguments in these set, the indices is 1; otherwise 2 (representing unknown)*
- virtual void task_EE ()

  *Problem [EE-$$] Print all extensions.*
- virtual void **task_EX** ()
- virtual void task_EC (const std::set< string > &argset)

  *Problem [EC-$$] Given an ${AF}=< {X}, {R}>$ and an argument $s {X}$ (respectively, a set of arguments $S {X}$), enumerate all sets $E {X}$ such that $E {E}_(AF)$ and $s E$ (respectively, $S E$).*
- virtual void task_SC (const std::set< string > &argset)

  *Problem [SC-$$] Given an ${AF}=< {X}, {R}>$ and an argument $s {X}$ (respectively, a set of arguments $S {X}$), enumerate some set $E {X}$ such that $E {E}_(AF)$ and $s E$ (respectively, $S E$).*

- virtual void task_SE ()

  *Problem [SE-$$] Given an ${AF}=< {X}, {R}>$, enumerate some set $E {X}$ that are in ${E}_(AF)$.*
- virtual void task_DE (const std::set< string > &argset)

  *Problem [DE-$$] Given an $AF = <X,R>$ and a set of arguments $S X$. Decide whether S is a -extension of AF, i.e., S E_(AF).*
- virtual void task_DN ()

  *Problem [DN-$$] Given an $AF = <X,R>$ and a set of arguments $S X$. Decide whether there exist a non-empty -extension for AF.*
- virtual void task_DC (const std::set< string > &argset)

  *Problem [DC-$$] Given an $AF = <X,R>$ and an argument s X (respectively, a set of arguments $S X$). Decide whether s contained (respectively, S included) in some E E_(AF) (i.e., credulously justified).*
- virtual void task_DS (const std::set< string > &argset)

  *Problem [DS-$$] Given an $AF = <X,R>$ and an argument s X (respectively, a set of arguments $S X$). Decide whether s contained (respectively, S included) in each E E_(AF) (i.e., skeptically justified).*

**Protected Attributes**

- bitmatrix m_BmAtkMtx
- size_type m_argNum
- const DungAF & m_daf
- set< bitvector > **m_extensions**
- vector< std::string > m_argLabels
- std::ostream m_output

### 5.18.1 Detailed Description

Definition at line 34 of file Reasoner.hpp.

### 5.18.2 Member Function Documentation

#### 5.18.2.1 __inline **argumatrix::bitvector** argumatrix::Reasoner::characteristic ( const **bitvector** & *_bv* )

The characteristic function of an abstract argumentation framework: $F_{AF}(S) = A|Aisacceptablewrt.S$. F_AF(↩ S_bv) = not{ R^+[ not( R^+(S_bv) ) ] }.

**Parameters**

| | |
|---|---|
| *extension* | an extension (a set of arguments), the default is empty set. |

**Returns**

a set of arguments in bitvector form.

Definition at line 368 of file Reasoner.hpp.

#### 5.18.2.2 __inline **argumatrix::bitvector** argumatrix::Reasoner::characteristic ( )

The characteristic function of an abstract argumentation framework with the initialization of empty set.

**Returns**

a set of arguments in bitvector form.

Definition at line 375 of file Reasoner.hpp.

Referenced by getGroundedExtension(), and is_acceptable().

#### 5.18.2.3 __inline argumatrix::bitvector argumatrix::Reasoner::getAttacked ( const bitvector & *_bv* )

Get the attacked arguments by arguments in _bv, $R^+(S)$ $R^+(S) = x|xisattackedbyS$. $R^+(S_bv) = D * S_bv$.

**Parameters**

| | |
|---|---|
| *_bv* | the bitvector with respect to the set $S$. |

**Returns**

a set of arguments in bitvector form.

Definition at line 360 of file Reasoner.hpp.

Referenced by getGroundedIntVector(), is_conflict_free(), and neutrality().

#### 5.18.2.4 __inline const set< bitvector > & argumatrix::Reasoner::getBvExtensions ( )

Get all extensions given a specific semantics in format of bitvector. Each extension is a set of arguments which can "survive the conflict together". Here, we use a bitvector to represent an extension. Therefore, all extensions are formed an set of bit vectors, i.e., set<bitvector>. Each bitvector in set is an extension w.r.t. a given semantics. The computed extensions are stored in m_extensions, therefore, to get all extensions, it must first invoke the function computeExtension()

**Returns**

a set of bitvector, i.e., set<bitvector>.

Definition at line 400 of file Reasoner.hpp.

#### 5.18.2.5 argumatrix::bitvector argumatrix::Reasoner::getGroundedExtension ( )

Get the grounded extension.

**Returns**

a set of arguments in bitvector form.

Definition at line 484 of file Reasoner.hpp.

Referenced by is_grounded(), and printGroundedExt().

#### 5.18.2.6 vector< int > argumatrix::Reasoner::getGroundedIntVector ( )

Get a vector of integers {0, 1, 2}: 2 – Unknown, 1 – in grounded extension, and 0 – attacked by the grounded extension.

**Returns**

a vector of integers {0, 1, 2}.

Definition at line 499 of file Reasoner.hpp.

#### 5.18.2.7 __inline argumatrix::bitvector argumatrix::Reasoner::getSelfAttackingArguments ( )

Get all arguments which are self-attacking. Obviously, if an argument i attacks itself, the entry[i][i] of the attack matrix must be 1 (true). Therefore to get the set of self-attacking arguments is to get the diagonal elements of the attack matrix.

**Returns**

a set of arguments in bitvector form, if bitvector[i] = true representing the argument (with index) i is a self-attacking argument, otherwise is not.

Definition at line 414 of file Reasoner.hpp.

#### 5.18.2.8 __inline bool argumatrix::Reasoner::is_acceptable ( const bitvector & *S*, const bitvector & *A* )

Argument set A is acceptable w.r.t S? Alternately, S defends A? A can be an argument or an argument set. S defends argument (set) A iff for any argument x in X, if attacks A (or a in A) then there is an argument y in S such that y attacks x.

**Parameters**

| | |
|---|---|
| *S* | a set of arguments. |
| *A* | an argument or an argument set |

**Returns**

true if S defends A.

Definition at line 394 of file Reasoner.hpp.

#### 5.18.2.9 __inline bool argumatrix::Reasoner::is_admissible ( const bitvector & *_bv* )

To decide whether a set of arguments (with the form of bitvector) is an admissible extension. $S$ is an admissible extension iff $S F(S) N(S)$.

**Returns**

true if _bv is admissible; otherwise returns false.

Definition at line 553 of file Reasoner.hpp.

#### 5.18.2.10 __inline bool argumatrix::Reasoner::is_complete ( const bitvector & *_bv* )

To decide whether a set of arguments (with the form of bitvector) is a complete extension. $S$ is a complete extension iff $S == F(S) N(S)$.

**Returns**

true if _bv is complete; otherwise returns false.

Definition at line 564 of file Reasoner.hpp.

#### 5.18.2.11 __inline bool argumatrix::Reasoner::is_conflict_free ( const bitvector & *_bv* )

Argument set S is conflict-free? S is said to be conflict-free iff for any two arguments a,b in S such that a does not attack b.

**Parameters**

| | |
|---|---|
| *extension* | an extension (a set of arguments). |

**Returns**

true if S is conflict-free.

Definition at line 381 of file Reasoner.hpp.

#### 5.18.2.12 __inline bool argumatrix::Reasoner::is_grounded ( const bitvector & *_bv* )

To decide whether a set of arguments (with the form of bitvector) is a grounded extension. $S$ is a grounded extension iff it is the least fixed point of the characteristic function F.

**Returns**

true if _bv is grounded; otherwise returns false.

Definition at line 584 of file Reasoner.hpp.

#### 5.18.2.13 __inline bool argumatrix::Reasoner::is_self_attacking ( size_type *idx* )

Argument a is self-attacking iff it attacks itself.

**Parameters**

| | |
|---|---|
| *idx* | the index of the argument w.r.t the attack matrix |

**Returns**

true if S defends A.

Definition at line 406 of file Reasoner.hpp.

#### 5.18.2.14 __inline bool argumatrix::Reasoner::is_stable ( const bitvector & *_bv* )

To decide whether a set of arguments (with the form of bitvector) is a stable extension. $S$ is a stable extension iff $S == N(S)$.

**Returns**

true if _bv is stable; otherwise returns false.

Definition at line 575 of file Reasoner.hpp.

#### 5.18.2.15 vector< int > argumatrix::Reasoner::labelSet2IntVector ( const std::set< string > & *label_set* )

Convert a set of argument (string) labels to an integer vector of {0,1,2}, All arguments in these set, the indices is 1; otherwise 2 (representing unknown)

**Returns**

no return.

Definition at line 526 of file Reasoner.hpp.

#### 5.18.2.16 __inline argumatrix::bitvector argumatrix::Reasoner::neutrality ( const bitvector & *_bv* )

The neutrality function of an abstract argumentation framework: N_AF(S) = {A| All arguments that not attacked by S}. N_AF(S_bv) = not( R^+(S_bv) )

**Parameters**

| | |
|---|---|
| *extension* | an extension (a set of arguments). |

**Returns**

a set of arguments in bitvector form.

Definition at line 547 of file Reasoner.hpp.

Referenced by characteristic(), getGroundedIntVector(), is_admissible(), is_complete(), and is_stable().

#### 5.18.2.17 __inline void argumatrix::Reasoner::printBvExts ( )

Output all extensions given a specific semantics with an ostream. Each extension is a set of arguments which can "survive the conflict together". Here, we return a string to represent all extensions. For example, if there are two extensions, [a,b] and [b, d], for some problem w.r.t. a given semantics, then we return string "[[a,b],[d,c]]". If extensions is not existing, the string "[ ]" will return.

**Parameters**

| | |
|---|---|
| *ostream&* | os = std::cout, output the resluts into the ostream os. |

**Returns**

a set of bitvector, i.e., set<bitvector>.

Definition at line 428 of file Reasoner.hpp.

#### 5.18.2.18 __inline void argumatrix::Reasoner::printGroundedExt ( )

Print the grounded extension.

**Returns**

no return.

Definition at line 539 of file Reasoner.hpp.

#### 5.18.2.19 void argumatrix::Reasoner::printLabSet ( const bitvector & *bv_ext* )

Print an extension with the form of bitvector. Assume the bitvector is [0, 1, 0, 1], the arguments corresponding to entry 1 will be print. The member *m_output* will determine where to print.

**Parameters**

| | |
|---|---|
| *bitvector&* | bv_ext : The bv_ext is a bitvector, which represents an extension. |

**Returns**

no return.

**See also**

[setOutput]

Definition at line 433 of file Reasoner.hpp.

Referenced by printGroundedExt().

#### 5.18.2.20 void argumatrix::Reasoner::printLabSet ( const std::set< string > & *labset* )

Print a set of arguments. The member *m_output* will determine where to print.

**Parameters**

| | |
|---|---|
| *const* | std::set<string>& labset |

**Returns**

no return.

**See also**

setOutput(streambuf∗ strbuf = std::cout.rdbuf());

Definition at line 454 of file Reasoner.hpp.

#### 5.18.2.21 __inline streambuf ∗ argumatrix::Reasoner::setOutput ( streambuf ∗ *strbuf* )

Redirect the output stream to streambuf∗ strbuf or ostream& os. If strbuf = cout.rdbuf(), then the output is standard output. It can also redirect the output to a file by the following codes:

```
ofstream ofile("output.txt");
streambuf* oldsb = xxx.setOutput(ofile.rdbuf());
```

or

```
streambuf* oldsb = xxx.setOutput(ofile);
```

.

**Parameters**

| | |
|---|---|
| *strbuf* | is a streambuf pointer. |

**Returns**

return the old streambuf, which can be used to redirect the old streambuf.

Definition at line 473 of file Reasoner.hpp.

#### 5.18.2.22 __inline streambuf ∗ argumatrix::Reasoner::setOutput ( ostream & *os* = `std::cout` )

Method: setOutput FullName: public argumatrix::Reasoner::setOutput

**See also**

streambuf∗ setOutput(streambuf∗ strbuf = std::cout.rdbuf());

**Parameters**

| | |
|---|---|
| *ostream* | & os |

**Returns**

streambuf∗

Definition at line 479 of file Reasoner.hpp.

#### 5.18.2.23 virtual void argumatrix::Reasoner::task_DE ( const std::set< string > & *argset* ) [inline], [virtual]

Problem [DE-$$] Given an $AF = <X,R>$ and a set of arguments $S X$. Decide whether S is a -extension of AF, i.e., S E_(AF).

**Parameters**

| | |
|---|---|
| *a* | set of arguments. |

**Returns**

true if argset is a -extension; otherwise return false.

**Note**

For this task, the *set<string>& argset* can be empty, which means to decide whether the empty set is a $$-extension of AF. This indicates that the option -a is not necessary, then the default *set<string>& argset* is empty.

Reimplemented in argumatrix::CompletePIReasoner, argumatrix::ConflictfreePIReasoner, argumatrix::↩ AdmissiblePIReasoner, argumatrix::StablePIReasoner, and argumatrix::GroundedReasoner.

Definition at line 303 of file Reasoner.hpp.

#### 5.18.2.24 virtual void argumatrix::Reasoner::task_DN ( ) [inline], [virtual]

Problem [DN-$$] Given an $AF = <X,R>$ and a set of arguments $S X$. Decide whether there exist a non-empty -extension for AF.

**Returns**

true if there exist a non-empty -extension; otherwise return false.

Reimplemented in argumatrix::CompletePIReasoner, argumatrix::ConflictfreePIReasoner, argumatrix::↩ AdmissiblePIReasoner, argumatrix::StablePIReasoner, and argumatrix::GroundedReasoner.

Definition at line 313 of file Reasoner.hpp.

#### 5.18.2.25 virtual void argumatrix::Reasoner::task_EC ( const std::set< string > & *argset* ) [inline], [virtual]

Problem [EC-$$] Given an ${AF}=< {X}, {R}>$ and an argument $s {X}$ (respectively, a set of arguments $S {X}$), enumerate all sets $E {X}$ such that $E {E}_(AF)$ and $s E$ (respectively, $S E$).

**Parameters**

| | |
|---|---|
| *set<string>,or* | a Boolean vector: a set of argument |

**Returns**

no return.

Reimplemented in argumatrix::CompletePlReasoner, argumatrix::AdmissiblePlReasoner, argumatrix::↩ ConflictfreePlReasoner, argumatrix::StablePlReasoner, and argumatrix::GroundedReasoner.

Definition at line 267 of file Reasoner.hpp.

#### 5.18.2.26 virtual void argumatrix::Reasoner::task_EE ( ) [inline], [virtual]

Problem [EE-$$] Print all extensions.

**Parameters**

| | |
|---|---|
| *no* | argument |

**Returns**

no return.

Reimplemented in argumatrix::CompletePlReasoner, argumatrix::AdmissiblePlReasoner, argumatrix::↩ ConflictfreePlReasoner, argumatrix::StablePlReasoner, and argumatrix::GroundedReasoner.

Definition at line 254 of file Reasoner.hpp.

#### 5.18.2.27 virtual void argumatrix::Reasoner::task_SC ( const std::set< string > & *argset* ) [inline], [virtual]

Problem [SC-$$] Given an ${AF}=< {X}, {R}>$ and an argument $s {X}$ (respectively, a set of arguments $S {X}$), enumerate some set $E {X}$ such that $E {E}_(AF)$ and $s E$ (respectively, $S E$).

**Parameters**

| | |
|---|---|
| *a* | set of arguments. |

**Returns**

no return.

Reimplemented in argumatrix::CompletePlReasoner, argumatrix::AdmissiblePlReasoner, argumatrix::↩ ConflictfreePlReasoner, argumatrix::StablePlReasoner, and argumatrix::GroundedReasoner.

Definition at line 280 of file Reasoner.hpp.

#### 5.18.2.28 virtual void argumatrix::Reasoner::task_SE ( ) [inline], [virtual]

Problem [SE-$$] Given an ${AF}=< {X}, {R}>$, enumerate some set $E {X}$ that are in ${E}_(AF)$.

**Parameters**

| | |
|---|---|
| *a* | set of arguments. |

**Returns**

no return.

Reimplemented in argumatrix::CompletePlReasoner, argumatrix::AdmissiblePlReasoner, argumatrix::↩ ConflictfreePlReasoner, argumatrix::StablePlReasoner, and argumatrix::GroundedReasoner.

Definition at line 290 of file Reasoner.hpp.

### 5.18.3 Member Data Documentation

#### 5.18.3.1 vector< std::string > argumatrix::Reasoner::m_argLabels [protected]

Dung's abstract argumentation framework

Definition at line 346 of file Reasoner.hpp.

Referenced by printLabSet().

#### 5.18.3.2 size_type argumatrix::Reasoner::m_argNum [protected]

The number of arguments

Definition at line 339 of file Reasoner.hpp.

Referenced by characteristic(), getGroundedExtension(), getGroundedIntVector(), and labelSet2IntVector().

#### 5.18.3.3 bitmatrix argumatrix::Reasoner::m_BmAtkMtx [protected]

The bitmatrix of the Dung Abstract argumentation framework. We can access all attackers of an argument. The attackers of the argument with index i is m_BmAtkMtx[i].

Definition at line 334 of file Reasoner.hpp.

Referenced by getAttacked(), getSelfAttackingArguments(), and is_self_attacking().

#### 5.18.3.4 const DungAF& argumatrix::Reasoner::m_daf [protected]

Dung's abstract argumentation framework

Definition at line 342 of file Reasoner.hpp.

Referenced by is_conflict_free(), labelSet2IntVector(), and printBvExts().

#### 5.18.3.5 std::ostream argumatrix::Reasoner::m_output [protected]

Where to output

Definition at line 348 of file Reasoner.hpp.

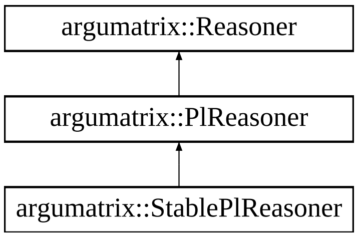
Referenced by printBvExts(), printGroundedExt(), printLabSet(), and setOutput().

The documentation for this class was generated from the following file:

- F:/Codespace/BoostProject/argumatrix/dung_theory/Reasoner.hpp

## 5.19 argumatrix::StablePlReasoner Class Reference

Inheritance diagram for argumatrix::StablePlReasoner:



### Public Member Functions

- **StablePlReasoner** (const DungAF &daf, const string &sm_task="ST", streambuf ∗osbuff=std::cout.rdbuf())
- void task_EE ()
- void **task_EX** ()
- void task_EC (const std::set< string > &argset)

  *Given an AF $= \langle \mathcal{X}, \mathcal{R} \rangle$ and an argument $s \in \mathcal{X}$ (respectively, a set of arguments $S \subseteq \mathcal{X}$), enumerate all sets $E \subseteq \mathcal{X}$ such that $E \in \mathcal{E}_\sigma(AF)$ and $s \in E$ (respectively, $S \subseteq E$).*
- void task_SC (const std::set< string > &argset)

  *Given an AF $= \langle \mathcal{X}, \mathcal{R} \rangle$ and an argument $s \in \mathcal{X}$ (respectively, a set of arguments $S \subseteq \mathcal{X}$), enumerate some set $E \subseteq \mathcal{X}$ such that $E \in \mathcal{E}_\sigma(AF)$ and $s \in E$ (respectively, $S \subseteq E$).*
- void task_SE ()

  *Given an AF $= \langle \mathcal{X}, \mathcal{R} \rangle$, enumerate some set $E \subseteq \mathcal{X}$ that are in $\mathcal{E}_\sigma(AF)$.*
- void findAllExts ()
- void task_DE (const std::set< string > &argset)

  *Given an $AF = \langle X, R \rangle$ and a set of arguments $S \subseteq X$. Decide whether S is a $\sigma$-extension of $AF$, i.e., $S \in E_\sigma(AF)$.*
- void task_DN ()

  *Given an $AF = \langle X, R \rangle$ and a set of arguments $S \subseteq X$. Decide whether there exist a non-empty -extension for AF.*
- void task_DC (const std::set< string > &argset)

  *Given an $AF = \langle X, R \rangle$ and an argument $s \in X$ (respectively, a set of arguments $S \subseteq X$). Decide whether $s$ contained (respectively, $S$ included) in some $E \in E_\sigma(AF)$ (i.e., credulously justified).*
- void task_DS (const std::set< string > &argset)

  *Given an $AF = \langle X, R \rangle$ and an argument s X (respectively, a set of arguments $S \subseteq X$). Decide whether s contained (respectively, S included) in each $E \in E_\sigma(AF)$ (i.e., skeptically justified).*
- void consultPlFile (std::string plFile=ARG_PROLOG_FILE)

  *Loading the Prolog source file [argmat-clpb].*
- void **Test_time** ()
- void printAllExts (const std::string &predct)

  *Print all extensions with respect to semantic predct.*
- void printAllExts (const std::string &predct, const std::set< string > &argset)

  *Given an ${AF}=< {X}, {R}>$ and an argument $s {X}$ (respectively, a set of arguments $S {X}$), enumerate all sets $E {X}$ such that $E {E}_(AF)$ and $s E$ (respectively, $S E$).*
- void **printAllExts** (const std::string &predct, const std::vector< int > &vecii)
- void **printAllExts2** (const std::string &predct)
- void fetchAllExts (const std::string &predct)
- void printSomeExt (const std::string &predct, const std::set< string > &argset)

  *Given an ${AF}=< {X}, {R}>$ and an argument $s {X}$ (respectively, a set of arguments $S {X}$), enumerate some set $E {X}$ such that $E {E}_(AF)$ and $s E$ (respectively, $S E$).*
- void **printSomeExt** (const std::string &predct, const vector< int > &vecii)

- bool verifyNonemptyExt (const std::string &predct)

  *Given an $AF = <X,R>$ and a set of arguments $S X$. Decide whether there exist a non-empty $$-extension for AF.*
- bool verifyExtension (const std::string &predct, std::set< string > &argset)

  *Given an $AF = <X,R>$ and a set of arguments $S X$. Decide whether S is a $$-extension of AF, i.e., $S E_(AF)$.*
- bool **verifyExtension** (const std::string &predct, bitvector &bvecii)
- bool isCredulouslyJustified (const std::string &predct, const std::set< string > &argset)

  *Given an $AF = <X,R>$ and an argument s X (respectively, a set of arguments $S X$). Decide whether s contained (respectively, S included) in some E E_(AF) (i.e., credulously justified).*
- bool isSkepticallyJustified (const std::string &predct, const std::set< string > &argset)

  *Given an $AF = <X,R>$ and an argument s X (respectively, a set of arguments $S X$). Decide whether s contained (respectively, S included) in each E E_(AF) (i.e., skeptically justified).*
- bitvector getAttacked (const bitvector &_bv)

  *Get the attacked arguments by arguments in _bv,* $R^+(S)$ $R^+(S) = x|xisattackedbyS.$ $R^+(S_bv) = D * S_bv.$
- bitvector characteristic (const bitvector &_bv)

  *The characteristic function of an abstract argumentation framework:* $F_{AF}(S) = A|Aisacceptablewrt.S.$ *F_AF(↩ S_bv) = not{ R^+[ not( R^+(S_bv) ) ] }.*
- bitvector characteristic ()

  *The characteristic function of an abstract argumentation framework with the initialization of empty set.*
- bitvector neutrality (const bitvector &_bv)

  *The neutrality function of an abstract argumentation framework: N_AF(S) = {A| All arguments that not attacked by S}. N_AF(S_bv) = not( R^+(S_bv) )*
- bool is_conflict_free (const bitvector &_bv)

  *Argument set S is conflict-free? S is said to be conflict-free iff for any two arguments a,b in S such that a does not attack b.*
- bool **is_conflict_free** (const set< string > &argset)
- bool is_acceptable (const bitvector &S, const bitvector &A)

  *Argument set A is acceptable w.r.t S? Alternately, S defends A? A can be an argument or an argument set. S defends argument (set) A iff for any argument x in X, if attacks A (or a in A) then there is an argument y in S such that y attacks x.*
- bool is_self_attacking (size_type idx)

  *Argument a is self-attacking iff it attacks itself.*
- bool is_admissible (const bitvector &_bv)

  *To decide whether a set of arguments (with the form of bitvector) is an admissible extension. $S$ is an admissible extension iff $S F(S) N(S)$.*
- bool is_complete (const bitvector &_bv)

  *To decide whether a set of arguments (with the form of bitvector) is a complete extension. $S$ is a complete extension iff $S == F(S) N(S)$.*
- bool is_stable (const bitvector &_bv)

  *To decide whether a set of arguments (with the form of bitvector) is a stable extension. $S$ is a stable extension iff $S == N(S)$.*
- bool is_grounded (const bitvector &_bv)

  *To decide whether a set of arguments (with the form of bitvector) is a grounded extension. $S$ is a grounded extension iff it is the least fixed point of the characteristic function F.*
- bitvector getSelfAttackingArguments ()

  *Get all arguments which are self-attacking. Obviously, if an argument i attacks itself, the entry[i][i] of the attack matrix must be 1 (true). Therefore to get the set of self-attacking arguments is to get the diagonal elements of the attack matrix.*
- const set< bitvector > & getBvExtensions ()

  *Get all extensions given a specific semantics in format of bitvector. Each extension is a set of arguments which can "survive the conflict together". Here, we use a bitvector to represent an extension. Therefore, all extensions are formed an set of bit vectors, i.e., set<bitvector>. Each bitvector in set is an extension w.r.t. a given semantics. The computed extensions are stored in m_extensions, therefore, to get all extensions, it must first invoke the function computeExtension()*

- vector< int > getGroundedIntVector ()

  *Get a vector of integers {0, 1, 2}: 2 – Unknown, 1 – in grounded extension, and 0 – attacked by the grounded extension.*

- streambuf ∗ setOutput (streambuf ∗strbuf)

  *Redirect the output stream to streambuf∗ strbuf or ostream& os. If strbuf = cout.rdbuf(), then the output is standard output. It can also redirect the output to a file by the following codes:*
  ```
  ofstream ofile("output.txt");
  streambuf* oldsb = xxx.setOutput(ofile.rdbuf());
  ```
  *or*
  ```
  streambuf* oldsb = xxx.setOutput(ofile);
  ```
  .

- streambuf ∗ setOutput (ostream &os=std::cout)
- void printLabSet (const bitvector &bv_ext)

  *Print an extension with the form of bitvector. Assume the bitvector is [0, 1, 0, 1], the arguments corresponding to entry 1 will be print. The member m_output will determine where to print.*

- void printLabSet (const std::set< string > &labset)

  *Print a set of arguments. The member m_output will determine where to print.*

- void printBvExts ()

  *Output all extensions given a specific semantics with an ostream. Each extension is a set of arguments which can "survive the conflict together". Here, we return a string to represent all extensions. For example, if there are two extensions, [a,b] and [b, d], for some problem w.r.t. a given semantics, then we return string "[[a,b],[d,c]]". If extensions is not existing, the string "[ ]" will return.*

- bitvector getGroundedExtension ()

  *Get the grounded extension.*

- void printGroundedExt ()

  *Print the grounded extension.*

- vector< int > labelSet2IntVector (const std::set< string > &label_set)

  *Convert a set of argument (string) labels to an integer vector of {0,1,2}, All arguments in these set, the indices is 1; otherwise 2 (representing unknown)*

## Protected Member Functions

- void findAllExts (const std::string &predct)

  *Find all extensions and store them in m_extensions with the form of bitvector.*

- void createPlAttackMatrix ()

  *Create the attack matrix with the form of PlTerm, which is an input of SWI-Prolog.*

- void printLableExtByBlistTerm (const PlTerm &plt)

  *Print an extension with the form of the bool list term. Assume a bool list term is [0, 1, 0, 1], the arguments corresponding to entry 1 will be print. The member m_output will determine where to print.*

- void printLableExtByBmatrixTerm (const PlTerm &pmtx)

  *Print an extension with the form of the bool matrix term. Assume a bool list term is [[0, 1, 0, 1], [0, 1, 0, 1]], the arguments corresponding to entry 1 will be print. The member m_output will determine where to print.*

- void addBlTermToBvExts (const PlTerm &plt)

  *Convert a bool list term into a bitvertor, and add it to extension.*

- bool verifyInclusion (const std::string &predct, const vector< int > &vecii)

  *Given an $AF = <X,R>$ and a set of arguments $S X$. Decide whether there exist a -extension that includes S.*

- bool verifyExclusion (const std::string &predct, const bitvector &vecB)

  *Given an $AF = <X,R>$ and a set of arguments $S X$. Decide whether there exist a -extension that excludes S.*

**Protected Attributes**

- string **m_predicate**
- PlTerm **m_PlAtkMtx**
- bitmatrix m_BmAtkMtx
- size_type m_argNum
- const DungAF & m_daf
- set< bitvector > **m_extensions**
- vector< std::string > m_argLabels
- std::ostream m_output

### 5.19.1 Detailed Description

Definition at line 37 of file StablePlReasoner.hpp.

### 5.19.2 Member Function Documentation

#### 5.19.2.1 void argumatrix::PlReasoner::addBlTermToBvExts ( const PlTerm & *plt* ) [protected], [inherited]

Convert a bool list term into a bitvertor, and add it to extension.

**Parameters**

| | |
|---|---|
| *PlTerm&* | plt : The PlTerm is a bool list term, which represents an extension. |

**Returns**

no return. The results are added into m_extensions.

Definition at line 303 of file PlReasoner.hpp.

#### 5.19.2.2 __inline argumatrix::bitvector argumatrix::Reasoner::characteristic ( const bitvector & *_bv* ) [inherited]

The characteristic function of an abstract argumentation framework: $F_{AF}(S) = A|Aisacceptablewrt.S$. F_AF(↩
S_bv) = not{ R^+[ not( R^+(S_bv) ) ] }.

**Parameters**

| | |
|---|---|
| *extension* | an extension (a set of arguments), the default is empty set. |

**Returns**

a set of arguments in bitvector form.

Definition at line 368 of file Reasoner.hpp.

#### 5.19.2.3 __inline argumatrix::bitvector argumatrix::Reasoner::characteristic ( ) [inherited]

The characteristic function of an abstract argumentation framework with the initialization of empty set.

**Returns**

a set of arguments in bitvector form.

Definition at line 375 of file Reasoner.hpp.

Referenced by argumatrix::Reasoner::getGroundedExtension(), and argumatrix::Reasoner::is_acceptable().

#### 5.19.2.4 void argumatrix::PlReasoner::consultPlFile ( std::string *plFile* = ARG_PROLOG_FILE ) [inherited]

Loading the Prolog source file [argmat-clpb].

Method: consultPlFile

**Parameters**

| | |
|---|---|
| *std::string* | plFile |

**Note**

SWI-Prolog supports compilation of individual or multiple Prolog source files into ‘Quick Load Files’. A ‘Quick Load File’ (.qlf file) stores the contents of the file in a precompiled format. These files load considerably faster than source files and are normally more compact. They are machine-independent and may thus be loaded on any implementation of SWI-Prolog. Note, however, that clauses are stored as virtual machine instructions. Changes to the compiler will generally make old compiled files unusable. Quick Load Files are created using qcompile/1. They are loaded using consult/1 or one of the other file-loading predicates described in section 4.3. If consult/1 is given an explicit .pl file, it will load the Prolog source. When given a .qlf file, it will load the file. When no extension is specified, it will load the .qlf file when present and the .pl file otherwise.

Definition at line 253 of file PlReasoner.hpp.

#### 5.19.2.5 void argumatrix::PlReasoner::createPlAttackMatrix ( ) [protected], [inherited]

Create the attack matrix with the form of PlTerm, which is an input of SWI-Prolog.

**Returns**

no return. The result is stored in member variable *m_PlAtkMtx*.

Definition at line 243 of file PlReasoner.hpp.

#### 5.19.2.6 void argumatrix::PlReasoner::fetchAllExts ( const std::string & *predct* ) [inherited]

Method: fetchAllExts FullName: public argumatrix::PlReasoner::fetchAllExts

**Parameters**

| | |
|---|---|
| *const* | std::string & predct |

**Returns**

void

**Return values**

| | |
|---|---|
| | |

Definition at line 701 of file PlReasoner.hpp.

#### 5.19.2.7 __inline void argumatrix::StablePlReasoner::findAllExts ( )

Find all extensions and store them in *m_extensions* with the form of bitvector.

**Parameters**

| | |
|---|---|
| *no* | argument. |

**Returns**

no return. The results are added into *m_extensions*.

Definition at line 169 of file StablePlReasoner.hpp.

#### 5.19.2.8 void argumatrix::PlReasoner::findAllExts ( const std::string & *predct* ) [protected], [inherited]

Find all extensions and store them in *m_extensions* with the form of bitvector.

**Parameters**

| | |
|---|---|
| *string&* | predct: The predicate. |

**Returns**

no return. The results are added into *m_extensions*.

Definition at line 669 of file PlReasoner.hpp.

Referenced by argumatrix::AdmissiblePlReasoner::findAllExts(), findAllExts(), argumatrix::ConflictfreePl↩ Reasoner::findAllExts(), and argumatrix::CompletePlReasoner::findAllExts().

#### 5.19.2.9 __inline argumatrix::bitvector argumatrix::Reasoner::getAttacked ( const bitvector & *_bv* ) [inherited]

Get the attacked arguments by arguments in _bv, $R^+(S)$ $R^+(S) = x|xisattackedbyS.$ $R^+(S_bv) = D * S_bv.$

**Parameters**

| | |
|---|---|
| *_bv* | the bitvector with respect to the set $S$. |

**Returns**

a set of arguments in bitvector form.

Definition at line 360 of file Reasoner.hpp.

Referenced by argumatrix::Reasoner::getGroundedIntVector(), argumatrix::Reasoner::is_conflict_free(), and argumatrix::Reasoner::neutrality().

#### 5.19.2.10 __inline const set< bitvector > & argumatrix::Reasoner::getBvExtensions ( ) [inherited]

Get all extensions given a specific semantics in format of bitvector. Each extension is a set of arguments which can "survive the conflict together". Here, we use a bitvector to represent an extension. Therefore, all extensions are formed an set of bit vectors, i.e., set<bitvector>. Each bitvector in set is an extension w.r.t. a given semantics. The computed extensions are stored in m_extensions, therefore, to get all extensions, it must first invoke the function computeExtension()

**Returns**

a set of bitvector, i.e., set<bitvector>.

Definition at line 400 of file Reasoner.hpp.

#### 5.19.2.11 argumatrix::bitvector argumatrix::Reasoner::getGroundedExtension ( ) [inherited]

Get the grounded extension.

**Returns**

a set of arguments in bitvector form.

Definition at line 484 of file Reasoner.hpp.

Referenced by argumatrix::Reasoner::is_grounded(), and argumatrix::Reasoner::printGroundedExt().

#### 5.19.2.12 vector< int > argumatrix::Reasoner::getGroundedIntVector ( ) [inherited]

Get a vector of integers {0, 1, 2}: 2 – Unknown, 1 – in grounded extension, and 0 – attacked by the grounded extension.

**Returns**

a vector of integers {0, 1, 2}.

Definition at line 499 of file Reasoner.hpp.

#### 5.19.2.13 __inline **argumatrix::bitvector** argumatrix::Reasoner::getSelfAttackingArguments ( ) [inherited]

Get all arguments which are self-attacking. Obviously, if an argument i attacks itself, the entry[i][i] of the attack matrix must be 1 (true). Therefore to get the set of self-attacking arguments is to get the diagonal elements of the attack matrix.

**Returns**

a set of arguments in bitvector form, if bitvector[i] = true representing the argument (with index) i is a self-attacking argument, otherwise is not.

Definition at line 414 of file Reasoner.hpp.

#### 5.19.2.14 __inline bool argumatrix::Reasoner::is_acceptable ( const **bitvector** & *S*, const **bitvector** & *A* ) [inherited]

Argument set A is acceptable w.r.t S? Alternately, S defends A? A can be an argument or an argument set. S defends argument (set) A iff for any argument x in X, if attacks A (or a in A) then there is an argument y in S such that y attacks x.

**Parameters**

| | |
|---|---|
| *S* | a set of arguments. |
| *A* | an argument or an argument set |

**Returns**

true if S defends A.

Definition at line 394 of file Reasoner.hpp.

#### 5.19.2.15 __inline bool argumatrix::Reasoner::is_admissible ( const **bitvector** & *_bv* ) [inherited]

To decide whether a set of arguments (with the form of bitvector) is an admissible extension. $S$ is an admissible extension iff $S F(S) N(S)$.

**Returns**

true if _bv is admissible; otherwise returns false.

Definition at line 553 of file Reasoner.hpp.

#### 5.19.2.16 __inline bool argumatrix::Reasoner::is_complete ( const **bitvector** & *_bv* ) [inherited]

To decide whether a set of arguments (with the form of bitvector) is a complete extension. $S$ is a complete extension iff $S == F(S) N(S)$.

**Returns**

true if _bv is complete; otherwise returns false.

Definition at line 564 of file Reasoner.hpp.

#### 5.19.2.17 __inline bool argumatrix::Reasoner::is_conflict_free ( const **bitvector** & *_bv* ) [inherited]

Argument set S is conflict-free? S is said to be conflict-free iff for any two arguments a,b in S such that a does not attack b.

**Parameters**

|  |  |
|---|---|
| *extension* | an extension (a set of arguments). |

**Returns**

true if S is conflict-free.

Definition at line 381 of file Reasoner.hpp.

#### 5.19.2.18 __inline bool argumatrix::Reasoner::is_grounded ( const bitvector & *_bv* ) `[inherited]`

To decide whether a set of arguments (with the form of bitvector) is a grounded extension. $S$ is a grounded extension iff it is the least fixed point of the characteristic function F.

**Returns**

true if _bv is grounded; otherwise returns false.

Definition at line 584 of file Reasoner.hpp.

#### 5.19.2.19 __inline bool argumatrix::Reasoner::is_self_attacking ( size_type *idx* ) `[inherited]`

Argument a is self-attacking iff it attacks itself.

**Parameters**

|  |  |
|---|---|
| *idx* | the index of the argument w.r.t the attack matrix |

**Returns**

true if S defends A.

Definition at line 406 of file Reasoner.hpp.

#### 5.19.2.20 __inline bool argumatrix::Reasoner::is_stable ( const bitvector & *_bv* ) `[inherited]`

To decide whether a set of arguments (with the form of bitvector) is a stable extension. $S$ is a stable extension iff $S == N(S)$.

**Returns**

true if _bv is stable; otherwise returns false.

Definition at line 575 of file Reasoner.hpp.

#### 5.19.2.21 bool argumatrix::PlReasoner::isCredulouslyJustified ( const std::string & *predct,* const std::set< string > & *argset* ) [inherited]

Given an $AF = <X,R>$ and an argument s X (respectively, a set of arguments $S X$). Decide whether s contained (respectively, S included) in some E E_(AF) (i.e., credulously justified).

Problem [DC-$$]

Definition at line 642 of file PlReasoner.hpp.

#### 5.19.2.22 bool argumatrix::PlReasoner::isSkepticallyJustified ( const std::string & *predct,* const std::set< string > & *argset* ) [inherited]

Given an $AF = <X,R>$ and an argument s X (respectively, a set of arguments $S X$). Decide whether s contained (respectively, S included) in each E E_(AF) (i.e., skeptically justified).

Problem [DS-$$]

Definition at line 654 of file PlReasoner.hpp.

#### 5.19.2.23 vector< int > argumatrix::Reasoner::labelSet2IntVector ( const std::set< string > & *label_set* ) [inherited]

Convert a set of argument (string) labels to an integer vector of {0,1,2}, All arguments in these set, the indices is 1; otherwise 2 (representing unknown)

**Returns**

no return.

Definition at line 526 of file Reasoner.hpp.

#### 5.19.2.24 __inline argumatrix::bitvector argumatrix::Reasoner::neutrality ( const bitvector & *_bv* ) [inherited]

The neutrality function of an abstract argumentation framework: N_AF(S) = {A| All arguments that not attacked by S}. N_AF(S_bv) = not( R^+(S_bv) )

**Parameters**

| | |
|---|---|
| *extension* | an extension (a set of arguments). |

**Returns**

a set of arguments in bitvector form.

Definition at line 547 of file Reasoner.hpp.

Referenced by argumatrix::Reasoner::characteristic(), argumatrix::Reasoner::getGroundedIntVector(), argumatrix↩ ::Reasoner::is_admissible(), argumatrix::Reasoner::is_complete(), and argumatrix::Reasoner::is_stable().

#### 5.19.2.25 void argumatrix::PlReasoner::printAllExts ( const std::string & *predct* ) [inherited]

Print all extensions with respect to semantic predct.

Problem [EE-$$]

**Parameters**

| | |
|---|---|
| *no* | argument |

**Returns**

no return.

Definition at line 430 of file PlReasoner.hpp.

Referenced by argumatrix::PlReasoner::printAllExts(), task_EC(), argumatrix::AdmissiblePlReasoner::task_E↩ C(), argumatrix::ConflictfreePlReasoner::task_EC(), argumatrix::CompletePlReasoner::task_EC(), argumatrix::↩ ConflictfreePlReasoner::task_EE(), argumatrix::AdmissiblePlReasoner::task_EE(), task_EE(), and argumatrix::↩ CompletePlReasoner::task_EE().

#### 5.19.2.26 __inline void argumatrix::PlReasoner::printAllExts ( const std::string & *predct,* const std::set< string > & *argset* ) `[inherited]`

Given an ${AF}=< {X}, {R}>$ and an argument $s {X}$ (respectively, a set of arguments $S {X}$), enumerate all sets $E {X}$ such that $E {E}_(AF)$ and $s E$ (respectively, $S E$).

Problem [EC-$$]

**Parameters**

| | |
|---|---|
| *set<string>,or* | a Boolean vector: a set of argument |

**Returns**

no return.

Definition at line 386 of file PlReasoner.hpp.

#### 5.19.2.27 __inline void argumatrix::Reasoner::printBvExts ( ) `[inherited]`

Output all extensions given a specific semantics with an ostream. Each extension is a set of arguments which can "survive the conflict together". Here, we return a string to represent all extensions. For example, if there are two extensions, [a,b] and [b, d], for some problem w.r.t. a given semantics, then we return string "[[a,b],[d,c]]". If extensions is not existing, the string "[ ]" will return.

**Parameters**

| | |
|---|---|
| *ostream&* | os = std::cout, output the resluts into the ostream os. |

**Returns**

a set of bitvector, i.e., set<bitvector>.

Definition at line 428 of file Reasoner.hpp.

#### 5.19.2.28 __inline void argumatrix::Reasoner::printGroundedExt ( ) [inherited]

Print the grounded extension.

**Returns**

no return.

Definition at line 539 of file Reasoner.hpp.

#### 5.19.2.29 void argumatrix::PlReasoner::printLableExtByBlistTerm ( const PlTerm & *plt* ) [protected], [inherited]

Print an extension with the form of the bool list term. Assume a bool list term is [0, 1, 0, 1], the arguments corresponding to entry 1 will be print. The member *m_output* will determine where to print.

**Parameters**

| | |
|---|---|
| *PlTerm&* | plt : The PlTerm is a bool list term, which represents an extension. |

**Returns**

no return.

Definition at line 271 of file PlReasoner.hpp.

#### 5.19.2.30 void argumatrix::PlReasoner::printLableExtByBmatrixTerm ( const PlTerm & *pmtx* ) [protected], [inherited]

Print an extension with the form of the bool matrix term. Assume a bool list term is [[0, 1, 0, 1], [0, 1, 0, 1]], the arguments corresponding to entry 1 will be print. The member *m_output* will determine where to print.

**Parameters**

| | |
|---|---|
| *PlTerm&* | plt : The PlTerm is a bool list term, which represents an extension. |

**Returns**

no return.

Definition at line 725 of file PlReasoner.hpp.

#### 5.19.2.31 void argumatrix::Reasoner::printLabSet ( const bitvector & *bv_ext* ) [inherited]

Print an extension with the form of bitvector. Assume the bitvector is [0, 1, 0, 1], the arguments corresponding to entry 1 will be print. The member *m_output* will determine where to print.

**Parameters**

| | |
|---|---|
| *bitvector&* | bv_ext : The bv_ext is a bitvector, which represents an extension. |

**Returns**

no return.

**See also**

[setOutput]

Definition at line 433 of file Reasoner.hpp.

Referenced by argumatrix::Reasoner::printGroundedExt().

#### 5.19.2.32 void argumatrix::Reasoner::printLabSet ( const std::set< string > & *labset* ) [inherited]

Print a set of arguments. The member *m_output* will determine where to print.

**Parameters**

| | |
|---|---|
| *const* | std::set<string>& labset |

**Returns**

no return.

**See also**

setOutput(streambuf∗ strbuf = std::cout.rdbuf());

Definition at line 454 of file Reasoner.hpp.

#### 5.19.2.33 __inline void argumatrix::PlReasoner::printSomeExt ( const std::string & *predct,* const std::set< string > & *argset* ) [inherited]

Given an ${AF}=< {X}, {R}>$ and an argument $s {X}$ (respectively, a set of arguments $S {X}$), enumerate some set $E {X}$ such that $E {E}_(AF)$ and $s E$ (respectively, $S E$).

Problem [SC-$$]

**Parameters**

| | |
|---|---|
| *a* | set of arguments. |

**Returns**

no return.

Definition at line 468 of file PlReasoner.hpp.

Referenced by argumatrix::AdmissiblePlReasoner::task_SC(), task_SC(), argumatrix::CompletePlReasoner::task↩ _SC(), task_SE(), and argumatrix::CompletePlReasoner::task_SE().

#### 5.19.2.34 __inline streambuf $*$ argumatrix::Reasoner::setOutput ( streambuf $*$ *strbuf* ) [inherited]

Redirect the output stream to streambuf$*$ strbuf or ostream& os. If strbuf = cout.rdbuf(), then the output is standard output. It can also redirect the output to a file by the following codes:

```
ofstream ofile("output.txt");
streambuf* oldsb = xxx.setOutput(ofile.rdbuf());
```

or

```
streambuf* oldsb = xxx.setOutput(ofile);
```

.

**Parameters**

| | |
|---|---|
| *strbuf* | is a streambuf pointer. |

**Returns**

return the old streambuf, which can be used to redirect the old streambuf.

Definition at line 473 of file Reasoner.hpp.

#### 5.19.2.35 __inline streambuf $*$ argumatrix::Reasoner::setOutput ( ostream & *os* = std::cout ) [inherited]

Method: setOutput FullName: public argumatrix::Reasoner::setOutput

**See also**

streambuf$*$ setOutput(streambuf$*$ strbuf = std::cout.rdbuf());

**Parameters**

| | |
|---|---|
| *ostream* | & os |

**Returns**

streambuf$*$

Definition at line 479 of file Reasoner.hpp.

#### 5.19.2.36 __inline void argumatrix::StablePIReasoner::task_DC ( const std::set$<$ string $>$ & *argset* ) [virtual]

Given an $AF = \langle X, R \rangle$ and an argument $s \in X$ (respectively, a set of arguments $S \subseteq X$). Decide whether $s$ contained (respectively, $S$ included) in some $E \in E_\sigma(AF)$ (i.e., credulously justified).

Problem [DC- $\sigma$]

Reimplemented from argumatrix::Reasoner.

Definition at line 219 of file StablePIReasoner.hpp.

#### 5.19.2.37 __inline void argumatrix::StablePIReasoner::task_DE ( const std::set< string > & *argset* ) [virtual]

Given an $AF = \langle X, R\rangle$ and a set of arguments $S \subseteq X$. Decide whether S is a $\sigma$-extension of $AF$, i.e., $S \in E_\sigma(AF)$.

Problem [DE- $\sigma$]

**Parameters**

| | |
|---|---|
| *a* | set of arguments. |

**Returns**

true if argset is a -extension; otherwise return false.

Reimplemented from argumatrix::Reasoner.

Definition at line 175 of file StablePIReasoner.hpp.

#### 5.19.2.38 void argumatrix::StablePIReasoner::task_DN ( ) [virtual]

Given an $AF = \langle X, R\rangle$ and a set of arguments $S \subseteq X$. Decide whether there exist a non-empty -extension for AF.

Problem [DN- $\sigma$]

**Returns**

true if there exist a non-empty $\sigma$-extension; otherwise return false.

Reimplemented from argumatrix::Reasoner.

Definition at line 191 of file StablePIReasoner.hpp.

#### 5.19.2.39 __inline void argumatrix::StablePIReasoner::task_DS ( const std::set< string > & *argset* ) [virtual]

Given an $AF = \langle X, R\rangle$ and an argument s X (respectively, a set of arguments $S \subseteq X$). Decide whether s contained (respectively, S included) in each $E \in E_\sigma(AF)$ (i.e., skeptically justified).

Problem [DS- $\sigma$]

Reimplemented from argumatrix::Reasoner.

Definition at line 234 of file StablePIReasoner.hpp.

#### 5.19.2.40 __inline void argumatrix::StablePIReasoner::task_EC ( const std::set< string > & *argset* ) [virtual]

Given an *AF* $= \langle \mathcal{X}, \mathcal{R}\rangle$ and an argument $s \in \mathcal{X}$ (respectively, a set of arguments $S \subseteq \mathcal{X}$), enumerate all sets $E \subseteq \mathcal{X}$ such that $E \in \mathcal{E}_\sigma(AF)$ and $s \in E$ (respectively, $S \subseteq E$).

Problem [EC- $\sigma$]

**Parameters**

| | |
|---|---|
| *set<string>,or* | a Boolean vector: a set of argument |

**Returns**

no return.

Reimplemented from argumatrix::Reasoner.

Definition at line 158 of file StablePIReasoner.hpp.

#### 5.19.2.41 __inline void argumatrix::StablePIReasoner::task_EE ( ) [virtual]

Problem [EE- $\sigma$] Print all extensions (with a vector of integers {0,1,2})

**Parameters**

| | |
|---|---|
| *no* | argument |

**Returns**

no return.

Reimplemented from argumatrix::Reasoner.

Definition at line 151 of file StablePIReasoner.hpp.

#### 5.19.2.42 __inline void argumatrix::StablePIReasoner::task_SC ( const std::set< string > & *argset* ) [virtual]

Given an *AF* $= \langle \mathcal{X}, \mathcal{R} \rangle$ and an argument $s \in \mathcal{X}$ (respectively, a set of arguments $S \subseteq \mathcal{X}$), enumerate some set $E \subseteq \mathcal{X}$ such that $E \in \mathcal{E}_\sigma(AF)$ and $s \in E$ (respectively, $S \subseteq E$).

Problem [SC- $\sigma$]

**Parameters**

| | |
|---|---|
| *a* | set of arguments. |

**Returns**

no return.

Reimplemented from argumatrix::Reasoner.

Definition at line 202 of file StablePIReasoner.hpp.

#### 5.19.2.43 __inline void argumatrix::StablePIReasoner::task_SE ( ) [virtual]

Given an *AF* $= \langle \mathcal{X}, \mathcal{R} \rangle$, enumerate some set $E \subseteq \mathcal{X}$ that are in $\mathcal{E}_\sigma(AF)$.

Problem [SE- $\sigma$]

**Parameters**

| | |
|---|---|
| *a* | set of arguments. |

**Returns**

no return.

Reimplemented from argumatrix::Reasoner.

Definition at line 213 of file StablePlReasoner.hpp.

#### 5.19.2.44 bool argumatrix::PlReasoner::verifyExclusion ( const std::string & *predct,* const **bitvector** & *vecB* ) `[protected], [inherited]`

Given an $AF = <X,R>$ and a set of arguments $S X$. Decide whether there exist a -extension that excludes S.

**Parameters**

| | |
|---|---|
| *a* | set of arguments. |

**Returns**

true if there exist a -extension that excludes S; otherwise return false.

Definition at line 576 of file PlReasoner.hpp.

#### 5.19.2.45 __inline bool argumatrix::PlReasoner::verifyExtension ( const std::string & *predct,* std::set< string > & *argset* ) `[inherited]`

Given an $AF = <X,R>$ and a set of arguments $S X$. Decide whether S is a $$-extension of AF, i.e., $S E_(AF)$.

Problem [DE-$$]

**Parameters**

| | |
|---|---|
| *a* | set of arguments. |

**Returns**

true if argset is a -extension; otherwise return false.

Definition at line 609 of file PlReasoner.hpp.

#### 5.19.2.46 bool argumatrix::PlReasoner::verifyInclusion ( const std::string & *predct,* const vector< int > & *vecii* ) `[protected], [inherited]`

Given an $AF = <X,R>$ and a set of arguments $S X$. Decide whether there exist a -extension that includes S.

**Parameters**

| | |
|---|---|
| *a* | set of arguments. |

**Returns**

true if there exist a -extension that includes S; otherwise return false.

Definition at line 543 of file PlReasoner.hpp.

#### 5.19.2.47 bool argumatrix::PlReasoner::verifyNonemptyExt ( const std::string & *predct* ) [inherited]

Given an $AF = <X,R>$ and a set of arguments $S X$. Decide whether there exist a non-empty $$-extension for AF.

Problem [DN-$$]

**Returns**

true if there exist a non-empty -extension; otherwise return false.

Definition at line 510 of file PlReasoner.hpp.

### 5.19.3 Member Data Documentation

#### 5.19.3.1 vector< std::string > argumatrix::Reasoner::m_argLabels [protected], [inherited]

Dung's abstract argumentation framework

Definition at line 346 of file Reasoner.hpp.

Referenced by argumatrix::Reasoner::printLabSet().

#### 5.19.3.2 size_type argumatrix::Reasoner::m_argNum [protected], [inherited]

The number of arguments

Definition at line 339 of file Reasoner.hpp.

Referenced by argumatrix::Reasoner::characteristic(), argumatrix::Reasoner::getGroundedExtension(), argumatrix↩ ::Reasoner::getGroundedIntVector(), and argumatrix::Reasoner::labelSet2IntVector().

#### 5.19.3.3 bitmatrix argumatrix::Reasoner::m_BmAtkMtx [protected], [inherited]

The bitmatrix of the Dung Abstract argumentation framework. We can access all attackers of an argument. The attackers of the argument with index i is m_BmAtkMtx[i].

Definition at line 334 of file Reasoner.hpp.

Referenced by argumatrix::Reasoner::getAttacked(), argumatrix::Reasoner::getSelfAttackingArguments(), and argumatrix::Reasoner::is_self_attacking().

#### 5.19.3.4 const DungAF& argumatrix::Reasoner::m_daf [protected], [inherited]

Dung's abstract argumentation framework

Definition at line 342 of file Reasoner.hpp.

Referenced by argumatrix::Reasoner::is_conflict_free(), argumatrix::Reasoner::labelSet2IntVector(), and argumatrix::Reasoner::printBvExts().

#### 5.19.3.5 std::ostream argumatrix::Reasoner::m_output [protected], [inherited]

Where to output

Definition at line 348 of file Reasoner.hpp.

Referenced by argumatrix::Reasoner::printBvExts(), argumatrix::Reasoner::printGroundedExt(), argumatrix::↩
Reasoner::printLabSet(), and argumatrix::Reasoner::setOutput().

The documentation for this class was generated from the following file:

- F:/Codespace/BoostProject/argumatrix/PlReasoner/StablePlReasoner.hpp

# 6 File Documentation

## 6.1 F:/Codespace/BoostProject/argumatrix/argmat-clpb/ClpbProblem.hpp File Reference

TODO: long description.

```
#include <map>
#include <string>
#include <iostream>
#include "dung_theory/DungAF.hpp"
#include "dung_theory/GroundedReasoner.hpp"
#include "config/config.hpp"
#include "parser/parser.hpp"
#include "PlReasoner/CompletePlReasoner.hpp"
#include "PlReasoner/ConflictfreePlReasoner.hpp"
#include "PlReasoner/StablePlReasoner.hpp"
#include "PlReasoner/AdmissiblePlReasoner.hpp"
```

### Classes

- struct argumatrix::ClpbProblem

### Functions

- void argumatrix::printVersionInfo ()

  *Print the version information.*
- void argumatrix::printHelpInfo ()

  *Print the help/usage information.*
- void argumatrix::printProblemOptions ()

  *Print the ProblemOptions information.*
- void argumatrix::printFileFormatOptions ()

  *Print the FileFormatOptions information.*
- ostream & **argumatrix::operator**<< (ostream &out, ClpbProblem &problem)

### 6.1.1 Detailed Description

TODO: long description.

**Date**

2016/05/04 15:20

**Author**

Fuan Pu Contact: `Pu.Fuan@gmail.com`

**Note**

# Index

addBlTermToBvExts
    argumatrix::AdmissiblePlReasoner, 12
    argumatrix::CompletePlReasoner, 39
    argumatrix::ConflictfreePlReasoner, 57
    argumatrix::PlReasoner, 93
    argumatrix::StablePlReasoner, 138
argumatrix::AdmissiblePlReasoner, 9
    addBlTermToBvExts, 12
    characteristic, 12, 13
    consultPlFile, 13
    createPlAttackMatrix, 13
    fetchAllExts, 13
    findAllExts, 14
    getAttacked, 14
    getBvExtensions, 15
    getGroundedExtension, 15
    getGroundedIntVector, 15
    getSelfAttackingArguments, 15
    is_acceptable, 16
    is_admissible, 16
    is_complete, 16
    is_conflict_free, 16
    is_grounded, 17
    is_self_attacking, 17
    is_stable, 17
    isCredulouslyJustified, 17
    isSkepticallyJustified, 18
    labelSet2IntVector, 18
    m_BmAtkMtx, 27
    m_argLabels, 27
    m_argNum, 27
    m_daf, 27
    m_output, 28
    neutrality, 18
    printAllExts, 18, 19
    printBvExts, 19
    printGroundedExt, 19
    printLabSet, 20, 21
    printLableExtByBlistTerm, 20
    printLableExtByBmatrixTerm, 20
    printSomeExt, 21
    setOutput, 21, 22
    task_DC, 22
    task_DE, 22
    task_DN, 23
    task_DS, 23
    task_EC, 23
    task_EE, 24
    task_SC, 24
    task_SE, 24
    verifyExclusion, 26
    verifyExtension, 26
    verifyInclusion, 26
    verifyNonemptyExt, 27
argumatrix::ArgumatrixPlEngine, 28
    ArgumatrixPlEngine, 29, 30
argumatrix::ArgumentProperty, 31
argumatrix::AttackProperty, 32
argumatrix::ClpbProblem, 35
    parseAddtionalParams, 35
argumatrix::CompletePlReasoner, 36
    addBlTermToBvExts, 39
    characteristic, 40
    consultPlFile, 40
    createPlAttackMatrix, 41
    fetchAllExts, 41
    findAllExts, 41, 42
    getAttacked, 42
    getBvExtensions, 42
    getGroundedExtension, 42
    getGroundedIntVector, 43
    getSelfAttackingArguments, 43
    is_acceptable, 43
    is_admissible, 43
    is_complete, 44
    is_conflict_free, 44
    is_grounded, 44
    is_self_attacking, 44
    is_stable, 45
    isCredulouslyJustified, 45
    isSkepticallyJustified, 45
    labelSet2IntVector, 45
    m_BmAtkMtx, 54
    m_argLabels, 53
    m_argNum, 53
    m_daf, 54
    m_output, 54
    neutrality, 45
    printAllExts, 46
    printBvExts, 46
    printGroundedExt, 47
    printLabSet, 48
    printLableExtByBlistTerm, 47
    printLableExtByBmatrixTerm, 47
    printSomeExt, 48
    setOutput, 49
    task_DC, 50
    task_DE, 50
    task_DN, 50
    task_DS, 50
    task_EC, 51
    task_EE, 51
    task_SC, 51
    task_SE, 52
    verifyExclusion, 52
    verifyExtension, 52
    verifyInclusion, 53
    verifyNonemptyExt, 53
argumatrix::ConflictfreePlReasoner, 54
    addBlTermToBvExts, 57

characteristic, 58
consultPlFile, 58
createPlAttackMatrix, 59
fetchAllExts, 59
findAllExts, 59, 60
getAttacked, 60
getBvExtensions, 60
getGroundedExtension, 60
getGroundedIntVector, 61
getSelfAttackingArguments, 61
is_acceptable, 61
is_admissible, 61
is_complete, 62
is_conflict_free, 62
is_grounded, 62
is_self_attacking, 62
is_stable, 63
isCredulouslyJustified, 63
isSkepticallyJustified, 63
labelSet2IntVector, 63
m_BmAtkMtx, 73
m_argLabels, 72
m_argNum, 72
m_daf, 73
m_output, 73
neutrality, 63
printAllExts, 64
printBvExts, 64
printGroundedExt, 65
printLabSet, 66
printLableExtByBlistTerm, 65
printLableExtByBmatrixTerm, 65
printSomeExt, 66
setOutput, 67
task_DC, 69
task_DE, 69
task_DN, 69
task_DS, 69
task_EC, 70
task_EE, 70
task_SC, 70
task_SE, 71
verifyExclusion, 71
verifyExtension, 71
verifyInclusion, 72
verifyNonemptyExt, 72

argumatrix::DungAF, 74
getArgumentProperty, 74
outputBv, 75
outputBvSet, 75
set2bv, 75

argumatrix::GroundedReasoner, 76
characteristic, 78
getAttacked, 79
getBvExtensions, 79
getGroundedExtension, 79
getGroundedIntVector, 79
getSelfAttackingArguments, 80
is_acceptable, 80
is_admissible, 80
is_complete, 80
is_conflict_free, 81
is_grounded, 81
is_self_attacking, 81
is_stable, 81
labelSet2IntVector, 82
m_BmAtkMtx, 87
m_argLabels, 87
m_argNum, 87
m_daf, 88
m_output, 88
neutrality, 82
printBvExts, 82
printGroundedExt, 83
printLabSet, 83
setOutput, 84
task_DC, 84
task_DE, 85
task_DN, 85
task_DS, 85
task_EC, 86
task_EE, 86
task_SC, 86
task_SE, 87

argumatrix::PlReasoner, 90
addBlTermToBvExts, 93
characteristic, 94
consultPlFile, 94
createPlAttackMatrix, 95
fetchAllExts, 95
findAllExts, 95
getAttacked, 95
getBvExtensions, 96
getGroundedExtension, 96
getGroundedIntVector, 96
getSelfAttackingArguments, 96
is_acceptable, 97
is_admissible, 97
is_complete, 97
is_conflict_free, 97
is_grounded, 98
is_self_attacking, 98
is_stable, 98
isCredulouslyJustified, 98
isSkepticallyJustified, 98
labelSet2IntVector, 99
m_BmAtkMtx, 107
m_argLabels, 106
m_argNum, 106
m_daf, 107
m_output, 107
neutrality, 99
printAllExts, 99, 100
printBvExts, 100
printGroundedExt, 100
printLabSet, 101

printLableExtByBlistTerm, 100
printLableExtByBmatrixTerm, 101
printSomeExt, 102
setOutput, 102, 103
task_DE, 103
task_DN, 103
task_EC, 104
task_EE, 104
task_SC, 104
task_SE, 105
verifyExclusion, 105
verifyExtension, 105
verifyInclusion, 106
verifyNonemptyExt, 106
argumatrix::PlTimeQry, 107
next_solution, 110
PlTimeQry, 108
argumatrix::PreferredReasoner, 110
characteristic, 113
computeExtensions, 113
getAttacked, 113
getAttackers, 114
getBvExtensions, 114
getGroundedExtension, 114
getGroundedIntVector, 114
getSelfAttackingArguments, 115
is_acceptable, 115
is_admissible, 115
is_complete, 115
is_conflict_free, 116
is_grounded, 116
is_self_attacking, 116
is_stable, 116
labelSet2IntVector, 117
m_BmAtkMtx, 122
m_argLabels, 122
m_argNum, 122
m_daf, 122
m_output, 122
neutrality, 117
printBvExts, 117
printGroundedExt, 118
printLabSet, 118
setOutput, 119
task_DE, 119
task_DN, 120
task_EC, 120
task_EE, 121
task_SC, 121
task_SE, 121
argumatrix::Reasoner, 123
characteristic, 125
getAttacked, 126
getBvExtensions, 126
getGroundedExtension, 126
getGroundedIntVector, 126
getSelfAttackingArguments, 127
is_acceptable, 127
is_admissible, 127
is_complete, 127
is_conflict_free, 128
is_grounded, 128
is_self_attacking, 128
is_stable, 128
labelSet2IntVector, 129
m_BmAtkMtx, 134
m_argLabels, 134
m_argNum, 134
m_daf, 134
m_output, 134
neutrality, 129
printBvExts, 129
printGroundedExt, 130
printLabSet, 130
setOutput, 131
task_DE, 131
task_DN, 132
task_EC, 132
task_EE, 133
task_SC, 133
task_SE, 133
argumatrix::StablePlReasoner, 135
addBlTermToBvExts, 138
characteristic, 138
consultPlFile, 139
createPlAttackMatrix, 139
fetchAllExts, 139
findAllExts, 140
getAttacked, 140
getBvExtensions, 141
getGroundedExtension, 141
getGroundedIntVector, 141
getSelfAttackingArguments, 141
is_acceptable, 142
is_admissible, 142
is_complete, 142
is_conflict_free, 142
is_grounded, 143
is_self_attacking, 143
is_stable, 143
isCredulouslyJustified, 143
isSkepticallyJustified, 144
labelSet2IntVector, 144
m_BmAtkMtx, 152
m_argLabels, 152
m_argNum, 152
m_daf, 152
m_output, 153
neutrality, 144
printAllExts, 144, 145
printBvExts, 145
printGroundedExt, 145
printLabSet, 146, 147
printLableExtByBlistTerm, 146
printLableExtByBmatrixTerm, 146
printSomeExt, 147

setOutput, 147, 148
task_DC, 148
task_DE, 148
task_DN, 149
task_DS, 149
task_EC, 149
task_EE, 150
task_SC, 150
task_SE, 150
verifyExclusion, 151
verifyExtension, 151
verifyInclusion, 151
verifyNonemptyExt, 152
argumatrix::bitmatrix, 32
diag, 33
argumatrix::bitvector, 33
EmptySet, 33
Increase, 34
is_emptyset, 34
operator∗, 34
argumatrix::parser, 89
Aspartix2DungAF, 89
TrivialGraph2DungAF, 90
ArgumatrixPlEngine
argumatrix::ArgumatrixPlEngine, 29, 30
Aspartix2DungAF
argumatrix::parser, 89

boost::graph::internal_vertex_constructor< argumatrix↩ ::ArgumentProperty >, 88
boost::graph::internal_vertex_name< argumatrix::↩ ArgumentProperty >, 89

characteristic
argumatrix::AdmissiblePlReasoner, 12, 13
argumatrix::CompletePlReasoner, 40
argumatrix::ConflictfreePlReasoner, 58
argumatrix::GroundedReasoner, 78
argumatrix::PlReasoner, 94
argumatrix::PreferredReasoner, 113
argumatrix::Reasoner, 125
argumatrix::StablePlReasoner, 138
computeExtensions
argumatrix::PreferredReasoner, 113
consultPlFile
argumatrix::AdmissiblePlReasoner, 13
argumatrix::CompletePlReasoner, 40
argumatrix::ConflictfreePlReasoner, 58
argumatrix::PlReasoner, 94
argumatrix::StablePlReasoner, 139
createPlAttackMatrix
argumatrix::AdmissiblePlReasoner, 13
argumatrix::CompletePlReasoner, 41
argumatrix::ConflictfreePlReasoner, 59
argumatrix::PlReasoner, 95
argumatrix::StablePlReasoner, 139

diag
argumatrix::bitmatrix, 33

EmptySet
argumatrix::bitvector, 33

F:/Codespace/BoostProject/argumatrix/argmat-clpb/↩ ClpbProblem.hpp, 153
fetchAllExts
argumatrix::AdmissiblePlReasoner, 13
argumatrix::CompletePlReasoner, 41
argumatrix::ConflictfreePlReasoner, 59
argumatrix::PlReasoner, 95
argumatrix::StablePlReasoner, 139
findAllExts
argumatrix::AdmissiblePlReasoner, 14
argumatrix::CompletePlReasoner, 41, 42
argumatrix::ConflictfreePlReasoner, 59, 60
argumatrix::PlReasoner, 95
argumatrix::StablePlReasoner, 140

getArgumentProperty
argumatrix::DungAF, 74
getAttacked
argumatrix::AdmissiblePlReasoner, 14
argumatrix::CompletePlReasoner, 42
argumatrix::ConflictfreePlReasoner, 60
argumatrix::GroundedReasoner, 79
argumatrix::PlReasoner, 95
argumatrix::PreferredReasoner, 113
argumatrix::Reasoner, 126
argumatrix::StablePlReasoner, 140
getAttackers
argumatrix::PreferredReasoner, 114
getBvExtensions
argumatrix::AdmissiblePlReasoner, 15
argumatrix::CompletePlReasoner, 42
argumatrix::ConflictfreePlReasoner, 60
argumatrix::GroundedReasoner, 79
argumatrix::PlReasoner, 96
argumatrix::PreferredReasoner, 114
argumatrix::Reasoner, 126
argumatrix::StablePlReasoner, 141
getGroundedExtension
argumatrix::AdmissiblePlReasoner, 15
argumatrix::CompletePlReasoner, 42
argumatrix::ConflictfreePlReasoner, 60
argumatrix::GroundedReasoner, 79
argumatrix::PlReasoner, 96
argumatrix::PreferredReasoner, 114
argumatrix::Reasoner, 126
argumatrix::StablePlReasoner, 141
getGroundedIntVector
argumatrix::AdmissiblePlReasoner, 15
argumatrix::CompletePlReasoner, 43
argumatrix::ConflictfreePlReasoner, 61
argumatrix::GroundedReasoner, 79
argumatrix::PlReasoner, 96
argumatrix::PreferredReasoner, 114
argumatrix::Reasoner, 126
argumatrix::StablePlReasoner, 141
getSelfAttackingArguments

argumatrix::AdmissiblePlReasoner, 15
argumatrix::CompletePlReasoner, 43
argumatrix::ConflictfreePlReasoner, 61
argumatrix::GroundedReasoner, 80
argumatrix::PlReasoner, 96
argumatrix::PreferredReasoner, 115
argumatrix::Reasoner, 127
argumatrix::StablePlReasoner, 141

Increase
argumatrix::bitvector, 34
is_acceptable
argumatrix::AdmissiblePlReasoner, 16
argumatrix::CompletePlReasoner, 43
argumatrix::ConflictfreePlReasoner, 61
argumatrix::GroundedReasoner, 80
argumatrix::PlReasoner, 97
argumatrix::PreferredReasoner, 115
argumatrix::Reasoner, 127
argumatrix::StablePlReasoner, 142
is_admissible
argumatrix::AdmissiblePlReasoner, 16
argumatrix::CompletePlReasoner, 43
argumatrix::ConflictfreePlReasoner, 61
argumatrix::GroundedReasoner, 80
argumatrix::PlReasoner, 97
argumatrix::PreferredReasoner, 115
argumatrix::Reasoner, 127
argumatrix::StablePlReasoner, 142
is_complete
argumatrix::AdmissiblePlReasoner, 16
argumatrix::CompletePlReasoner, 44
argumatrix::ConflictfreePlReasoner, 62
argumatrix::GroundedReasoner, 80
argumatrix::PlReasoner, 97
argumatrix::PreferredReasoner, 115
argumatrix::Reasoner, 127
argumatrix::StablePlReasoner, 142
is_conflict_free
argumatrix::AdmissiblePlReasoner, 16
argumatrix::CompletePlReasoner, 44
argumatrix::ConflictfreePlReasoner, 62
argumatrix::GroundedReasoner, 81
argumatrix::PlReasoner, 97
argumatrix::PreferredReasoner, 116
argumatrix::Reasoner, 128
argumatrix::StablePlReasoner, 142
is_emptyset
argumatrix::bitvector, 34
is_grounded
argumatrix::AdmissiblePlReasoner, 17
argumatrix::CompletePlReasoner, 44
argumatrix::ConflictfreePlReasoner, 62
argumatrix::GroundedReasoner, 81
argumatrix::PlReasoner, 98
argumatrix::PreferredReasoner, 116
argumatrix::Reasoner, 128
argumatrix::StablePlReasoner, 143
is_self_attacking
argumatrix::AdmissiblePlReasoner, 17
argumatrix::CompletePlReasoner, 44
argumatrix::ConflictfreePlReasoner, 62
argumatrix::GroundedReasoner, 81
argumatrix::PlReasoner, 98
argumatrix::PreferredReasoner, 116
argumatrix::Reasoner, 128
argumatrix::StablePlReasoner, 143
is_stable
argumatrix::AdmissiblePlReasoner, 17
argumatrix::CompletePlReasoner, 45
argumatrix::ConflictfreePlReasoner, 63
argumatrix::GroundedReasoner, 81
argumatrix::PlReasoner, 98
argumatrix::PreferredReasoner, 116
argumatrix::Reasoner, 128
argumatrix::StablePlReasoner, 143
isCredulouslyJustified
argumatrix::AdmissiblePlReasoner, 17
argumatrix::CompletePlReasoner, 45
argumatrix::ConflictfreePlReasoner, 63
argumatrix::PlReasoner, 98
argumatrix::StablePlReasoner, 143
isSkepticallyJustified
argumatrix::AdmissiblePlReasoner, 18
argumatrix::CompletePlReasoner, 45
argumatrix::ConflictfreePlReasoner, 63
argumatrix::PlReasoner, 98
argumatrix::StablePlReasoner, 144

labelSet2IntVector
argumatrix::AdmissiblePlReasoner, 18
argumatrix::CompletePlReasoner, 45
argumatrix::ConflictfreePlReasoner, 63
argumatrix::GroundedReasoner, 82
argumatrix::PlReasoner, 99
argumatrix::PreferredReasoner, 117
argumatrix::Reasoner, 129
argumatrix::StablePlReasoner, 144

m_BmAtkMtx
argumatrix::AdmissiblePlReasoner, 27
argumatrix::CompletePlReasoner, 54
argumatrix::ConflictfreePlReasoner, 73
argumatrix::GroundedReasoner, 87
argumatrix::PlReasoner, 107
argumatrix::PreferredReasoner, 122
argumatrix::Reasoner, 134
argumatrix::StablePlReasoner, 152
m_argLabels
argumatrix::AdmissiblePlReasoner, 27
argumatrix::CompletePlReasoner, 53
argumatrix::ConflictfreePlReasoner, 72
argumatrix::GroundedReasoner, 87
argumatrix::PlReasoner, 106
argumatrix::PreferredReasoner, 122
argumatrix::Reasoner, 134
argumatrix::StablePlReasoner, 152
m_argNum

argumatrix::AdmissiblePlReasoner, 27
argumatrix::CompletePlReasoner, 53
argumatrix::ConflictfreePlReasoner, 72
argumatrix::GroundedReasoner, 87
argumatrix::PlReasoner, 106
argumatrix::PreferredReasoner, 122
argumatrix::Reasoner, 134
argumatrix::StablePlReasoner, 152
m_daf
argumatrix::AdmissiblePlReasoner, 27
argumatrix::CompletePlReasoner, 54
argumatrix::ConflictfreePlReasoner, 73
argumatrix::GroundedReasoner, 88
argumatrix::PlReasoner, 107
argumatrix::PreferredReasoner, 122
argumatrix::Reasoner, 134
argumatrix::StablePlReasoner, 152
m_output
argumatrix::AdmissiblePlReasoner, 28
argumatrix::CompletePlReasoner, 54
argumatrix::ConflictfreePlReasoner, 73
argumatrix::GroundedReasoner, 88
argumatrix::PlReasoner, 107
argumatrix::PreferredReasoner, 122
argumatrix::Reasoner, 134
argumatrix::StablePlReasoner, 153

neutrality
argumatrix::AdmissiblePlReasoner, 18
argumatrix::CompletePlReasoner, 45
argumatrix::ConflictfreePlReasoner, 63
argumatrix::GroundedReasoner, 82
argumatrix::PlReasoner, 99
argumatrix::PreferredReasoner, 117
argumatrix::Reasoner, 129
argumatrix::StablePlReasoner, 144
next_solution
argumatrix::PlTimeQry, 110

operator∗
argumatrix::bitvector, 34
outputBv
argumatrix::DungAF, 75
outputBvSet
argumatrix::DungAF, 75

parseAddtionalParams
argumatrix::ClpbProblem, 35
PlTimeQry
argumatrix::PlTimeQry, 108
printAllExts
argumatrix::AdmissiblePlReasoner, 18, 19
argumatrix::CompletePlReasoner, 46
argumatrix::ConflictfreePlReasoner, 64
argumatrix::PlReasoner, 99, 100
argumatrix::StablePlReasoner, 144, 145
printBvExts
argumatrix::AdmissiblePlReasoner, 19
argumatrix::CompletePlReasoner, 46
argumatrix::ConflictfreePlReasoner, 64
argumatrix::GroundedReasoner, 82
argumatrix::PlReasoner, 100
argumatrix::PreferredReasoner, 117
argumatrix::Reasoner, 129
argumatrix::StablePlReasoner, 145
printGroundedExt
argumatrix::AdmissiblePlReasoner, 19
argumatrix::CompletePlReasoner, 47
argumatrix::ConflictfreePlReasoner, 65
argumatrix::GroundedReasoner, 83
argumatrix::PlReasoner, 100
argumatrix::PreferredReasoner, 118
argumatrix::Reasoner, 130
argumatrix::StablePlReasoner, 145
printLabSet
argumatrix::AdmissiblePlReasoner, 20, 21
argumatrix::CompletePlReasoner, 48
argumatrix::ConflictfreePlReasoner, 66
argumatrix::GroundedReasoner, 83
argumatrix::PlReasoner, 101
argumatrix::PreferredReasoner, 118
argumatrix::Reasoner, 130
argumatrix::StablePlReasoner, 146, 147
printLableExtByBlistTerm
argumatrix::AdmissiblePlReasoner, 20
argumatrix::CompletePlReasoner, 47
argumatrix::ConflictfreePlReasoner, 65
argumatrix::PlReasoner, 100
argumatrix::StablePlReasoner, 146
printLableExtByBmatrixTerm
argumatrix::AdmissiblePlReasoner, 20
argumatrix::CompletePlReasoner, 47
argumatrix::ConflictfreePlReasoner, 65
argumatrix::PlReasoner, 101
argumatrix::StablePlReasoner, 146
printSomeExt
argumatrix::AdmissiblePlReasoner, 21
argumatrix::CompletePlReasoner, 48
argumatrix::ConflictfreePlReasoner, 66
argumatrix::PlReasoner, 102
argumatrix::StablePlReasoner, 147

set2bv
argumatrix::DungAF, 75
setOutput
argumatrix::AdmissiblePlReasoner, 21, 22
argumatrix::CompletePlReasoner, 49
argumatrix::ConflictfreePlReasoner, 67
argumatrix::GroundedReasoner, 84
argumatrix::PlReasoner, 102, 103
argumatrix::PreferredReasoner, 119
argumatrix::Reasoner, 131
argumatrix::StablePlReasoner, 147, 148

task_DC
argumatrix::AdmissiblePlReasoner, 22
argumatrix::CompletePlReasoner, 50
argumatrix::ConflictfreePlReasoner, 69

argumatrix::GroundedReasoner, 84
argumatrix::StablePlReasoner, 148
task_DE
argumatrix::AdmissiblePlReasoner, 22
argumatrix::CompletePlReasoner, 50
argumatrix::ConflictfreePlReasoner, 69
argumatrix::GroundedReasoner, 85
argumatrix::PlReasoner, 103
argumatrix::PreferredReasoner, 119
argumatrix::Reasoner, 131
argumatrix::StablePlReasoner, 148
task_DN
argumatrix::AdmissiblePlReasoner, 23
argumatrix::CompletePlReasoner, 50
argumatrix::ConflictfreePlReasoner, 69
argumatrix::GroundedReasoner, 85
argumatrix::PlReasoner, 103
argumatrix::PreferredReasoner, 120
argumatrix::Reasoner, 132
argumatrix::StablePlReasoner, 149
task_DS
argumatrix::AdmissiblePlReasoner, 23
argumatrix::CompletePlReasoner, 50
argumatrix::ConflictfreePlReasoner, 69
argumatrix::GroundedReasoner, 85
argumatrix::StablePlReasoner, 149
task_EC
argumatrix::AdmissiblePlReasoner, 23
argumatrix::CompletePlReasoner, 51
argumatrix::ConflictfreePlReasoner, 70
argumatrix::GroundedReasoner, 86
argumatrix::PlReasoner, 104
argumatrix::PreferredReasoner, 120
argumatrix::Reasoner, 132
argumatrix::StablePlReasoner, 149
task_EE
argumatrix::AdmissiblePlReasoner, 24
argumatrix::CompletePlReasoner, 51
argumatrix::ConflictfreePlReasoner, 70
argumatrix::GroundedReasoner, 86
argumatrix::PlReasoner, 104
argumatrix::PreferredReasoner, 121
argumatrix::Reasoner, 133
argumatrix::StablePlReasoner, 150
task_SC
argumatrix::AdmissiblePlReasoner, 24
argumatrix::CompletePlReasoner, 51
argumatrix::ConflictfreePlReasoner, 70
argumatrix::GroundedReasoner, 86
argumatrix::PlReasoner, 104
argumatrix::PreferredReasoner, 121
argumatrix::Reasoner, 133
argumatrix::StablePlReasoner, 150
task_SE
argumatrix::AdmissiblePlReasoner, 24
argumatrix::CompletePlReasoner, 52
argumatrix::ConflictfreePlReasoner, 71
argumatrix::GroundedReasoner, 87
argumatrix::PlReasoner, 105
argumatrix::PreferredReasoner, 121
argumatrix::Reasoner, 133
argumatrix::StablePlReasoner, 150
TrivialGraph2DungAF
argumatrix::parser, 90

verifyExclusion
argumatrix::AdmissiblePlReasoner, 26
argumatrix::CompletePlReasoner, 52
argumatrix::ConflictfreePlReasoner, 71
argumatrix::PlReasoner, 105
argumatrix::StablePlReasoner, 151
verifyExtension
argumatrix::AdmissiblePlReasoner, 26
argumatrix::CompletePlReasoner, 52
argumatrix::ConflictfreePlReasoner, 71
argumatrix::PlReasoner, 105
argumatrix::StablePlReasoner, 151
verifyInclusion
argumatrix::AdmissiblePlReasoner, 26
argumatrix::CompletePlReasoner, 53
argumatrix::ConflictfreePlReasoner, 72
argumatrix::PlReasoner, 106
argumatrix::StablePlReasoner, 151
verifyNonemptyExt
argumatrix::AdmissiblePlReasoner, 27
argumatrix::CompletePlReasoner, 53
argumatrix::ConflictfreePlReasoner, 72
argumatrix::PlReasoner, 106
argumatrix::StablePlReasoner, 152