

librv

1.0rc1

Generated by Doxygen 1.5.8

Mon Feb 23 14:43:07 2009

Contents

1	librv documentation	1
1.1	Introduction	1
1.2	Installation	1
1.3	Licence	1
1.4	Authors	1
2	File Index	3
2.1	File List	3
3	File Documentation	5
3.1	rv-DAG.h File Reference	5
3.1.1	Detailed Description	12
3.1.2	Define Documentation	13
3.1.2.1	boolean_connector	13
3.1.2.2	boolean_constant	13
3.1.2.3	quantifier	14
3.1.2.4	arith_function	14
3.1.3	Typedef Documentation	14
3.1.3.1	Tsort	14
3.1.3.2	Tsymb_type	15
3.1.4	Enumeration Type Documentation	15
3.1.4.1	Esymb_type	15
3.1.5	Function Documentation	15
3.1.5.1	DAG_init	15
3.1.5.2	DAG_done	16
3.1.5.3	DAG_sort_new	16
3.1.5.4	DAG_sort_new_args	16
3.1.5.5	DAG_sort_lookup	16
3.1.5.6	DAG_sort_name	17

3.1.5.7	DAG_sort_arity	17
3.1.5.8	DAG_sort_sub	17
3.1.5.9	DAG_sort_predefined	17
3.1.5.10	DAG_sort_com	18
3.1.5.11	DAG_symb_new	18
3.1.5.12	DAG_symb_lookup	18
3.1.5.13	DAG_symb_name	18
3.1.5.14	DAG_symb_type	19
3.1.5.15	DAG_symb_sort	19
3.1.5.16	DAG_symb_set_bind_DAG	19
3.1.5.17	DAG_symb_get_bind_DAG	19
3.1.5.18	DAG_symb_set_interpreted	19
3.1.5.19	DAG_symb_interpreted	20
3.1.5.20	DAG_symb_skolem	20
3.1.5.21	DAG_symb_variable	20
3.1.5.22	DAG_symb_predicate	20
3.1.5.23	DAG_new	20
3.1.5.24	DAG_new_args	21
3.1.5.25	DAG_new_integer	21
3.1.5.26	DAG_new_rational	21
3.1.5.27	DAG_new_integer_str	22
3.1.5.28	DAG_new_float_str	22
3.1.5.29	DAG_new_rational_str	22
3.1.5.30	DAG_dup	22
3.1.5.31	DAG_free	23
3.1.5.32	DAG_count_nodes	23
3.1.5.33	DAG_count_nodes_tree	23
3.1.5.34	DAG_count_atoms	23
3.1.5.35	DAG_subst	23
3.1.5.36	DAG_subst_multiple	24
3.1.5.37	DAG_contain	24
3.2	rv-status.h File Reference	25
3.2.1	Detailed Description	25
3.2.2	Enumeration Type Documentation	25
3.2.2.1	Estatus	25
3.3	rv.h File Reference	26

3.3.1	Detailed Description	26
3.3.2	Function Documentation	26
3.3.2.1	rv_init	26
3.3.2.2	rv_done	26
3.3.2.3	rv_interactive	27
3.3.2.4	rv_add_formula	27
3.3.2.5	rv_solve	27
3.3.2.6	rv_status	27
3.3.2.7	rv_reset	27
3.3.2.8	rv_get_model	28

Chapter 1

librv documentation

1.1 Introduction

The librv library provides a C-language API to access harvey's Satisfiability Modulo Theory (SMT) solver.

The solver is currently integrating decision procedures for the following: uninterpreted functions with equality, difference arithmetics (integers and reals). The superposition-based prover E is integrated as a fall-back verification engine for verification conditions with a unique sort. The solver integrates some level of quantifier reasoning, using Skolemization and instantiation heuristics.

1.2 Installation

...

1.3 Licence

The library uses third-party components and, as such, is subject to some constraints. To relieve our potential users from such constraints we are providing librv under two licenses BSD and LGPL.

The functionality between these licenses is the same.

LGPL-librv has arbitrary precision arithmetics.

BSD-librv has fixed precision arithmetics, using native data types. Should overflow occur at run time, they are detected and an error is reported.

1.4 Authors

Pascal Fontaine, David Deharbe are the two main developers. Diego Caminha has developed the difference logic verification engine and contributed to the design of the combination schema of the different "little engines" for reasoning. Thomas Bouton has contributed improvements to the interaction with the Boolean satisfiability engine as well as the QA infrastructure.

This library is a product of the collaboration between the [ForAll](#) group at [Universidade Federal do Rio Grande do Norte](#) (Brazil) and the [Mosel](#) group at [Université Nancy 2](#) and [LORIA](#)

(France).

Chapter 2

File Index

2.1 File List

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

rv-DAG.h (API to build formulas, terms, symbols and sorts to use with librv)	5
rv-status.h (Proof status in librv)	25
rv.h (API for the SMT solver VERI(T))	26

Chapter 3

File Documentation

3.1 rv-DAG.h File Reference

API to build formulas, terms, symbols and sorts to use with librv.

```
#include <stdarg.h>
```

Defines

- #define `boolean_connector(c)`
Tests if symbol `c` is a boolean connector.
- #define `boolean_constant(c)`
Tests if symbol `c` is a boolean constant.
- #define `quantifier(c)`
Tests if symbol `c` is a quantifier.
- #define `arith_function(c)`
Tests if symbol `c` is an arithmetic operator.
- #define `DAG_eq(A, B) DAG_new_args(PREDICATE_EQ, A, B, NULL)`
Conveniency macro constructor for equalities.
- #define `DAG_not(A) DAG_new_args(CONNECTOR_NOT, A, NULL)`
Conveniency macro constructor for negations.

Typedefs

- typedef struct TSort * `Tsort`
Type for sorts of symbols and DAGs.
- typedef struct TSymb * `Tsymb`
The type for symbol representation.

- typedef struct TSDAG * [TDAG](#)
The type for terms and formulas representation.
- typedef enum [Esymb_type](#) [Tsymb_type](#)

Enumerations

- enum [Esymb_type](#) {
 [SYMB_UNKNOWN](#),
 [SYMB_PREDICATE](#),
 [SYMB_FUNCTION](#),
 [SYMB_VARIABLE](#),
 [SYMB_ITE_FUNC](#),
 [SYMB_ITE](#),
 [SYMB_QUANTIFIER](#),
 [SYMB_BOOLEAN](#),
 [SYMB_BOOLEAN_CONSTANT](#),
 [SYMB_INTEGER](#),
 [SYMB_RATIONAL](#),
 [SYMB_LAMBDA](#),
 [SYMB_APPLY](#),
 [SYMB_SKOLEM](#),
 [SYMB_MACRO](#),
 [SYMB_TYPE_NB](#) }
the different kinds of symbols.

Functions

- void [DAG_init](#) (void)
Initializes rv-DAG module.
- void [DAG_done](#) (void)
Closes rv-DAG module, frees all allocated data structures.
- [Tsort](#) [DAG_sort_new](#) (const char *name, int arity, [Tsort](#) *sub)
sort constructor
- [Tsort](#) [DAG_sort_new_args](#) (const char *name, int arity,...)
- [Tsort](#) [DAG_sort_lookup](#) (const char *name)
Gets sort with name.
- char * [DAG_sort_name](#) ([Tsort](#) sort)
Accesses the name of the sort.

- int [DAG_sort_arity](#) (Tsort sort)
Accesses the arity of the sort.
- Tsort [DAG_sort_sub](#) (Tsort sort, int i)
Accesses the i-th sub-sort of the sort.
- int [DAG_sort_predefined](#) (Tsort sort)
Tests if sort is predefined.
- Tsort [DAG_sort_com](#) (Tsort sort1, Tsort sort2)
- Tsymb [DAG_symb_new](#) (const char *name, Tsymb_type type, Tsort sort)
Constructor.
- Tsymb [DAG_symb_lookup](#) (char *name, int nb_arg, TDAG *PDAG, Tsort sort)
Gets symbol with given name.
- char * [DAG_symb_name](#) (Tsymb symb)
Accesses the name of the symbol.
- Tsymb_type [DAG_symb_type](#) (Tsymb symb)
Accesses the kind of the symbol.
- Tsort [DAG_symb_sort](#) (Tsymb symb)
Accesses the sort of the symbol.
- void [DAG_symb_set_bind_DAG](#) (Tsymb symb, TDAG DAG)
Associates a DAG to symbol.
- TDAG [DAG_symb_get_bind_DAG](#) (Tsymb symb)
Gets DAG associated DAG to symbol.
- void [DAG_symb_set_interpreted](#) (Tsymb symb)
Marks symbol as being interpreted.
- int [DAG_symb_interpreted](#) (Tsymb symb)
Tests if symbol is interpreted.
- Tsymb [DAG_symb_skolem](#) (Tsort sort)
Specialized constructor.
- Tsymb [DAG_symb_variable](#) (Tsort sort)
Specialized constructor.
- Tsymb [DAG_symb_predicate](#) (Tsort sort)
Specialized constructor.
- TDAG [DAG_new](#) (Tsymb symb, int arity, TDAG *PDAG)
DAG constructor.

- [TDAG DAG_new_args](#) (Tsymp symb,...)
DAG constructor.
- [TDAG DAG_new_integer](#) (long value)
DAG constructor.
- [TDAG DAG_new_rational](#) (long num, long den)
DAG constructor.
- [TDAG DAG_new_integer_str](#) (char *value)
DAG constructor.
- [TDAG DAG_new_float_str](#) (char *value)
DAG constructor.
- [TDAG DAG_new_rational_str](#) (char *value)
DAG constructor.
- [TDAG DAG_dup](#) (TDAG DAG)
Reference counter increment.
- void [DAG_free](#) (TDAG DAG)
Destructor.
- int [DAG_count_nodes](#) (TDAG DAG)
DAG size.
- int [DAG_count_nodes_tree](#) (TDAG DAG)
Expanded DAG size.
- int [DAG_count_atoms](#) (TDAG DAG)
Expanded DAG boolean size.
- [TDAG DAG_subst](#) (TDAG src, TDAG origin, TDAG subst)
Simple substitution.
- [TDAG DAG_subst_multiple](#) (TDAG src, int n, TDAG *origin, TDAG *subst)
Multiple substitution.
- int [DAG_contain](#) (TDAG src, TDAG find)
Tests DAG inclusion.
- int [DAG_is_rational](#) (TDAG DAG)
Tests if DAG is a rational literal.
- int [DAG_is_integer](#) (TDAG DAG)
Tests if DAG is an integer literal.
- int [DAG_is_number](#) (TDAG DAG)
Tests if DAG is a numeric literal.

- int [DAG_is_atom](#) (TDAG DAG)
Tests if DAG is a boolean atom.
- Tsort [DAG_sort](#) (TDAG DAG)
Returns the sort of the DAG.

Variables

- const char * [SORT_BOOLEAN_STR](#)
Name of predefined boolean sort.
- const char * [SORT_INTEGER_STR](#)
Name of predefined integer sort.
- const char * [SORT_REAL_STR](#)
Name of predefined real/rational sort.
- const char * [SORT_UNINTERPRETED_STR](#)
Name of predefined sort for uninterpreted values.
- const char * [SORT_ARRAY_STR](#)
Name of predefined array sort.
- Tsort [SORT_BOOLEAN](#)
Predefined boolean sort.
- Tsort [SORT_INTEGER](#)
Predefined integer sort.
- Tsort [SORT_REAL](#)
Predefined real/rational sort.
- Tsort [SORT_UNINTERPRETED](#)
Predefined sort for uninterpreted values.
- Tsort [SORT_ARRAY](#)
Predefined array sort.
- const char * [BOOLEAN_TRUE_STR](#)
String for the predefined symbol TRUE.
- const char * [BOOLEAN_FALSE_STR](#)
String for the predefined symbol FALSE.
- Tsymb [BOOLEAN_TRUE](#)
Predefined symbol for boolean constant TRUE.

- **Tsymb BOOLEAN_FALSE**
Predefined symbol for boolean constant FALSE.
- const char * **CONNECTOR_NOT_STR**
Name of predefined negation symbol.
- const char * **CONNECTOR_OR_STR**
Name of predefined disjunction symbol.
- const char * **CONNECTOR_XOR_STR**
Name of predefined exclusive disjunction symbol.
- const char * **CONNECTOR_AND_STR**
Name of predefined conjunction symbol.
- const char * **CONNECTOR_IMPLIES_STR**
Name of predefined implication symbol.
- const char * **CONNECTOR_EQUIV_STR**
Name of predefined equivalence symbol.
- const char * **CONNECTOR_ITE_STR**
Name of predefined (boolean) conditional symbol.
- **Tsymb CONNECTOR_NOT**
Predefined symbol for negation.
- **Tsymb CONNECTOR_OR**
Predefined symbol for disjunction (n-ary).
- **Tsymb CONNECTOR_XOR**
Predefined symbol for exclusive or (n-ary).
- **Tsymb CONNECTOR_AND**
Predefined symbol for conjunction (n-ary).
- **Tsymb CONNECTOR_IMPLIES**
Predefined symbol for implication.
- **Tsymb CONNECTOR_EQUIV**
Predefined symbol for equivalence.
- **Tsymb CONNECTOR_ITE**
Predefined symbol for (boolean) conditional.
- const char * **PREDICATE_EQ_STR**
String for the equality symbol.
- **Tsymb PREDICATE_EQ**
Predefined symbol for the equality operator.

- const char * [PREDICATE_LESS_STR](#)
String for the predefined relational symbol "smaller than".
- const char * [PREDICATE_LEQ_STR](#)
String for the predefined relational symbol "smaller or equal".
- [Tsymb PREDICATE_LESS](#)
Symbol for the strictly smaller than relational operator.
- [Tsymb PREDICATE_LEQ](#)
Symbol for the smaller or equal relational operator.
- const char * [FUNCTION_ITE_STR](#)
String of the predefined symbol for the functional conditional.
- [Tsymb FUNCTION_ITE](#)
Symbol for the functional operator IF THEN ELSE.
- const char * [FUNCTION_SUM_STR](#)
String of the predefined symbol for addition.
- const char * [FUNCTION_PROD_STR](#)
String of the predefined symbol for multiplication.
- const char * [FUNCTION_UNARY_MINUS_STR](#)
String of the predefined symbol for opposite.
- const char * [FUNCTION_MINUS_STR](#)
String of the predefined symbol for subtraction.
- const char * [FUNCTION_DIV_STR](#)
String of the predefined symbol for division.
- [Tsymb FUNCTION_SUM](#)
Predefined symbol for arithmetic sum (n-ary).
- [Tsymb FUNCTION_PROD](#)
Predefined symbol for arithmetic product (n-ary).
- [Tsymb FUNCTION_UNARY_MINUS](#)
Predefined symbol for arithmetic unary minus.
- [Tsymb FUNCTION_MINUS](#)
Predefined symbol for arithmetic binary minus.
- [Tsymb FUNCTION_DIV](#)
Predefined symbol for arithmetic division minus.
- const char * [QUANTIFIER_EXISTS_STR](#)

String of the predefined symbol for existential quantification.

- `const char * QUANTIFIER_FORALL_STR`
String of the predefined symbol for universal quantification.
- `Tsymb QUANTIFIER_EXISTS`
Predefined symbol for existential quantification.
- `Tsymb QUANTIFIER_FORALL`
Predefined symbol for universal quantification.
- `const char * LAMBDA_STR`
String of the predefined symbol for lambda abstraction.
- `const char * APPLY_LAMBDA_STR`
String of the predefined symbol for beta reduction.
- `Tsymb LAMBDA`
Predefined symbol for lambda-abstraction operator.
- `Tsymb APPLY_LAMBDA`
Predefined symbol for beta reduction.
- `const char * FUNCTION_SELECT_STR`
String of predefined symbol for array element selection.
- `const char * FUNCTION_STORE_STR`
String of predefined symbol for array element assignment.
- `Tsymb FUNCTION_SELECT`
Predefined symbol for array element selection.
- `Tsymb FUNCTION_STORE`
Predefined symbol for array element assignment.
- `TDAG DAG_TRUE`
Represents the formula TRUE.
- `TDAG DAG_FALSE`
Represents the formula FALSE.

3.1.1 Detailed Description

API to build formulas, terms, symbols and sorts to use with `librv`.

Author:

Pascal Fontaine

Type Tsort is for the representation of sorts, type Tsymb is for symbols, and type TDAG is for terms and formulas.

A term or formula is made of a symbol and the sub-terms or sub-formulas. A symbol has a sort. The sort for predicate and function symbols is a tuple of sorts, where the last element of the tuple is the sort of the result and the remaining elements of the type are the sorts of the parameters.

Some technical details:

- Formulas and terms are represented by DAGs.
- Maximal sharing is used (two identical DAGs are merged.)
- Facilities for sorts are provided.
- Facilities for symbols are provided. Each symbol has a sort.
- Declaring several times a same symbol is allowed, as long as declaration are coherent (i.e. the same).
- Using an undeclared symbol issues a warning, but no error message.
- DAGs are associated reference counters. For any used DAG, its reference counter should be greater than 0. DAGs are nonetheless created with a reference counter set to zero in the case of DAGs that are immediately set as subDAGs (think bottom-up construction of terms in a parser). In other situations, the `reference_counter` should be set explicitly to one by `DAG_dup`.

3.1.2 Define Documentation

3.1.2.1 #define boolean_connector(c)

Value:

```
((c == CONNECTOR_NOT) || \
  (c == CONNECTOR_OR) || \
  (c == CONNECTOR_XOR) || \
  (c == CONNECTOR_AND) || \
  (c == CONNECTOR_IMPLIES) || \
  (c == CONNECTOR_EQUIV) || \
  (c == CONNECTOR_ITE))
```

Tests if symbol `c` is a boolean connector.

Parameters:

`c` shall be an expression of type Tsymb.

3.1.2.2 #define boolean_constant(c)

Value:

```
((c == BOOLEAN_TRUE) || \
  (c == BOOLEAN_FALSE))
```

Tests if symbol *c* is a boolean constant.

Parameters:

c shall be an expression of type *Tsymb*.

3.1.2.3 #define quantifier(*c*)

Value:

```
((c == QUANTIFIER_EXISTS) || \
  (c == QUANTIFIER_FORALL))
```

Tests if symbol *c* is a quantifier.

Parameters:

c shall be an expression of type *Tsymb*.

3.1.2.4 #define arith_function(*c*)

Value:

```
((c == FUNCTION_SUM) || \
  (c == FUNCTION_PROD) || \
  (c == FUNCTION_UNARY_MINUS) || \
  (c == FUNCTION_MINUS) || \
  (c == FUNCTION_DIV))
```

Tests if symbol *c* is an arithmetic operator.

Parameters:

c shall be an expression of type *Tsymb*.

3.1.3 Typedef Documentation

3.1.3.1 typedef struct *TSsort** *TSort*

Type for sorts of symbols and DAGs.

Facilities for disjoint sorts. A sort is either

- scalar : This is just a name.
- functional : $\text{sort}_1 \times \dots \times \text{sort}_{\{n-1\}} \rightarrow \text{sort}_{\{n\}}$, with $n > 1$.
- n-ary : $\text{sort}_1 \times \dots \times \text{sort}_1 \rightarrow \text{sort}_2$

arity is respectively 0, *n*, -1 in those cases.

Compound sorts are "non-scalar" sorts.

There may be aliases (i.e. names) to sorts (that is, even the compound ones). An alias may be bound to only one sort. `DAG_sort_new(_args)` fails when using several times the same name. However, it will not fail when declaring the same sort twice, i.e. it fails only when declaring different sorts with the same name.

Any compound sort should be defined uniquely. That is two aliases should never refer to similar compound sorts.

Notice that arity (for a functional sort) is not the number of arguments of the function having such a sort, but the number of arguments + 1 since the domain is also taken into account.

Boolean sort is just as any other sort. Predicates are thus a special kind of function.

3.1.3.2 typedef enum Esymb_type Tsymb_type

Type for the different kinds of symbols

3.1.4 Enumeration Type Documentation

3.1.4.1 enum Esymb_type

the different kinds of symbols.

Enumerator:

SYMB_UNKNOWN Should never be.

SYMB_PREDICATE Identifies predicate symbols.

SYMB_FUNCTION Identifies function symbols.

SYMB_VARIABLE Quantified variables.

SYMB_ITE_FUNC ITE functor.

SYMB_ITE boolean-valued ITE.

SYMB_QUANTIFIER Universal or existential quantifiers.

SYMB_BOOLEAN Boolean connectors.

SYMB_BOOLEAN_CONSTANT True or False.

SYMB_INTEGER Integer numeric values.

SYMB_RATIONAL Rational numeric values.

SYMB_LAMBDA Lambda abstraction operator.

SYMB_APPLY Application of a lambda expression (beta reduction).

SYMB_SKOLEM Skolem constants.

SYMB_MACRO Macro symbols.

SYMB_TYPE_NB Should never be. Trick to known number of enumerators in `Tsymb_type` at compile time.

3.1.5 Function Documentation

3.1.5.1 void DAG_init (void)

Initializes rv-DAG module.

Module options must have been initialized before rv-DAG module.

3.1.5.2 void DAG_done (void)

Closes rv-DAG module, frees all allocated data structures.

Module options must be closed after rv-DAG module

3.1.5.3 Tsort DAG_sort_new (const char * name, int arity, Tsort * sub)

sort constructor

Parameters:

name pointer to string naming the sort (may be NULL)

arity number of sub-sorts in a compound sort; if arity is -1, then symbol of this sort may have any number of arguments of sort sub[0], and returns argument of sort sub[1].

sub array storing sub-sorts in compound sorts

Returns:

a new sort

Destructive for sub

3.1.5.4 Tsort DAG_sort_new_args (const char * name, int arity, ...)

Sort constructor

Parameters:

name pointer to string naming the sort (may be NULL)

arity number of sub-sorts in a compound sort; if arity is -1, then symbol of this sort may have any number of arguments of sort sub[0], and returns argument of sort sub[1].

... for compound sorts, sub-sorts are given as arguments, followed by NULL

Returns:

a new sort

3.1.5.5 Tsort DAG_sort_lookup (const char * name)

Gets sort with name.

Parameters:

name of the searched sort

Returns:

the sort named name, or NULL if not found

3.1.5.6 char* DAG_sort_name (Tsort *sort*)

Accesses the name of the sort.

Parameters:

sort

Returns:

a pointer to the name string of the sort, if declared and named, NULL otherwise.

3.1.5.7 int DAG_sort_arity (Tsort *sort*)

Accesses the arity of the sort.

Parameters:

sort

Returns:

the arity of the sort.

3.1.5.8 Tsort DAG_sort_sub (Tsort *sort*, int *i*)

Accesses the *i*-th sub-sort of the sort.

Parameters:

sort The accessed sort.

i The accessed position.

Precondition:

i is a valid sub-sort position.

Returns:

compound sort *i*. This routine may be used to access the elements of a functional sort.

3.1.5.9 int DAG_sort_predefined (Tsort *sort*)

Tests if sort is predefined.

Parameters:

sort

Returns:

0 if not predefined, non-zero otherwise.

3.1.5.10 Tsort DAG_sort_com (Tsort *sort1*, Tsort *sort2*)

get the sort that include the other when ordered sort

Parameters:

sort1

sort2 Mainly for Integer and Real. NULL if disjoint.

3.1.5.11 Tsymb DAG_symb_new (const char * *name*, Tsymb_type *type*, Tsort *sort*)

Constructor.

Parameters:

name string naming the symbol

type identifies the kind of symbols that needs to be created

sort the symbol sort

Returns:

returns the declared symbol

Declares a new symbol

3.1.5.12 Tsymb DAG_symb_lookup (char * *name*, int *nb_arg*, TDAG * *PDAG*, Tsort *sort*)

Gets symbol with given name.

Parameters:

name string naming the symbol

nb_arg the number of subterms

PDAG the array of *nb_arg* subterms (optional)

sort the symbols sort (optional)

Returns:

Returns the appropriate Tsymb for name, if declared, or NULL if zero or several symbol match.

PDAG and sort are used for taking the right symbol if name is overloaded.

3.1.5.13 char* DAG_symb_name (Tsymb *symb*)

Accesses the name of the symbol.

Parameters:

symb

Returns:

The name of symb

3.1.5.14 Tsymb_type DAG_symb_type (Tsymb *symb*)

Accesses the kind of the symbol.

Parameters:

symb

Returns:

The kind of symb

3.1.5.15 Tsort DAG_symb_sort (Tsymb *symb*)

Accesses the sort of the symbol.

Parameters:

symb

Returns:

The kind of symb

3.1.5.16 void DAG_symb_set_bind_DAG (Tsymb *symb*, TDAG *DAG*)

Associates a DAG to symbol.

Parameters:

symb

DAG This is useful for instantiations (think *symb* is a variable).

3.1.5.17 TDAG DAG_symb_get_bind_DAG (Tsymb *symb*)

Gets DAG associated DAG to symbol.

Parameters:

symb

Returns:

DAG associated to symb This is useful for instantiations (think *symb* is a variable).

3.1.5.18 void DAG_symb_set_interpreted (Tsymb *symb*)

Marks symbol as being interpreted.

Parameters:

symb a symbol

Macros need to be marked as interpreted symbols.

3.1.5.19 int DAG_symb_interpreted (Tsymb *symb*)

Tests if symbol is interpreted.

Parameters:

symb a symbol

Macros are considered interpreted symbols.

3.1.5.20 Tsymb DAG_symb_skolem (Tsort *sort*)

Specialized constructor.

Parameters:

sort sort of the symbol to create

Returns:

symbol of a fresh skolem symbol of the given sort

3.1.5.21 Tsymb DAG_symb_variable (Tsort *sort*)

Specialized constructor.

Parameters:

sort sort of the symbol to create

Returns:

symbol of a fresh variable of the given sort

3.1.5.22 Tsymb DAG_symb_predicate (Tsort *sort*)

Specialized constructor.

Parameters:

sort sort of the symbol to create

Returns:

symbol of a fresh predicate of the given sort

3.1.5.23 TDAG DAG_new (Tsymb *symb*, int *arity*, TDAG * *PDAG*)

DAG constructor.

Parameters:

symb topmost symbol

arity number of sub-terms

PDAG array of subterms

Returns:

Creates (if new) and returns TDAG from *symb* and *PDAG*.

Destructive for *PDAG*

3.1.5.24 TDAG DAG_new_args (T*symb symb*, ...)

DAG constructor.

Parameters:

symb The topmost symbol of the constructed term.

... subterms, followed by NULL.

Precondition:

The number of subterms needs to be compatible with the arity of *symb*.

Returns:

Creates (if new) and returns TDAG from *symb* and given arguments.

3.1.5.25 TDAG DAG_new_integer (long *value*)

DAG constructor.

Parameters:

value an integer

Returns:

Creates (if new) and returns DAG representing integer value.

3.1.5.26 TDAG DAG_new_rational (long *num*, long *den*)

DAG constructor.

Parameters:

num an integer interpreted as numerator

den an integer interpreted as denominator

Returns:

Creates (if new) and returns DAG representing rational *num/den*.

User is responsible for overflow, if using version with native data types.

3.1.5.27 TDAG DAG_new_integer_str (char * value)

DAG constructor.

Parameters:

value textual representation of an integer $0 \mid [1-9][0-9]^*$

Returns:

Creates (if new) and returns DAG representing integer value.

The given string is checked for conformance.

3.1.5.28 TDAG DAG_new_float_str (char * value)

DAG constructor.

Parameters:

value textual representation of a floating point $0.[0-9]^+ \mid [1-9][0-9]^*.[0-9]^+$

Returns:

Creates (if new) and returns DAG representing floating point value.

The given string is checked for conformance.

3.1.5.29 TDAG DAG_new_rational_str (char * value)

DAG constructor.

Parameters:

value textual representation of a rational $[1-9][0-9]^* / [0-9]^+[1-9]$ or $[1-9][0-9]^*$

Returns:

Creates (if new) and returns DAG representing rational value.

The given string is checked for conformance.

3.1.5.30 TDAG DAG_dup (TDAG DAG)

Reference counter increment.

Parameters:

DAG its reference counter will be incremented

Returns:

the result is the same as the argument.

3.1.5.31 void DAG_free (TDAG *DAG*)

Destructor.

Parameters:

DAG to be freed

The reference counter of DAG is decremented. If the resulting value is zero, then DAG is freed.

3.1.5.32 int DAG_count_nodes (TDAG *DAG*)

DAG size.

Parameters:

DAG to be measured.

Returns:

Number of nodes in DAG as a DAG representation.

3.1.5.33 int DAG_count_nodes_tree (TDAG *DAG*)

Expanded DAG size.

Parameters:

DAG to be measured.

Returns:

Number of nodes in DAG as a tree representation.

3.1.5.34 int DAG_count_atoms (TDAG *DAG*)

Expanded DAG boolean size.

Parameters:

DAG to be measured.

Returns:

Number of atoms in DAG as a tree representation.

3.1.5.35 TDAG DAG_subst (TDAG *src*, TDAG *origin*, TDAG *subst*)

Simple substitution.

Parameters:

src Term where the substitution is realized.

origin Term that will be substituted.

subst Term that will replace origin.

See also:

[DAG_subst_multiple](#)

3.1.5.36 TDAG DAG_subst_multiple (TDAG *src*, int *n*, TDAG * *origin*, TDAG * *subst*)

Multiple substitution.

Parameters:

src Term where the substitution is realized.

n The number of terms that will be substituted.

origin Array of *n* terms that will be substituted.

subst Array of *n* term that will replace substituted terms.

3.1.5.37 int DAG_contain (TDAG *src*, TDAG *find*)

Tests DAG inclusion.

Parameters:

src

find

Returns:

Zero if *find* is not a subterm of *src*, non-zero otherwise.

3.2 rv-status.h File Reference

Proof status in librv.

Typedefs

- typedef enum [Estatus](#) [Tstatus](#)
Type of proof status in librv.

Enumerations

- enum [Estatus](#) {
 [SAT](#),
 [UNSAT](#),
 [UNDEC](#),
 [OPEN](#) }
Enumeration of the different possible proof status in harvey.

3.2.1 Detailed Description

Proof status in librv.

Author:

David Deharbe and Pascal Fontaine

This file only contains the definition of the type of the proof status in the solver. This is in a separate file as the same status is also used in different internal proof engines.

3.2.2 Enumeration Type Documentation

3.2.2.1 enum Estatus

Enumeration of the different possible proof status in harvey.

Enumerator:

SAT satisfiable

UNSAT unsatisfiable

UNDEC undecided

The proof obligation is not within the theories where the solver is complete, and the solver was not able to show unsatisfiability.

OPEN not verified yet

Run `rv_solve` to (semi)decide

3.3 rv.h File Reference

API for the SMT solver VERI(T).

```
#include "rv-DAG.h"  
#include "rv-status.h"
```

Functions

- void [rv_init](#) (void)
- void [rv_done](#) (void)
- void [rv_interactive](#) (void)
- void [rv_add_formula](#) (TDAG formula)
- Tstatus [rv_solve](#) (void)
- Tstatus [rv_status](#) (void)
- void [rv_reset](#) (void)
- void [rv_get_model](#) (int *Pnb_literals, TDAG **Pliterals)

3.3.1 Detailed Description

API for the SMT solver VERI(T).

Author:

David Deharbe, Pascal Fontaine

This module provides satisfiability modulo theory (SMT) solving functionality and can be accessed from any tool requiring formal verification capabilities that fall within the scope of VERI(T), namely: quantified, or quantifier-free, first-order logic with uninterpreted functions, and difference arithmetics over integers or arithmetics.

The module requires that formulas representations with the TDAG type be built with rv-DAG API.

3.3.2 Function Documentation

3.3.2.1 void [rv_init](#) (void)

Initializes the module.

Must be called before any other function of the module.

3.3.2.2 void [rv_done](#) (void)

Closes down the module.

Frees all the memory allocated by module functions.

3.3.2.3 void rv_interactive (void)

use rv in interactive mode.

In non-interactive mode (default), only one rv_add_formula can occur. Some simplifications may only occur in non-interactive mode. In interactive mode, formulas can be added conjunctively using rv_add_formula. This action is irreversible.

3.3.2.4 void rv_add_formula (TDAG formula)

Adds conjunctively a formula in the input.

Parameters:

formula represents the formula to be checked for satisfiability.

Use this function to add conjunctively a formula in the input. Non-destructive for formula. These formulas are added in the context without proof. Caution: DO NOT USE FOR INTERNALLY-PRODUCED FORMULA (e.g. lemmas).

3.3.2.5 Tstatus rv_solve (void)

Runs solver on the formula(s)

This function runs the solver on the current set of formulas and returns the result:

- SAT: satisfiable,
- UNSAT: unsatisfiable,
- UNDECIDED: not within the complete fragment handled, and solver was not able to conclude.

3.3.2.6 Tstatus rv_status (void)

Returns the current status of the solver.

The status may be:

- SAT: satisfiable,
- UNSAT: unsatisfiable,
- UNDECIDED: not within the complete fragment handled, and solver was not able to conclude,
- OPEN: not verified yet, run rv_solve.

3.3.2.7 void rv_reset (void)

Resets module.

This function needs to be called to reset the library to add a new formula in non-interactive mode, or erase the added formulas, in interactive mode.

3.3.2.8 void rv_get_model (int * Pnb_literals, TDAG ** Pliterals)

Builds a model for the conjunction of formulas.

Precondition:

[rv_status\(\)](#) yields SAT.

Parameters:

Pnb_literals address where function stores the number of literals in the model

Pliterals address where function stores the model (an array of literals)

If status is SAT, *Pnb_literals gets the number of literals in model and *Pliterals is assigned an array of *Pnb_literals TDAG representing the FOL literals composing the model.

Index

- arith_function
 - rv-DAG.h, 14
- boolean_connector
 - rv-DAG.h, 13
- boolean_constant
 - rv-DAG.h, 13
- DAG_contain
 - rv-DAG.h, 24
- DAG_count_atoms
 - rv-DAG.h, 23
- DAG_count_nodes
 - rv-DAG.h, 23
- DAG_count_nodes_tree
 - rv-DAG.h, 23
- DAG_done
 - rv-DAG.h, 15
- DAG_dup
 - rv-DAG.h, 22
- DAG_free
 - rv-DAG.h, 22
- DAG_init
 - rv-DAG.h, 15
- DAG_new
 - rv-DAG.h, 20
- DAG_new_args
 - rv-DAG.h, 21
- DAG_new_float_str
 - rv-DAG.h, 22
- DAG_new_integer
 - rv-DAG.h, 21
- DAG_new_integer_str
 - rv-DAG.h, 21
- DAG_new_rational
 - rv-DAG.h, 21
- DAG_new_rational_str
 - rv-DAG.h, 22
- DAG_sort_arity
 - rv-DAG.h, 17
- DAG_sort_com
 - rv-DAG.h, 17
- DAG_sort_lookup
 - rv-DAG.h, 16
- DAG_sort_name
 - rv-DAG.h, 16
- DAG_sort_new
 - rv-DAG.h, 16
- DAG_sort_new_args
 - rv-DAG.h, 16
- DAG_sort_predefined
 - rv-DAG.h, 17
- DAG_sort_sub
 - rv-DAG.h, 17
- DAG_subst
 - rv-DAG.h, 23
- DAG_subst_multiple
 - rv-DAG.h, 24
- DAG_symb_get_bind_DAG
 - rv-DAG.h, 19
- DAG_symb_interpreted
 - rv-DAG.h, 19
- DAG_symb_lookup
 - rv-DAG.h, 18
- DAG_symb_name
 - rv-DAG.h, 18
- DAG_symb_new
 - rv-DAG.h, 18
- DAG_symb_predicate
 - rv-DAG.h, 20
- DAG_symb_set_bind_DAG
 - rv-DAG.h, 19
- DAG_symb_set_interpreted
 - rv-DAG.h, 19
- DAG_symb_skolem
 - rv-DAG.h, 20
- DAG_symb_sort
 - rv-DAG.h, 19
- DAG_symb_type
 - rv-DAG.h, 18
- DAG_symb_variable
 - rv-DAG.h, 20
- Estatus
 - rv-status.h, 25
- Esymb_type
 - rv-DAG.h, 15
- OPEN
 - rv-status.h, 25

- quantifier
 - rv-DAG.h, 14
- rv-DAG.h, 5
 - arith_function, 14
 - boolean_connector, 13
 - boolean_constant, 13
 - DAG_contain, 24
 - DAG_count_atoms, 23
 - DAG_count_nodes, 23
 - DAG_count_nodes_tree, 23
 - DAG_done, 15
 - DAG_dup, 22
 - DAG_free, 22
 - DAG_init, 15
 - DAG_new, 20
 - DAG_new_args, 21
 - DAG_new_float_str, 22
 - DAG_new_integer, 21
 - DAG_new_integer_str, 21
 - DAG_new_rational, 21
 - DAG_new_rational_str, 22
 - DAG_sort_arity, 17
 - DAG_sort_com, 17
 - DAG_sort_lookup, 16
 - DAG_sort_name, 16
 - DAG_sort_new, 16
 - DAG_sort_new_args, 16
 - DAG_sort_predefined, 17
 - DAG_sort_sub, 17
 - DAG_subst, 23
 - DAG_subst_multiple, 24
 - DAG_symb_get_bind_DAG, 19
 - DAG_symb_interpreted, 19
 - DAG_symb_lookup, 18
 - DAG_symb_name, 18
 - DAG_symb_new, 18
 - DAG_symb_predicate, 20
 - DAG_symb_set_bind_DAG, 19
 - DAG_symb_set_interpreted, 19
 - DAG_symb_skolem, 20
 - DAG_symb_sort, 19
 - DAG_symb_type, 18
 - DAG_symb_variable, 20
 - Esymb_type, 15
 - quantifier, 14
 - SYMB_APPLY, 15
 - SYMB_BOOLEAN, 15
 - SYMB_BOOLEAN_CONSTANT, 15
 - SYMB_FUNCTION, 15
 - SYMB_INTEGER, 15
 - SYMB_ITE, 15
 - SYMB_ITE_FUNC, 15
 - SYMB_LAMBDA, 15
 - SYMB_MACRO, 15
 - SYMB_PREDICATE, 15
 - SYMB_QUANTIFIER, 15
 - SYMB_RATIONAL, 15
 - SYMB_SKOLEM, 15
 - SYMB_TYPE_NB, 15
 - SYMB_UNKNOWN, 15
 - SYMB_VARIABLE, 15
 - Tsort, 14
 - Tsymb_type, 15
- rv-status.h, 25
 - Estatus, 25
 - OPEN, 25
 - SAT, 25
 - UNDEC, 25
 - UNSAT, 25
- rv.h, 26
 - rv_add_formula, 27
 - rv_done, 26
 - rv_get_model, 27
 - rv_init, 26
 - rv_interactive, 26
 - rv_reset, 27
 - rv_solve, 27
 - rv_status, 27
- rv_add_formula
 - rv.h, 27
- rv_done
 - rv.h, 26
- rv_get_model
 - rv.h, 27
- rv_init
 - rv.h, 26
- rv_interactive
 - rv.h, 26
- rv_reset
 - rv.h, 27
- rv_solve
 - rv.h, 27
- rv_status
 - rv.h, 27
- SAT
 - rv-status.h, 25
- SYMB_APPLY
 - rv-DAG.h, 15
- SYMB_BOOLEAN
 - rv-DAG.h, 15
- SYMB_BOOLEAN_CONSTANT
 - rv-DAG.h, 15
- SYMB_FUNCTION
 - rv-DAG.h, 15
- SYMB_INTEGER
 - rv-DAG.h, 15

SYMB_ITE
rv-DAG.h, [15](#)

SYMB_ITE_FUNC
rv-DAG.h, [15](#)

SYMB_LAMBDA
rv-DAG.h, [15](#)

SYMB_MACRO
rv-DAG.h, [15](#)

SYMB_PREDICATE
rv-DAG.h, [15](#)

SYMB_QUANTIFIER
rv-DAG.h, [15](#)

SYMB_RATIONAL
rv-DAG.h, [15](#)

SYMB_SKOLEM
rv-DAG.h, [15](#)

SYMB_TYPE_NB
rv-DAG.h, [15](#)

SYMB_UNKNOWN
rv-DAG.h, [15](#)

SYMB_VARIABLE
rv-DAG.h, [15](#)

Tsort
rv-DAG.h, [14](#)

Tsymb_type
rv-DAG.h, [15](#)

UNDEC
rv-status.h, [25](#)

UNSAT
rv-status.h, [25](#)