

libveriT
201107

Generated by Doxygen 1.7.4

Mon Jul 11 2011 10:34:13

Contents

1	libveriT documentation	1
1.1	Introduction	1
1.2	Installation	1
1.3	Licence	1
1.4	Authors	2
2	File Index	3
2.1	File List	3
3	File Documentation	5
3.1	veriT-DAG.h File Reference	5
3.1.1	Detailed Description	14
3.1.2	Define Documentation	15
3.1.2.1	boolean_connector	15
3.1.2.2	boolean_constant	15
3.1.2.3	quantifier	16
3.1.2.4	binder	16
3.1.2.5	arith_function	16
3.1.2.6	arith_comparison	17
3.1.2.7	unary_minus	17
3.1.3	Typedef Documentation	17
3.1.3.1	Tsort	17
3.1.3.2	Tsymb_type	18
3.1.3.3	Tpid	18
3.1.4	Enumeration Type Documentation	18
3.1.4.1	Esymb_type	18

3.1.5	Function Documentation	19
3.1.5.1	DAG_init	19
3.1.5.2	DAG_done	19
3.1.5.3	DAG_sort_new	19
3.1.5.4	DAG_sort_new_args	20
3.1.5.5	DAG_sort_new_var	20
3.1.5.6	DAG_sort_new_param	20
3.1.5.7	DAG_sort_new_inst	21
3.1.5.8	DAG_sort_lookup	21
3.1.5.9	DAG_sort_name	21
3.1.5.10	DAG_sort_arity	22
3.1.5.11	DAG_sort_sub	22
3.1.5.12	DAG_sort_predefined	22
3.1.5.13	DAG_sort_parametric	22
3.1.5.14	DAG_sort_polymorphic	23
3.1.5.15	DAG_sort_instance	23
3.1.5.16	DAG_sort_variable	23
3.1.5.17	DAG_sort_com	23
3.1.5.18	DAG_sort_subsumes	23
3.1.5.19	DAG_sort_substitute	24
3.1.5.20	DAG_sort_in_free_variables	24
3.1.5.21	DAG_symb_new	24
3.1.5.22	DAG_symb_lookup	25
3.1.5.23	DAG_symb_lookup_sort	25
3.1.5.24	DAG_symb_name	25
3.1.5.25	DAG_symb_type	26
3.1.5.26	DAG_symb_sort	26
3.1.5.27	DAG_symb_set_bind_DAG	26
3.1.5.28	DAG_symb_get_bind_DAG	26
3.1.5.29	DAG_symb_set_interpreted	27
3.1.5.30	DAG_symb_interpreted	27
3.1.5.31	DAG_symb_skolem	27
3.1.5.32	DAG_symb_const	27
3.1.5.33	DAG_symb_variable	27

3.1.5.34	DAG_symb_predicate	28
3.1.5.35	DAG_symb_prop_get	28
3.1.5.36	DAG_symb_prop_check	28
3.1.5.37	DAG_new	29
3.1.5.38	DAG_new_args	29
3.1.5.39	DAG_new_integer	29
3.1.5.40	DAG_new_rational	29
3.1.5.41	DAG_new_integer_str	30
3.1.5.42	DAG_new_decimal_str	30
3.1.5.43	DAG_new_binary_str	31
3.1.5.44	DAG_new_hex_str	31
3.1.5.45	DAG_new_rational_str	31
3.1.5.46	DAG_new_str	31
3.1.5.47	DAG_dup	32
3.1.5.48	DAG_free	32
3.1.5.49	DAG_count_nodes	32
3.1.5.50	DAG_count_nodes_tree	32
3.1.5.51	DAG_count_atoms	33
3.1.5.52	DAG_depth	33
3.1.5.53	DAG_subst_Pflag	33
3.1.5.54	DAG_subst	34
3.1.5.55	DAG_subst_multiple	34
3.1.5.56	DAG_contain	34
3.1.5.57	DAG_smtlib_logic_set	34
3.1.5.58	DAG_prop_new	35
3.1.5.59	DAG_prop_set	35
3.1.5.60	DAG_prop_remove	35
3.1.5.61	DAG_prop_get	35
3.1.5.62	DAG_prop_check	36
3.2	veriT-status.h File Reference	36
3.2.1	Detailed Description	36
3.2.2	Enumeration Type Documentation	37
3.2.2.1	Estatus	37
3.3	veriT.h File Reference	37

3.3.1	Detailed Description	38
3.3.2	Function Documentation	38
3.3.2.1	veriT_init	38
3.3.2.2	veriT_done	38
3.3.2.3	veriT_logic	39
3.3.2.4	veriT_push	39
3.3.2.5	veriT_pop	39
3.3.2.6	veriT_assert	39
3.3.2.7	veriT_check_sat	40
3.3.2.8	veriT_complete	40
3.3.2.9	veriT_status	40
3.3.2.10	veriT_interactive	41
3.3.2.11	veriT_add_formula	41
3.3.2.12	veriT_solve	41
3.3.2.13	veriT_reset	42
3.3.2.14	veriT_get_model	42

Chapter 1

libveriT documentation

1.1 Introduction

The libveriT library provides a C-language API to access veriT'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), linear arithmetics (integers and reals). The solver integrates some level of quantifier reasoning, using Skolemization and instantiation heuristics based on triggers. The superposition-based prover E is integrated as a fall-back verification engine for verification conditions with a unique sort.

1.2 Installation

There is currently no installation procedure available for using veriT as a library. From the distribution, all the src directory, but the parsers sub-directory and the main.c file, are required to package veriT as a library.

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 libveriT under two licenses BSD and LGPL.

The functionality between these licenses is the same.

LGPL-libveriT has arbitrary precision arithmetics.

BSD-libveriT 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:

veriT-DAG.h (API to build formulas, terms, symbols and sorts to use with lib- veriT)	5
veriT-status.h (Proof status in libveriT)	36
veriT.h (API for the SMT solver VERI(T))	37

Chapter 3

File Documentation

3.1 veriT-DAG.h File Reference

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

```
#include <stdarg.h>
#include <assoc.h>
#include "types.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 **binder**(c)
Tests if symbol c is a quantifier.
- #define **arith_function**(c)
Tests if symbol c is an arithmetic operator.
- #define **arith_symbol**(c) (arith_function(c))
- #define **arith_comparison**(c)
Tests if symbol c is a arithmetic comparison operator.
- #define **unary_minus**(c)
- #define **DAG_eq**(A, B) DAG_new_args(PREDICATE_EQ, A, B, NULL)
Conveniency macro constructor for equalities.
- #define **DAG_equiv**(A, B) DAG_new_args(CONNECTOR_EQUIV, A, B, NULL)
Conveniency macro constructor for equivalences.
- #define **DAG_not**(A) DAG_new_args(CONNECTOR_NOT, A, NULL)

Conveniency macro constructor for negations.

- #define `DAG_neq(A, B) DAG_new_args(CONNECTOR_NOT, DAG_new_args(PREDICATE_EQ, A, B, NULL), NULL)`

Conveniency macro constructor for negations of equality.

- #define `DAG_and2(A, B) DAG_new_args(CONNECTOR_AND, A, B, NULL)`

Conveniency macro constructor for binary conjunctions.

- #define `DAG_plus(A, B) DAG_new_args(FUNCTION_SUM, A, B, NULL)`

Conveniency macro constructor for addition.

- #define `DAG_minus(A, B) DAG_new_args(FUNCTION_MINUS, A, B, NULL)`

Conveniency macro constructor for subtraction.

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`
- typedef unsigned `Tpid`
type for property identifiers

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_BINARY,`
`SYMB_HEX,`
`SYMB_RATIONAL,`
`SYMB_STRING,`
`SYMB_LAMBDA,`

```

SYMB_LET,
SYMB_APPLY,
SYMB_SKOLEM,
SYMB_MACRO,
SYMB_TYPE_NB }

```

the different kinds of symbols.

Functions

- void [DAG_init](#) (void)
Initializes veriT-DAG module.
- void [DAG_done](#) (void)
Closes veriT-DAG module, frees all allocated data structures.
- [Tsort DAG_sort_new](#) (const char *name, int arity, [Tsort](#) *sub)
old sort constructor
- [Tsort DAG_sort_new_args](#) (const char *name, int arity,...)
- [Tsort DAG_sort_new_var](#) (char *name)
creates a sort variable with the given name
- [Tsort DAG_sort_new_param](#) (char *name, int arity)
creates a parametric sort constructor
- [Tsort DAG_sort_new_inst](#) (char *name, [Tsort](#) sort, int arity, [Tsort](#) *sub)
creates an instance of a parametric sort constructor
- [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, unsigned i)
Accesses the i-th sub-sort of the sort.
- int [DAG_sort_predefined](#) ([Tsort](#) sort)
Tests if sort is predefined.
- int [DAG_sort_parametric](#) ([Tsort](#) sort)
Tests if sort is parametric.
- int [DAG_sort_polymorphic](#) ([Tsort](#) sort)
Tests if sort is polymorphic.
- int [DAG_sort_instance](#) ([Tsort](#) sort)
Tests if sort is an instance of a parametric sort constructor.
- int [DAG_sort_variable](#) (const [Tsort](#) sort)
tests if a sort is a sort variable
- [Tsort DAG_sort_com](#) ([Tsort](#) sort1, [Tsort](#) sort2)
- int [DAG_sort_subsumes](#) ([Tsort](#) sort1, [Tsort](#) sort2)

Checks if sort1 subsumes sort2.

- [TDAG DAG_sort_substitute](#) (const [TDAG](#) src)
- bool [DAG_sort_in_free_variables](#) (const [Tsort](#) sort1, const [Tsort](#) sort2)
- Tassoc [sort_constraint_new](#) (const [Tsort](#) sort1, const [Tsort](#) sort2)
- [Tsymb DAG_symb_new](#) (const char *name, [Tsymb_type](#) type, [Tsort](#) sort)

Constructor.

- [Tsymb DAG_symb_lookup](#) (char *name, unsigned nb_arg, [TDAG](#) *PDAG, [Tsort](#) sort)

Gets symbol with given name.

- [Tsymb DAG_symb_lookup_sort](#) (char *name, [Tsort](#) sort)

Gets symbol with given name and sort.

- 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.

- bool [DAG_symb_interpreted](#) ([Tsymb](#) symb)

Tests if symbol is interpreted.

- [Tsymb DAG_symb_skolem](#) ([Tsort](#) sort)

Specialized constructor.

- [Tsymb DAG_symb_const](#) ([Tsort](#) sort)

Specialized constructor.

- [Tsymb DAG_symb_variable](#) ([Tsort](#) sort)

Specialized constructor.

- [Tsymb DAG_symb_predicate](#) ([Tsort](#) sort)

Specialized constructor.

- void [DAG_symb_reset_misc](#) ([Tsymb](#) symb)

Reset the misc field.

- void [DAG_symb_prop_set](#) ([Tsymb](#) symb, [Tpid](#) pid, void *value)

sets the value of a property

- int [DAG_symb_prop_remove](#) ([Tsymb](#) symb, [Tpid](#) pid)

remove the value of a property

- int [DAG_symb_prop_get](#) ([Tsymb](#) symb, [Tpid](#) pid, void ***P)

gets the value of a property

- int [DAG_symb_prop_check](#) ([Tsymb](#) symb, [Tpid](#) pid)

check if there is a value for a property

- [TDAG DAG_new](#) (Tsymp symb, unsigned 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_decimal_str](#) (char *value)
DAG constructor.
- [TDAG DAG_new_binary_str](#) (char *value)
DAG constructor.
- [TDAG DAG_new_hex_str](#) (char *value)
DAG constructor.
- [TDAG DAG_new_float_str](#) (char *value)
- [TDAG DAG_new_rational_str](#) (char *value)
DAG constructor.
- [TDAG DAG_new_str](#) (char *value)
DAG constructor.
- [TDAG DAG_dup](#) (TDAG DAG)
Reference counter increment.
- void [DAG_free](#) (TDAG DAG)
Destructor.
- unsigned [DAG_count_nodes](#) (TDAG DAG)
DAG size.
- int [DAG_count_nodes_tree](#) (TDAG DAG)
Expanded DAG size.
- unsigned [DAG_count_atoms](#) (TDAG DAG)
Expanded DAG boolean size.
- unsigned [DAG_depth](#) (TDAG DAG)
Depth of a DAG.
- int [DAG_subst_Pflag](#) (TDAG src)
substitutes every node by node->Pflag in a term
- [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.
- void `DAG_smtlib_logic_init` (void)
- void `DAG_smtlib_logic_set` (const char *str, int version)
Sets the logic.
- char * `DAG_smtlib_logic` (void)
- void `DAG_smtlib_logic_done` (void)
- Tpid `DAG_prop_new` (TFfree f)
creates a property identifier
- void `DAG_prop_set` (TDAG DAG, Tpid pid, void *value)
sets the value of a property
- int `DAG_prop_remove` (TDAG DAG, Tpid pid)
remove the value of a property
- int `DAG_prop_get` (TDAG DAG, Tpid pid, void ***P)
gets the value of a property
- int `DAG_prop_check` (TDAG DAG, Tpid pid)
check if there is a value for a property

Variables

- char * `SORT_BOOLEAN_STR`
Name of predefined boolean sort.
- char * `SORT_INTEGER_STR`
Name of predefined integer sort.
- char * `SORT_REAL_STR`
Name of predefined real/rational sort.
- char * `SORT_UNINTERPRETED_STR`
Name of predefined sort for uninterpreted values.
- char * `SORT_ARRAY_STR`
Name of predefined array sort.
- char * `SORT_ELEMENT_STR`
Name of predefined (array) element sort.
- char * `SORT_INDEX_STR`
Name of predefined (array) index 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.*

 - [Tsort SORT_ARRAY1](#)
- Predefined array sort.*

 - [Tsort SORT_ARRAY2](#)
- Predefined array sort.*

 - [Tsort SORT_ELEMENT](#)
- Predefined element sort.*

 - [Tsort SORT_INDEX](#)
- Predefined (array) index sort.*

 - char * [BOOLEAN_TRUE_STR](#)
String for the predefined symbol TRUE.
 - 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.
 - char * [CONNECTOR_NOT_STR](#)
Name of predefined negation symbol.
 - char * [CONNECTOR_OR_STR](#)
Name of predefined disjunction symbol.
 - char * [CONNECTOR_XOR_STR](#)
Name of predefined exclusive disjunction symbol.
 - char * [CONNECTOR_AND_STR](#)
Name of predefined conjunction symbol.
 - char * [CONNECTOR_IMPLIES_STR](#)
Name of predefined implication symbol.
 - char * [CONNECTOR_EQUIV_STR](#)
Name of predefined equivalence symbol.
 - 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.
- char * [PREDICATE_EQ_STR](#)
String for the equality symbol.
- [Tsymb PREDICATE_EQ](#)
Predefined symbol for the equality operator.
- char * [PREDICATE_DISTINCT_STR](#)
String for the distinct symbol.
- [Tsymb PREDICATE_DISTINCT](#)
Predefined symbol for the distinct operator.
- char * [PREDICATE_LESS_STR](#)
String for the predefined relational symbol "smaller than".
- char * [PREDICATE_LEQ_STR](#)
String for the predefined relational symbol "smaller or equal".
- char * [PREDICATE_GREATER_STR](#)
String for the predefined relational symbol "greater than".
- char * [PREDICATE_GREATEREQ_STR](#)
String for the predefined relational symbol "greater or equal".
- char * [PREDICATE_ISINT_STR](#)
String for the predefined predicate IsInt.
- [Tsymb PREDICATE_LESS](#)
Symbol for the strictly smaller than relational operator.
- [Tsymb PREDICATE_LEQ](#)
Symbol for the smaller or equal relational operator.
- [Tsymb PREDICATE_GREATER](#)
Symbol for the strictly greater than relational operator.
- [Tsymb PREDICATE_GREATEREQ](#)
Symbol for the greater or equal relational operator.
- [Tsymb PREDICATE_ISINT](#)
Symbol for the predicate testing integrality on real numbers.
- char * [PREDICATE_ARITH_DLEQ_STR](#)
String for difference logic operator.
- [Tsymb PREDICATE_ARITH_DLEQ](#)
Symbol for difference logic operator.
- char * [PREDICATE_ARITH_DLST_STR](#)
String for strict difference logic operator.
- [Tsymb PREDICATE_ARITH_DLST](#)

- Symbol for strict difference logic operator.*

 - char * [FUNCTION_ZERO_VARIABLE_STR](#)
String for variable 0 (for difference logic).
 - [Tsymb FUNCTION_ZERO_VARIABLE](#)
Symbol for variable 0 (for difference logic).
- char * [FUNCTION_ITE_STR](#)
String of the predefined symbol for the functional conditional.
- [Tsymb FUNCTION_ITE](#)
Symbol for the functional operator IF THEN ELSE.
- char * [FUNCTION_SUM_STR](#)
String of the predefined symbol for addition.
- char * [FUNCTION_PROD_STR](#)
String of the predefined symbol for multiplication.
- char * [FUNCTION_UNARY_MINUS_STR](#)
String of the predefined symbol for opposite.
- char * [FUNCTION_MINUS_STR](#)
String of the predefined symbol for subtraction.
- 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_UNARY_MINUS_ALT](#)
Alternative predefined symbol for arithmetic unary minus.
- [Tsymb FUNCTION_MINUS](#)
Predefined symbol for arithmetic binary minus.
- [Tsymb FUNCTION_DIV](#)
Predefined symbol for arithmetic division minus.
- char * [QUANTIFIER_EXISTS_STR](#)
String of the predefined symbol for existential quantification.
- 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.
- char * [LET_STR](#)
String of the predefined symbol for let construction.
- [Tsymb LET](#)
Predefined symbol for let construction.

- char * [LAMBDA_STR](#)
String of the predefined symbol for lambda abstraction.
- 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.
- char * [FUNCTION_SELECT_STR](#)
String of predefined symbol for array element selection.
- 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.
- [Tsymb FUNCTION_SELECT1](#)
- [Tsymb FUNCTION_STORE1](#)
- [Tsymb FUNCTION_SELECT2](#)
- [Tsymb FUNCTION_STORE2](#)
- [TDAG DAG_TRUE](#)
Represents the formula TRUE.
- [TDAG DAG_FALSE](#)
Represents the formula FALSE.
- [TDAG DAG_ONE](#)
Represents the formula TRUE.
- [TDAG DAG_ZERO](#)
Represents the formula FALSE.
- [Tpid DAG_PROP_TRIGGER](#)
Identifier for property storing quantifier triggers.
- [Tpid DAG_PROP_NOTRIGGER](#)
- [Tpid DAG_PROP_NAMED](#)

3.1.1 Detailed Description

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

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

<code>c</code>	shall be an expression of type <code>Tsymb</code> .
----------------	---

3.1.2.2 #define boolean_constant(c)

Value:

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

Tests if symbol *c* is a boolean constant.

Parameters

<i>c</i>	shall be an expression of type <i>T</i> symb.
----------	---

3.1.2.3 #define quantifier(*c*)

Value:

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

Tests if symbol *c* is a quantifier.

Parameters

<i>c</i>	shall be an expression of type <i>T</i> symb.
----------	---

3.1.2.4 #define binder(*c*)

Value:

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

Tests if symbol *c* is a quantifier.

Parameters

<i>c</i>	shall be an expression of type <i>T</i> symb.
----------	---

3.1.2.5 #define arith_function(*c*)

Value:

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

Tests if symbol *c* is an arithmetic operator.

Parameters

<code>c</code>	shall be an expression of type <code>Tsymb</code> .
----------------	---

3.1.2.6 #define arith_comparison(c)**Value:**

```
((c == PREDICATE_LESS) || \
  (c == PREDICATE_LEQ) || \
  (c == PREDICATE_GREATER) || \
  (c == PREDICATE_GREATEREQ))
```

Tests if symbol `c` is a arithmetic comparison operator.

Parameters

<code>c</code>	shall be an expression of type <code>Tsymb</code> .
----------------	---

3.1.2.7 #define unary_minus(c)**Value:**

```
((c == FUNCTION_UNARY_MINUS) || \
  (c == FUNCTION_UNARY_MINUS_ALT))
```

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 : `sort_1 x ... sort_{n-1} -> sort_{n}`, with $n > 1$.
- n-ary : `sort_1 x ... (arbitrary number of times) x sort_1 -> 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

3.1.3.3 typedef unsigned **Tpid**

type for property identifiers

Note

properties are stored as linked list and shall be used for rare attributes

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_BINARY*** Binary numeric values.
- SYMB_HEX*** Hexadecimal numeric values.
- SYMB_RATIONAL*** Rational numeric values.
- SYMB_STRING*** string values.
- SYMB_LAMBDA*** Lambda abstraction operator.
- SYMB_LET*** Let 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 veriT-DAG module.

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

3.1.5.2 void DAG_done (void)

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

Module options must be closed after veriT-DAG module

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

old sort constructor

Parameters

<i>name</i>	pointer to string naming the sort (may be NULL)
<i>arity</i>	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 <code>sub[0]</code> , and returns argument of sort <code>sub[1]</code> .
<i>sub</i>	array storing sub-sorts in compound sorts

Returns

a new sort

Remarks

Two sorts are equal if they have the same arity and the same sub-sorts.

- If an equal sort of the same name as already been created, it is returned.
- If a different sort with the same name has already been created, it is an error.
- If an equal sort with a different, not null, name has already been created, it is an error.
- If an equal sort with a null name has already been created, its name is set to the given name and it is returned.

- Destructive for sub
- The created sort is functional. Set functional field to 0 to change.

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

Sort constructor

Parameters

<i>name</i>	pointer to string naming the sort (may be NULL)
<i>arity</i>	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_new_var (char * name)

creates a sort variable with the given name

Parameters

<i>name</i>	
-------------	--

Precondition

name must either be NULL or a valid string.

Returns

It returns a sort variable:

- when called with NULL, it generates a fresh sort variable, the name of this variable is '_' (single-quote underscore) followed by a positive integer.
- when called with a string, it generates a sort variable of the given name.

Remarks

If called twice with the same (non-NULL) string, then returns the same sort.

3.1.5.6 Tsort DAG_sort_new_param (char * name, int arity)

creates a parametric sort constructor

Parameters

<i>name</i>	
<i>arity</i>	

Remarks

If there is already a constructor of the same name and arity, then it is returned.
 If name is NULL, an error is printed and execution halts.
 If there is already a constructor of the same name and different arity, an error is printed to stderr and execution halts.
 If arity is 0, then the result is DAG_sort_new_func(name, 0, NULL)

3.1.5.7 Tsort DAG_sort_new_inst (char * name, Tsort sort, int arity, Tsort * sub)

creates an instance of a parametric sort constructor

Parameters

<i>the</i>	name of the resulting sort (optional: may be NULL)
<i>sort</i>	the parametric sort constructor
<i>arity</i>	the number of arguments
<i>the</i>	arguments

Remarks

If there is already a constructor of the same constructor and arguments, then it is returned.
 If arity != sort->arity, an error is printed and execution halts.

3.1.5.8 Tsort DAG_sort_lookup (const char * name)

Gets sort with name.

Parameters

<i>name</i>	of the searched sort
-------------	----------------------

Returns

the sort named name, or NULL if not found

3.1.5.9 char* DAG_sort_name (Tsort sort)

Accesses the name of the sort.

Parameters

<i>sort</i>	
-------------	--

Returns

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

3.1.5.10 int DAG_sort_arity (Tsort *sort*)

Accesses the arity of the sort.

Parameters

<i>sort</i>	
-------------	--

Returns

the arity of the sort.

3.1.5.11 Tsort DAG_sort_sub (Tsort *sort*, unsigned *i*)

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

Parameters

<i>sort</i>	The accessed sort.
<i>i</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.

3.1.5.12 int DAG_sort_predefined (Tsort *sort*)

Tests if *sort* is predefined.

Parameters

<i>sort</i>	
-------------	--

3.1.5.13 int DAG_sort_parametric (Tsort *sort*)

Tests if *sort* is parametric.

Parameters

<i>sort</i>	the sort
-------------	----------

Remarks

(List 1) is parametric, (List Int) and Int are not.

3.1.5.14 int DAG_sort_polymorphic (Tsort *sort*)

Tests if sort is polymorphic.

Parameters

<i>sort</i>	the sort
-------------	----------

3.1.5.15 int DAG_sort_instance (Tsort *sort*)

Tests if sort is an instance of a parametric sort constructor.

Parameters

<i>sort</i>	
-------------	--

3.1.5.16 int DAG_sort_variable (const Tsort *sort*)

tests if a sort is a sort variable

Parameters

<i>sort</i>	The tested sort
-------------	-----------------

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

get the sort that include the others when ordered sort

Parameters

<i>sort1</i>	
<i>sort2</i>	Mainly for Integer and Real. NULL if disjoint.

3.1.5.18 int DAG_sort_subsumes (Tsort *sort1*, Tsort *sort2*)

Checks if sort1 subsumes sort2.

Parameters

<i>sort1</i>	a sort
<i>sort2</i>	a sort

Returns

1 if *sort1* subsumes *sort2*, 0 otherwise. *sort1* subsumes *sort2* iff there is a substitution of sort variables *s* such that $s(\text{sort1}) = \text{sort2}$

3.1.5.19 TDAG DAG_sort_substitute (const TDAG *src*)

computes a sort substitution on a DAG

Parameters

<i>src</i>	a DAG representing some term
------------	------------------------------

Returns

Assuming a sort substitution, i.e. that some (variable) sorts are bound to sorts, creates a copy of *src*, where all symbols of such variable sorts are substituted with new symbols, matching the sort substitution.

3.1.5.20 bool DAG_sort_in_free_variables (const Tsort *sort1*, const Tsort *sort2*)

tests if a sort variable appears in a sort

Parameters

<i>sort1</i>	is a sort variable
<i>sort2</i>	is a sort

Returns

1 if *sort1* occurs in *sort2*, 0 otherwise

3.1.5.21 T symb DAG_symb_new (const char * *name*, T symb_type *type*, Tsort *sort*)

Constructor.

Parameters

<i>name</i>	string naming the symbol
<i>type</i>	identifies the kind of symbols that needs to be created
<i>sort</i>	the symbol sort

Returns

returns the declared symbol

Declares a new symbol

3.1.5.22 **Tsymb** DAG_symb_lookup (char * *name*, unsigned *nb_arg*, TDAG * *PDAG*, Tsort *sort*)

Gets symbol with given name.

Parameters

<i>name</i>	string naming the symbol
<i>nb_arg</i>	the number of subterms
<i>PDAG</i>	the array of <i>nb_arg</i> subterms (optional)
<i>sort</i>	the symbols sort (optional)

Returns

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

Remarks

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

3.1.5.23 **Tsymb** DAG_symb_lookup_sort (char * *name*, Tsort *sort*)

Gets symbol with given name and sort.

Parameters

<i>name</i>	string naming the symbol
<i>sort</i>	the symbols sort

Returns

A symbol **s1** of sort **sort1** is candidate for the result if **s1** subsumes **sort** and there is no other symbol **s2** of sort **sort2** such that **sort1** subsumes **sort2** and **sort2** subsumes **sort**. Returns NULL if there are 0 or several candidates, and the candidate symbol otherwise.

3.1.5.24 char* DAG_symb_name (**Tsymb** *symp*)

Accesses the name of the symbol.

Parameters

<i>symp</i>	
-------------	--

Returns

The name of symb

3.1.5.25 Tsymb_type DAG_symb_type (Tsymb symb)

Accesses the kind of the symbol.

Parameters

<i>symb</i>	
-------------	--

Returns

The kind of symb

3.1.5.26 Tsort DAG_symb_sort (Tsymb symb)

Accesses the sort of the symbol.

Parameters

<i>symb</i>	
-------------	--

Returns

The kind of symb

3.1.5.27 void DAG_symb_set_bind_DAG (Tsymb symb, TDAG DAG)

Associates a DAG to symbol.

Parameters

<i>symb</i>	
<i>DAG</i>	This is useful for instantiations (think symb is a variable).

3.1.5.28 TDAG DAG_symb_get_bind_DAG (Tsymb symb)

Gets DAG associated DAG to symbol.

Parameters

<i>symb</i>	
-------------	--

Returns

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

3.1.5.29 void DAG_symb_set_interpreted (*Tsymb symb*)

Marks symbol as being interpreted.

Parameters

<i>symb</i>	a symbol
-------------	----------

Macros need to be marked as interpreted symbols.

3.1.5.30 bool DAG_symb_interpreted (*Tsymb symb*)

Tests if symbol is interpreted.

Parameters

<i>symb</i>	a symbol
-------------	----------

Macros are considered interpreted symbols.

3.1.5.31 *Tsymb* DAG_symb_skolem (*Tsort sort*)

Specialized constructor.

Parameters

<i>sort</i>	sort of the symbol to create
-------------	------------------------------

Returns

symbol of a fresh skolem symbol of the given sort

3.1.5.32 *Tsymb* DAG_symb_const (*Tsort sort*)

Specialized constructor.

Parameters

<i>sort</i>	sort of the symbol to create
-------------	------------------------------

Returns

symbol of a fresh constant symbol of the given sort

3.1.5.33 *Tsymb* DAG_symb_variable (*Tsort sort*)

Specialized constructor.

Parameters

<i>sort</i>	sort of the symbol to create
-------------	------------------------------

Returns

symbol of a fresh variable of the given sort

3.1.5.34 Tsymb DAG_symb_predicate (Tsort *sort*)

Specialized constructor.

Parameters

<i>sort</i>	sort of the symbol to create
-------------	------------------------------

Returns

symbol of a fresh predicate of the given sort

3.1.5.35 int DAG_symb_prop_get (Tsymb *symp*, Tpid *pid*, void * *P*)**

gets the value of a property

Parameters

<i>symp</i>	the symbol on which the property is read
<i>pid</i>	the identifier of the property
<i>P</i>	pointer to the address where the value of the property is assigned.

Returns

0 if property is not found, 1 otherwise.

3.1.5.36 int DAG_symb_prop_check (Tsymb *symp*, Tpid *pid*)

check if there is a value for a property

Parameters

<i>symp</i>	the symbol on which the property is read
<i>pid</i>	the identifier of the property

Returns

0 if property is not found, 1 otherwise.

3.1.5.37 TDAG DAG_new (Tsymp *symp*, unsigned *arity*, TDAG * *PDAG*)

DAG constructor.

Parameters

<i>symp</i>	topmost symbol
<i>arity</i>	number of sub-terms
<i>PDAG</i>	array of subterms

Returns

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

Destructive for *PDAG*

3.1.5.38 TDAG DAG_new_args (Tsymp *symp*, ...)

DAG constructor.

Parameters

<i>symp</i>	The topmost symbol of the constructed term.
...	subterms, followed by NULL.

Precondition

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

Returns

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

3.1.5.39 TDAG DAG_new_integer (long *value*)

DAG constructor.

Parameters

<i>value</i>	an integer
--------------	------------

Returns

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

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

DAG constructor.

Parameters

<i>num</i>	an integer interpreted as numerator
<i>den</i>	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.

3.1.5.41 TDAG DAG_new_integer_str (char * value)

DAG constructor.

Parameters

<i>value</i>	textual representation of an integer $0 [1-9][0-9]^*$
--------------	---

Returns

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

The given string is checked for conformance.

3.1.5.42 TDAG DAG_new_decimal_str (char * value)

DAG constructor.

Parameters

<i>value</i>	textual representation of a decimal $(0 [1-9][0-9]^*)\.[0-9]^+$
--------------	---

Returns

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

The given string is checked for conformance.

Parameters

<i>value</i>	textual representation of a floating point $0.[0-9]^+ [1-9][0-9]^*\.[0-9]^+$
--------------	--

Returns

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

The given string is checked for conformance.

3.1.5.43 TDAG DAG_new_binary_str (char * value)

DAG constructor.

Parameters

<i>value</i>	textual representation of a binary #b[01]+
--------------	--

Returns

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

The given string is checked for conformance.

3.1.5.44 TDAG DAG_new_hex_str (char * value)

DAG constructor.

Parameters

<i>value</i>	textual representation of an hexadecimal #x[0-9A-Fa-f]+
--------------	---

Returns

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

The given string is checked for conformance.

3.1.5.45 TDAG DAG_new_rational_str (char * value)

DAG constructor.

Parameters

<i>value</i>	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.

3.1.5.46 TDAG DAG_new_str (char * value)

DAG constructor.

Parameters

<i>value</i>	string
--------------	--------

Returns

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

3.1.5.47 TDAG DAG_dup (TDAG DAG)

Reference counter increment.

Parameters

<i>DAG</i>	its reference counter will be incremented
------------	---

Returns

the result is the same as the argument.

3.1.5.48 void DAG_free (TDAG DAG)

Destructor.

Parameters

<i>DAG</i>	to be freed
------------	-------------

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

3.1.5.49 unsigned DAG_count_nodes (TDAG DAG)

DAG size.

Parameters

<i>DAG</i>	to be measured.
------------	-----------------

Returns

Number of nodes in DAG as a DAG representation.

3.1.5.50 int DAG_count_nodes_tree (TDAG DAG)

Expanded DAG size.

Parameters

<i>DAG</i>	to be measured.
------------	-----------------

Returns

Number of nodes in DAG as a tree representation (-1 for overflow)

3.1.5.51 unsigned DAG_count_atoms (TDAG DAG)

Expanded DAG boolean size.

Parameters

DAG	to be measured.
-----	-----------------

Returns

Number of atoms in DAG as a tree representation.

3.1.5.52 unsigned DAG_depth (TDAG DAG)

Depth of a DAG.

Parameters

DAG	to be measured.
-----	-----------------

Returns

Depth of a DAG as a tree representation (depth is 1 for leaves).

3.1.5.53 int DAG_subst_Pflag (TDAG src)

substitutes every node by node->Pflag in a term

Author

David Deharbe, Pascal Fontaine

Parameters

src	the term
-----	----------

Returns

1 iff the term is modified (0 otherwise)

Attention

the user should restore the Pflag of every intermediate node for instance using DAG_reset_Pflag(src)

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

Simple substitution.

Parameters

<i>src</i>	Term where the substitution is realized.
<i>origin</i>	Term that will be substituted.
<i>subst</i>	Term that will replace origin.

See also

[DAG_subst_multiple](#)

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

Multiple substitution.

Parameters

<i>src</i>	Term where the substitution is realized.
<i>n</i>	The number of terms that will be substituted.
<i>origin</i>	Array of n terms that will be substituted.
<i>subst</i>	Array of n term that will replace substituted terms.

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

Tests DAG inclusion.

Parameters

<i>src</i>	
<i>find</i>	

Returns

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

3.1.5.57 void DAG_smtlib_logic_set (const char * *str*, int *version*)

Sets the logic.

Parameters

<i>str</i>	is the name of the logic, NULL for default setup.
<i>version</i>	the version of SMT-LIB language

Note

This function must be called at most once.

3.1.5.58 Tpid DAG_prop_new (TFfree *f*)

creates a property identifier

Parameters

<i>f</i>	is the destructor for property values
----------	---------------------------------------

3.1.5.59 void DAG_prop_set (TDAG *DAG*, Tpid *pid*, void * *value*)

sets the value of a property

Parameters

<i>DAG</i>	the DAG on which the property is set
<i>pid</i>	the identifier of the property
<i>value</i>	a pointer to its value

3.1.5.60 int DAG_prop_remove (TDAG *DAG*, Tpid *pid*)

remove the value of a property

Parameters

<i>DAG</i>	the DAG on which the property is removed
<i>pid</i>	the identifier of the property

3.1.5.61 int DAG_prop_get (TDAG *DAG*, Tpid *pid*, void * *P*)**

gets the value of a property

Parameters

<i>DAG</i>	the DAG on which the property is read
<i>pid</i>	the identifier of the property
<i>P</i>	pointer to the address where the value of the property is assigned.

Returns

0 if property is not found, 1 otherwise

3.1.5.62 int DAG_prop_check (TDAG DAG, Tpid pid)

check if there is a value for a property

Parameters

<i>DAG</i>	the DAG on which the property is read
<i>pid</i>	the identifier of the property

Returns

0 if property is not found, 1 otherwise

3.2 veriT-status.h File Reference

Proof status in libveriT.

Typedefs

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

Enumerations

- enum [Estatus](#) {
[SAT](#),
[UNSAT](#),
[UNDEF](#),
[OPEN](#) }

Enumeration of the different possible proof status in veriT.

3.2.1 Detailed Description

Proof status in libveriT.

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 veriT.

Enumerator:

SAT satisfiable

UNSAT unsatisfiable

UNDEF undefined The proof obligation is not within the theories for which the solver is complete, and the solver was not able to show unsatisfiability.

OPEN not verified yet Run veriT_solve to (semi)decide

3.3 veriT.h File Reference

API for the SMT solver VERI(T).

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

Functions

- void `veriT_init` (void)
Initializes the module.
- void `veriT_done` (void)
Closes down the module.
- void `veriT_logic` (char *logic, int version)
implement the SMT-LIB 2.0 set-logic command
- void `veriT_push` (int n)
implement the SMT-LIB 2.0 push command
- void `veriT_pop` (int n)
implement the SMT-LIB 2.0 pop command
- void `veriT_assert` (TDAG DAG)
implement the SMT-LIB 2.0 assert command
- Tstatus `veriT_check_sat` (void)
implement the SMT-LIB 2.0 check-sat command
- int `veriT_complete` (void)
check if the formulas given to veriT fall in a complete fragment handled by the tool
- Tstatus `veriT_status` (void)
Returns the current status of the solver.
- void `veriT_interactive` (void)
use veriT in interactive mode.

- void `veriT_add_formula` (TDAG formula)
Adds conjunctively a formula in the input.
- `Tstatus veriT_solve` (void)
Runs solver on the formula(s)
- void `veriT_reset` (void)
Resets module.
- void `veriT_get_model` (int *Pnb_literals, TDAG **Pliterals)
Builds a model for the conjunction of formulas.

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 linear arithmetics over integers or reals.

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

3.3.2 Function Documentation

3.3.2.1 void `veriT_init` (void)

Initializes the module.

Remarks

Must be called before any other function of the module.

3.3.2.2 void `veriT_done` (void)

Closes down the module.

Remarks

Frees all the memory allocated by module functions.

3.3.2.3 void veriT_logic (char * *logic*, int *version*)

implement the SMT-LIB 2.0 set-logic command

Author

David Deharbe, Pascal Fontaine

Parameters

<i>logic</i>	is the name of the logic using the SMT-LIB nomenclature, or NULL to set up the default logic.
<i>version</i>	the version of SMT-LIB language

Remarks

Fails if unkown logic

3.3.2.4 void veriT_push (int *n*)

implement the SMT-LIB 2.0 push command

Author

Pascal Fontaine

Parameters

<i>n</i>	an integer argument.
----------	----------------------

3.3.2.5 void veriT_pop (int *n*)

implement the SMT-LIB 2.0 pop command

Author

Pascal Fontaine

Parameters

<i>n</i>	an integer argument.
----------	----------------------

Remarks

Fails (with error) if initial assertion-set popped, i.e. if more pops than pushes

3.3.2.6 void veriT_assert (TDAG *DAG*)

implement the SMT-LIB 2.0 assert command

Author

Pascal Fontaine

Parameters

<i>DAG</i> a formula

Remarks

Fails if formula is not Boolean

3.3.2.7 Tstatus veriT_check_sat (void)

implement the SMT-LIB 2.0 check-sat command

Author

Pascal Fontaine

Returns

The status of the verification, i.e.:

- UNSAT, if the current set of assertions is unsatisfiable,
- SAT, if the current set of assertions is satisfiable,
- UNKNOWN if veriT fails to show either of the two previous outcomes.

See also

[veriT_status](#)

3.3.2.8 int veriT_complete (void)

check if the formulas given to veriT fall in a complete fragment handled by the tool

Author

Pascal Fontaine

Returns

1 if so, 0 if not

3.3.2.9 Tstatus veriT_status (void)

Returns the current status of the solver.

Author

David Deharbe, Pascal Fontaine

The status may be:

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

3.3.2.10 `void veriT_interactive (void)`

use veriT in interactive mode.

Remarks

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

3.3.2.11 `void veriT_add_formula (TDAG formula)`

Adds conjunctively a formula in the input.

Parameters

<i>formula</i>	represents the formula to be checked for satisfiability.
----------------	--

Use this function to add conjunctively a formula in the input. Non-destructive for formula.

Remarks

These formulas are added in the context without proof.
Caution: DO NOT USE FOR INTERNALLY-PRODUCED FORMULA (e.g. lemmas).

3.3.2.12 `Tstatus veriT_solve (void)`

Runs solver on the formula(s)

Returns

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

- SAT: satisfiable,
- UNSAT: unsatisfiable,

- UNDEF: not within the complete fragment handled, and solver was not able to conclude.

3.3.2.13 void veriT_reset (void)

Resets module.

Remarks

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.

3.3.2.14 void veriT_get_model (int * Pnb_literals, TDAG ** Pliterals)

Builds a model for the conjunction of formulas.

Precondition

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

Parameters

<i>Pnb_literals</i>	address where function stores the number of literals in the model
<i>Pliterals</i>	address where function stores the model (an array of literals)

Remarks

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_comparison
 veriT-DAG.h, 17

arith_function
 veriT-DAG.h, 16

binder
 veriT-DAG.h, 16

boolean_connector
 veriT-DAG.h, 15

boolean_constant
 veriT-DAG.h, 15

DAG_contain
 veriT-DAG.h, 34

DAG_count_atoms
 veriT-DAG.h, 33

DAG_count_nodes
 veriT-DAG.h, 32

DAG_count_nodes_tree
 veriT-DAG.h, 32

DAG_depth
 veriT-DAG.h, 33

DAG_done
 veriT-DAG.h, 19

DAG_dup
 veriT-DAG.h, 32

DAG_free
 veriT-DAG.h, 32

DAG_init
 veriT-DAG.h, 19

DAG_new
 veriT-DAG.h, 28

DAG_new_args
 veriT-DAG.h, 29

DAG_new_binary_str
 veriT-DAG.h, 30

DAG_new_decimal_str
 veriT-DAG.h, 30

DAG_new_hex_str
 veriT-DAG.h, 31

DAG_new_integer
 veriT-DAG.h, 29

DAG_new_integer_str
 veriT-DAG.h, 30

DAG_new_rational
 veriT-DAG.h, 29

DAG_new_rational_str
 veriT-DAG.h, 31

DAG_new_str
 veriT-DAG.h, 31

DAG_prop_check
 veriT-DAG.h, 35

DAG_prop_get
 veriT-DAG.h, 35

DAG_prop_new
 veriT-DAG.h, 35

DAG_prop_remove
 veriT-DAG.h, 35

DAG_prop_set
 veriT-DAG.h, 35

DAG_smtlib_logic_set
 veriT-DAG.h, 34

DAG_sort_arity
 veriT-DAG.h, 22

DAG_sort_com
 veriT-DAG.h, 23

DAG_sort_in_free_variables
 veriT-DAG.h, 24

DAG_sort_instance
 veriT-DAG.h, 23

DAG_sort_lookup
 veriT-DAG.h, 21

DAG_sort_name
 veriT-DAG.h, 21

DAG_sort_new
 veriT-DAG.h, 19

DAG_sort_new_args
 veriT-DAG.h, 20

DAG_sort_new_inst
 veriT-DAG.h, 21

DAG_sort_new_param
 veriT-DAG.h, 20

- DAG_sort_new_var
veriT-DAG.h, [20](#)
- DAG_sort_parametric
veriT-DAG.h, [22](#)
- DAG_sort_polymorphic
veriT-DAG.h, [23](#)
- DAG_sort_predefined
veriT-DAG.h, [22](#)
- DAG_sort_sub
veriT-DAG.h, [22](#)
- DAG_sort_substitute
veriT-DAG.h, [24](#)
- DAG_sort_subsumes
veriT-DAG.h, [23](#)
- DAG_sort_variable
veriT-DAG.h, [23](#)
- DAG_subst
veriT-DAG.h, [33](#)
- DAG_subst_multiple
veriT-DAG.h, [34](#)
- DAG_subst_Pflag
veriT-DAG.h, [33](#)
- DAG_symb_const
veriT-DAG.h, [27](#)
- DAG_symb_get_bind_DAG
veriT-DAG.h, [26](#)
- DAG_symb_interpreted
veriT-DAG.h, [27](#)
- DAG_symb_lookup
veriT-DAG.h, [25](#)
- DAG_symb_lookup_sort
veriT-DAG.h, [25](#)
- DAG_symb_name
veriT-DAG.h, [25](#)
- DAG_symb_new
veriT-DAG.h, [24](#)
- DAG_symb_predicate
veriT-DAG.h, [28](#)
- DAG_symb_prop_check
veriT-DAG.h, [28](#)
- DAG_symb_prop_get
veriT-DAG.h, [28](#)
- DAG_symb_set_bind_DAG
veriT-DAG.h, [26](#)
- DAG_symb_set_interpreted
veriT-DAG.h, [27](#)
- DAG_symb_skolem
veriT-DAG.h, [27](#)
- DAG_symb_sort
veriT-DAG.h, [26](#)
- DAG_symb_type
veriT-DAG.h, [26](#)
- DAG_symb_variable
veriT-DAG.h, [27](#)
- Estatus
veriT-status.h, [37](#)
- Esymb_type
veriT-DAG.h, [18](#)
- OPEN
veriT-status.h, [37](#)
- quantifier
veriT-DAG.h, [16](#)
- SAT
veriT-status.h, [37](#)
- SYMB_APPLY
veriT-DAG.h, [18](#)
- SYMB_BINARY
veriT-DAG.h, [18](#)
- SYMB_BOOLEAN
veriT-DAG.h, [18](#)
- SYMB_BOOLEAN_CONSTANT
veriT-DAG.h, [18](#)
- SYMB_FUNCTION
veriT-DAG.h, [18](#)
- SYMB_HEX
veriT-DAG.h, [18](#)
- SYMB_INTEGER
veriT-DAG.h, [18](#)
- SYMB_ITE
veriT-DAG.h, [18](#)
- SYMB_ITE_FUNC
veriT-DAG.h, [18](#)
- SYMB_LAMBDA
veriT-DAG.h, [18](#)
- SYMB_LET
veriT-DAG.h, [18](#)
- SYMB_MACRO
veriT-DAG.h, [19](#)
- SYMB_PREDICATE
veriT-DAG.h, [18](#)
- SYMB_QUANTIFIER
veriT-DAG.h, [18](#)
- SYMB_RATIONAL
veriT-DAG.h, [18](#)
- SYMB_SKOLEM
veriT-DAG.h, [18](#)

- SYMB_STRING
 - veriT-DAG.h, 18
- SYMB_TYPE_NB
 - veriT-DAG.h, 19
- SYMB_UNKNOWN
 - veriT-DAG.h, 18
- SYMB_VARIABLE
 - veriT-DAG.h, 18
- Tpid
 - veriT-DAG.h, 18
- Tsort
 - veriT-DAG.h, 17
- Tsymb_type
 - veriT-DAG.h, 18
- unary_minus
 - veriT-DAG.h, 17
- UNDEF
 - veriT-status.h, 37
- UNSAT
 - veriT-status.h, 37
- veriT-DAG.h, 5
 - arith_comparison, 17
 - arith_function, 16
 - binder, 16
 - boolean_connector, 15
 - boolean_constant, 15
 - DAG_contain, 34
 - DAG_count_atoms, 33
 - DAG_count_nodes, 32
 - DAG_count_nodes_tree, 32
 - DAG_depth, 33
 - DAG_done, 19
 - DAG_dup, 32
 - DAG_free, 32
 - DAG_init, 19
 - DAG_new, 28
 - DAG_new_args, 29
 - DAG_new_binary_str, 30
 - DAG_new_decimal_str, 30
 - DAG_new_hex_str, 31
 - DAG_new_integer, 29
 - DAG_new_integer_str, 30
 - DAG_new_rational, 29
 - DAG_new_rational_str, 31
 - DAG_new_str, 31
 - DAG_prop_check, 35
 - DAG_prop_get, 35
 - DAG_prop_new, 35
 - DAG_prop_remove, 35
 - DAG_prop_set, 35
 - DAG_smtlib_logic_set, 34
 - DAG_sort_arity, 22
 - DAG_sort_com, 23
 - DAG_sort_in_free_variables, 24
 - DAG_sort_instance, 23
 - DAG_sort_lookup, 21
 - DAG_sort_name, 21
 - DAG_sort_new, 19
 - DAG_sort_new_args, 20
 - DAG_sort_new_inst, 21
 - DAG_sort_new_param, 20
 - DAG_sort_new_var, 20
 - DAG_sort_parametric, 22
 - DAG_sort_polymorphic, 23
 - DAG_sort_predefined, 22
 - DAG_sort_sub, 22
 - DAG_sort_substitute, 24
 - DAG_sort_subsumes, 23
 - DAG_sort_variable, 23
 - DAG_subst, 33
 - DAG_subst_multiple, 34
 - DAG_subst_Pflag, 33
 - DAG_symb_const, 27
 - DAG_symb_get_bind_DAG, 26
 - DAG_symb_interpreted, 27
 - DAG_symb_lookup, 25
 - DAG_symb_lookup_sort, 25
 - DAG_symb_name, 25
 - DAG_symb_new, 24
 - DAG_symb_predicate, 28
 - DAG_symb_prop_check, 28
 - DAG_symb_prop_get, 28
 - DAG_symb_set_bind_DAG, 26
 - DAG_symb_set_interpreted, 27
 - DAG_symb_skolem, 27
 - DAG_symb_sort, 26
 - DAG_symb_type, 26
 - DAG_symb_variable, 27
 - Esymb_type, 18
 - quantifier, 16
 - SYMB_APPLY, 18
 - SYMB_BINARY, 18
 - SYMB_BOOLEAN, 18
 - SYMB_BOOLEAN_CONSTANT, 18
 - SYMB_FUNCTION, 18
 - SYMB_HEX, 18
 - SYMB_INTEGER, 18

SYMB_ITE, 18
SYMB_ITE_FUNC, 18
SYMB_LAMBDA, 18
SYMB_LET, 18
SYMB_MACRO, 19
SYMB_PREDICATE, 18
SYMB_QUANTIFIER, 18
SYMB_RATIONAL, 18
SYMB_SKOLEM, 18
SYMB_STRING, 18
SYMB_TYPE_NB, 19
SYMB_UNKNOWN, 18
SYMB_VARIABLE, 18
Tpid, 18
Tsort, 17
Tsymb_type, 18
unary_minus, 17
veriT-status.h, 36
 Estatus, 37
 OPEN, 37
 SAT, 37
 UNDEF, 37
 UNSAT, 37
veriT.h, 37
 veriT_add_formula, 41
 veriT_assert, 39
 veriT_check_sat, 40
 veriT_complete, 40
 veriT_done, 38
 veriT_get_model, 42
 veriT_init, 38
 veriT_interactive, 41
 veriT_logic, 38
 veriT_pop, 39
 veriT_push, 39
 veriT_reset, 42
 veriT_solve, 41
 veriT_status, 40
veriT_add_formula
 veriT.h, 41
veriT_assert
 veriT.h, 39
veriT_check_sat
 veriT.h, 40
veriT_complete
 veriT.h, 40
veriT_done
 veriT.h, 38
veriT_get_model
 veriT.h, 42
 veriT_init
 veriT.h, 38
veriT_interactive
 veriT.h, 41
veriT_logic
 veriT.h, 38
veriT_pop
 veriT.h, 39
veriT_push
 veriT.h, 39
veriT_reset
 veriT.h, 42
veriT_solve
 veriT.h, 41
veriT_status
 veriT.h, 40