

veriT  
201107

Generated by Doxygen 1.7.4

Mon Jul 11 2011 10:34:13



# Contents

<b>1</b>	<b>VERI(T) documentation</b>	<b>1</b>
1.1	Introduction . . . . .	1
1.2	Execution . . . . .	1
1.3	Support for SMT-LIB 1.2 . . . . .	2
1.4	Support for SMT-LIB 2.0 . . . . .	3
1.5	Installation . . . . .	5
1.6	Licence . . . . .	5
1.7	Authors . . . . .	5
<b>2</b>	<b>Module Index</b>	<b>7</b>
2.1	Modules . . . . .	7
<b>3</b>	<b>Module Documentation</b>	<b>9</b>
3.1	Options . . . . .	9
3.2	Developer options . . . . .	9



# Chapter 1

## VERI(T) documentation

### 1.1 Introduction

VERI(T) is a Satisfiability Modulo Theory (SMT) solver.

The solver is currently integrating a quantifier instantiation mechanism with decision procedures for the following: uninterpreted functions with equality, difference arithmetics (integers and reals), linear arithmetics (integers and reals). The superposition-based prover E is also integrated as a fall-back verification engine for verification conditions with a unique sort.

### 1.2 Execution

The input of the solver is a proof obligation in one of the following formats:

- SMT-LIB 1.2: This is the default input format and is still the best supported. It should be deprecated in the next release though. See <http://www.smtlib.org> for information on the format.
- SMT-LIB 2.0: This format may be employed as long as interactive features and incrementality are not used. It should be the default format in the next release. See <http://www.smtlib.org> for information on this format.
- DIMACS: This format may be employed when one wants to use veriT as a SAT-solver (with proof production capabilities).

In the case of the SMT-LIB formats, the input language is actually slightly expanded and includes parametric sorts, macro definitions and lambda expressions. Note the "dot" separator in notation for lambda abstractions, as shown in the following example (in format SMT-LIB 1.2):

```
(benchmark macro
```

```

:logic UNKNOWN
:status unknown
:extrasorts ((ELEMENT)
  (List 1))
:extrafuns ((nil (List 's))
  (cons 's (List 's) (List 's))
  (car (List 's) 's)
  (cdr (List 's) (List 's)))
:extrafuns ((length (List 's) Int))
:extrafuns ((x ELEMENT))
:extramacros ((in (lambda (?x 's) (?p ('s boolean)) . (?p ?x))
  (singleton (lambda (?x 's) . (lambda (?y 's) . (= ?x ?y))))))
:assumption (forall (?x 's) (?l (List 's))
  (and
    (= (car (cons ?x ?l)) ?x)
    (= (cdr (cons ?x ?l)) ?l)))
:assumption (forall (?x 's) (?l (List 's))
  (not (= nil (car (cons ?x ?l)))))
:formula
(not (in x (car (cons (singleton x) nil))))
)

```

The solver outputs the result on stdout as a one-line message, which is either:

- sat when the formula in the proof obligation is satisfiable;
- unsat when the formula in the proof obligation is unsatisfiable;
- unknown when the solver has not been able to conclude.

Also possible is a run-time error. The authors of the tool made their best to catch all possible errors and output a meaningful message.

The solver may be executed in two modes: batch and interactive.

In batch mode the solver expects as argument the name of a file name in one of the supported formats. In interactive mode the solver expects a series of clauses in SMT-LIB 2.0 format. Several formulas may be added, and they are conjunctively checked for satisfiability. There is no provision for backtracking.

veriT may be compiled with proof-producing capabilities (this is optional). When the result is unsat, a derivation of the result might be produced and checked by a third party.

Command-line options may be used to output information about the proof obligation or about the proof process itself into files or to the standard output stream. See the specific documentation modules for [Options](#) and [Developer options](#).

### 1.3 Support for SMT-LIB 1.2

- AUFLIA Incomplete
- AUFLIRA Incomplete

- AUFNIRA Incomplete
- LIA Incomplete
- LRA Incomplete
- QF\_A Incomplete
- QF\_AUFBV Not supported
- QF\_AUFLIA Incomplete
- QF\_AX Incomplete
- QF\_BV Not supported
- QF\_IDL Complete
- QF\_LIA Incomplete
- QF\_LRA Complete
- QF\_NIA Incomplete
- QF\_RDL Complete
- QF\_UF Complete, proof-producing
- QF\_UFIDL Complete
- QF\_UFLIA Incomplete
- QF\_UFLRA Complete

Using the logic "UNKNOWN" provides a logic for the SMT-LIB theories Reals and ArraysEx, and veriT is incomplete.

## 1.4 Support for SMT-LIB 2.0

- AUFLIA Incomplete
- AUFLIRA Incomplete
- AUFNIRA Incomplete
- LRA Incomplete
- QF\_ABV Not supported
- QF\_AUFBV Not supported
- QF\_AUFLIA Incomplete
- QF\_AX Incomplete
- QF\_BV Not supported

- QF\_IDL Complete
- QF\_LIA Incomplete
- QF\_LRA Complete
- QF\_NIA Incomplete
- QF\_RDL Complete
- QF\_UF Complete, proof-producing
- QF\_UFIDL Complete

Using the logic "UNKNOWN" provides a logic with the SMT-LIB theories Reals and ArraysEx, and veriT is then incomplete.

The following commands are supported:

- set-logic
- set-info: :status and :version are supported.
- set-option: :diagnostic-output-channel, :regular-output-channel, :print-success are supported.
- get-option: only for the options listed with set-option
- declare-sort
- define-sort
- declare-fun
- define-fun
- assert
- check-sat
- exit

The following commands are parsed but their interpretation is not compliant:

- push
- pop

The following commands are parsed but are not supported:

- set-option
- get-assertions
- get-proof (but see proof production for SMT-LIB 1.2)
- get-unsat-core

- get-value
- get-assignment

veriT currently does not support indexed symbols as identifiers.

## 1.5 Installation

The source code is available on the Web at <http://www.verit-solver.org/>, along with a series of binaries.

To install the solver from the sources, once it is downloaded and unpacked, change to the top level directory and set the values in the file "Makefile.variables". Run the command line make. This will fetch some third-party components and compile the sources. Once the build is complete, copy the binary program, named "verit", to the location of your choice.

## 1.6 Licence

VERI(T) 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 licences BSD and LGPL.

The functionality between these licenses is the same.

LGPL-VERI(T) links against the GMP library to handle arbitrary precision arithmetics.

BSD-VERI(T) uses native data types and only supports fixed precision arithmetics. However, should an overflow occur at runtime, it will be detected and an error will be reported.

## 1.7 Authors

Pascal Fontaine, David Deharbe are the two main developpers. Diego Caminha has developed the arithmetic decision procedures 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.

The solver is being developed by 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

# Module Index

### 2.1 Modules

Here is a list of all modules:

Options . . . . .	9
Developer options . . . . .	9



## Chapter 3

# Module Documentation

### 3.1 Options

- `--disable-banner`

The message identifying the program is not printed to stdout.

- `--input=(smtlib1|smtlib2|dimacs)`

Sets the input format (smtlib1 is default).

- `--input=(smtlib1|smtlib2|b)`

Sets the output format (smtlib1 is default). Meaningful only when output formulas are produced.

- `--max-time=n`

Sets maximal execution time to n (in seconds). Caveat: the execution time limit is individual to each process (i.e. the main process and the possible calls to external provers.)

- `--disable-print-success`

Overrides the default SMT-LIB 2 behavior regarding the option `:print-success`.

### 3.2 Developer options

- `--print-and-exit`

Loads formula, expands macros and print on stdout in SMT format.

- `--parse-and-exit`

Loads formula and exits.

- `--print-proof_format-and-exit`

Loads formula, expands macros, applies selected simplifications, and prints on stdout in SMT format.

- `--print-simp-and-exit`

Loads formula, expands macros, applies selected simplifications, and prints on stdout in SMT format.

- `--simp-and-exit`

Loads formula, expands macros, and applies selected simplifications.

- `--print-atoms-and-exit`

Loads formula, expands macros, and prints atoms on stdout in SMT format.

# Index

Developer options, [9](#)

Options, [9](#)