

VERI(T)

1.0rc1

Generated by Doxygen 1.5.8

Mon Feb 23 14:43:08 2009



# Contents

<b>1</b>	<b>VERI(T) documentation</b>	<b>1</b>
1.1	Introduction . . . . .	1
1.2	Execution . . . . .	1
1.3	Installation . . . . .	2
1.4	Licence . . . . .	2
1.5	Authors . . . . .	2
<b>2</b>	<b>Module Index</b>	<b>3</b>
2.1	Modules . . . . .	3
<b>3</b>	<b>Module Documentation</b>	<b>5</b>
3.1	Options . . . . .	5
3.2	Developer options . . . . .	6



# Chapter 1

## VERI(T) documentation

### 1.1 Introduction

VERI(T) is a 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 also integrated as a fall-back verification engine for verification conditions with a unique sort. The solver also integrates some level of quantifier reasoning, using Skolemization and instantiation heuristics.

### 1.2 Execution

The input of the solver is a proof obligation in the SMT format (see <http://www.smtlib.org>). The input language is actually slightly expanded and includes macro definitions and lambda expressions (please note the "dot" separator in notation for lambda abstractions):

```
(benchmark macro
  :logic UNKNOWN
  :status unknown
  :extrasorts (ELEMENT)
  :extrafuns ((x ELEMENT))
  :extramacros ((in (lambda (?x ELEMENT) (?p (ELEMENT boolean)) . (?p ?x)))
                (singleton (lambda (?x ELEMENT) . (lambda (?y ELEMENT) . (= ?x ?y)))))
  :formula
  (not (in x (singleton x)))
)
```

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 SMT format. In interactive mode the solver expects a series of clauses in SMT format. However several formulas may be added, and they are conjunctively checked for satisfiability. There is no provision for backtracking.

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 Installation

The source code is available on the Web at <http://harvey.loria.fr/> , 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.

## 1.4 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 librv 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.

## 1.5 Authors

Pascal Fontaine, David Deharbe are the two main developpers. 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.

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 . . . . .	5
Developer options . . . . .	6



# Chapter 3

## Module Documentation

### 3.1 Options

- `-disable-banner`

The message identifying the program is not printed to stdout.

- `-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-instance-polarity`

Disables the quantifier instantiation heuristics based on polarity.

- `-disable-instance-equality`

Disables the quantifier instantiation heuristics based on equality.

- `-disable-sym-eq`

Disables symmetry of equality (EXPERIMENTAL - DO NOT USE).

- `-enable-simp`

Enables application of simplifications

- `-enable-qnt-simp`

Enables application of simplifications for quantified expressions.

## 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-simp-and-exit`

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

- `-print-proof_format-and-exit`

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

- `-print-atoms-and-exit`

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

- `-check-deduced`

Generates files with verification conditions in SMT format for each conflict clause, model and lemma found. Useful to debug consistency of the theory solver (only available when compiled with `DEBUG` defined).

- `-dump-abstract-models`

Generates files in DIMACS format with the propositional abstraction of the model, the lemmas and conflict clauses that have been given to the boolean engine. Useful to debug consistency (only available when compiled with `DEBUG` defined).

# Index

Developer options, [6](#)

Options, [5](#)