SRI International

CSL Technical Report • April 26, 2012

Yices 2 Manual

Bruno Dutertre
Computer Science Laboratory
SRI International
Menlo Park CA 94025 USA

Computer Science Laboratory • 333 Ravenswood Ave. • Menlo Park, CA 94025 • (650) 326-6200 • Facsimile: (650) 859-2844

Contents

Chapter 1

Introduction

This manual is an introduction to the logic, language, and architecture of the Yices 2 SMT solver. Yices is developed in SRI International's Computer Science Laboratory and is distributed free-of-charge for personnal use, under the terms of the Yices License 7. To discuss alternative license terms, please contact us at `fm-license@csl.sri.com`.

Yices can be downloaded at `http://yices.csl.sri.com`. The Yices website provides the latest release and information about Yices. For bug reports and questions about Yices, please contact us via the Yices mailing lists:

- To report a bug, send e-mail to `yices-bugs@csl.sri.com`.

  Please include enough information in your bug rpeort to enable us to reproduce the problem.

- If you have any questions about Yices usage or installation, send e-mail to `yices-help@csl.sri.com`.

Chapter 2

Yices 2 Language

Yices 2 specifications are written in a typed logic. The language is intended to be simple enough for efficient processing by the tool and expressive enough for most applications. The Yices 2 language is similar to the logic supported by Yices 1, but the most complex type constructs have been removed.

2.1 Type System

Yices 2 has a few built-in types for primitive objects:

- The arithmetic types `int` and `real`
- The boolean type `bool`
- The type `(bitvector k)` of bitvectors of size k, where k is a positive integer.

All these built-in types are *atomic*. The set of atomic types can be extended by declaring new *uninterpreted types* and *scalar types*. An uninterpreted type denotes a nonempty collection of objects with no cardinality constraint. A scalar type denotes a nonempty, *finite* set of objects. The cardinality of a scalar type is defined when the type is created.

In addition to the atomic types, Yices 2 provides constructors for tuple and function types. The set of all Yices 2 types can be defined inductively as follows:

- Any atomic type τ is a type.
- If $n > 0$ and $\sigma_1, \ldots, \sigma_n$ are n types, then $\sigma = (\sigma_1 \times \ldots \times \sigma_n)$ is a type. Objects of type σ are tuples $(x_1, \ldots, x_n)$ where x_i is an object of type σ_i.
- If $n > 0$ and $\sigma_1, \ldots, \sigma_n$ and τ are types, then $\sigma = (\sigma_1 \times \ldots \times \sigma_n \rightarrow \tau)$ is a type. Objects of type σ are functions of domain $\sigma_1 \times \ldots \times \sigma_n$ and range τ.

By construction, all the types are nonempty. Yices does not have a specific type constructor for arrays since the logic does not distinguish between arrays and functions. For example, an array indexed by integers is simply a function of domain `int`.

Yices 2 uses a simple form of subtyping. Given two types σ and τ, let $\sigma \sqsubset \tau$ denote that σ is a subtype of τ. Then the subtype relation is defined by the following rules:

- $\tau \sqsubset \tau$ (any type is a subtype of itself)
- $\texttt{int} \sqsubset \texttt{real}$ (the integers form a subtype of the reals)
- If $\sigma_1 \sqsubset \tau_1, \ldots, \sigma_n \sqsubset \tau_n$ then $(\sigma_1 \times \ldots \times \sigma_n) \sqsubset (\tau_1 \times \ldots \times \tau_n)$.
- If $\tau \sqsubset \tau'$ then $(\sigma_1 \times \ldots \times \sigma_n \rightarrow \tau) \sqsubset (\sigma_1 \times \ldots \times \sigma_n \rightarrow \tau')$.

For example, the type $(\texttt{int} \times \texttt{int})$ (pairs of integers) is a subtype of $(\texttt{real} \times \texttt{real})$ (pairs of reals).

Two types, τ and τ', are said to be *compatible* if they have a common supertype, that is, if there exists a type σ such that $\tau \sqsubset \sigma$ and $\tau' \sqsubset \sigma$. If that is the case, then there exists a unique minimal supertype among all the common supertypes. We denote the minimal supertype of τ and τ' by $\tau \sqcup \tau'$. By definition, we then have

$$\tau \sqsubset \sigma \ \texttt{and} \ \tau' \sqsubset \sigma \ \Rightarrow \ \tau \sqcup \tau' \ \sqsubset \ \sigma.$$

For example, the tuple types $\tau = (\texttt{int} \times \texttt{real} \times \texttt{int})$ and $\tau = (\texttt{int} \times \texttt{int} \times \texttt{real})$ are compatible. Their minimal supertype is $\tau \sqcup \tau' = (\texttt{int} \times \texttt{real} \times \texttt{real})$. The type $(\texttt{real} \times \texttt{real} \times \texttt{real})$ is also a common supertype of τ and τ' but it is not minimal.

2.2 Terms and Formulas

In Yices 2, the atomic terms include the boolean constants (`true` and `false`) as well as arithmetic and bitvector constants.

When a scalar type τ of cardinality n is declared, n distinct constant $c_1, \ldots, c_n$ of type τ are also implicitly defined. In the Yices 2 syntax, this is done via a declaration of the form:

```
(define-type tau (scalar c1 ... cn))
```

An equivalent functionality is provided by the Yices API. The API allows one to create a new scalar type and to access n constants of that type indexed by integers between 0 and $n-1$ (check file `include/yices.h` for explanations).

The user can also declare *uninterpreted constants* of arbitrary types. Informally, uninterpreted constants of type τ can be considered like global variables, but Yices (in particular the Yices API) makes a distinction between *variables* of type τ and *uninterpreted constants* of type τ. In the Yices API, variables are used to build quantified expressions and to support term substitutions. Free variables are not allowed to occur in assertions.

The term constructors include the common Boolean operators (conjunction, disjunction, negation, implication, etc.), an if-then-else constructor, equality, function application, and tuple constructor and projection. In addition, Yices provides an `update` operator that can be applied to arbitrary functions. The type-checking rules for these primitive operators are described in Figure 2.1, where the notation $t :: \tau$ means "term t has type τ".

There are no separate syntax or constructors for formulas. In Yices 2, a formula is simply a term of Boolean type.

The semantics of most of these operators is standard. The update operator for functions is characterized by the following axioms[1]:

$$
\begin{array}{rcl}
((\texttt{update}\ f\ t_1 \ldots t_n\ v)\ t_1 \ldots t_n) & = & v \\
u_1 \neq t_1 \vee \ldots \vee u_n \neq t_n \Rightarrow ((\texttt{update}\ f\ t_1 \ldots t_n\ v)\ u_1 \ldots u_n) & = & (f\ u_1 \ldots u_n)
\end{array}
$$

In other words, $(\texttt{update}\ f\ t_1 \ldots t_n\ v)$ is the function equal to f at all points except $(t_1, \ldots, t_n)$. Informally, if f is interpreted as an array then the update corresponds to "storing" v at position $t_1, \ldots, t_n$ in the array. Reading the content of the array is nothing other than function application: $(f\ i_1 \ldots i_n)$ is the content of the array at position $i_1, \ldots, i_n$.

The full Yices 2 language has a few more operators not described here, and it includes existential and universal quantifiers. We do not describe the type-checking rules for quantifiers here since Yices 2 does not have a solver for quantified formulas at this point.

2.3 Supported Theories

In addition to the generic operators presented previously, the Yices language includes the standard arithmetic operators and a rich set of bitvector operators.

2.3.1 Arithmetic

Arithmetic constants are arbitrary precision integers and rationals. Although Yices uses exact arithmetic, rational constants can be written using standard floating-point notation. Internally, Yices converts floating-point input to rationals. For example, the floating-point expression $3.04e - 1$ is converted to $304/1000$.

The Yices language supports the traditional arithmetic operators (i.e., addition, subtraction, multiplication) with the exception that it does not allow division by a non constant, to avoid issues related to division by zero. For example, the expression $(x + 4y)/3$ is allowed, but $3/(x + 4y)$ is not. The arithmetic predicates are the usual comparison operators, including both strict and nonstrict inequalities.

[1]These are the main axioms of the McCarthy theory of arrays.

Boolean Operators

$$\frac{t :: \texttt{bool}}{(\texttt{not}\ t) :: \texttt{bool}} \qquad \frac{t_1 :: \texttt{bool} \quad t_2 :: \texttt{bool}}{(\texttt{implies}\ t_1\ t_2) :: \texttt{bool}}$$

$$\frac{t_1 :: \texttt{bool} \ldots t_n :: \texttt{bool}}{(\texttt{or}\ t_1 \ldots t_n) :: \texttt{bool}} \qquad \frac{t_1 :: \texttt{bool} \ldots t_n :: \texttt{bool}}{(\texttt{and}\ t_1 \ldots t_n) :: \texttt{bool}}$$

Equality

$$\frac{t_1 :: \tau_1 \quad t_2 :: \tau_2}{(t_1 = t_2) :: \texttt{bool}} \text{ provided } \tau_1 \text{ and } \tau_2 \text{ are compatible}$$

If-then-else

$$\frac{c :: \texttt{bool} \quad t_1 :: \tau_1 \quad t_2 :: \tau_2}{(\texttt{ite}\ c\ t_1\ t_2) :: \tau_1 \sqcup \tau_2} \text{ provided } \tau_1 \text{ and } \tau_2 \text{ are compatible}$$

Tuple Constructor and Projection

$$\frac{t_1 :: \tau_1 \ldots t_n :: \tau_n}{(\texttt{tuple}\ t_1 \ldots t_n) :: (\tau_1 \times \ldots \times \tau_n)} \qquad \frac{t :: (\tau_1 \times \ldots \times \tau_n)}{(\texttt{select}_i\ t) :: \tau_i}$$

Function Application

$$\frac{f :: (\tau_1 \times \ldots \times \tau_n \to \tau) \quad t_1 :: \sigma_1 \ldots t_n :: \sigma_n \quad \sigma_1 \sqsubset \tau_1 \ldots \sigma_n \sqsubset \tau_n}{(f\ t_1 \ldots t_n) :: \tau}$$

Function Update

$$\frac{f :: (\tau_1 \times \ldots \times \tau_n \to \tau) \quad t_1 :: \sigma_1 \ldots t_n :: \sigma_n \quad v :: \sigma \quad \sigma_i \sqsubset \tau_i \quad \sigma \sqsubset \tau}{(\texttt{update}\ f\ t_1 \ldots t_n\ v) :: (\tau_1 \times \ldots \times \tau_n \to \tau)}$$

Figure 2.1: Primitive Operators and Type Checking

The language allows nonlinear polynomials but this is not fully supported by the tool at this time. Yices 2 can solve problems involving real and integer linear arithmethic, but it does not yet include a solver for nonlinear arithmetic.

2.3.2 Bitvectors

Yices supports all the bitvector operators defined in the SMT-LIB standard [RT06]. The most commonly used operators are listed in Table 2.1. They include bitvector arithmetic (where bitvectors are interpreted either as unsigned integers or as signed integers in two's complement representation), logical operators such as bitwise OR or AND, logical and arithmetic shifts, concatenation, and extraction of subvectors. Other operators are defined in the theory QF_BV of SMT-LIB (cf. http://combination.cs.uiowa.edu/smtlib); all of them are supported by Yices 2.

The semantics of all the bitvector operators is defined in the SMT-LIB 1.2 standard. Yices 2 follows the standard except for the case of division by zero. In SMT-LIB, the result of a division by zero is undefined but one must ensure that the division operators are functional. In other words, SMT-LIB does not specify the result of $(\texttt{bvudiv}\ a\ b)$ if b is the zero vector, but $(\texttt{bvudiv}\ a\ b)$ and $(\texttt{bvudiv}\ c\ b)$ must be equal whenever $a = c$, even if b is the zero vector. Yices 2 uses a simpler semantics (inspired from the BTOR format [BBL08]):

- **Unsigned Division:** If b is the zero bitvector of n bits then

  $$\begin{aligned} (\texttt{bvudiv}\ a\ b) &= \texttt{0b111...1} \\ (\texttt{bvurem}\ a\ b) &= a \end{aligned}$$

  In general, the quotient $(\texttt{bvudiv}\ a\ b)$ is the largest unsigned integer that can be represented on n bits, and is smaller than a/b, and the following idendity for all bitvectors a and b

  $$a = (\texttt{bvadd}\ (\texttt{bvmul}\ (\texttt{bvudiv}\ a\ b)\ b)\ (\texttt{bvurem}\ a\ b))$$

- **Signed Division:** If b is the zero bitvector of n bits then

  $$\begin{aligned} (\texttt{bvsdiv}\ a\ b) &= \texttt{0b000..01} \text{ if } a \text{ is negative} \\ (\texttt{bvsdiv}\ a\ b) &= \texttt{0b111...1} \text{ if } a \text{ is non-negative} \\ (\texttt{bvsrem}\ a\ b) &= a \\ (\texttt{bvsmod}\ a\ b) &= a \end{aligned}$$

Operator and Type	Meaning
$\texttt{bvadd} :: ((\texttt{bv}\ n) \times (\texttt{bv}\ n) \to (\texttt{bv}\ n))$	addition
$\texttt{bvsub} :: ((\texttt{bv}\ n) \times (\texttt{bv}\ n) \to (\texttt{bv}\ n))$	subtraction
$\texttt{bvmul} :: ((\texttt{bv}\ n) \times (\texttt{bv}\ n) \to (\texttt{bv}\ n))$	multiplication
$\texttt{bvneg} :: \texttt{bv}\ n) \to (\texttt{bv}\ n))$	2's complement opposite
$\texttt{bvudiv} :: ((\texttt{bv}\ n) \times (\texttt{bv}\ n) \to (\texttt{bv}\ n))$	quotient in unsigned division
$\texttt{bvudiv} :: ((\texttt{bv}\ n) \times (\texttt{bv}\ n) \to (\texttt{bv}\ n))$	remainder in unsigned division
$\texttt{bvsdiv} :: ((\texttt{bv}\ n) \times (\texttt{bv}\ n) \to (\texttt{bv}\ n))$	quotient in signed division with rounding toward zero
$\texttt{bvsrem} :: ((\texttt{bv}\ n) \times (\texttt{bv}\ n) \to (\texttt{bv}\ n))$	remainder in signed division with rounding toward zero
$\texttt{bvsmod} :: ((\texttt{bv}\ n) \times (\texttt{bv}\ n) \to (\texttt{bv}\ n))$	remainder in signed division with rounding toward $-\infty$
$\texttt{bvule} :: ((\texttt{bv}\ n) \times (\texttt{bv}\ n) \to \texttt{bool}$	unsigned less than or equal
$\texttt{bvuge} :: ((\texttt{bv}\ n) \times (\texttt{bv}\ n) \to \texttt{bool}$	unsigned greater than or equal
$\texttt{bvult} :: ((\texttt{bv}\ n) \times (\texttt{bv}\ n) \to \texttt{bool}$	unsigned less thanl
$\texttt{bvugt} :: ((\texttt{bv}\ n) \times (\texttt{bv}\ n) \to \texttt{bool}$	unsigned greater than
$\texttt{bvsle} :: ((\texttt{bv}\ n) \times (\texttt{bv}\ n) \to \texttt{bool}$	unsigned less than or equal
$\texttt{bvsge} :: ((\texttt{bv}\ n) \times (\texttt{bv}\ n) \to \texttt{bool}$	unsigned greater than or equal
$\texttt{bvslt} :: ((\texttt{bv}\ n) \times (\texttt{bv}\ n) \to \texttt{bool}$	unsigned less thanl
$\texttt{bvsgt} :: ((\texttt{bv}\ n) \times (\texttt{bv}\ n) \to \texttt{bool}$	unsigned greater than
$\texttt{bvand} :: ((\texttt{bv}\ n) \times (\texttt{bv}\ n) \to (\texttt{bv}\ n))$	bitwise and
$\texttt{bvor} :: ((\texttt{bv}\ n) \times (\texttt{bv}\ n) \to (\texttt{bv}\ n))$	bitwise or
$\texttt{bvnot} :: ((\texttt{bv}\ n) \to (\texttt{bv}\ n))$	bitwise negation
$\texttt{bvxor} :: ((\texttt{bv}\ n) \times (\texttt{bv}\ n) \to (\texttt{bv}\ n))$	bitwise exclusive or
$\texttt{bvshl} :: ((\texttt{bv}\ n) \times (\texttt{bv}\ n) \to (\texttt{bv}\ n))$	shift left
$\texttt{bvlshr} :: ((\texttt{bv}\ n) \times (\texttt{bv}\ n) \to (\texttt{bv}\ n))$	logical shift right
$\texttt{bvashr} :: ((\texttt{bv}\ n) \times (\texttt{bv}\ n) \to (\texttt{bv}\ n))$	arithmetic shift right
$\texttt{bvconcat} :: ((\texttt{bv}\ n) \times (\texttt{bv}\ m) \to (\texttt{bv}\ n+m))$	concatenation
$\texttt{bvextract}_{i,j}((\texttt{bv}\ n) \to (\texttt{bv}\ m))$	extract bits i down to j form a bitvector of size n

Table 2.1: Bitvector Operators

Chapter 3

Yices 2 Architecture

Yices 2 relies on a simpler language and type system than Yices 1. We have also completely redesigned the architecture to make Yices 2 easier to maintain and develop. The new architecture supports new features, such as the possibility to maintain several contexts in parallel.

3.1 Main Components

The Yices 2 software can be conceptually decomposed into three main modules:

Term Database Yices 2 maintains a global database in which all terms and types are stored. Yices 2 provides an API for constructing terms, formulas, and types in this database.

Context Management A context is a central data structure that stores asserted formulas. Each context contains a set of assertions to be checked for satisfiability. The context-management API supports operations for creating and initializing contexts, for asserting formulas into a context, and for checking the satisfiability of the asserted formulas. Several contexts can be constructed and manipulated independently.

Contexts are highly customizable. Each context can be configured to support a specific theory, and to use a specific solver or combination of solvers.

Model Management If the set of formulas asserted in a context is satisfiable, then one can construct a model of the formulas. The model maps symbols of the formulas to concrete values (e.g., integer or rational values or bitvector constants). The API provides functions to build and query models.

Figure 3.1 shows the top-level architecture of Yices 2, divided into the three main modules. Each context consists of two separate components: The *solver* employs a Boolean satisfiability solver and decision procedures for determining whether the formulas asserted

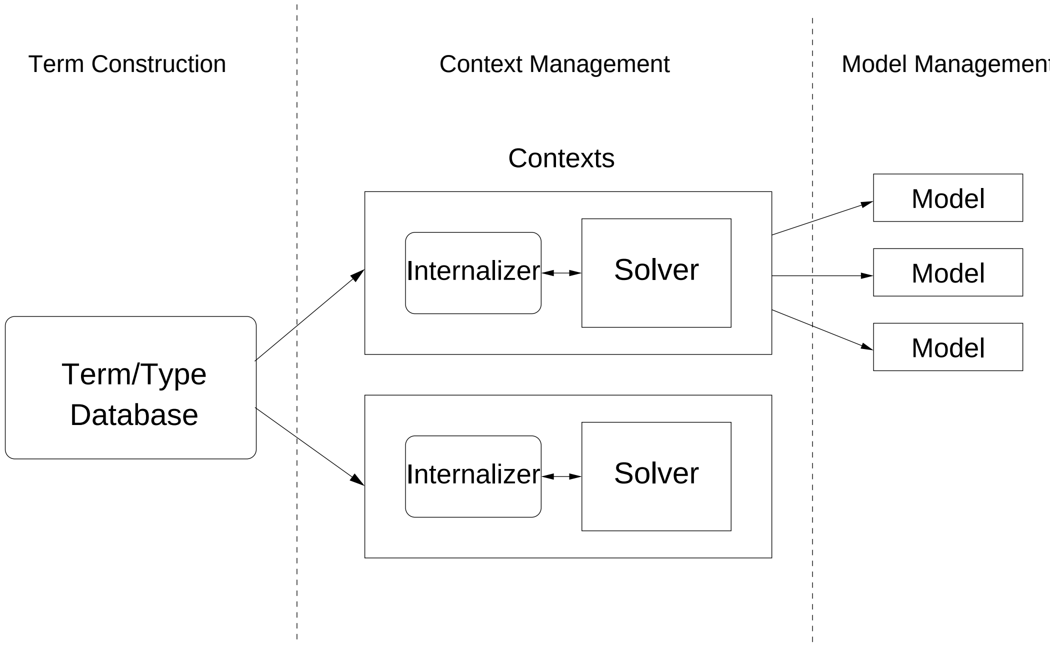

Figure 3.1: Top-level Yices 2 Architecture

in the context are satisfiable. The *internalizer* converts the format used by the term database into the internal format used by the solver. In particular, the internalizer rewrites all formulas in conjunctive normal form, which is used by the internal SAT solver.

3.2 Solvers

In Yices 2, it is possible to select a different solver (or combination of solvers) for the problem of interest. Each context can thus be configured for a specific class of formulas. For example, one can use a solver specialized for linear arithmetic, or use a solver that supports the full Yices 2 language. Figure 3.2 shows how the most general solver is built. A major component of all solvers is a SAT solver based on the Davis-Putnam-Logemann-Loveland (DPLL) procedure. The SAT solver is coupled with one or more so-called *theory solvers*. Each theory solver implements a decision procedure for a particular theory. Currently, Yices 2 includes four main theory solvers:

- The *UF Solver* deals with the theory of uninterpreted functions with equality[1]. It implements a decision procedure based on computing congruence closures, similar to the Simplify system [DNS05].

[1] UF stands for uninterpreted functions.

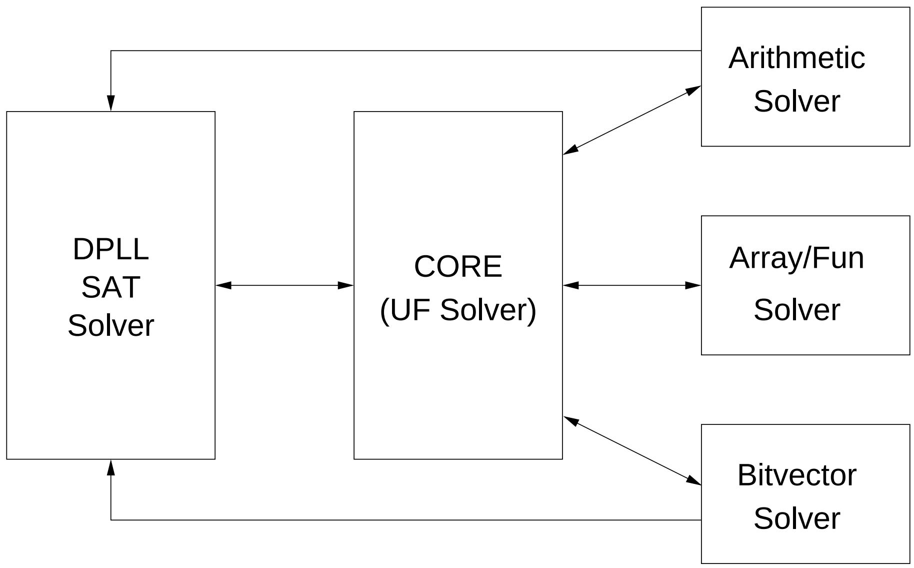

Figure 3.2: Solver Components

- The *Arithmetic Solver* deals with linear integer and real arithmetic. It implements a decision procedure based on the Simplex algorithm [DdM06a, DdM06b].
- The *Bitvector Solver* deals with the theory of bitvectors.
- The *Array Solver* implements a decision procedure for McCarthy's theory of arrays.

Yices 2 employs a modular solver architecture. It is possible to remove some of the components of Figure 3.2 to build simpler and more efficient solvers that are specialized for specific classes of formulas. For example, a solver for pure arithmethic can be built by directly attaching the arithmetic solver to the DPLL SAT solver. Similarly, Yices 2 can be specialized for pure bitvector problems, or for problems combining uninterpreted functions, arrays, and bitvectors (by removing the arithmetic solver).

Yices 2 combines several theory solver using the Nelson-Oppen method [NO79]. The UF solver is essential for this purpose; it coordinates the different theory solvers and ensures global consistency. The other solvers (for arithmetic, arrays, and bitvectors) communicate only with the central UF solver and never directly with each other. This property considerably simplifies the design and implementation of theory solvers.

Chapter 4

`yices`

The Yices 2 distribution includes a tool for processing input written in the Yices 2 language. This tool is callled `yices` (or `yices.exe` in the Windows and Cygwin distributions). The syntax and the set of commands supported by `yices` are explained in the file `doc/YICES-LANGUAGE` included in the distribution. Seveval example specifications are also included in the `examples/` directory.

By default, the `yices` tool supports the combination of arithmetic, uninterpreted functions and arrays. It builds a context that includes the Simplex, UF, and Array solvers. This can be changed by giving command-line arguments to the tool. Try `yices --help` for more details.

Chapter 5

yices-smt

Another tool included in the distribution can process input written in the SMT-LIB notation. This tool is called `yices-smt` (or `yices-smt.exe`). It is included in the `bin` directory. Currently, this tool supports version 1.2 of SMT-LIB. Support for the more recent SMT-LIB 2 will be provided in future releases.

Chapter 6

Yices API

The distribution includes a library and header files for embedding Yices in other software. The main header file is `yices.h` which includes all the API. The API functions are documented in this header file. More complete and detailed documentation on the Yices 2 API will be provided at the Yices website http://yices.csl.sri.com/.

Chapter 7

Yices License Terms

Before downloading and using Yices, you will be asked to agree to the Yices license terms reproduced below. SRI is open to distributing Yices under other agreements. Contact us at `fm-licencing@csl.sri.com` to discuss alternative licence terms.

```
END-USER LICENSE AGREEMENT

IMPORTANT - READ CAREFULLY.  Be  sure to carefully read and understand
all of the rights and  restrictions described in this End-User License
Agreement ("EULA").  You will be  asked to review and either accept or
not accept the terms of the EULA.  You will not be permitted to access
or use the Software unless or  until you accept the terms of the EULA.
Alternative  license  terms may  be  available  to  you by  contacting
fm-licensing@csl.sri.com.

This EULA is a legal agreement  between you (either an individual or a
single entity) and SRI International ("SRI") for the software referred
to by SRI as "Yices",  which includes the computer software accessible
via  this web  browser interface,  and may  include  associated media,
printed  materials  and   any  "online"  or  electronic  documentation
("Software").  By utilizing the Software, you agree to be bound by the
terms of this  EULA.  If you do  not agree to the terms  of this EULA,
you may not access or use the Software.

GRANT  OF LIMITED  LICENSE.   SRI  hereby grants  to  you a  personal,
non-exclusive,  non-transferable, royalty-free  license to  access and
use  the Software  for your  own internal  purposes.  The  Software is
licensed to  you, and such license  does not constitute a  sale of the
Software.   SRI  reserves the  right  to  release  the Software  under
different license terms or to stop distributing or providing access to
the Software at any time.

RESTRICTIONS.  You may not:  (i) distribute, sublicense, rent or lease
the  Software;  (ii)  modify,  adapt,  translate,  reverse  engineer,
decompile,  disassemble  or  create  derivative  works  based  on  the
```

Software; or (iii) create more than one (1) copy of the Software or any related documentation.

OWNERSHIP. SRI is the sole owner of the Software. You agree that SRI retains title to and ownership of the Software and that you will keep confidential and use your best efforts to prevent and protect the Software from unauthorized access, use or disclosure. All trademarks, service marks, and trade names are proprietary to SRI. All rights not expressly granted herein are hereby reserved.

TERMINATION. The EULA is effective upon the date you first use the Software and shall continue until terminated as specified below. You may terminate the EULA at any time prior to the natural expiration date by destroying the Software and any and all related documentation and copies and installations thereof, whether made under the terms of these terms or otherwise. SRI may terminate the EULA if you fail to comply with any condition of the EULA or at SRI's discretion for good cause. Upon termination, you must destroy the Software in your possession, if any, and any and all copies thereof. In the event of termination for any reason, the provisions set forth under the paragraphs entitled DISCLAIMER OF ALL WARRANTIES, EXCLUSION OF ALL DAMAGES, and LIMITATION AND RELEASE OF LIABILITY shall survive.

U.S. GOVERNMENT RESTRICTED RIGHTS. The Software is deemed to be "commercial software" and "commercial computer software documentation", respectively, pursuant to DFARS 227.7202 and FAR 12.212, as applicable. Any use, modification, reproduction, release, performance, display, or disclosure of the Software by the U.S. Government or any of its agencies or by a U.S. Government prime contractor or subcontractor (at any tier) shall have only "Restricted Rights", shall be governed solely by the terms of this EULA, and shall be prohibited except to the extent expressly permitted by the terms of this EULA.

DISCLAIMER OF ALL WARRANTIES. SRI PROVIDES THE SOFTWARE "AS IS" AND WITH ALL FAULTS, AND HEREBY DISCLAIMS ALL OTHER WARRANTIES AND CONDITIONS, EITHER EXPRESS, IMPLIED OR STATUTORY, INCLUDING BUT NOT LIMITED TO ANY (IF ANY) IMPLIED WARRANTIES OR CONDITIONS OF MERCHANTABILITY, OF FITNESS FOR A PARTICULAR PURPOSE, OF LACK OF VIRUSES AND OF LACK OF NEGLIGENCE OR LACK OF WORKMANLIKE EFFORT. ALSO, THERE IS NO WARRANTY OR CONDITION OF TITLE, OF QUIET ENJOYMENT OR OF NON-INFRINGEMENT. THE ENTIRE RISK ARISING OUT OF THE USE OR PERFORMANCE OF THE SOFTWARE IS WITH YOU.

EXCLUSION OF ALL DAMAGES. TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, IN NO EVENT SHALL SRI BE LIABLE FOR ANY CONSEQUENTIAL, INCIDENTAL, DIRECT, INDIRECT, SPECIAL, PUNITIVE OR OTHER DAMAGES WHATSOEVER (INCLUDING, WITHOUT LIMITATION, DAMAGES FOR ANY INJURY TO PERSON OR PROPERTY, DAMAGES FOR LOSS OF PROFITS, BUSINESS INTERRUPTION, LOSS OF BUSINESS INFORMATION, FOR LOSS OF PRIVACY FOR

FAILURE TO MEET ANY DUTY INCLUDING OF GOOD FAITH OR OF REASONABLE CARE, FOR NEGLIGENCE AND FOR ANY PECUNIARY OR OTHER LOSS WHATSOEVER) ARISING OUT OF OR IN ANY WAY RELATED TO THE USE OF OR INABILITY TO USE THE SOFTWARE, EVEN IF SRI HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. THIS EXCLUSION OF DAMAGES SHALL BE EFFECTIVE EVEN IF ANY REMEDY FAILS OF ITS ESSENTIAL PURPOSE.

LIMITATION AND RELEASE OF LIABILITY. SRI has included in this EULA terms that disclaim all warranties and liability for the Software. To the full extent allowed by law, YOU HEREBY RELEASE SRI FROM ANY AND ALL LIABILITY ARISING FROM OR RELATED TO ALL CLAIMS CONCERNING THE SOFTWARE OR ITS USE. If you do not wish to accept access to the Software under the terms of this EULA, do not access or use the Software. No refund will be made because the SOFTWARE was provided to you at no charge. Independent of, severable from, and to be enforced independently of any other provision of this EULA, UNDER NO CIRCUMSTANCE SHALL SRI’S aggregate LIABILITY TO YOU (INCLUDING LIABILITY TO ANY THIRD PERSON OR PERSONS WHOSE CLAIM OR CLAIMS ARE BASED ON OR DERIVED FROM A RIGHT OR RIGHTS CLAIMED BY YOU), WITH RESPECT TO ANY AND ALL CLAIMS AT ANY AND ALL TIMES ARISING FROM OR RELATED TO THE SUBJECT MATTER OF THIS EULA, IN CONTRACT, TORT, OR OTHERWISE, EXCEED THE TOTAL AMOUNT ACTUALLY PAID BY YOU to SRI pursuant to THIS EULA, IF ANY.

JURISDICTIONAL ISSUES. This Software is controlled by SRI from its offices within the State of California. SRI makes no representation that the Software is appropriate or available for use in other locations. Those who choose to access this Software from other locations do so at their own initiative and are responsible for compliance with local laws, if and to the extent local laws are applicable. You hereby acknowledge that the rights and obligations of the EULA are subject to the laws and regulations of the United States relating to the export of products and technical information. Without limitation, you shall comply with all such laws and regulations, including the restriction that the Software may not be accessed from, used or otherwise exported or reexported (i) into (or to a national or resident of) any country to which the U.S. has embargoed goods; or (ii) to anyone on the U.S. Treasury Department’s list of Specialty Designated Nationals or the U.S. Commerce Department’s Table of Deny Orders. By accessing or using the Software, you represent and warrant that you are not located in, under the control of, or a national or resident of any such country on any such list.

Notice and Procedure for Making Claims of Copyright Infringement. Pursuant to Title 17, United States Code, Section 512(c)(2), notifications of claimed copyright infringement should be sent to SRI International, Office of the General Counsel, 333 Ravenswood Ave., Menlo Park, CA 94025.

SUPPORT, UPDATES AND NEW RELEASES. The EULA does not grant you any rights to any software support, enhancements or updates. Any updates or new releases of the Software which SRI chooses at its own discretion to distribute or provide access to shall be subject to the terms hereof.

GENERAL INFORMATION. The EULA constitutes the entire agreement between you and SRI and governs your access to and use of the Software. The EULA shall not be modified except in writing by both parties.

The EULA shall be governed by and construed in accordance with the laws of the State of California, without regard to the conflicts of law principles thereof. The parties shall resolve any disputes arising out of this EULA, including disputes about the scope of this arbitration provision, by final and binding arbitration seated and held in San Francisco, California before a single arbitrator. JAMS shall administer the arbitration under its comprehensive arbitration rules and procedures. The arbitrator shall aware the prevailing party its reasonable attorney's fees and expenses, and its arbitration fees and associated costs. Any court of competent jurisdiction may enter judgment on the award.

If any provision of the EULA shall be deemed unlawful, void, or for any reason unenforceable, then that provision shall be deemed severable from these terms and shall not affect the validity and enforceability of any remaining provisions.

In consideration of your use of the Software, you represent that you are of legal age to form a binding contract and are not a person barred from receiving services under the laws of the United States or other applicable jurisdiction.

The failure of SRI to exercise or enforce any right or provision of the EULA shall not constitute a waiver of such right or provision.

Bibliography

[BBL08] R. Brummayer, A. Biere, and F. Lonsing. BTOR: Bit-Precise Modelling of Word-Level Problems for Model Checking. In *First International Workshop on Bit-Precise Reasoning*, pages 53–64, 2008. Available at http://fmv.jku.at/BrummayerBiereLonsing-BPR08.pdf.

[DdM06a] Bruno Dutertre and Leonardo de Moura. A fast linear-arithmetic solver for DPLL(T). In *Computer-Aided Verification (CAV'2006)*, volume 4144 of *Lecture Notes in Computer Science*, pages 81–94. Springer Verlag, August 2006.

[DdM06b] Bruno Dutertre and Leonardo de Moura. Integrating Simplex with DPLL(T). Technical Report SRI-CSL-06-01, Computer Science Laboratory, SRI International, May 2006. Available at http://yices.csl.sri.com/sri-csl-06-01.pdf.

[DNS05] D. Detlefs, G. Nelson, and J. B. Saxe. Simplify: a Theorem Prover for Program Checking. *Journal of the ACM*, 52(3):365–473, May 2005.

[NO79] G. Nelson and D. C. Oppen. Simplification by Cooperating Decision Procedures. *ACM Transactions on Programming Languages and Systems*, 1(2):245–257, 1979.

[RT06] Silvio Ranise and Cesare Tinelli. The SMT-LIB Standard: Version 1.2. Technical report, SMT-LIB Initiative, 2006. Available at http://www.smtlib.org.