

System Description: Yices 1.0.10

Bruno Dutertre

Computer Science Laboratory, SRI International
333 Ravenswood Avenue, Menlo Park, CA 94025 - USA
`bruno@csl.sri.com`

1 Introduction

Yices 1.0.10 is the most recent distribution of the Yices decision procedure. It can decide formulas containing uninterpreted function symbols, linear real and integer arithmetic, fixed-size bitvectors, extensional arrays, lambda expressions, tuples, records, quantifiers, scalar types, recursive datatypes, and dependent types. Compared to Yices 1.0—the version that participated in SMT-COMP’06—the current version includes many bug fixes, some improvements to the bitvector solver, and extensions to support the new SMT-LIB logics and bitvector operators. Yices can be downloaded at <http://yices.csl.sri.com>. An improved version is under development and should be released by the end of the year.

2 Algorithms and Implementation

Yices is implemented in C++. The architecture integrates a modern DPLL-based SAT solver, a *core theory solver* that handles equalities and uninterpreted functions, and *satellite solvers* (for arithmetic, arrays, tuples, etc.). Yices uses an extension to the standard Nelson-Oppen combination method: The core and satellite theories communicate via *offset equalities*, that is, equalities of the form $x = y + k$ where x and y are terms and k is a rational constant. By using offset equalities, the core handles simple arithmetic constraints directly, which, in many cases, avoids the overhead of communicating with a dedicated solver. The core algorithm is similar to the one used by Simplify [2], with extensions for producing precise explanations and for handling offset equalities.

Yices case splits on (offset) equalities between shared variables to achieve completeness. Each theory is responsible for creating the required case splits, and simple filters are used to minimize the number of case splits. For example, if the core contains four terms $f(x_1, x_2)$, $f(x_3, x_4)$, $g(x_5)$, and $g(x_6)$, and x_1 to x_6 are the only shared variables then case splitting on the three interface equalities $x_1 = x_3$, $x_2 = x_4$ and $x_5 = x_6$ is sufficient.

The linear arithmetic solver uses the Simplex-based algorithm described in [3,4]. This algorithm is efficient for sparse problems in both full linear arithmetic and the difference logic fragment. For dense difference-logic problems, Yices uses a specialized algorithm based on incremental Floyd-Warshall.

Yices employs a dynamic form of Ackermann’s reduction when uninterpreted functions are present. Yices creates the clause $x \neq y \vee f(x) = f(y)$ whenever

the congruence rule $x = y \rightsquigarrow f(x) = f(y)$ is used to deduce a conflict. Using this technique, Yices can perform the propagation $f(x) \neq f(y) \rightsquigarrow x \neq y$, which is missed by traditional congruence-closure algorithms. This propagation rule has a dramatic performance benefit on many problems. Creating Ackermann clauses during the search rather than all from the start avoids flooding the SAT solver with unnecessary instances. Furthermore, the DPLL solver clause-deletion heuristics can safely remove any of the dynamically created instances since they are not required for completeness.

Yices uses three methods for handling universally quantified expressions. The main approach is an extension of egraph-matching [2] that supports offset equalities and terms. Yices can use several triggers for each universally quantified expression, and the triggers are fired using a heuristic that gives preference to the most conservative ones. Yices also uses Fourier-Motzkin elimination to simplify quantified expressions involving linear arithmetic. Finally, Yices uses an instantiation heuristic based on the approach described in [1].

The array-theory solver uses lazy instantiation of the array axioms. The fixed-size bit-vectors theory applies bit-blasting to all bitvector operators but equality. This solver just implements a bridge between the core theory and the encoding of the bitvector operations in the SAT solver.

3 Problem Divisions

Yices will participate in all divisions of SMT-COMP'07.

References

1. A. R. Bradley, Z. Manna, and H. B. Sipma. What's decidable about arrays? In *In Proc. Verification, Model-Checking, and Abstract-Interpretation (VMCAI'06)*, volume 3855 of *LNCS*. Springer, 2006. 2
2. D. Detlefs, G. Nelson, and J. B. Saxe. Simplify: A theorem prover for program checking. Technical Report HPL-2003-148, HP Labs, 2003. 1, 2
3. B. Dutertre and L. de Moura. A Fast Linear-Arithmetic Solver for DPLL(T). In T. Ball and R.B. Jones, editors, *Int. Conference on Computer Aided Verification (CAV'06)*, volume 4144 of *LNCS*, pages 81–94. Springer, 2006. 1
4. B. Dutertre and L. de Moura. Integrating simplex with DPLL(T). Technical Report CSL-06-01, CSL, SRI International, 2006. 1