The YICES SMT Solver

Bruno Dutertre and Leonardo de Moura

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

Abstract. SMT stands for Satisfiability Modulo Theories. An SMT solver decides the satisfiability of propositionally complex formulas in theories such as arithmetic and uninterpreted functions with equality. SMT solving has numerous applications in automated theorem proving, in hardware and software verification, and in scheduling and planning problems.

This paper describes Yices, an efficient SMT solver developed at SRI International. Yices supports a rich combination of first-order theories that occur frequently in software and hardware modeling: arithmetic, uninterpreted functions, bit vectors, arrays, recursive datatypes, and more. Beyond pure SMT solving, Yices can solve weighted MAX-SMT problems, compute unsatisfiable cores, and construct models. Yices is the main decision procedure used by the SAL model checking environment, and it is being integrated to the PVS theorem prover. As a MAX-SMT solver, Yices is the main component of the probabilistic consistency engine used in SRI's CALO system.

1 Introduction

Recent breakthroughs in boolean satisfiability (SAT) solving have enabled new approaches to software and hardware verification. Existing SAT solvers can handle problems with millions of clauses and variables that are encountered in bounded model checking, test-case generation, and certain types of planning problems. SAT solving has thus become a major tool in automated analysis of hardware and other finite systems. Satisfiability modulo theories (SMT) generalizes SAT by adding equality reasoning, arithmetic, and other useful first-order theories. An SMT solver is a tool for deciding the satisfiability (or dually the validity) of formulas in these theories. SMT solvers enable application of bounded model checking to infinite systems. They have numerous applications in theorem proving and other domains such as real-time scheduling, temporal or metric planning, and test-case generation.

This paper describes Yices —an SMT solver developed at SRI International— that is capable of handling large and propositionally complex formulas in a rich combination of theories. Yices can be downloaded free of charge at `http://yices.csl.sri.com/`. It is available for common hardware platforms and operating systems. Tutorial material, documentation, and examples are also available on the Yices website.

Yices is the default decision procedure used by the SAL finite and infinite-state bounded model checkers, and it is included in the latest SAL distribution (`http://sal.csl.sri.com/`). Yices is also integrated into SRI's PVS theorem prover, which is available at `http://pvs.csl.sri.com/`.

2 Architecture and Algorithms

The main components of Yices are depicted in Figure 1. Yices integrates an efficient SAT solver based on the Davis-Putnam-Logemann-Loveland (DPLL) algorithm with specialized theory solvers that handle the first-order theories. A *core theory solver* handles equalities and uninterpreted functions. It is complemented by *satellite solvers* for other theories such as arithmetic, bit vectors, or arrays.

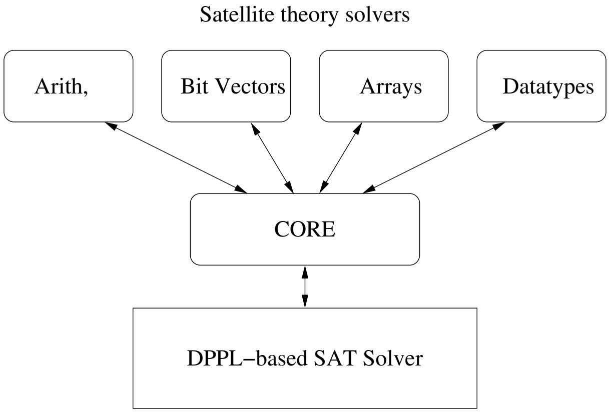

Fig. 1. Yices Architecture

The SAT solving algorithm used in Yices is a modern variant of DPLL that relies on techniques introduced by Chaff [8]: two watched literals, non-chronological backtracking based on learned clauses and UIP computation, and a VSIDS-like search heuristic. However, the Yices DPLL implementation is considerably more flexible than traditional SAT solvers to support efficient interaction with the core and satellite solvers. The latter are allowed to dynamically create literals and add clauses during the DPLL search, and to send *explanations* to the SAT solver when they detect inconsistencies or propagate implied literals (see [7,3]).

Yices uses a generalization of the Nelson-Oppen method for combining decision procedures [9]. The core and satellite solvers exchange *offset equalities*, that is, equalities of the form $x = y + k$ where x and y are terms and k is a rational constant. This extends the standard Nelson-Oppen approach in which decision procedures communicate equalities of the form $x = y$. By using offset equalities, the core solver can perform simple arithmetic reasoning, which in many cases avoids the overhead of communicating with a dedicated arithmetic solver. The core solver implements a congruence closure algorithm similar to the one used in Simplify [4], with extensions for producing precise explanations and for handling offset equalities.

Completeness in the Nelson-Oppen method requires the satellite and core solvers to agree on equalities between interface variables, that is, on variables that occur in more than one theory. To achieve this agreement, Yices generates a sufficient set of interface offset equalities and performs case-splits on these. This is similar to the method discussed in [1]. As a consequence, satellite solvers are not required to perform complete equality propagation, which can be expensive. Yices uses a filtering mechanism to avoid creating unnecessary interface equalities. 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 a novel Simplex-based algorithm [5,6]. This algorithm is very efficient for sparse problems in both full linear arithmetic and the difference logic fragment. On sparse problems, this solver is competitive with (and often outperforms) state-of-the-art tools specialized for difference logic. For dense difference-logic problems, Yices uses a specialized solver based on an incremental form of the Floyd-Warshall algorithm.

Yices employs a dynamic form of Ackermann's reduction when uninterpreted functions are present. The core 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* [4] 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. To complement egraph matching, Yices applies Fourier-Motzkin elimination to simplify quantified expressions involving linear arithmetic, and uses an instantiation heuristic based on the approach described in [2] for arrays.

The array-theory solver uses lazy instantiation of the array axioms. The fixed-size bitvectors theory relies on rewriting to perform simplifications, then 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 Using Yices

Yices is distributed both as a standalone tool and as a library. In its basic use, Yices reads a formula from a file and checks whether that formula is satisfiable. Optionally, Yices can output a model if the formula is satisfiable. In semi-decidable theories —for example, if the formula includes quantifiers— Yices returns "unknown" if it fails to show that the formula is unsatisfiable.

Yices can also be used in a more interactive fashion where the user has finer control. Yices maintains an internal logical context— that is, a set of declarations and assertions— and provides commands for manipulating the context by adding or retracting assertions, checking satisfiability of the current set of assertions, and backtracking. User-controlled backtracking is available via a push/pop mechanism.

Yices has its own input language but it also accepts specifications written in the SMT-LIB notation. Yices supports all the theories currently defined in SMT-LIB [10], including uninterpreted functions, difference logic, linear real and integer arithmetic, extensional arrays, and bit vectors. First order quantification is supported.

More logical theories are available when Yices's native language is used. This language supports recursive datatypes, tuples, records, and lambda expressions in addition to the previous theories. This language is also more flexible than SMT-LIB as theories can be mixed arbitrarily. For example, mixed (real and integer) arithmetic is supported. The Yices language is related to the PVS and SAL languages but it has a Lisp-like syntax. It is strongly typed and provides constructs for defining subtypes and dependent types. The language also provides commands that give access to the full range of Yices functionalities including model construction, MAX-SMT, and manipulation of the logical context.

In addition to pure SMT solving, Yices includes a MAX-SMT solver. The user can assign numerical weights to assertions and ask Yices to search for a satisfying assignment of maximal weight. In this mode, Yices is similar to a MAX-SAT solver that works on first-order formulas rather than simply booleans. Finally, Yices can be used as a regular SAT or MAX-SAT solver; it accepts input in the DIMACS CNF format.

Yices can be embedded in user software as a library that provides the same functionalities as the standalone tool. The library API is documented at `http://yices.csl.sri.com/`.

4 Applications

Bounded Model Checking. The latest version of SAL 2.4 includes Yices 1.0 as a backend solver. In SAL, Yices is used both as a regular SAT solver for traditional, finite-state bounded model checking and as the default SMT solver for the SAL infinite-state bounded model checker. In these roles, Yices is replacing ICS 2.0 an older SMT solver developed at SRI. Yices 1.0 is more robust and several order faster than ICS.

Theorem Proving. Yices 1.0 is embedded as a new decision procedure in PVS. In this application, Yices complements the existing PVS decision procedure that is primarily designed to support interactive theorem proving but can be overwhelmed by large proof goals with a complex propositional structure. Yices provides the capability to automatically discharge such goals in a "fire-and-forget" mode. The expressiveness of the Yices language and type system are essential in such applications, to enable arbitrary PVS formulas to be translated to and solved by Yices.

Integrated Learning and Reasoning. As part the CALO system (`http://caloproject.sri.com/`), Yices is the main component of a probabilistic consistency module. CALO stands for Cognitive Assistant that Learns and Organizes. It is a system based on learning and reasoning algorithms whose goal is to learn tasks by interacting with its users, adapts to user preferences, and assist them by automating routing tasks. Part of the CALO challenge involves combining results from several learning algorithms. This requires taking into account both logical facts that are known to be true and uncertain and possibly conflicting facts that different learning algorithm may produce. Yices handles this task as a MAX-SMT problem: the learned facts are asserted with different weights depending that reflect degrees of uncertainty. Yices resolve conflicts by constructing a model of maximal weight, that attempts to satisfy all assertions or when that is not possible discounts the most uncertain facts.

5 Acknowledgments

This work was supported by the National Science Foundation, under Grants CCR-ITR-0326540 and CCR-0311348, and by the Defense Advanced Research Projects Agency (DARPA) under Contract No. NBCHD030010. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of DARPA or the Department of Interior National Business Center (DOI-NBC).

References

1. M. Bozzano, R. Bruttomesso, A. Cimatti, T. Junttila, P. van Rossum, S. Ranise, and R. Sebastiani. Efficient satisfiability modulo theories via delayed theory combination. In *Int. Conf. on Computer-Aided Verification (CAV)*, volume 3576 of *LNCS*. Springer, 2005.
2. 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.
3. Leonardo de Moura and Harald Rueß. Lemmas on demand for satisfiability solvers. In *Proceedings of the Fifth International Symposium on the Theory and Applications of Satisfiability Testing (SAT 2002)*. Cincinnati, Ohio, 2002.
4. D. Detlefs, G. Nelson, and J. B. Saxe. Simplify: A theorem prover for program checking. Technical Report HPL-2003-148, HP Labs, 2003.
5. 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.
6. B. Dutertre and L. de Moura. Integrating simplex with DPLL(T). Technical Report CSL-06-01, CSL, SRI International, 2006.
7. H. Ganzinger, G. Hagen, R. Nieuwenhuis, A. Oliveras, and C. Tinelli. DPLL(T): Fast decision procedures. In R. Alur and D. Peled, editors, *Int. Conference on Computer Aided Verification (CAV'04)*, volume 3114 of *LNCS*, pages 175–188. Springer, 2004.
8. Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: Engineering an Efficient SAT Solver. In *Proc. of DAC'01*, 2001.
9. G. Nelson and D. C. Oppen. Simplification by cooperating decision procedures. *ACM Transactions on Programming Languages and Systems*, 1(2):245–257, 1979.
10. Silvio Ranise and Cesare Tinelli. The satisfiability modulo theories library (smt-lib), 2006. Available at `http://goedel.cs.uiowa.edu/smtib`.