Yices 2.2 *

Bruno Dutertre

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

Abstract. Yices is an SMT solver developed by SRI International. The first version of Yices was released in 2006 and has been continuously updated since then. In 2007, we started a complete re-implementation of the solver to improve performance and increase modularity and flexibility. We describe the latest release of Yices, namely, Yices 2.2. We present the tool's architecture and discuss the algorithms it implements, and we describe recent developments such as support for the SMT-LIB 2.0 notation and various performance improvements.

1 Introduction

SRI International has a long history in developing formal verification tools. Shostak developed his decision procedures and combination method while at SRI in the 1980s [1]. Since then, SRI has continuously extended and supported decision procedures as part the PVS theorem prover [2, 3]. Methods for combining Boolean satisfiability solvers and decision procedures were also pioneered at SRI in the ICS solver [4]. In 2006, we released Yices 1, an efficient SMT solver that was the state of the art. Yices 1 introduced an innovative Simplex-based decision procedure designed to efficiently integrate with a SAT solver [5], included a congruence-closure algorithm inspired by Simplify's E-graph [6], and used an approach for theory combination based on the Nelson-Oppen method [7] complemented with lazy generation of interface equalities (an optimization of the method proposed by Bozzano et al. [8]). These main ingredients and others introduced by Yices 1 are now common in general-purpose SMT solvers such as Z3, CVC, MathSAT, VeriT, and SMTInterpol [9–13].

Although Yices 1 remains a decent SMT solver to this day, it has some limitations:

- Yices 1 relies on a complex type system that includes predicate subtypes as in PVS. This logic is very expressive but it has a major drawback: type-correctness is undecidable in general. This is very confusing for users and leads to many problems as the behavior of Yices 1 on specifications that are not type correct is chaotic.
- Yices 1 was designed to be used mostly via a textual interface (i.e., by reading specifications from files), but many applications require close interaction with an SMT solver via a programmable interface. Yices 1 can be used as a library but its

* This work was supported in part by DARPA under contract FA8750-12-C-0284 and by the NSF Grant SHF:CSR-1017483. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author and do not necessarily reflect the views of the funding agencies.

API is cumbersome and incomplete, which makes it difficult to integrate Yices 1 in other software.
- Yices 1 has poor performance on certain classes of problems, most notably problems that involve bitvectors.

To address these issues, we started a complete re-implementation of Yices in 2007. Prototypes of the resulting new Yices 2 solver entered the SMT Competition in 2008 and in 2009. We then released a full-featured version of Yices 2 in May 2012, with a few updates for bug fixes since then. This paper describes our latest SMT solver—Yices 2.2—the first solver in the Yices family to support the SMT-LIB 2.0 notation.

2 Logic

The Yices 2 logic is the Yices 1 logic without the most complex type constructs. Primitive types include the arithmetic types `int` and `real`, bitvectors, and Boolean. One can extend this set by declaring new *uninterpreted* and *scalar* types. An uninterpreted type denotes a nonempty collection of object with no cardinality constraint. A scalar type denotes a nonempty finite collection of objects. In addition to these atomic types, Yices 2 provides constructors for function and tuple types. Yices 2 uses a simple form of subtyping: `int` is a subtype of `real`, and the subtyping relation extends in a natural way to tuple and function types. Details are given in the Yices 2 manual [14].

Yices 2 supports the usual Boolean and arithmetic operators, and all the bitvector operators defined in the SMT-LIB 1.2 and SMT-LIB 2.0 specifications [15, 16]. It also includes operations on tuples, and an `update` operation that applies to any function type. If `f` is a function, then the term $(\texttt{update}\ f\ t_1 \ldots t_n\ v)$ is the function that maps $(t_1, \ldots, t_n)$ to v and is equal to f at all other points. This generalizes the SMT-LIB `store` operation to arbitrary function types.

In summary, the Yices 2 logic is broadly similar to the array, arithmetic, and bitvector logics defined in SMT-LIB 2.0, with extensions to support tuples and scalar types, and with a more general function-update operation. Yices 2's subtyping mechanism allows arithmetic terms of integer and real types to be mixed arbitrarily (whereas `Int` and `Real` are disjoint types in SMT-LIB 2.0).

3 System Architecture

Figure 1 shows the core architecture of the Yices 2 library. The software is decomposed into three main modules for manipulating terms and types, contexts, and models, which are the main data types available in the Yices API. Additional components include the front ends that process specifications in different input languages, but these components are not part of the library.

Internally, Yices 2 maintains a global database of terms and types. The API provides a large number of functions for constructing terms and types, for pretty printing, and so forth. Unlike Yices 1, Yices 2 provides a complete API: all term and type constructors defined in the Yices 2 language are present in the API. We have paid special attention

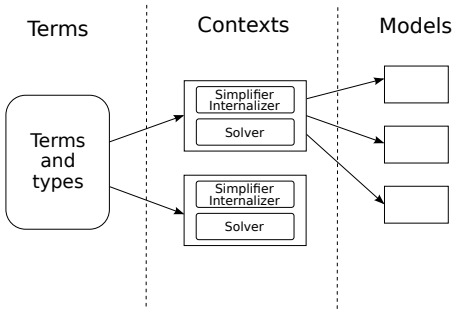

Fig. 1. Toplevel Architecture

to memory consumption by using compact data structures for terms and types, and by employing hash-consing to maximize sharing of subterms.

The second main module implements operations on *contexts*. A context is a central data structure that stores assertions to be checked for satisfiability. The API includes operations for creating and configuring contexts, adding and removing assertions, and for checking satisfiability of the asserted formulas. Internally, a context includes a solver and a module to simplify assertions and convert them into the internal form used by the solver. Contexts are highly flexible and can be configured to support a specific class of formulas, to apply different preprocessing and simplification procedures, and to use a specific solver or combination of solvers.

If the set of assertions in a context is satisfiable, then one can build a *model* of the formulas. Such a model maps uninterpreted symbols present in the assertions to concrete values such as rational or bitvector constants. A model is a separate object that can be queried and examined independently of the context from which it was built. Once a model is created from a context, it is not affected by further operations on this context. The model can remain in existence after the context is deleted.

A particular focus is to make Yices 2 easy to use as a library and enable flexible operations on multiple contexts and models while using a shared set of terms. Efficient operation on multiple contexts is crucial to applications related to software or control synthesis, such as exists/forall SMT solving [17–19].

4 Solvers

Yices includes a Boolean satisfiability solver and theory solvers for four main theories: uninterpreted functions with equalities, linear arithmetic, bitvectors, and arrays. These solvers can be combined as illustrated in Figure 2. It is also possible to select different solvers or combinations depending on the problem. For example, a specialized solver can be built by attaching the arithmetic solver directly to the SAT solver. The API provides functions to select the right solver combination when a context is created.

The SAT solver uses the CDCL approach. It is similar in performance and implementation to solvers such as Minisat 1.4 [20] or Picosat [21], with extensions to com-

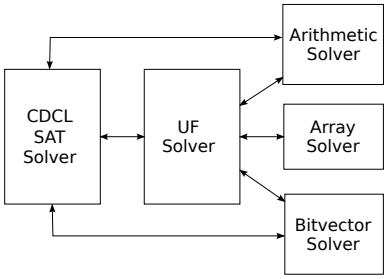

Fig. 2. Solver Architecture

municate with the theory solvers. For example, theory solvers can dynamically create literals and add clauses during the CDCL search, and can assign literals via theory propagation.

The solver for uninterpreted functions (UF) implements a congruence closure algorithm. The implementation is inspired by Simplify [6] with support for explanations [22], and a heuristic for *dynamic Ackermannization* [23, 24]. This UF solver supports Boolean terms. This enables the UF solver to store equalities as binary terms of the form $(\texttt{eq}\ t\ u)$ and efficiently perform propagation by congruence closure. A simple example is the following propagation

$$(\texttt{eq}\ t\ u) = \texttt{false} \ \wedge\ t = v\ \wedge\ u = w\ \Rightarrow\ (\texttt{eq}\ w\ v) = \texttt{false},$$

effectively deducing that $w \neq v$ follows from $t \neq u$ by congruence. This idea was introduced by the first Yices 2 prototype in 2008 and has since been adopted by other solvers.

The main arithmetic solver implements a decision procedure based on Simplex [5]. Yices also includes two specialized solvers for the *difference-logic fragments* of linear arithmetic. These two solvers rely on a variant of the Floyd-Warshall algorithm. One deals with integer difference logic and the other with real difference logic.

The bitvector solver is based on the "bit-blasting" approach. It applies various simplifications to bitvector constraints, then convert them to a pure Boolean SAT problem that is handled by the CDCL solver. In problems that combine uninterpreted functions and bitvectors or arrays and bitvectors, the bitvector solver dynamically adds constraints in the SAT solver as it receives equalities from the UF solver.

The array solver relies on instantiating the classic array axioms:

$$((\texttt{update}\ f\ i\ v)\ i) = v \tag{1}$$

$$((\texttt{update}\ f\ i\ v)\ j) = (f\ j)\ \text{ if } i \neq j \tag{2}$$

The solver eagerly generates instances of axiom (1) for every `update` term. On the other hand, it uses a lazy strategy for generating instances of axiom (2). After the UF and other theory solvers have built a consistent model (as explained below), the array

solver searches for instances of axiom (2) that are false in this model. It adds these instances to the clause database, which triggers search for a different model.

5 Recent Developments

Yices 2.1 was released in August 2012. Since then, we have implemented new features, most notably a front end for SMT-LIB 2.0. Yices 2.2 supports most of the SMT-LIB 2.0 specification, except proof generation and construction of unsat cores.

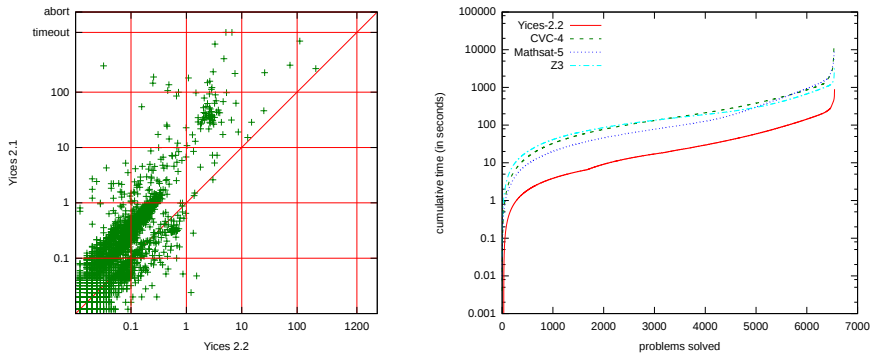

Fig. 3. Yices-2.2 on QF_UF Benchmarks

We have also added new preprocessing procedures, such as, the symmetry-breaking algorithm of Déharbe et al. [25]. Figure 3 shows the resulting performance improvement on the QF_UF benchmarks of SMT-LIB. The left part is a "scatter plot" comparing Yices-2.2 and Yices-2.1. Every point above the diagonal is a benchmark that Yices-2.2 solves faster than Yices-2.1. Yices-2.2 solves all benchmarks, whereas Yices-2.1 has two timeouts. The right part of the figure compares Yices-2.2 with other solvers[1] on the same benchmarks. All solvers in this graph use symmetry breaking, but Yices-2.2 is significantly faster. This data was collected on Linux machines (Ubuntu 12.04) with a timeout of 20 min. and a memory limit of 6 GB.

Another recent development is a new theory-combination method. In Yices, theory combination always involves UF on one side and another theory T (arithmetic or bitvector) on the other. Given two sets of formulas Γ_1 and Γ_2, such that Γ_1 is satisfiable in UF and Γ_2 is satisfiable in T, the goal is to ensure that $\Gamma_1 \cup \Gamma_2$ is satisfiable in the combined theory. For this purpose, Yices 2.2 uses a model-based approach [24]. Given a model M_1 of Γ_1 (computed by the UF solver) and a model M_2 of Γ_2, we must ensure that M_1 and M_2 agree on equalities between variables that occur in $\Gamma_1 \cap \Gamma_2$. By construction, the UF solver propagates all implied equalities to the bitvector and arithmetic solvers.

[1] All experiments mentioned in this paper used solvers that entered the main track of SMT-COMP 2012, plus Z3 version 4.2.

The only conflicts between M_1 and M_2 are then pairs of shared variables (x, y) such that

$$M_2 \models x = y \quad \text{but} \quad M_1 \models x \neq y.$$

To fix this conflict, Yices 2.2 attempts to modify M_1 locally by merging the congruence classes of x and y while keeping M_2 unchanged. This merging is applied if it does not conflict with existing disequalities in the UF solver, and if it does imply an equality $u = v$ where u and v are shared variables that have distinct values in M_2 (i.e., merging of x and y would cause more variables to become equal in theory T). If the conflict cannot be solved, Yices 2.2 generates an *interface lemma* that forces backtracking and search for different models. For example, in linear arithmetic, an interface lemma has the form

$$(\texttt{eq}\ x\ y) \vee (x < y) \vee (y < x),$$

which includes the UF atom $(\texttt{eq}\ x\ y)$ and two arithmetic atoms. However, local modification of M_1 is often successful and can often make M_1 and M_2 consistent without generating any lemmas. This algorithm is particularly effective on the SMT-LIB benchmarks that mix arrays and bitvectors. As shown in Figure 4, Yices 2.2 is competitive on such benchmarks with solvers specialized for bitvector problems. Yices 2.2 is generally fast, but Boolector 1.5 solves the most benchmarks (one more than Yices 2.2).

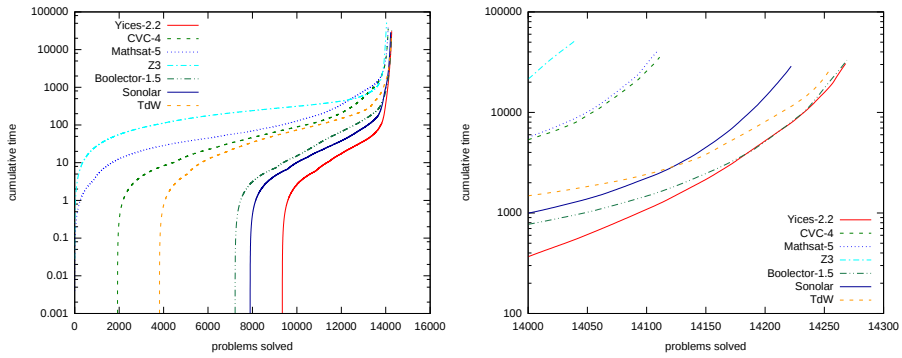

Fig. 4. Yices-2.2. vs. other Solvers on QF_AUFBV Problems

6 Conclusion

Yices 2 is a complete re-implementation of the Yices 1 solver. It is designed to be modular and extensible, to be efficient on a large class of problems, and to provide a rich API to enable advanced applications of SMT solving such as exists/forall SMT. Yices 2 now supports both versions of the SMT-LIB notation in addition to its own input language. Yices 2.2 is distributed at `http://yices.csl.sri.com`. Precompiled binaries are available for common operating systems such as Linux, Windows, Mac OS X and FreeBSD. Yices 2.2 is free for research and other non-commercial use.

References

1. Shostak, R.E.: Deciding Combinations of Theories. In: Loveland, D.W. (ed.) 6th Conference on Automated Deduction (CADE), LNCS, vol. 138, pp. 209–222. Springer, Heidelberg (1982)
2. Owre, S., Rushby, J.M., Shankar, N.: PVS: A Prototype Verification System. In: Kapur, D. (ed.), Automated Deduction—CADE–11, LNCS, vol. 607, pp. 748–752. Springer, Heidelberg (1992)
3. Owre, S., Rushby, J., Shankar, N., von Henke, F.: Formal Verification of Fault-Tolerant Architectures: Prolegomena to the Design of PVS. IEEE Transactions on Software Engineering **21**(2), 107–125 (February 1995)
4. de Moura, L., Owre, S., Rueß, H., Rushby, J., Shankar, N.: The ICS Decision Procedures for Embedded Deduction. In: Basin, D., Rusinowitch, M. (eds.) Automated Reasoning (IJCAR 2004), LNCS, vol. 3097, pp. 218–222. Springer, Heidelberg (2004)
5. Dutertre, B., de Moura, L.: A fast linear-arithmetic solver for DPLL(T). In: Ball, T., Jones, R.B. (eds.) Computer-Aided Verification (CAV'2006), LNCS, vol 4144, pp. 81–94. Springer, Heidelberg (2006)
6. Detlefs, D., Nelson, G., Saxe, J.B.: Simplify: A Theorem Prover for Program Checking. Journal of the ACM **52**(3), 365–473 (May 2005)
7. Nelson, G., Oppen, D.: Simplification by cooperating decision procedures. ACM Transactions on Programming Languages and Systems **1**(2), 245–257 (1979)
8. Bozzano, M., Bruttomesso, R., Cimatti, A., Junttila, T., Ranise, S., van Rossum, P., Sebastiani, R.: Efficient Satisfiability Modulo Theories via Delayed Theory Combination. In: Etessami, K., Rajamani, S.K. (eds.) Computer Aided Verification (CAV 2005), LNCS, vol. 3576, pp. 335–349. Springer, Heidelberg (2005)
9. de Moura, L., Bjørner, N.: Z3: An Efficient SMT Solver. In: Ramakrishnan, C., Rehof, J. (eds.) Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2008), LNCS, vol 4963, pp. 337–340. Springer, Heidelberg (2008)
10. Barrett, C., Conway, C.L., Deters, M., Hadarean, L., Jovanović, D., King, T., Reynolds, A., Tinelli, C.: CVC4. In: Gopalakrishnan, G., Qadeer, S. (eds.) Computer Aided Verification (CAV 2011), LNCS, vol 6806, pp. 171–177. Springer, Heidelberg (2011)
11. Cimatti, A., Griggio, A., Schaafsma, B.J., Sebastiani, R.: The MathSAT5 SMT Solver. In: Piterman, N., Smolka, S.A. (eds.) Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2013), LNCS, vol 7795, pp. 93–107. Springer, Heidelberg (2013)
12. Bouton, T., de Oliveira, D.C.B., Déharbe, D., Fontaine, P.: veriT: An Open, Trustable, and Efficient SMT Solver. In Schmidt, R.A. (ed.) Automated Deduction—CADE–22, LNCS, vol 5663, pp. 151–156. Springer, Heidelberg (2009)
13. Christ, J., Hoenicke, J., Nutz, A.: SMTInterpol: An Interpolating SMT Solver. In: Donaldson, A., Parker, D. (eds.) Model Checking Software (SPIN Workshop 2012), LNCS, vol 7385, pp. 248–254. Springer, Heidelberg (2012)
14. Dutertre, B.: Yices 2 Manual. Technical report, Computer Science Laboratory, SRI International (2014) Included in the Yices 2 distribution.
15. Ranise, S., Tinelli, C.: The SMT-LIB Standard: Version 1.2. Technical report, SMT-LIB Initiative (2006) Available at `http://www.smtlib.org`.
16. Barrett, C., Sump, A., Tinelli, C.: The SMT-LIB Standard: Version 2.0. Technical report, SMT-LIB Initiative (2012) Available at `http://www.smtlib.org`.
17. Jha, S., Gulwani, S., Seshia, S.A., Tiwari, A.: Oracle-Guided Component-Based Program Synthesis. In: Kramer, J., Bishop, J., Devanbu, P. T., Uchitel, S. (eds) Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering (ICSE), 215–224. ACM (2010)

18. Taly, A., Gulwani, S., Tiwari, A.: Synthesizing Switching Logic using Constraint Solving. In: Jones, N.D., Müller-Olm, M. (eds.) Verification, Model Checking, and Abstract Interpretation (VMCAI), LNCS, vol 5403, pp. 305–319. Springer, Heidelberg (2009)
19. Cheng, C.H., Shankar, N., Rueß, H., Bensalem, S.: EFSMT: A Logical Framework for Cyber-Physical Systems. arXiv:1306:3456b2 and http://www6.in.tum.de/~chengch/efsmt/ (June 2014)
20. Eén, N., Sörensson, N.: An Extensible SAT Solver. In: Giunchiglia, E., Tacchella, A. (eds.) Theory and Applications of Satisfiability Testing (SAT 2003), LNCS, vol 2919, pp. 502–518. Springer, Heidelberg (2003)
21. Biere, A.: PicoSAT Essentials. Journal on Satisfiability, Boolean Modelling, and Computation (JSAT) **4**, 75–97 (2008)
22. Nieuwenhuis, R., Oliveras, A.: Fast Congruence Closure and Extensions. Information and Computation **205**(4), 557–580 (2007)
23. Dutertre, B., de Moura, L.: The Yices SMT Solver. http://yices.csl.sri.com/tool-paper.pdf (2006)
24. de Moura, L., Bjørner, N.: Model-Based Theory Combination. Electronic Notes on Theoretical Computer Science **198**(2), 37–49 (2008)
25. Déharbe, D., Fontaine, P., Merz, S., Woltzenlogel Paleo, B.: Exploiting Symmetry in SMT Problems. In: Bjørner, N., Sofronie-Stokkermans, V. (eds.) Automated Deduction—CADE–23, LNCS, vol. 6803, pp. 222–236. Springer, Heidelberg (2011)