

An Update on Yices

Bruno Dutertre, SRI International

NASA Langley Formal Methods Workshop

May 2nd, 2008

Outline

Background

- Decision Procedures
- SMT Solvers

Yices 1

- Supported theories
- Applications
- Some Issues

Yices 2

- New features
- Architecture

Decision Procedures

Definition

- Algorithm to determine whether a formula ϕ (in a first-order theory T) is satisfiable.

Examples

- **Congruence closure**: for quantifier-free formulas, uninterpreted functions
- **Simplex methods** for quantifier-free linear arithmetic
- **Cylindrical algebraic decomposition** for real closed fields

More useful versions

- Decision procedures for **combinations of theories**:

$$2.\text{car}(x) - 3.\text{cdr}(x) = f(\text{cdr}(x)) \Rightarrow \\ g(\text{cons}(4.\text{car}(x) - 2.f(\text{cdr}(x)), y)) = g(\text{cons}(6.\text{cdr}(x)), y)$$

- **Example**: the decision procedures of PVS (based on Shostak)

Dealing with Boolean Structure

Many decision procedures (e.g., congruence closure, simplex) work on **conjunctions of literals**

They can still be applied to arbitrary formula ϕ . For example, write ϕ in DNF:

$$(a_{11} \wedge \dots \wedge a_{1n}) \vee \dots \vee (a_{m1} \wedge \dots \wedge a_{mp})$$

Problem: this is highly inefficient

- DNF can explode
- If several conjuncts share identical literals, we prove the same thing many time:

$$(f(x, y) \neq f(y, x) \wedge z = 3x + 1 \wedge x = y \wedge z < 0) \vee \\ (t > g(y) \wedge x = y \wedge z + 3 \leq 0 \wedge f(x, y) \neq f(y, x)) \vee \dots$$

Better approach: use a Boolean SAT solver to enumerate the conjuncts

- This is done by tools called **Satisfiability Modulo Theory (SMT) Solvers**

Naïve SMT Solving

$$x + y \geq 0 \wedge (x = z \Rightarrow z + y = -1) \wedge z > 3t$$

1) Replace atoms by boolean variables

$$\begin{array}{ll} a \mapsto x + y \geq 0 & b \mapsto x = z \\ c \mapsto z + y = -1 & d \mapsto z > 3t \end{array}$$

2) Ask for a model of $a \wedge (b \Rightarrow c) \wedge d$ using a SAT solver

- Boolean model: $\{a, b, c, d\}$
- Convert the model back to arithmetic

$$x + y \geq 0 \wedge x = z \wedge z + y = -1 \wedge z > 3t$$

and check its consistency

Answer: not consistent

Explanation: $Arithmetic \models \neg(x + y \geq 0 \wedge x = z \wedge z + y = -1)$

Naïve SMT Solving (continued)

3) Feed the explanation to the SAT solver:

- add the clause $(\neg a \vee \neg b \vee \neg c)$

4) Get a model of $(a \wedge (b \Rightarrow c) \wedge d) \wedge (\neg a \vee \neg b \vee \neg c)$

- Boolean model: $\{a, \neg b, c, d\}$
- Convert back to arithmetic:

$$x + y \geq 0 \wedge \neg(x = z) \wedge z + y = -1 \wedge z > 3t$$

- Check consistency: **satisfiable**

Conclusion: The original formula is satisfiable

Improvements to Naïve SMT Solving

Make it incremental

- Don't wait for a full boolean model to check consistency: interleave boolean propagation and calls to the theory solver

Theory propagation

- **Example:** given partial model $\{a, d, c\}$ (i.e., $x + y \geq 0, z + y = -1, z > 3t$) linear arithmetic solver can deduce that **b must be false**
(since $Arithmetic \models x + y \geq 0 \wedge z + y = -1 \Rightarrow \neg(x = z)$)
- **Theory propagation:** detect this and assign $\neg b$ in the SAT solver.

Benefit of these improvements: prune the SAT solver search space

Yices

Yices is SRI's current SMT solver

- Successor of previous systems and prototypes (ICS, Yices 0.1, Simplics)
- Follows a long tradition of SRI's work on decision procedures (Shostak, PVS decision procedures)

A state-of-the-art SMT solver

- Yices won several categories in 2005, 2006, 2007 competition on SMT solving
- Uses recent advances in Boolean SAT solving (cf. Chaff, MiniSat, PicoSat)
- Can solve very large formulas (100 to 10,000 atoms/variables/terms)
- Supports all theories in SMT-LIB and more

Main Features

Supported Theories

- Uninterpreted functions
- Linear real and integer arithmetic
- Extensional arrays
- Fixed-size bit-vectors
- Scalar types
- Recursive datatypes, tuples, records
- Quantifiers and lambda expressions

Other Features

- Model generation, unsatisfiable cores
- Supports incremental assertions: push, pop, retract
- Dependent types (similar to PVS)

Applications of Yices

Backend Solver for the SAL Toolset

- Support bounded model checking of finite and infinite state systems:
 - Check satisfiability of formulas of the form

$$I(X_0) \wedge T(X_0, X_1) \wedge \dots \wedge T(X_{n-1}, X_n) \Rightarrow P(X_n)$$

where (X, I, T) encodes a state-transition system and P is a state property:
 X : state variables, I : initialization, T : transition relation

- Related applications: k -induction, test-case generation, planning

Integration to PVS

- There is a translation from PVS to Yices (in PVS4.0)
- Allows to use Yices as an endgame prover from PVS

Other Applications

Static Analysis

- Extended static checking (e.g., with the **Why** system)
- Symbolic simulation
- Support for invariant generation in hybrid systems (Gulwani and Tiwari, 2008)

Some Limitations of Yices 1

Type System

- Dependent subtypes cause problems:
 - Type correctness of a formula cannot be established cheaply (if at all)
 - Yices behavior on type incorrect formulas is chaotic

API Issues

- Yices 1 is mostly intended to be used via the `yices` executable
- Many user want to embed Yices in other system: use it as a library
- A Yices library exists but the API is not complete and fragile

Performance Issues

- Yices is among the best SMT solvers for arithmetic, arrays, uninterpreted functions
- Not as good for bitvectors and quantifiers

The Next Yices

Yices 2: complete redesign and new implementation

Goals:

- Increase flexibility and usability as a library
- Simplify the type system to ensure easy type checking
- Maintain or improve performance

Yices 2 Formulas

Type Systems

- Primitive types: `Int`, `Real`, `Bool`, `(Bitvector k)`
- Uninterpreted and scalar types:
- Tuple and function types: $(\tau_1 \times \dots \times \tau_n)$ and $(\tau_1 \times \dots \times \tau_n \rightarrow \tau_0)$

Subtype Relations

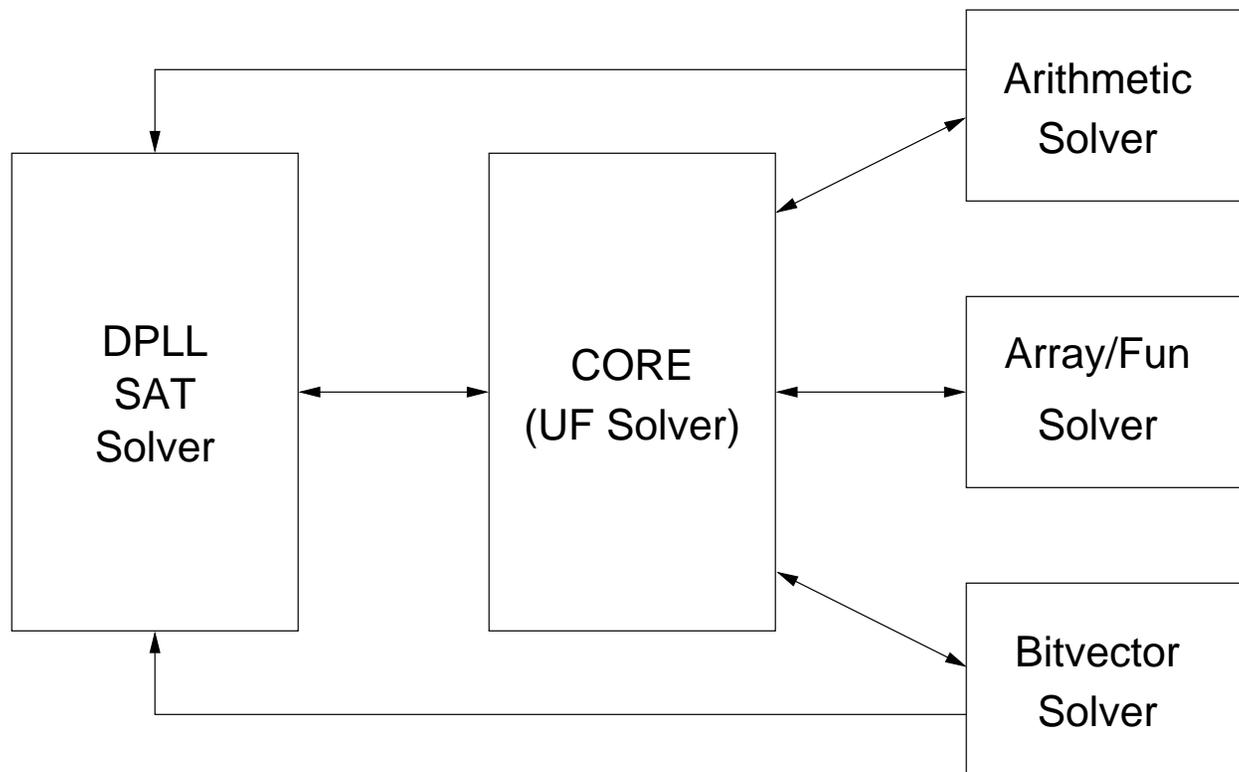
- $\text{Int} \sqsubseteq \text{Real}$
- If $\tau_1 \sqsubseteq \sigma_1, \dots, \tau_n \sqsubseteq \sigma_n$ then $(\tau_1 \times \dots \times \tau_n) \sqsubseteq (\sigma_1 \times \dots \times \sigma_n)$
- If $\tau_0 \sqsubseteq \sigma_0$ then $(\tau_1 \times \dots \times \tau_n \rightarrow \tau_0) \sqsubseteq (\tau_1 \times \dots \times \tau_n \rightarrow \sigma_0)$

Terms

- Boolean, rational, and bitvector constants, uninterpreted constants
- Non-primitive terms: $(t_1 = t_2)$ $(\text{ite } c \ t_1 \ t_2)$ $(\text{not } t)$ $(\text{or } t_1 \dots t_n)$
 $(f \ t_1 \dots t_n)$ $(\forall(x_1 : \tau_1, \dots, x_n : \tau_n)t) \dots$

Type checking is straightforward

Yices 2 Architecture



Main Components

SAT Solver

- Similar to state-of-the-art sat solvers (MiniSat, Picosat)
- Extensions for interaction with theory solvers:
 - support mapping of boolean variables to atoms
 - addition of clauses and boolean variables on the fly
 - support for theory propagation and theory conflict
 - many configurable parameters controllable via API

Core

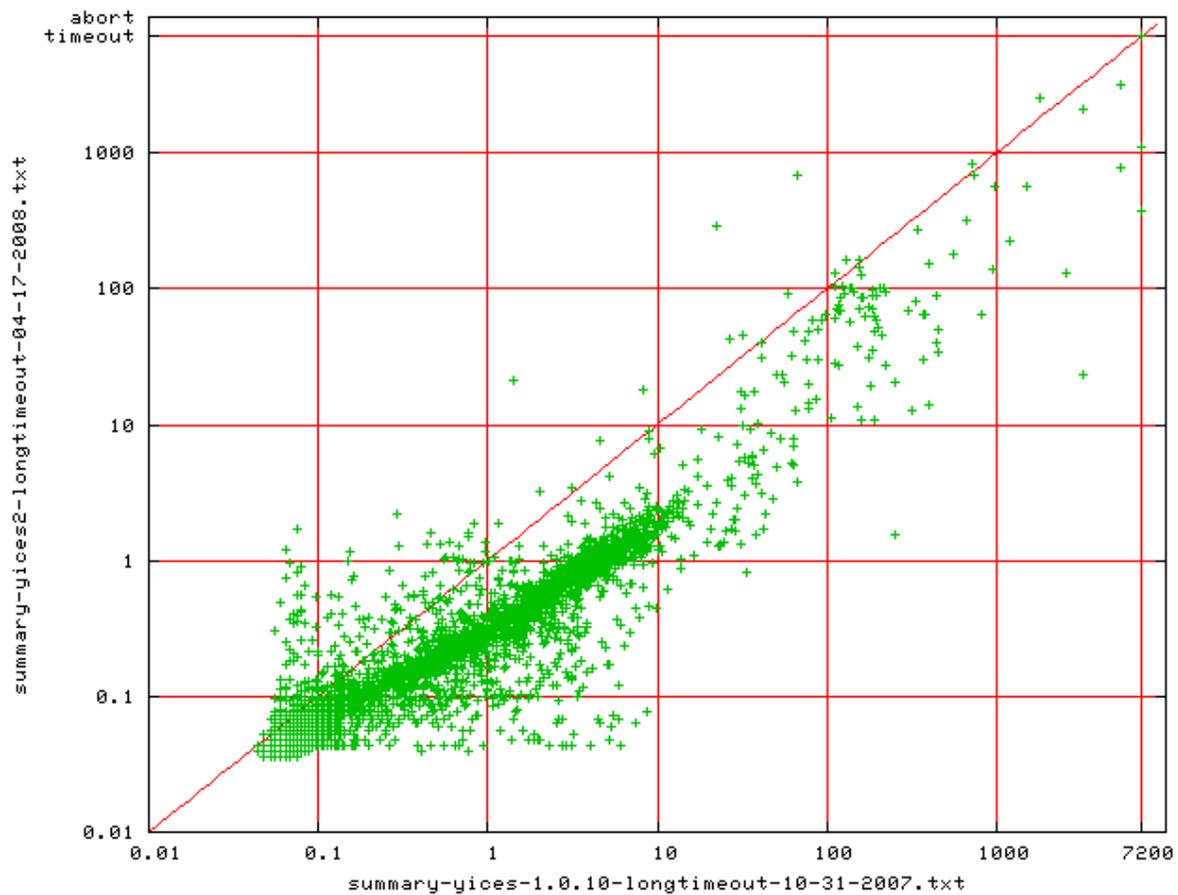
- Congruence-closure solver for uninterpreted functions and tuples
- Ensures consistency between solvers (Nelson-Oppen combination using interface equalities)
- Improvements over Yices 1 core
 - More efficient algorithm for equality propagation
 - Built-in support for if-then-else and boolean terms

Theory Solvers

Satellite Solvers

- Communicate with SAT solver and Core
- Each solver deals with a specific theory:
 - Arithmetic solver
 - The default solver uses the Simplex method
 - Optionally, it can be replaced by other solvers (based on the Floyd-Warshall algorithm) for difference logic
 - Bitvector solver
 - Bitvector arithmetic: bit-blasting + equality reasoning
 - Array/function theory solver
 - Extensionality
 - Array updates

Improved Performance (UF Benchmarks)



Improved API

Complete API

- All functionalities supported by the `yice2` solver are also available via a C-API
- Supports multiple independent contexts (in addition to push/pop)
- Increased flexibility and customization:
 - Heuristics parameters can be modified
 - The solver architecture is configurable (e.g., which arithmetic solver to use, whether the Core is needed or not, etc.)
 - Better support for model construction

Future Work

Short Term Plans

- Yices 2 still under development
- Our plan is to enter Yices 2 in this year's SMT Competition
- Release Yices 2 later this year

Future Extensions

- Support non-linear arithmetic
- Revisit the quantifier matching algorithms and heuristics

Conclusion

SMT solvers enable **new approaches to verification**:

- Bounded model checking requires powerful solvers capable of handling large, propositionally complex formulas

Yices 1 is one of the most efficient SMT solvers available

With Yices 2 we hope to continue improving:

- **Easier to use**: simpler language, correct typechecking, increased flexibility
- **Embeddable in other software**: improved API and library
- **Better performance**

Visit our website: <http://yices.csl.sri.com/>