# Challenging Problems for Yices

Bruno Dutertre, SRI International

Deduction at Scale Seminar

March, 2011

# SMT Solvers at SRI

2000-2004: Integrated Canonizer and Solver (ICS)

- Based on Shostak's method + a non-clausal SAT solver

2005: Two solvers in the SMT competition

- Simplics: linear arithmetic (Simplex based)
- Yices 0.1: linear arithmetic, arrays, uninterpreted functions

2006: Yices 1 released

- supported all SMT logics at that time: arithmetic, bitvectors, quantifiers
- main developer: Leonardo de Moura

Since 2006: Yices 1 maintained and developed

2008 and 2009: prototypes of a new solver (Yices 2) entered SMT-COMP

# Yices 1

## Yices 1 is SRI's current SMT solver

- Successor of previous systems and prototypes (ICS, Yices 0.1, Simplics)
- Current release: Yices 1.0.29
- Available for many platforms and OSs (Linux, Windows, MacOS X, Solaris)

## A state-of-the-art SMT solver

- Yices won several categories in 2005, 2006, 2007 competition on SMT solving
- Rely on modern Boolean SAT solving (cf. Chaff, MiniSat, PicoSat)
- Many users and applications

# Main Features of Yices 1

## 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
- Max SMT (weighted assertions)

# Some Limitations of Yices 1

## Input language and type system are too complex

- Type correctness of a formula cannot be established cheaply (if at all)
- Some language features not well supported (e.g., recursive functions)

## API Issues

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

## Performance Issues

- Yices is still a good solver for arithmetic, arrays, uninterpreted functions
- Not as good for bitvectors and quantifiers

## Portability/Maintainability

- Yices 1 is written in C++ (which changes too fast, we're already running into issues with deprecated C++ features)

# Yices 2: The New Yices

Started in 2008

- Complete redesign and new implementation
- Written entirely in C
- UF + arithmetic done in 2008, arrays + bitvectors added in 2009
- Developments since 2009:
  - model construction + queries
  - support for incremental use (push/pop)
  - better simplification/preprocessing
  - non-linear arithmetic (under development)

Goals:

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

# Yices 2 Language

## Types

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

## Subtype Relation

- `Int` $\sqsubset$ `Real`
- If $\tau_1 \sqsubset \sigma_1, \ldots, \tau_n \sqsubset \sigma_n$ then $(\tau_1 \times \ldots \times \tau_n) \sqsubset (\sigma_1 \times \ldots \times \sigma_n)$
- If $\tau_0 \sqsubset \sigma_0$ then $(\tau_1 \times \ldots \times \tau_n \to \tau_0) \sqsubset (\tau_1 \times \ldots \times \tau_n \to \sigma_0)$
- Two types $\tau$ and $\sigma$ are compatible if they have a common supertype

## Terms

- Boolean, rational, and bitvector constants
- Distinct constants $k_0, k_1, \ldots$ for an uninterpreted type $T$ (also for scalar types)
- Variables + usual term constructors

# Term Constructors + Type Checking

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

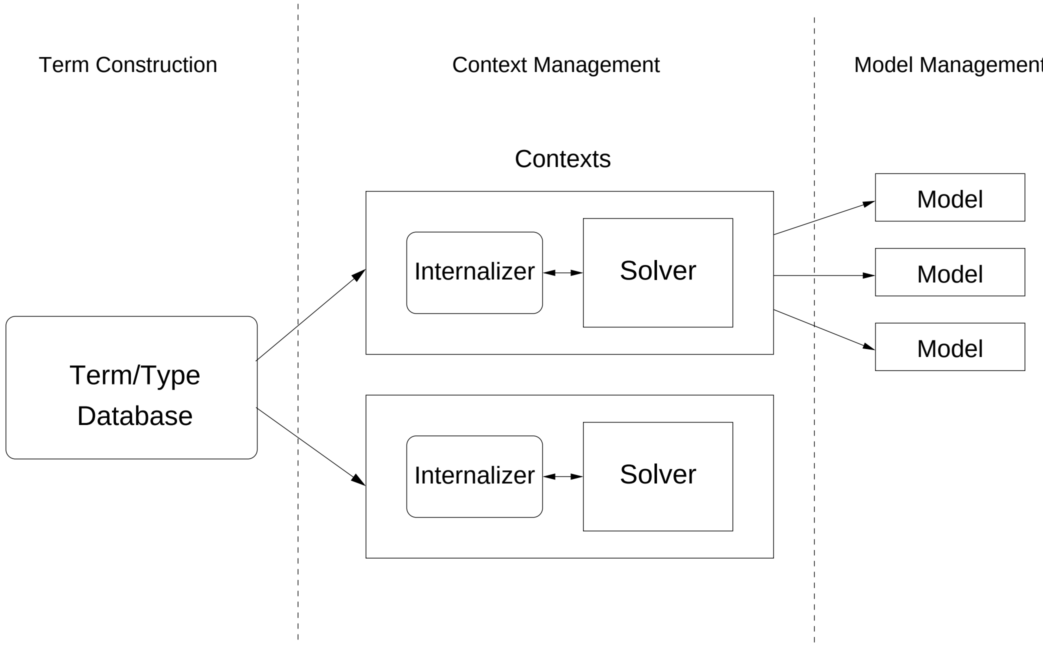
$$\frac{c :: \texttt{Bool} \quad t_1 :: \tau_1 \quad t_2 :: \tau_2}{(\texttt{ite}\ c\ t_1\ t_2) :: \tau_1 \sqcup \tau_2}$$ provided $\tau_1$ and $\tau_2$ are compatible

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

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

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

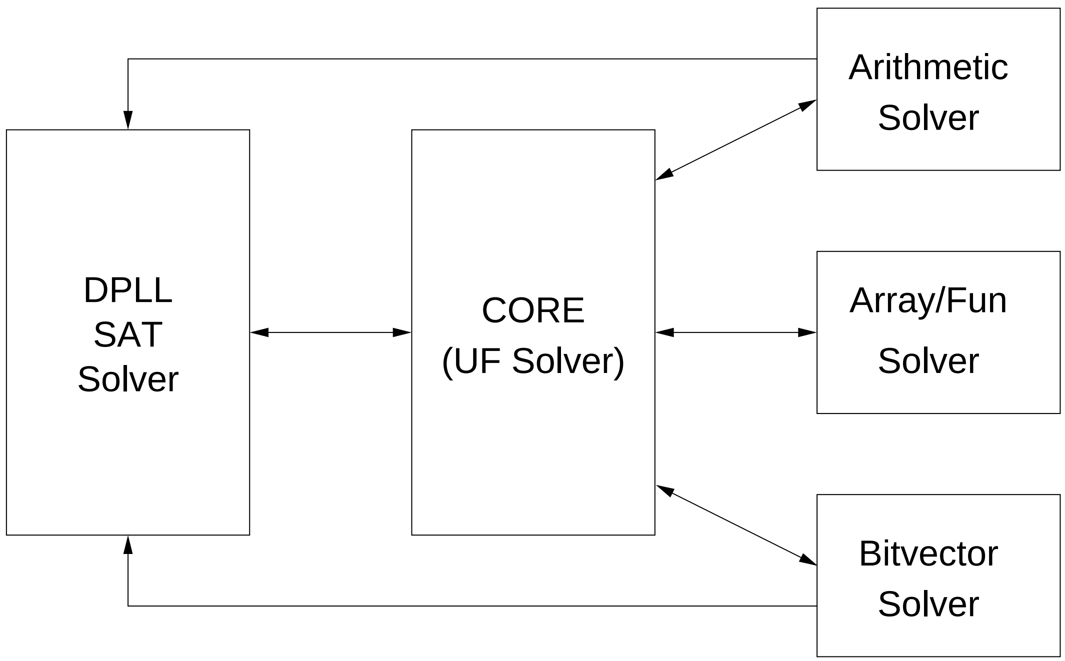
# Yices 2 Architecture




Three Main Modules: Type/Term database, Contexts, Models

- ○ Several contexts can coexist
- ○ Models are constructed from contexts but can be queried independently

# Solver Interaction




The actual solver combination used by a context can be configured via the API

# Current Solvers

SAT Solver

- Similar to MiniSat/Picosat, with extensions for interaction with theory solvers

Core/UF Solver

- Congruence-closure solver for uninterpreted functions and tuples
- Improvement over Yices 1: better equality propagation and support for theory combination (Nelson-Oppen, lazy generation of interface equalities)

Arithmetic Solvers

- Default: simplex
- Floyd-Warshall solvers for difference logic

Bitvector Solver: simplifier + bit blasting

Array Solver: lazy instantiation of array axioms

# Preprocessing and Simplification

Preprocessing and formula simplification are not glamorous but they are critical to SMT solving:

- Many SMT-LIB benchmarks are accidently hard: they become easy (sometimes trivial) with the right simplification trick
  - Examples: `eq_diamond`, `nec-smt` problems, `rings` problems, `unconstrained` family
- This is not just in the SMT-LIB benchmarks:
  - Bitvector problems are typically solved via bit-blasting (i.e., converted to Boolean SAT). But without simplification, bit-blasting can turn easy problems into exponential search
  - There are other problems that just can't be solved without the right simplifications

# Bitvector Example 1 (from a Yices user)

```
(define v1::(bitvector 32))
(define v2::(bitvector 32))
(define v3::(bitvector 32))

(assert (not(= v1 0x00000000)))
(assert (= v3 (bv-urem v2 v1)))
(assert (not (bv-lt v3 v1)))

(check)
```

# Bitvector Example 2 (from a Yices user)

```
(define-type bv-type-32 (bitvector 32))
(define EIP_0_1_0::bv-type-32)
(define temp-var-0::bv-type-32 (mk-bv 32 7))
(define temp-var-22::bv-type-32 (mk-bv 32 0))
(define temp-var-1::bool (= EIP_0_1_0 temp-var-0))
(define ESP_0_1_0::bv-type-32)
(define ESP_0_0_0::bv-type-32)
(define temp-var-2::bv-type-32 (mk-bv 32 4294967292))
(define temp-var-3::bv-type-32 (bv-add ESP_0_0_0 temp-var-2))
(define temp-var-4::bool (= ESP_0_1_0 temp-var-3))
(define temp-var-5::bool (and temp-var-1 temp-var-4))
(define temp-var-54::bv-type-32 (bv-mul ESP_0_1_0 (mk-bv 32 473028019)))
(define temp-var-55::bv-type-32 (bv-mul temp-var-0 (mk-bv 32 956831788)))
(define temp-var-56::bv-type-32 (bv-sub temp-var-54 temp-var-55))
(define temp-var-57::bv-type-32 (bv-mul ESP_0_0_0 (mk-bv 32 473028019)))
(define temp-var-58::bv-type-32 (bv-sub temp-var-56 temp-var-57))
(define temp-var-59::bool (= temp-var-22 temp-var-58))
(define temp-var-65::bool (not temp-var-59))
(define temp-var-66::bool (and temp-var-5 temp-var-65))
(assert temp-var-66)
(check)
```

# Example 3: Nested if-then-elses

How do we deal with non-boolean if-then-else?

- Lifting:
  - Rewrite `(>= (ite c t1 t2) u)` to `(ite c (>= t1 u) (>= t2 u))`
  - Risk exponential blow up if `t1` and `t2` are themselves if-then-else

- Use an auxiliary variable
  - Rewrite `(>= (ite c t1 t2) u)` to `(>= z u)` and add two constraints

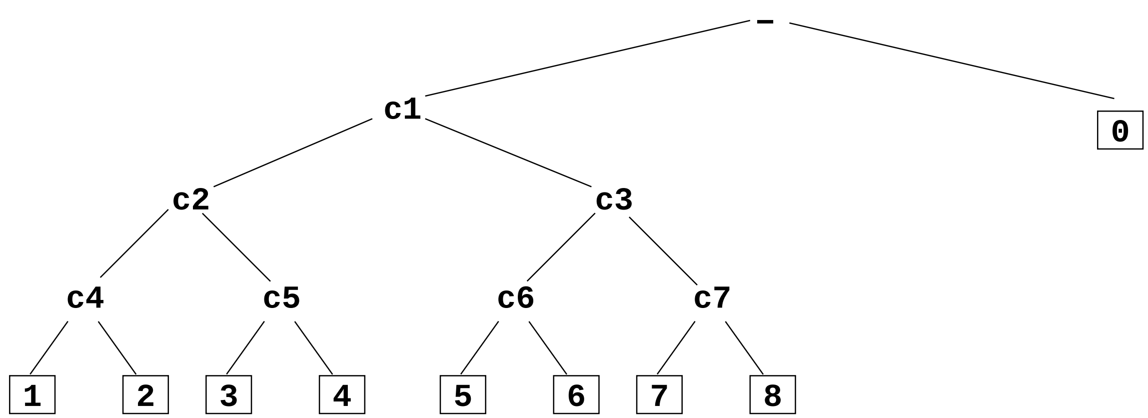
    ```
    (implies c (= z t1))
    (implies (not c) (= z t2))
    ```

  - Benefit: this does not blow up

# Nested if-then-else (cont'd)

## But lifting may still work better

- Example: `(= t1 a)` when `t1` is a nested if-then-else with all leaves trivially distinct from `a`.




- This type of constraints occurs a lot in the `nec-smt` benchmarks.
- That's why lift-if pays off on these benchmarks (cf., Kim et al, 2009)

# Two Sources of Hard Problems for Yices

There are real users with real hard problems (no known simplification trick for them!)

- Computational Biology: Flux Balance Analysis and related problems
- Scheduling Probems: Communication Schedules for Timed-Triggered Ethernet (Steiner, RTSS 2010).

Note: these users see Yices as a constraint solver (as opposed to a theorem proving tool). They care about finding models more than finding proofs.

# Flux Balance Analysis

Technique for modeling and analysis of metabolic pathways based on stoichiometry

- For an individual reaction:

$$\text{D-ribose} + \text{ATP} \longrightarrow \text{D-ribose-5-phosphate} + \text{ADP} + 2\text{H}^+$$

Let $\rho$ denote the reaction rate, then the molecule quantities vary according to

$$\begin{aligned} \frac{d[\text{D-ribose}]}{dt} &= \frac{d[\text{ATP}]}{dt} = -\rho \\ \frac{d[\text{D-ribose-5-phosphate}]}{dt} &= \frac{d[\text{ADP}]}{dt} = \rho \\ \frac{d[\text{H}^+]}{dt} &= 2\rho \end{aligned}$$

# Flux Balance Analysis (cont'd)

If a molecule (say $\mathrm{H}^+$) is involved in $n$ reactions, then we get

$$\frac{d[\mathrm{H}^+]}{dt} = a_1\rho_1 + \ldots + a_n\rho_n$$

where $\rho_i$s are reaction rates and $a_i$ are integer constants ($a_i$ is positive if reaction $i$ produces $\mathrm{H}^+$ and negative if reaction $i$ consumes $\mathrm{H}^+$).

Doing this for a full set of molecules, we get a stoichiometry matrix $S$ and an equation

$$\frac{d[C]}{dt} = SR$$

where $R$ is a vector of reaction rates and $C$ is a vector of molecule quantities

# Flux Balance Analysis (cont'd)

Flux balance analysis: looks for possible reaction rates when the system is at an equilibrium (more or less)

- At equilibrium $\frac{d[C]}{dt} = 0$
- So we search for solutions to the linear system: $SR = 0$

Which solutions?

- The system is underdetermined (many more reactions than chemical components)
- There's always a trivial solution: $R = 0$, but it's not interesting
- So more constraints are added to get solution that are "biologically interesting"
  - add bounds on rates
  - search for solutions that maximize some objective functions (i.e., biomass)

Beyond Flux-Balance Analysis

- add/search for missing reactions (i.e., errors in the pathway models): can be formulated as a MILP optimization problem with 0-1 variables.

# Solving FBA and Related Problems

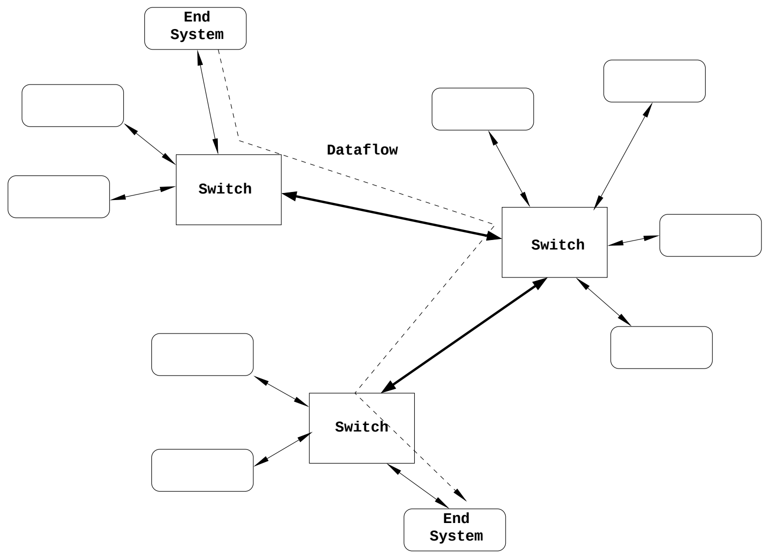
## Off-the-shelf LP and MILP solvers

- Typical problem size is about 10,000s reaction, 1,000s components
- CPLEX, SCIP solve them without much problems

## Using Yices?

- Motivation for trying Yices: it does exact arithmetic, off-the-shelf solvers have licensing restrictions
- But: results so far are disapointing.
  - Yices can't solve many of the MILP problems that are easy for SCIP.
  - Poor convergence of the pivoting heuristics used by Yices
  - Encoding using 0-1 variables is suboptimal for Yices

# Timed-Triggered Ethernet (TTE)




- Extension of standard Ethernet for real-time, distributed systems
- Guarantees for real-time messages: low jitter, predictable latency, no collisions
- All nodes are synchronized (fault-tolerant clock synchronization protocol)
- All communication and computation follow a system-wide, cyclic schedule

# Computing a Communication Schedule

## Input

- a set of virtual links: dataflows from one end system to one or more end systems
- the communication period

## Constraints

- no contention: all frames on every link are in a different time slot
- application constraints: one frame must be received at most $\Delta$ms after another
- path constraints: relayed frames must be scheduled after they are received
- other constraints: limits on switch memory, etc.

# TTE Scheduling as an SMT Problem

Large Difference Logic Problem (over the integers)

- Typical size: 10000-20000 variables, $10^6$ to $10^7$ constraints
- This depends on the network topology and number of virtual links

Solving this with Yices

- Yices 1 can solve moderate size instances (about 120 virtual links) out of the box
- In Wilfried Steiner's RTSS 2010 paper: incremental approach using push/pop can solve much larger instances (up to 1000 virtual links)
- Still, this may not be not quite enough for all TTE systems.

# Conclusion

SMT solvers are not just for proofs/verification

Many users see them as constraint-solving tools

- Their problem is to find models for a formula $\Phi$ (often in the less expressive SMT logics such as IDL or LIA)
- They want models and speed (don't care about proofs)

Many scalability problems to be addressed

- We're way behind state-of-the-art MILP solvers on many problems
- Naïve Simplex implementations are not good enough
- How to efficiently deal with integer arithmetic is not well understood in SMT
- We need to address optimization problems, not just feasibility