Yices and Applications

Bruno Dutertre, SRI International

SAT/SMT Solver Summer School
MIT, June 2011

Outline

Overview of Yices

Two Examples

- Scheduling for Timed-Triggered Ethernet (Steiner, 2010)
- Verification of timed systems (Brown & Pike, 2006)

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

Current Releases

Yices 1 is SRI's current SMT solver

- Current release: Yices 1.0.29
- Available for many platforms and OSs (Linux, Windows, MacOS X, Solaris)
- Supports SMT-LIB 1.2 or Yices language + usable via an API

Yices 2 Prototype

- This is a prerelease of Yices 2 that entered SMT COMP in 2009
- Input: SMT-LIB 1.2 input only (no API yet)

Both are available at `http://yices.csl.sri.com/`

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)

Yices 2: The New Yices

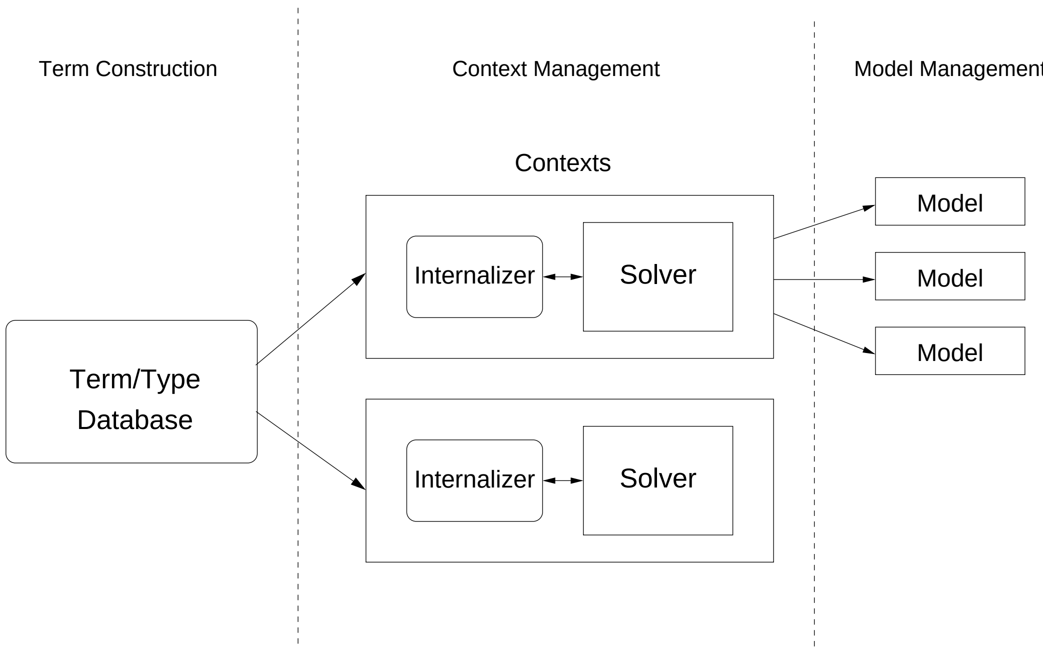
Started in 2008

- 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

Goals:

- Increase flexibility and usability as a library
- Simplify the type system to ensure easy type checking
- Improve performance over Yices 1

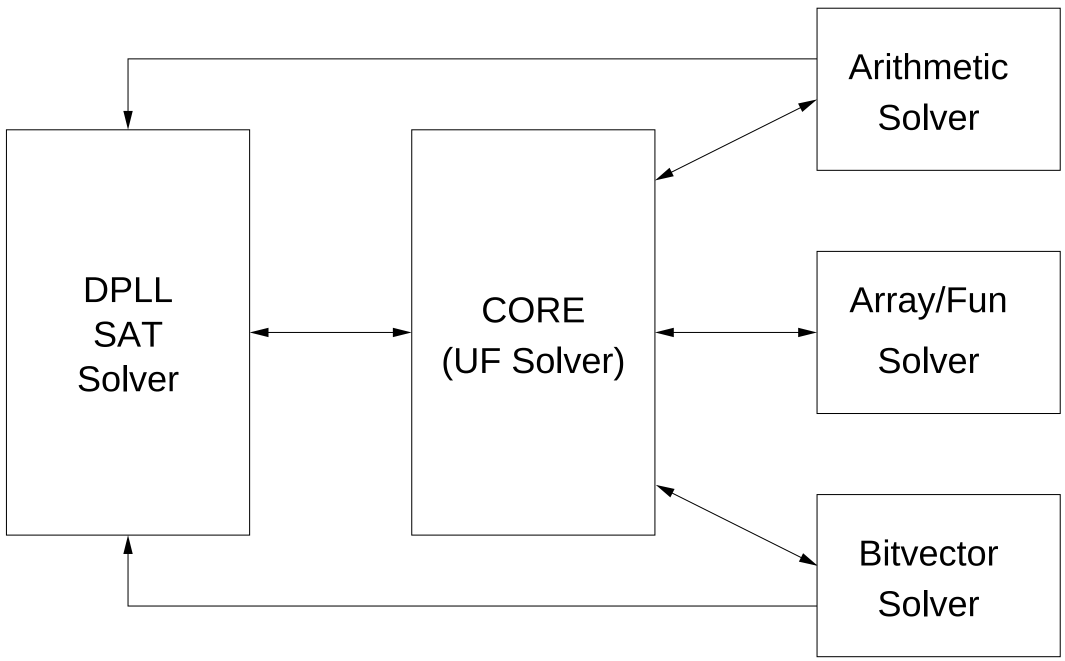
Yices 2 Architecture



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 in Yices 2

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

Example Uses of Yices

Model Checking

- Backend solver to the SAL model checkers (SRI)
- MCMT (Ghilardi & Ranise)
- Model checking of Lustre Programs (Hagen & Tinelli)

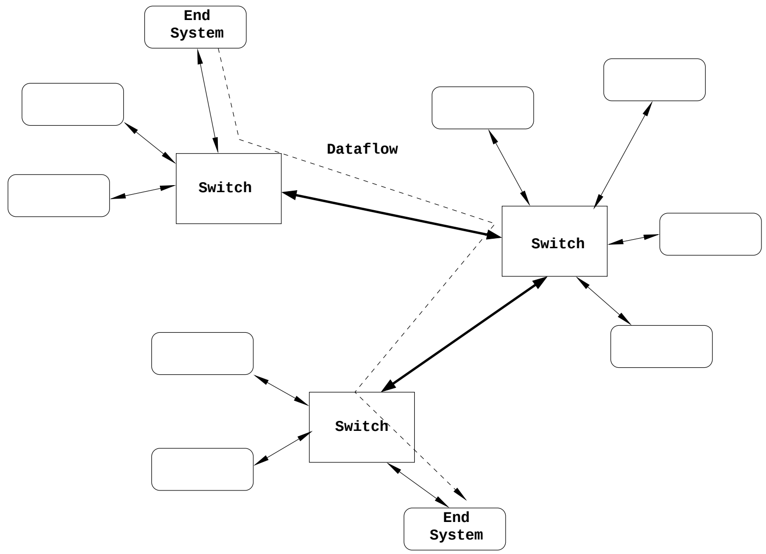
Program Analysis

- Symbolic Execution: Sireum/Kiasan (Deng, Robby, Hatcliff), JPF (Anand, Păsăreanu, Visser)
- Backend prover for SPARK-ADA (Jackson, Ellis, Sharp)

Within Interactive Theorem Provers

- PVS, Isabelle/HOL can use Yices as an *end-game* solver

Application 1: Scheduling for TTEthernet



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
- path constraints: relayed frames must be scheduled after they are received
- other constraints: limits on switch memory, application constraints, etc.

TTE Scheduling as an SMT Problem (Steiner, 2010)

Frames

- Messages are called frames in TTE.
- A frame f is characterized by its period $f.period$ and its length $f.length$.
- Routing is static: we know a priori the source of f, all receivers, and the set of communication links that will transport f.
- Given a link i, our goal is to compute when to send f over that link. The start of this transmission is denoted by $offset_{f,i}$

Simplification: in the simplest case, all frames have the same period (equal to the schedule cycle).

Example Scheduling Constraints

No Collisions: if distinct frames f and g use link i:

$$\mathit{offset}_{f,i} + \mathit{f.period} \leqslant \mathit{offset}_{g,i} \text{ or } \mathit{offset}_{g,i} + \mathit{g.period} \leqslant \mathit{offset}_{f,i}$$

Path Constraints: if a switch receives f on link i and relays it on link j

$$\mathit{offset}_{f,j} - \mathit{offset}_{f,i} \geqslant \mathit{maxhopdelay}$$

End-to-End Latency: along a path $i_0, i_1, \ldots, i_n$

$$\mathit{offset}_{f,i_n} - \mathit{offset}_{f,i_0} \leqslant \mathit{maxlatency}$$

Resulting 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)

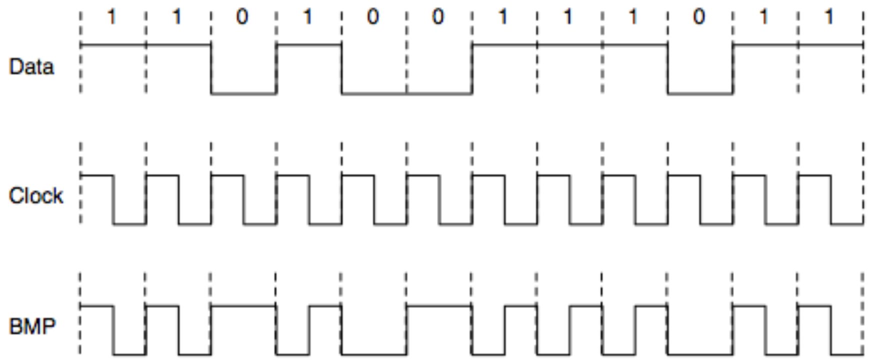
Application 2: Verification of Timed Systems

Yices used as backend to SAL

- SAL is a toolkit for modeling and verification of state-transition systems
- Specification language: guarded commands + extensions
- SAL supports both synchronous and asychronous composition
- Tools
  - BDD-based model checker: `sal-smc`
  - SAT-based bounded model checker: `sal-bmc` (for finite systems)
  - SMT-based bounded model checker: `sal-inf-bmc` (for infinite systems)
  - Test-case generation: `sal-atg`

Many timed systems can be modeled in SAL and verified using `sal-inf-bmc` and Yices

Example: Biphase Mark Protocol (BMP)



Biphase Mark: Physical layer protocol for data transmission (over serial links)

- transmitter and receiver have independent clocks
- encoding merges transmitter clock + data into a single bit stream
- decoding goal: recover the data from the bit stream
- Issues: must take into account jitter and sampling uncertainties

BMP: SAL Model

Output from the transmitter

```
WIRE:   TYPE = { Zero, One, ToZero, ToOne };
...
OUTPUT tdata : WIRE
...
     phase = Stable AND tstate = 1 -->
                  tdata' = ttoggle;
                  tstate' = 0;
  [] phase = Stable AND tstate = 0 -->
                  tdata' = IF (tbit = 1) THEN ttoggle ELSE tdata ENDIF;
                  tstate' = 1;
  [] phase = Settle -->
                  tdata' = IF tdata = ToOne THEN One
                           ELSIF tdata = ToZero THEN Zero
                           ELSE tdata
                           ENDIF;
```

Sampling

```
sample(w : WIRE) : [WIRE -> BOOLEAN] =
     IF (w = ToZero OR w = ToOne) THEN {Zero, One}
     ELSE {w}
     ENDIF;
```

SAL Model: Time and Clocks

Use a global real-valued `time` variable

Transmitter and receiver use *timeout* variables to schedule future discrete transitions:

```
   INPUT  time   : TIME
   OUTPUT tclk   : TIME
INITIALIZATION
   ...
   tclk  IN {x : TIME | 0 <= x AND x <= TSTABLE};
TRANSITION
   [ time = tclk AND phase = Stable -->
                 tclk' = time + TSETTLE;
                 phase' = Settle;
  [] time = tclk AND phase = Settle -->
                 tclk' = time + TSTABLE;
                 phase' = Stable;
```

SAL Model: Properties

Correct Reception Theorem

```
system : MODULE = clock [] rx [] tx;

BMP_Thm : THEOREM
  system |- G( rstate = 1 AND time = rclk =>
               (time /= tclk) AND (tstate = 1) AND X(rbit = tbit));
```

Conversion to SMT

State-transition systems

$$\mathcal{M} = \langle X, I(X), T(X, X') \rangle$$

- X set of state variables
- formula $I(X)$ defines the initial states
- formula $T(X, X')$ defines the transition relation

Traces

- Sequences of states $x_0 \to x_1 \to x_2 \ldots$ such that
  - x_0 satisfies $I(X)$
  - for every $t \in \mathbb{N}$, (x_t, x_{t+1}) satisfies $T(X, X')$

Bounded Model Checking

Goal

- Find counterexamples to a property
- Usually the property is an invariant $\Box P$
- The goal is then to find a reachable state that does not satisfy P.

Technique

- Fix a bound k
- Search for a state reachable in k steps that falsifies P
- This is the same as checking the satisfiability of the formula

$$I(x_0) \wedge T(x_0, x_1) \wedge T(x_1, x_2) \wedge \ldots \wedge T(x_{k-1}, x_k) \wedge \neg P(x_k)$$

Induction

Goal

- Prove that P is invariant

Standard Induction

- Show that the following formulas are valid (their negation is not satisfiable)

$$I(x_0) \to P(x_0)$$

$$P(x_0) \wedge T(x_0, x_1) \to P(x_1)$$

- Limitations:
  - This may fail even if P is invariant for $\mathcal{M}$
  - If the induction fails, P must be strengthened:
    find Q such that Q implies P and such that Q is an inductive invariant

k-induction

Generalizes induction to k steps

- Base case:

$$I(x_0) \wedge T(x_0, x_1) \wedge \ldots \wedge T(x_{k-1}, x_k) \Rightarrow P(x_0) \wedge \ldots \wedge P(x_k)$$

- Induction step:

$$T(x_0, x_1) \wedge \ldots \wedge T(x_k, x_{k+1}) \wedge P(x_0) \wedge \ldots \wedge P(x_k) \Rightarrow P(x_{k+1})$$

How good is it?

- In most cases, k-induction is stronger than standard induction (when $k \geqslant 2$) $\Box P$ is provable by k-induction iff $\Box(P \wedge \circ P \wedge \ldots \wedge \circ^k P)$ is provable by induction.

- There are counterexamples: For example, if T is reflexive, then $\Box P$ is provable by k-induction iff $\Box P$ is provable by standard induction.

BMP Verification

Proof Process

- The correctness property is not invariant (for any reasonable k)
- We need auxiliary lemmas:

```
l0 : LEMMA system |- G(phase = Settle OR tdata  = One OR tdata = Zero);
l1 : LEMMA system |- G(phase = Stable => (tclk <= (time + TSTABLE)));
l2 : LEMMA system |- G(phase = Settle => (tclk <= (time + TSETTLE)));
```

- The full proof requires four auxiliary lemmas, the main one is proved by k induction for $k = 5$.
- All proofs run in a few seconds.

Much Easier than Previous Proofs of BMP

- Vaandrager and de Groot, 2004, use PVS and Uppaal
  Difficult proof: need 37 invariants, 4000 proof steps, hours to run

Conclusion

Many Applications of SMT Solvers

- Backend/constraint solvers in another tool (e.g., static analysis, model checkers)
- Producing models is one of the most important features (e.g., test generation, scheduling, counterexamples)

Yices is becoming the backbone of SRI and others verification tools

- Solver for SAL
- Decision procedure for PVS