# Satisfiability Modulo Theories
# Applications and Challenges

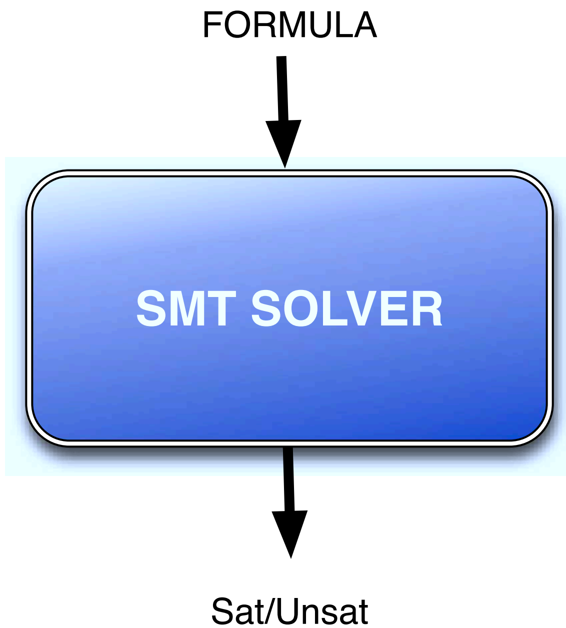
Summer School on Formal Techniques
Menlo Park, May 2012

Bruno Dutertre — SRI International

Leonardo de Moura — Microsoft Research

# Applications of SMT Solving

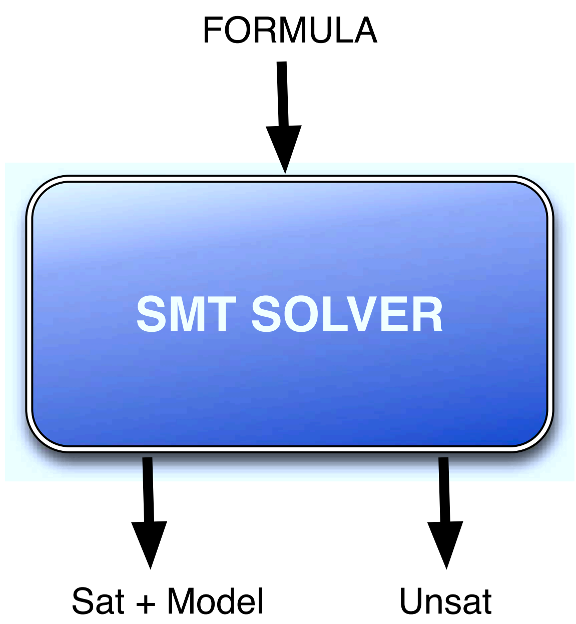
# What Can an SMT Solver Do?




Most Basic Solver: just gives a Yes/No answer (or Unknown)

Usage: theorem proving (e.g., for non-linear real arithmetic), abstraction

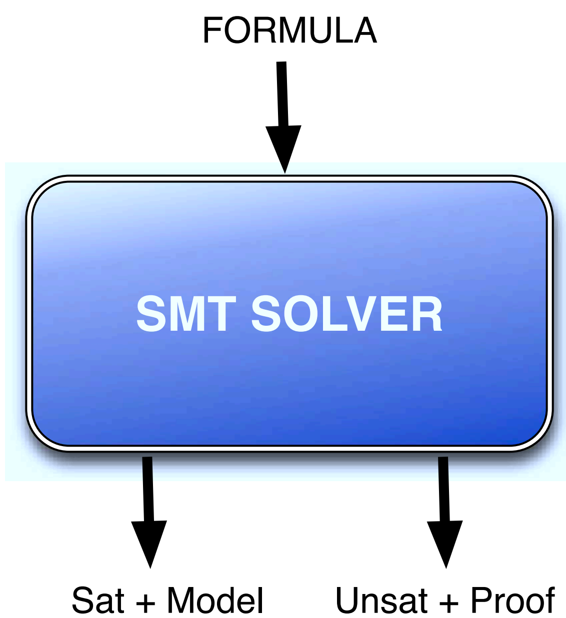
# What Can an SMT Solver Do?




Standard Solver: produces a model if the answer is 'sat'

Many Applications: test generation, constraint solving, etc.

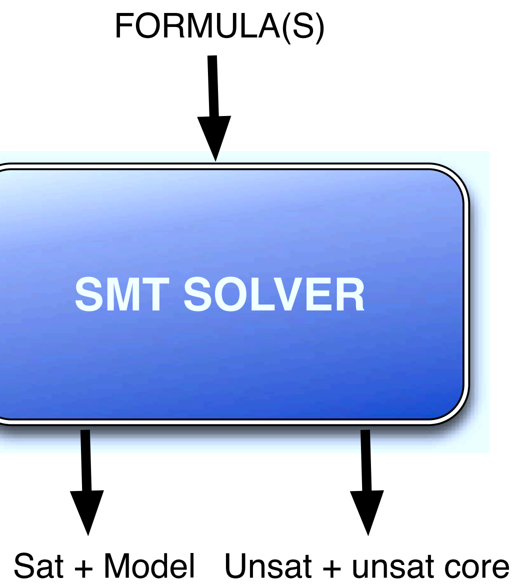
# What Can an SMT Solver Do?




Proof-Producing Solver: generates a proof if the answer is 'unsat'

Applications: system verification using interpolants, *paranoid* theorem proving

# What Can an SMT Solver Do?




Unsat Core: (small) subset of the input formulas that's unsatisfiable

Applications: optimization problems, diagnosis/debugging problems, etc.

# SMT Applications in Software Engineering

Test Generation and Bug Finding

- Example tools: SAGE, PEX, YOGI, CREST, KLEE, etc.

Static Analysis

- SMT solvers for predicate abstraction (e.g., SLAM)

Program Verification: SMT Solvers to discharge proof obligations

- Spec#, Boogie, etc.
- Why project; Frama-C, Krakatoa,
- ESC Java

Other: synthesis, symbolic execution, software model checking, termination proofs, compilation, type-checking, etc.

# Model Checking with SMT Solvers

## Applications

- Fault-tolerant systems and protocols
- Control systems and software
- Timed systems, hybrid systems

## Example Tools

- Symbolic Analysis Laboratory (SAL)
- K-Induction based Model Checker (KIND)
- MCMT (Model-Checking Modulo Theories)
- Kratos. etc.

# SMT Solvers as Proof Tools

## Integrated in Generic Theorem Provers

- SledgeHammer: Isabelle/HOL (Z3 and other solvers as backends)
- Yices in PVS (also in Isabelle/HOL)

## In Software Verification

- Spec#, Boogie, etc.
- Why project; Frama-C, Krakatoa,
- ESC Java

# SMT Solvers as Constraint Solvers

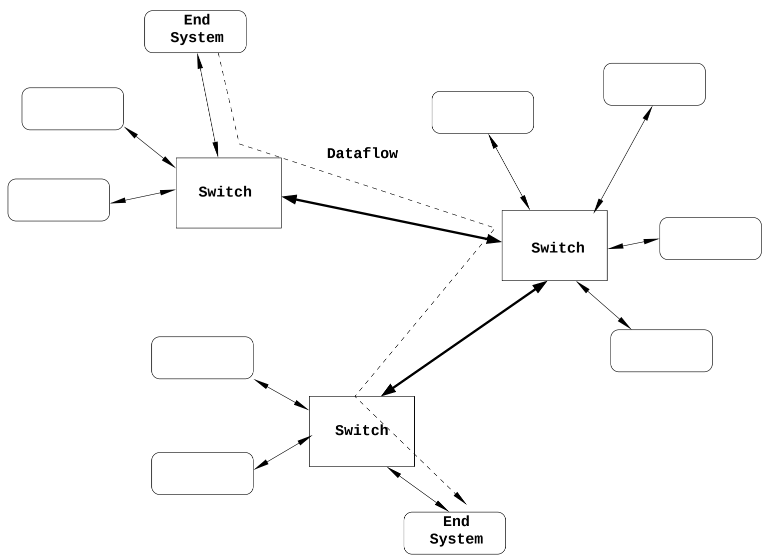
## Scheduling

- Communication Scheduling in Timed-Triggered Ethernet (Steiner)
- Control System Scheduler (Majumdar, et al.)

## General Constraint Solving/Programming

- Scala + Z3
- FlatZinc. etc.

# Application 1: Scheduling for Timed-Triggered Ethernet




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 $i$:

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

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)

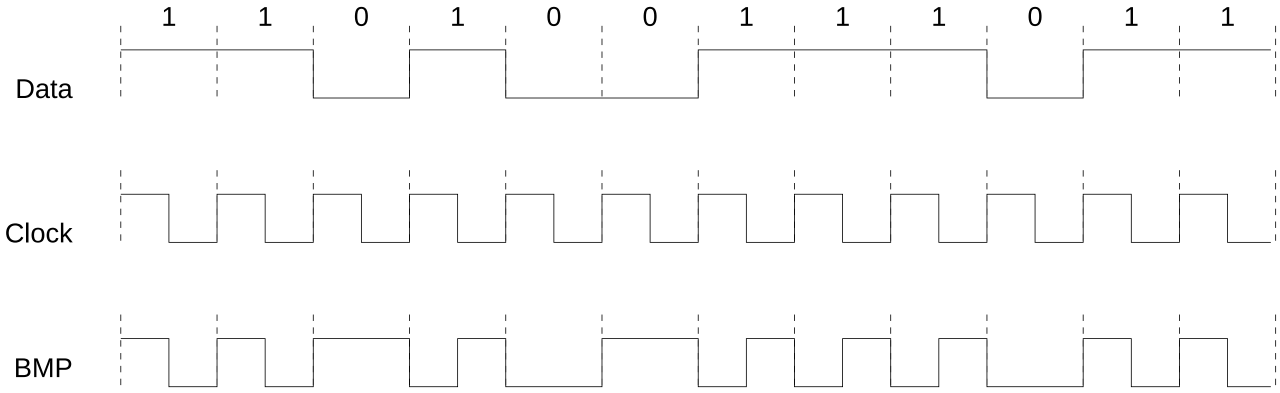
# Application 2: Verification of Timed Systems

## SAT & SMT Solvers as backends to SAL

- SAL is a toolkit for modeling and verification of state-transition systems
- Specification language: guarded commands + extensions
- SAL supports both synchronous and asynchronous 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` (with an SMT solver as backend)

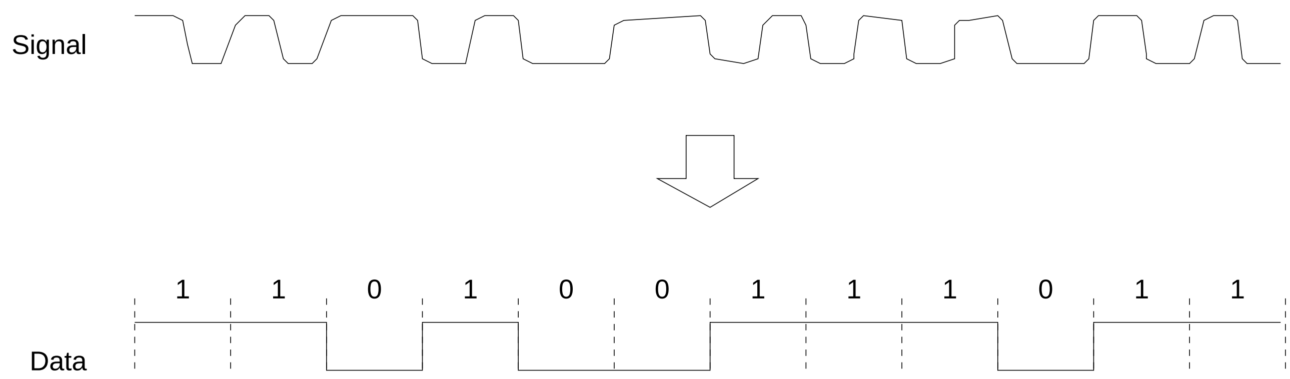
# Example: Biphase Mark Protocol (BMP)




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 signal

# BMP: Decoding Problem




Issues:

- jitter
- sampling uncertainties
- clock drift, phase shift

# 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)

$$I(x_0) \Rightarrow P(x_0)$$

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

- 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

# Application 3: Computational Biology

## 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 SMT Solver?

- Motivation for trying Yices: it does exact arithmetic, off-the-shelf solvers have licensing restrictions
- But: results are disappointing.
  - 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

# Problems and Challenges in SMT Solving

# Challenges: Better SMT Solving

## New Theories/Better Solvers for Hard Theories

- First-Order Logic
  - efficient/refutationally complete solvers?
  - conversely: model finding solvers
  - SMT + rewriting

- Beyond Linear Arithmetic
  - polynomial constraints and more (rational functions, trigonometry, log, etc.)
  - floating point verification

- Beyond Resolution
  - Resolution is at the core of CDCL but it has limitations (some problems have no short resolution proofs)

# Challenges: Better SMT Solving

Parallelization

- state of the art: portfolio approaches
- is there hope of better performance using lower-grain parallelism or other approaches?

# Challenges: Optimization using SMT-like Techniques

## Optimization Problems

- Minimize/maximize some objective function subject to some constraints
- Many applications
- Challenge: Can SMT solvers (or variants) beat known techniques?