Satisfiability Modulo Theories

Tjark Weber

UPPSALA
UNIVERSITET

Verification of Erlang Programs Day

January 31, 2012

Introduction

Satisfiability Modulo Theories =

Propositional satisfiability + background theories

Example: Job-Shop Scheduling

Given: n jobs, each composed of m tasks of varying duration, that must be performed consecutively on m machines; a total maximum time max.

$d_{i,j}$	Machine 1	Machine 2
Job 1	2	1
Job 2	3	1
Job 3	2	3

$max = 8$

Is there a schedule such that the end-time of every task is $\leq$ max?

Job-Shop Scheduling: SMT Encoding

The job-shop scheduling problem has a straightforward encoding in propositional logic + linear integer arithmetic.

A schedule is specified by the start time $t_{i,j}$ for the j-th task of every job i.

Precedence constraints:

$$t_{i,1} \geq 0 \ \wedge \ t_{i,2} \geq t_{i,1} + d_{i,1} \ \wedge \ t_{i,2} + d_{i,2} \leq max \qquad (\text{for } i = 1, 2, 3)$$

Resource constraints:

$$\begin{aligned}
&(t_{1,j} \geq t_{2,j} + d_{2,j} \ \vee \ t_{2,j} \geq t_{1,j} + d_{1,j}) \ \wedge \\
&(t_{1,j} \geq t_{3,j} + d_{3,j} \ \vee \ t_{3,j} \geq t_{1,j} + d_{1,j}) \ \wedge \\
&(t_{2,j} \geq t_{3,j} + d_{3,j} \ \vee \ t_{3,j} \geq t_{2,j} + d_{2,j}) \qquad (\text{for } j = 1, 2)
\end{aligned}$$

Job-Shop Scheduling: Solution

SMT formula encoding

$d_{i,j}$	Machine 1	Machine 2
Job 1	2	1
Job 2	3	1
Job 3	2	3

$max = 8$

Solution:

$$t_{1,1} = 5,\ t_{1,2} = 7$$
$$t_{2,1} = 2,\ t_{2,2} = 6$$
$$t_{3,1} = 0,\ t_{3,2} = 3$$

Background Theories

- EUF $\qquad x = y \implies f(x) = f(y)$
- Arithmetic $\qquad y < 0 \implies x + y < x$
- Arrays $\qquad \text{select}(\text{store}(a, i, x), i) = x$
- Bit-vectors $\qquad 2 \cdot x = x << 1$

- Quantifiers
- Algebraic data types
- ...

Applications

SMT solvers are the core engine of many tools for program analysis, testing and verification.

Dynamic Symbolic Execution

Task: To find input that can steer program execution into specific branches.

Program Model Checking

Task: To prove/refute conjectures about the values of program variables in order to characterize a finite-state abstraction.

Static Program Analysis

Task: To check feasibility of certain program paths.

Program Verification

Task: To prove verification conditions that arise from claims of functional correctness.

SMT Solver Use

We've seen what SMT solvers are good for. How do you actually interact with them?

The SMT-LIB Language

SMT solvers provide a textual interface. Most solvers support a standard language, SMT-LIB.

SMT-LIB defines

- concrete syntax for input formulas, and
- a command-based scripting language.

Solver-specific syntax is often available to extend SMT-LIB, e.g., for data types.

SMT-LIB: Example

```
; This example illustrates basic arithmetic and
; uninterpreted functions

(declare-fun x () Int)
(declare-fun y () Int)
(declare-fun z () Int)
(assert (>= (* 2 x) (+ y z)))
(declare-fun f (Int) Int)
(declare-fun g (Int Int) Int)
(assert (< (f x) (g x x)))
(assert (> (f y) (g x x)))
(check-sat)
(get-model)
(push)
(assert (= x y))
(check-sat)
(pop)
(exit)
```

ask z3 *Is this formula satisfiable?* ***Click 'ask z3'!***

tutorial home video

SMT-LIB: Example (Result)

ask z3

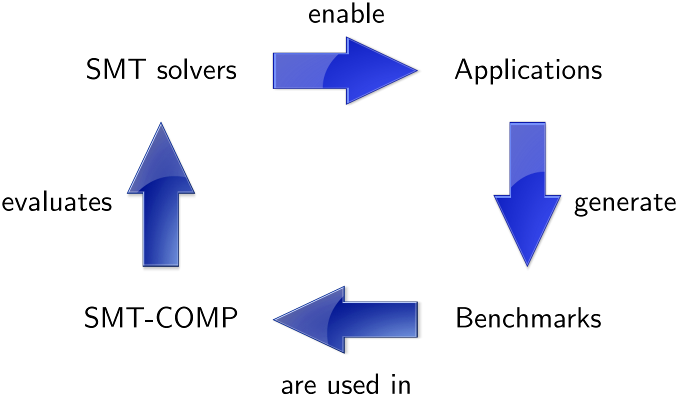
```
sat
(model
  (define-fun z () Int
    0)
  (define-fun y () Int
    (- 38))
  (define-fun x () Int
    0)
  (define-fun f ((x!1 Int)) Int
    (ite (= x!1 0) (- 1)
    (ite (= x!1 (- 38)) 1
      (- 1))))
  (define-fun g ((x!1 Int) (x!2 Int)) Int
    (ite (and (= x!1 0) (= x!2 0)) 0
      0))
)
unsat
```

Inter-Process Communication

- File-based (the basic solution)
- Stream-based (when you need online functionality)

- Web interface (mostly for quick experiments)
- In-memory API (the tightly integrated approach)
- ...

A Virtuous Circle



Choosing an SMT Solver

There are many good SMT solvers: Barcelogic, CVC, MathSAT, OpenSMT, Yices, Z3, ...

Differences:

- Supported background theories
- Platform, license, API, incrementality, quantifiers, ...
- Performance (cf. SMT-COMP)

Choosing an SMT Solver

Speaker's pick: Z3

- Good all-round solver, expressive input language
- Excellent performance
- Many other features: MaxSMT, fixed-point constraints, proofs
- Users can define custom theory solvers
- However, closed source (but free for academic use)

Your mileage may vary.

Algorithms

So far, we have considered SMT solvers as a black box.

SMT formula

sat (model)
unsat (proof)

This view is sufficient for many applications!

Conclusion

SMT solvers are expressive and easy to use. They scale orders of magnitude beyond custom ad hoc solvers.

Use them!

Do not write your own constraint solver.