Satisfiability Modulo Theories
and
Network Verification

Nikolaj Bjørner
Microsoft Research
Formal Methods and Networks Summer School
Ithaca, June 10-14 2013
Lectures

Wednesday 2:00pm-2:45pm:  
An Introduction to SMT with Z3

Thursday 11:00am-11:45am  
Algorithmic underpinnings of SAT/SMT

Friday 9:00am-9:45am  
Theories, Solvers and Applications
Plan

I. Satisfiability Modulo Theories in a nutshell

II. SMT solving in a nutshell

III. SMT by example
Takeaways:

• Modern SMT solvers are a often good fit for program analysis tools.
  – Handle domains found in programs directly.

• The selected examples are intended to show instances where sub-tasks are reduced to SMT/Z3.
Wasn’t that easy?!

Problems with bugs in your code?

Doctor Rustan’s tool to the rescue

Get to know how debugging your code gets the simple look and feel of spell checking in Word.* See some of the latest and most exciting research in formal verification employed in action. This will be a hands-on tutorial, so bring your own laptop to try it for yourself.

Jean Yang

I am a fifth-year Ph.D. student in Computer-Aided Programming at KTH.

If you use Z3, this could be you.

Rustan Leino from Microsoft Research is a world-leading expert in the area. Those who have seen his presentations know why programming is cool.

You don’t want to miss this!

When: Tuesday March 20, 2012 at 13:15 - 15:00
Where: E1, Osquars backe 2, KTH
http://www.cs.kth.se/tcs/seminarevents/rustanleino.php

* Your mileage may vary. Do not use when operating heavy machinery. Prolonged excitement from using programming tools may cause drowsiness. Some users report a sensation of increased and irresistible social attraction. If you experience bug withdrawal, consider collecting pet armadillidide.
Z3 – Backed by Proof Plumbers

Not all is hopeless

Leonardo de Moura, Nikolaj Bjørner, Christoph Wintersteiger
Background Reading: SAT

COMMUNICATIONS OF THE ACM

REVIEW ARTICLES

Boolean Satisfiability: From Theoretical Hardness to Practical Success

By Sharad Malik, Lintao Zhang
Communications of the ACM, Vol. 52 No. 8, Pages 76-82
10.1145/1536616.1536637

There are many practical situations where we need to satisfy several potentially conflicting constraints. Simple examples of this abound in daily life, for example, determining a schedule for a series of games that resolves the availability of players and venues, or finding a seating assignment at dinner consistent with various rules the host would like to impose. This also applies to applications in computing, for example, ensuring that a hardware/software system functions correctly with its overall behavior constrained by the behavior of its components and their composition, or finding a path for a robot to reach a goal that is
SAT solvers have been the focus of increased recent attention thanks to technological advances and industrial applications. Yet, they draw on a combination of some of the most fundamental areas in computer science as well as discoveries from the past century of symbolic logic. They combine the problem of Boolean Satisfiability with domains, such as, those studied in convex optimization and term-manipulating symbolic systems. They involve the decision problem, completeness and incompleteness of logical theories, and finally, complexity theory. In this article, we present an overview of the field of Satisfiability Modulo Theories, and some of its applications.

1.1 An SMT Application - Scheduling

Consider the classical job shop scheduling decision problem. In this problem, there are $n$ jobs, each composed of $m$ tasks of varying duration that have to be performed consecutively on $m$ machines. The start of a new task can be delayed as long as needed in order to wait for a machine to become available, but tasks cannot be interrupted once started.
SAT IN A NUTSHELL
SAT in a nutshell

\((\text{Tie} \lor \text{Shirt}) \land (\neg \text{Tie} \lor \neg \text{Shirt}) \land (\neg \text{Tie} \lor \text{Shirt})\)
SMT IN A NUTSHELL
Satisfiability Modulo Theories (SMT)

Is formula $\varphi$ satisfiable modulo theory $T$?

SMT solvers have specialized algorithms for $T$. 
Satisfiability Modulo Theories (SMT)

Array Theory
Arithmetic
Uninterpreted Functions

\[ x + 2 = y \Rightarrow f(\text{select}(\text{store}(a, x, 3), y - 2)) = f(y - x + 1) \]

select(store(a, i, v), i) = v
\[ i \neq j \Rightarrow \text{select}(\text{store}(a, i, v), j) = \text{select}(a, j) \]
SMT SOLVING IN A NUTSHELL

Job Shop Scheduling
Job Shop Scheduling

$\zeta(s) = 0 \Rightarrow s = \frac{1}{2} + ir$

$P = NP?$
Job Shop Scheduling

Constraints:

**Precedence:** between two tasks of the same job

\[ \text{start}_{2,2} \cap \text{end}_{2,2} = \emptyset \]

**Resource:** Machines execute at most one job at a time

\[ \text{start}_{4,2} \cap \text{end}_{4,2} = \emptyset \]
Job Shop Scheduling

Constraints:

Precedence:

Resource:

[\text{start}_{2,2} .. \text{end}_{2,2}] \cap [\text{start}_{4,2} .. \text{end}_{4,2}] = \emptyset

Encoding:

- start time of job 2 on mach 3
- duration of job 2 on mach 3

Not convex

\[ t_{2,2} + d_{2,2} \leq t_{4,2} \]
\[ t_{4,2} + d_{4,2} \leq t_{2,2} \]
Job Shop Scheduling

<table>
<thead>
<tr>
<th>$d_{i,j}$</th>
<th>Machine 1</th>
<th>Machine 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Job 1</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>Job 2</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>Job 3</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

$max = 8$

**Solution**

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

**Encoding**

$\begin{align*}
(t_{1,1} \geq 0) & \land (t_{1,2} \geq t_{1,1} + 2) \land (t_{1,2} + 1 \leq 8) \land \\
(t_{2,1} \geq 0) & \land (t_{2,2} \geq t_{2,1} + 3) \land (t_{2,2} + 1 \leq 8) \land \\
(t_{3,1} \geq 0) & \land (t_{3,2} \geq t_{3,1} + 2) \land (t_{3,2} + 3 \leq 8) \land \\
((t_{1,1} \geq t_{2,1} + 3) & \lor (t_{2,1} \geq t_{1,1} + 2)) \land \\
((t_{1,1} \geq t_{3,1} + 2) & \lor (t_{3,1} \geq t_{1,1} + 2)) \land \\
((t_{2,1} \geq t_{3,1} + 2) & \lor (t_{3,1} \geq t_{2,1} + 3)) \land \\
((t_{1,2} \geq t_{2,2} + 1) & \lor (t_{2,2} \geq t_{1,2} + 1)) \land \\
((t_{1,2} \geq t_{3,2} + 3) & \lor (t_{3,2} \geq t_{1,2} + 1)) \land \\
((t_{2,2} \geq t_{3,2} + 3) & \lor (t_{3,2} \geq t_{2,2} + 1))
\end{align*}$
Job Shop Scheduling

Efficient solvers:
- Floyd-Warshall algorithm
- Ford-Fulkerson algorithm

\[ z - t_{1,1} \leq 0 \]
\[ z - t_{2,1} \leq 0 \]
\[ z - t_{3,1} \leq 0 \]
\[ t_{3,2} - z \leq 5 \]
\[ t_{3,1} - t_{3,2} \leq -2 \]
\[ t_{2,1} - t_{3,1} \leq -3 \]
\[ t_{1,1} - t_{2,1} \leq -2 \]

\[ z - z = 5 - 2 - 3 - 2 = -2 < 0 \]
THEORIES
Theories

Uninterpreted functions
Explore the Z3 API using Python

```python
1  t11, t12, t21, t22, t31, t32 = Ints('t11 t12 t21 t22 t31 t32')
2  s = Solver()
3
4  s.add(And([t11 >= 0, t12 >= t11 + 2, t12 + 1 <= 8]))
5  s.add(And([t21 >= 0, t22 >= t21 + 3, t22 + 1 <= 8]))
6  s.add(And([t31 >= 0, t32 >= t31 + 2, t32 + 3 <= 8]))
7
8  s.add(Or(t11 >= t21 + 3, t21 >= t11 + 2))
9  s.add(Or(t11 >= t31 + 2, t31 >= t11 + 2))
10 s.add(Or(t21 >= t31 + 2, t31 >= t21 + 3))
11 s.add(Or(t21 >= t22 + 1, t22 >= t12 + 1))
12 s.add(Or(t12 >= t32 + 3, t32 >= t12 + 1))
13 s.add(Or(t22 >= t32 + 3, t32 >= t22 + 1))
14
15 |
16 print ">>", s.check()
17 print ">>", s.model()
18
19
```
Theories

Uninterpreted functions

Arithmetic (linear)

Bit-vectors

```python
x = BitVec('x', 32)
powers = [2**i for i in range(32)]
fast = And(x != 0, x & (x - 1) == 0)
slow = Or([x == p for p in powers])

prove(fast == slow)
print("buggy version...")

fast = x & (x - 1) == 0

prove(fast == slow)
```

proved

buggy version...

counterexample

[x = 0]
Theories

Uninterpreted functions
Arithmetic (linear)
Bit-vectors

Algebraic data-types
Theories

Uninterpreted functions

Arithmetic (linear)

Bit-vectors

Algebraic data-types

Arrays
Theories

Uninterpreted functions
Arithmetic (linear)
Bit-vectors
Algebraic data-types
Arrays

Polynomial Arithmetic

z3py

Explore the Z3 API using Python

```python
1 x, y, z = Reals('x y z')
2
3 solve(x**2 + y**2 < 1, x*y > 1,
4    show=True)
5
6 solve(x**2 + y**2 < 1, x*y > 0.4,
7    show=True)
8
9 solve(x**2 + y**2 < 1, x*y > 0.4, x < 0,
10   show=True)
11
12 solve(x**5 - x - y == 0, Or(y == 1, y == -1),
13   show=True)
14```

tutorial

about Z3Py - Python interface for the Z3 SMT solver.
Z3 supports extensional arrays, datatypes, uninterpreted functions, and more.
QUANTIFIERS
Equality-Matching

\[ g(c, x) \text{ matches } g(b, b) \]

with substitution \([x \rightarrow b]\)

modulo \(b = c\)

[de Moura, B. CADE 2007]
Quantifier Elimination

Presburger Arithmetic,
Algebraic Data-types,
Quadratic polynomials

SMT integration to prune branches [B. IJCAR 2010]
MBQI: Model based Quantifier Instantiation

(set-option :mbqi true)
(declare-fun f (Int Int) Int)
(declare-const a Int)
(declare-const b Int)

(assert (forall ((x Int)) (>= (f x x) (+ x a))))

(assert (< (f a b) a))
(assert (> a 0))
(check-sat)
(get-model)

(echo "evaluating (f (+ a 10) 20)"
(eval (f (+ a 10) 20))

[de Moura, Ge. CAV 2008]
[Bonachnia, Lynch, de Moura CADE 2009]
[de Moura, B. IJCAR 2010]
Superposition

1. \( \forall x. (x; id) = x \)
2. \( \forall x. (id; x) = x \)
3. \( \forall x. (id \mid x) = x \)
4. \( \forall x y z u. (x \mid y); (z \mid u) \leq (x, z) \mid (y, u) \) \( \forall p q. (p, q) \leq (p \mid q) \)

5. \( \forall x z u . x; (z \mid u) \leq (x, z) \mid (id; u) \) super-pose 1, 4
6. \( \forall x z u . x; (z \mid u) \leq (x, z) \mid u \) super-pose 2,5
7. \( \forall x z u . x; u \leq (x, id) \mid u \) super-pose 3,6
8. \( \forall x z u . x; u \leq x \mid u \) super-pose 1,7

[de Moura, B. IJCAR 2008]

(disabled in release 4.1)
Horn Clauses

\[ mc(x) = x-10 \quad \text{if } x > 100 \]
\[ mc(x) = mc(mc(x+11)) \quad \text{if } x \leq 100 \]

assert \( (mc(x) \geq 91) \)

\[ \forall X. \quad X > 100 \rightarrow mc(X, X-10) \]
\[ \forall X, Y, R. \quad X \leq 100 \land mc(X+11, Y) \land mc(Y, R) \rightarrow mc(X, R) \]
\[ \forall X, R. \quad mc(X, R) \land X \leq 101 \rightarrow R = 91 \]

Solver finds solution for \( mc \)  

[Hoder, B. SAT 2012]
MODELS, PROOFS, CORES & SIMPLIFICATION
Models

Logical Formula

Click on a tool to load a sample then ask:

- agl
- bek
- boogie
- code contracts
- concurrent revisions
- dafny
- esm
- fine
- heapdbg
- poirot
- pex
- rex
- spec#
- vcc

```
(define-sorts ((A (Array Int Int))))
(declare-funs ((x Int) (y Int) (z Int)))
(declare-funs ((a1 A) (a2 A) (a3 A)))
(assert (= (select a1 x) x))
(assert (= (store a1 x y) a1))
(check-sat)
(get-info model)
```

Is this SMT formula satisfiable? Click 'ask Z3'! Read more or watch the video.

```
sat
  ("model" 
  (define x 0)
  (define a1 as-array[k!0])
  (define y 0)
  (define (k!0 (x1 Int))
    (if (= x1 0) 0
     1)))
```
Proofs

(set-logic QF_LIA)
(declare-funs ((x Int) (x1 Int)))
(declare-funs ((x3 Int) (x2 Int)))
(declare-funs ((x4 Int) (x5 Int)))
(declare-funs ((y Int) (z Int) (u Int)))
(assert (> x y))
(assert (= (- (* x 3) (* y 3)) (- z u)))
(assert (= 0 z))
(assert (= 0 u))
(assert (< z 3))
(assert (< u 3))
(check-sat)
(get-proof)

proof.smt2 PROOF_MODE=2

ttd (<= 0 u) [rewrite (iff (<= 0 u) (>= u 0))] (>= u 0)
im

ertd (< (- (* x 3) (* y 3)) (< z u))
ns
nonotonicity
[trans
 [rewrite (<= (< (* x 3) (* y 3)) (< z u))]
 [rewrite (>= (x 3) (x 3 y))]
 [rewrite (<= (- (* x 3) (* y 3)) (< (* 3 x) (* 3 y)))]
 [rewrite (>= (- (* x 3) (* y 3)) (< (* 3 x) (* 3 y)))]
 [rewrite (<= (- (* x 3) (* y 3)) (< (* 3 x) (* 3 y)))]
 [rewrite (iff (= (- (* x 3) (* y 3)) (< z u))]
 [rewrite (iff (= (+ (* 3 x) (< z u)) (< z u))]
 [rewrite (iff (= (+ (* 3 x) (< z u)) (< z u))]
 [rewrite (iff (= (+ (* 3 x) (< z u)) (< z u))]
 [rewrite (iff (= (+ (* 3 x) (< z u)) (< z u))]
 [rewrite (iff (= (+ (* 3 x) (< z u)) (< z u))]
 [rewrite (iff (= (+ (* 3 x) (< z u)) (< z u))]
 [rewrite (iff (= (+ (* 3 x) (< z u)) (< z u))]
 [rewrite (iff (= (+ (* 3 x) (< z u)) (< z u))]
 [rewrite (iff (= (+ (* 3 x) (< z u)) (< z u))]
 [rewrite (iff (= (+ (* 3 x) (< z u)) (< z u))]
 [rewrite (iff (= (+ (* 3 x) (< z u)) (< z u))]
 [rewrite (iff (= (+ (* 3 x) (< z u)) (< z u))]
 [rewrite (iff (= (+ (* 3 x) (< z u)) (< z u))]
 [rewrite (iff (= (+ (* 3 x) (< z u)) (< z u))]
 [rewrite (iff (= (+ (* 3 x) (< z u)) (< z u))]
 [rewrite (iff (= (+ (* 3 x) (< z u)) (< z u))]
 [rewrite (iff (= (+ (* 3 x) (< z u)) (< z u))]
 [rewrite (iff (= (+ (* 3 x) (< z u)) (< z u))]
 [rewrite (iff (= (+ (* 3 x) (< z u)) (< z u))]
 [rewrite (iff (= (+ (* 3 x) (< z u)) (< z u))]
 [rewrite (iff (= (+ (* 3 x) (< z u)) (< z u))]
 [rewrite (iff (= (+ (* 3 x) (< z u)) (< z u))]
 [rewrite (iff (= (+ (* 3 x) (< z u)) (< z u))]
 [rewrite (iff (= (+ (* 3 x) (< z u)) (< z u))]
 [rewrite (iff (= (+ (* 3 x) (< z u)) (< z u))]
 [rewrite (iff (= (+ (* 3 x) (< z u)) (< z u))]
 [rewrite (iff (= (+ (* 3 x) (< z u)) (< z u))]
 [rewrite (iff (= (+ (* 3 x) (< z u)) (< z u))]
 [rewrite (iff (= (+ (* 3 x) (< z u)) (< z u))]
 [rewrite (iff (= (+ (* 3 x) (< z u)) (< z u))]
 [rewrite (iff (= (+ (* 3 x) (< z u)) (< z u))]
 [rewrite (iff (= (+ (* 3 x) (< z u)) (< z u))]
 [rewrite (iff (= (+ (* 3 x) (< z u)) (< z u))]
 [rewrite (iff (= (+ (* 3 x) (< z u)) (< z u))]
 [rewrite (iff (= (+ (* 3 x) (< z u)) (< z u))]
 [rewrite (iff (= (+ (* 3 x) (< z u)) (< z u))]
 [rewrite (iff (= (+ (* 3 x) (< z u)) (< z u))]
 [rewrite (iff (= (+ (* 3 x) (< z u)) (< z u))]
 [rewrite (iff (= (+ (* 3 x) (< z u)) (< z u))]
 [rewrite (iff (= (+ (* 3 x) (< z u)) (< z u))]
 [rewrite (iff (= (+ (* 3 x) (< z u)) (< z u))]
 solsp

Unsat/Proof
Simplification

R1SE4+un

gave 48,503 answers!

Click on a tool to load a sample then ask!

agl  bek  boogie  code contracts

concurrent revisions  dafny  esm  fine

heapdbg  poirof  pex  rex  spec#  vcc

z3

(declare-fun x () Real)
(declare-fun y () Real)
(simplify (>= x (+ x y)))

Is this SMT formula satisfiable? Click 'ask z3'! Read more or watch the video.

(<= y 0.0)

Simplify

Logical Formula
Cores

Logical Formula

(declare-preds ((p) (q) (r) (s)))
(set-option enable-cores)
(assert (or p q))
(assert (implies r s))
(assert (implies s (iff q r)))
(assert (or r p))
(assert (or r s))
(assert (not (and r q)))
(assert (not (and s p)))
(check-sat)
(get-unsat-core)

Is this SMT formula satisfiable? Click 'ask z3'! Read more or watch the video.

Unsat
((or p q)
(=> r s)
(or r p)
(or r s)
(not (and r q))
(not (and s p)))
TACTICS, SOLVERS
Tactics

(declare-const x (_ BitVec 16))
(declare-const y (_ BitVec 16))

(assert (= (bvor x y) (_ bv13 16)))
(assert (bvslt x y))

(check-sat-using (then simplify solve-eqs bit-blast sat))
(get-model)

Composition of tactics:
• (then t s)
• (par-then t s) applies t to the input goal and S to every subgoal produced by t in parallel.
• (or-else t s)
• (par-or t s) applies t and S in parallel until one of them succeed.
• (repeat t)
• (repeat t n)
• (try-for t ms)
• (using-params t params) Apply the given tactic using the given parameters.
Solvers

- Tactics take goals and reduce to sub-goals
- Solvers take tactics and serve as logical contexts.
  - push
  - add
  - check
  - model, core, proof
  - pop

```python
bv_solver = Then(With('simplify', mul2concat=True),
                 'solve-eqs',
                 'bit-blast',
                 'aig',
                 'sat').solver()
x, y = BitVecs('x y', 16)
bv_solver.add(x*32 + y == 13, x & y < 10, y > -100)
print bv_solver.check()
m = bv_solver.model()
print m
print x*32 + y, "==", m.evaluate(x*32 + y)
print x & y, "==", m.evaluate(x & y)
```
APIS
Summary

Z3 supports several theories
   – Using a default combination
   – Providing custom tactics for special combinations

Z3 is more than sat/unsat
   – Models, proofs, unsat cores,
   – simplification, quantifier elimination are tactics

Prototype with python/smt-lib2
   – Implement using smt-lib2/programmatic API