Software Verification with Satisfiability Modulo Theories

Nikolaj Bjørner
Microsoft Research
SSFT 2014, Menlo Park
Contents

A primer on SMT with Z3

SMT & Verification by Assertion Checking
- Boogie GC, Quantifiers, Theories

SMT & Verification by Assertion Inference
- Symbolic Software Model Checking, Horn Clauses
SMT & Verification

Program + Property

Verification with Auxiliary Invariants

Boogie/Dafny

SAGE, Pex

Synthesize Auxiliary invariants

Find bugs

SLAM, Yogi, DASH, Z3-Horn, Duality, HSF, Liquid
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.

If you use Z3, this could be you

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

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

Jean Yang

I am a fifth-year Ph.D. student in the Computer-Aided Programming group.

My goal is to automate the creative process using forward-chaining approaches to construct interesting programs.

To get an idea of the research I am working on, please visit my Programming Languages Superpages.

Research Projects.
- The Jeeves (programming language for automatically enforcing privacy policies)
- The Verve (computing system, the first automatically and ever-changing programming language)

Peer-Reviewed Publications.
- Safe to the Last Instruction: Automated Verification of Secure Compilers, with Chris Hawblitzel, PLDI 2010. [Paper: pdf, Slides: pptx]
- First! by Xavier Leroy. [Full text: html pdf | Technical Report]

Pustan Leino is a senior research scientist at Microsoft Research Cambridge. He is a leading expert in the area. The reason we have seen his presentations know why programming is cool.

You don’t want to miss this!
Not all is hopeless
A primer on SMT with Z3
Applications of SMT solvers to Program Verification

Rough notes for SSPT 2014
Prepared as part of a forthcoming revision of Daniel Kröning and Ofer Strichman’s book on Decision Procedures

May 16, 2014

Springer

1.1 An SMT Application - Scheduling
Consider the classical job shop scheduling decision problem. In this problem, there are n jobs, each composed of n 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

September 2011
Some Microsoft Tools based on Z3

Program Verification

Auditting

Type Safety

Testing

Analysis

Synthesis
## New Projects

**rise4fun**

A community of software engineering tools

- [fe](http://rise4fun.com/) A formal verification tool for higher-order elated programs
- [fast](http://rise4fun.com/) A domain-specific language for writing and analyzing tree manipulating programs
- [ie3](http://rise4fun.com/) Efficient and scalable theorem prover

### Microsoft

- [agl](http://rise4fun.com/) Automatic Graph Layout
- [bex](http://rise4fun.com/) A domain-specific language for writing and analyzing string matching
- [boogie](http://rise4fun.com/) Intermediate Verification Language

### Dafny

- [ckal](http://rise4fun.com/) A domain-specific language for writing and analyzing string matching
- [esm](http://rise4fun.com/) Efficient Software Engineering and Measurement Group
- [fast](http://rise4fun.com/) A domain-specific language for writing and analyzing tree manipulating programs

### Cilkal

- [hcac4bg](http://rise4fun.com/) A domain-specific language for writing and analyzing tree manipulating programs
- [i23](http://rise4fun.com/) An efficient theorem prover
- [koka](http://rise4fun.com/) A domain-specific language for writing and analyzing string matching

### Rex

- [seal](http://rise4fun.com/) A domain-specific language for writing and analyzing string matching
- [slayer](http://rise4fun.com/) An efficient theorem prover
- [spec](http://rise4fun.com/) A domain-specific language for writing and analyzing string matching

### Z3

- [z3bio](http://rise4fun.com/) A domain-specific language for writing and analyzing string matching
- [z3py](http://rise4fun.com/) Python Interface for the Z3 Theorem Prover

### Gravy

- [joogic](http://rise4fun.com/) Java and C++ static analysis

### Eth Zurich - Chair of Software Engineering

- [bongalo](http://rise4fun.com/) A domain-specific language for writing and analyzing string matching
- [javann](http://rise4fun.com/) A domain-specific language for writing and analyzing string matching
- [qfis](http://rise4fun.com/) A domain-specific language for writing and analyzing string matching

### KU Leuven

- [verifast](http://rise4fun.com/) A domain-specific language for writing and analyzing string matching
- [gpusverify-cuda](http://rise4fun.com/) A domain-specific language for writing and analyzing string matching

### Multicore Programming Group, Imperial College London

- [gpusverify-opencl](http://rise4fun.com/) A domain-specific language for writing and analyzing string matching

### University of Utah and IMDEA Software Institute

- [smack](http://rise4fun.com/) A domain-specific language for writing and analyzing string matching

---

**albert-ludwigs-university freiburg**

**eth zurich - chair of software engineering**

**ku leuven**

**multicore programming group, imperial college london**

**university of utah and imdea software institute**
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)

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

Array Theory  Arithmetic  Uninterpreted Functions

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

Job Shop Scheduling
Job Shop Scheduling

\[ \zeta(s) = 0 \Rightarrow s = \frac{1}{2} + ir \]
Job Shop Scheduling

Constraints:

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

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

\[
[start_{2,2} \ldots end_{2,2}] \cap [start_{4,2} \ldots end_{4,2}] = \emptyset
\]
Job Shop Scheduling

Constraints:

Precedence:

[Image of precedence diagram]

Resource:

[Image of resource diagram]

Encoding:

\( t_{2,3} \) - start time of job 2 on mach 3

\( d_{2,3} \) - duration of job 2 on mach 3

\( t_{2,3} + d_{2,3} \leq t_{2,4} \)

Not convex

\( [start_{2,2}..end_{2,2}] \cap [start_{4,2}..end_{4,2}] = \emptyset \)

\( t_{2,2} + d_{2,2} \leq t_{4,2} \v盍 t_{4,2} + d_{4,2} \leq t_{2,2} \)
### 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**

\[(t_{1,1} \geq 0) \land (t_{1,2} \geq t_{1,1} + 2) \land (t_{1,2} + 1 \leq 8) \land (t_{2,1} \geq 0) \land (t_{2,2} \geq t_{2,1} + 3) \land (t_{2,2} + 1 \leq 8) \land (t_{3,1} \geq 0) \land (t_{3,2} \geq t_{3,1} + 2) \land (t_{3,2} + 3 \leq 8) \land ((t_{1,1} \geq t_{2,1} + 3) \lor (t_{2,1} \geq t_{1,1} + 2)) \land ((t_{1,1} \geq t_{3,1} + 2) \lor (t_{3,1} \geq t_{1,1} + 2)) \land ((t_{2,1} \geq t_{3,1} + 2) \lor (t_{3,1} \geq t_{2,1} + 3)) \land ((t_{1,2} \geq t_{2,2} + 1) \lor (t_{2,2} \geq t_{1,2} + 1)) \land ((t_{1,2} \geq t_{3,2} + 3) \lor (t_{3,2} \geq t_{1,2} + 1)) \land ((t_{2,2} \geq t_{3,2} + 3) \lor (t_{3,2} \geq t_{2,2} + 1))\]
Job Shop Scheduling

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

\[(t_{1,1} \geq 0) \land (t_{1,2} \geq t_{1,1} + 2) \land (t_{1,2} + 1 \leq 8) \land
(t_{2,1} \geq 0) \land (t_{2,2} \geq t_{2,1} + 3) \land (t_{2,2} + 1 \leq 8) \land
(t_{3,1} \geq 0) \land (t_{3,2} \geq t_{3,1} + 2) \land (t_{3,2} + 3 \leq 8) \land
((t_{1,1} \geq t_{2,1} + 3) \lor (t_{2,1} \geq t_{1,1} + 2)) \land
((t_{1,1} \geq t_{3,1} + 2) \lor (t_{3,1} \geq t_{1,1} + 2)) \land
((t_{2,1} \geq t_{3,1} + 2) \lor (t_{3,1} \geq t_{2,1} + 3)) \land
((t_{1,2} \geq t_{2,2} + 1) \lor (t_{2,2} \geq t_{1,2} + 1)) \land
((t_{1,2} \geq t_{3,2} + 3) \lor (t_{3,2} \geq t_{1,2} + 1)) \land
((t_{2,2} \geq t_{3,2} + 3) \lor (t_{3,2} \geq t_{2,2} + 1))\]

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

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

Uninterpreted functions
Uninterpreted function

Arithmetic (linear)

```python
import z3

t11, t12, t21, t22, t31, t32 = [z3.Int(f't{i}) for i in range(1, 13)]
s = z3.Solver()

s.add(z3.And([t11 >= 0, t12 >= t11 + 2, t12 + 1 <= 8]))
s.add(z3.And([t21 >= 0, t22 >= t21 + 3, t22 + 1 <= 8]))
s.add(z3.And([t31 >= 0, t32 >= t31 + 2, t32 + 3 <= 8]))
s.add(z3.Or(t11 >= t21 + 3, t21 >= t11 + 2))
s.add(z3.Or(t11 >= t31 + 2, t31 >= t11 + 2))
s.add(z3.Or(t21 >= t31 + 2, t31 >= t21 + 3))
s.add(z3.Or(t21 >= t22 + 1, t22 >= t12 + 1))
s.add(z3.Or(t12 >= t32 + 3, t32 >= t12 + 1))
s.add(z3.Or(t22 >= t32 + 3, t32 >= t22 + 1))

print('sat', s.check())
print('model', s.model())
```

`tutorial`
Theories

Uninterpreted functions

Arithmetic (linear)

Bit-vectors

```
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..."
prove(fast == slow)
```
Theories

Uninterpreted functions

Arithmetic (linear)

Bit-vectors

Algebraic data-types

Explore the Z3 API using Python

```python
List = Datatype('List')
List.declare('cons', ('car', IntSort()), ('cdr', List))
List.declare('nil')
List = List.create()
cons = List.cons
car = List.car
```

```python
cdr = List.cdr
nil = List.nil
l1 = cons(10, cons(20, nil))
```

```python
print '>>', simplify(cdr(l1))
print '>>', simplify(car(l1))
print '>>', simplify(l1 == nil)
```

```python
x, y = Ints('x y')
l1 = Const('l1',List)
l2 = Const('l2',List)
s = Solver()
tutorial
```
Theories

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

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

Polynomial Arithmetic
QUANTIFIERS
Quantifier Elimination

1 (define-fun stamp () Bool
2   (forall ((x Int))
3     (=>
4       (>= x 8)
5       (exists ((u Int) (v Int))
6         (and (>= u 0) (>= v 0) (= x (+ (* 3 u) (* 5 v)))))))
7   (simplify stamp)
8
9 (elim-quantifiers stamp)

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]
MODELS, PROOFS, CORES & SIMPLIFICATION
Models

Logical Formula

(define-sortsa (Array Int Int))
(declare-funsa (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)

Logical Formula

proof.smt2 PROOF_MODE=2

Unsat/Proof
**Simplification**

**R1SE4+un**

Click on a tool to load a sample then ask!

- akl
- bek
- boogie
- code contracts
- concurrent revisions
- dafny
- esm
- fine
- heapdbg
- poirot
- pex
- rex
- spec#
- vcc
- z3

---

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

---

**ask z3**

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

---

**(<= y 0.0)**

---

**Logical Formula**

---

**Simplify**
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)
ping x & y, "==", m.evaluate(x & y)
```
APIS

C++  python  OCaml  .NET

Java
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