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

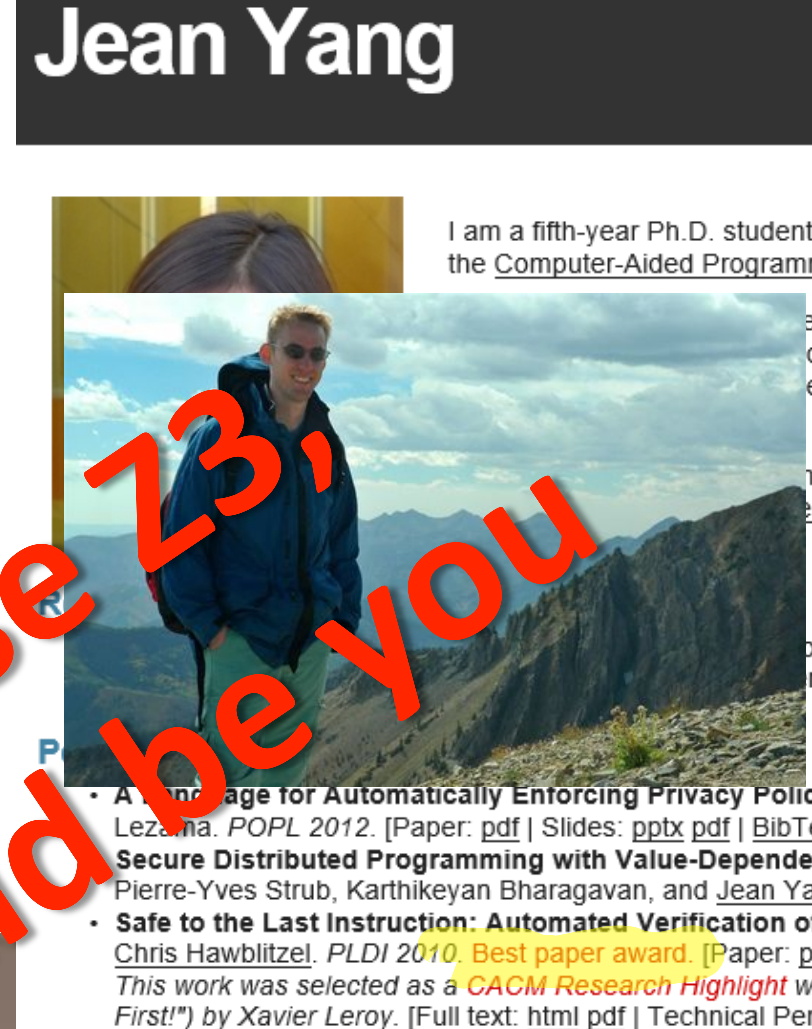
II. SMT solving in a nutshell

III. SMT by example

Takeaways:

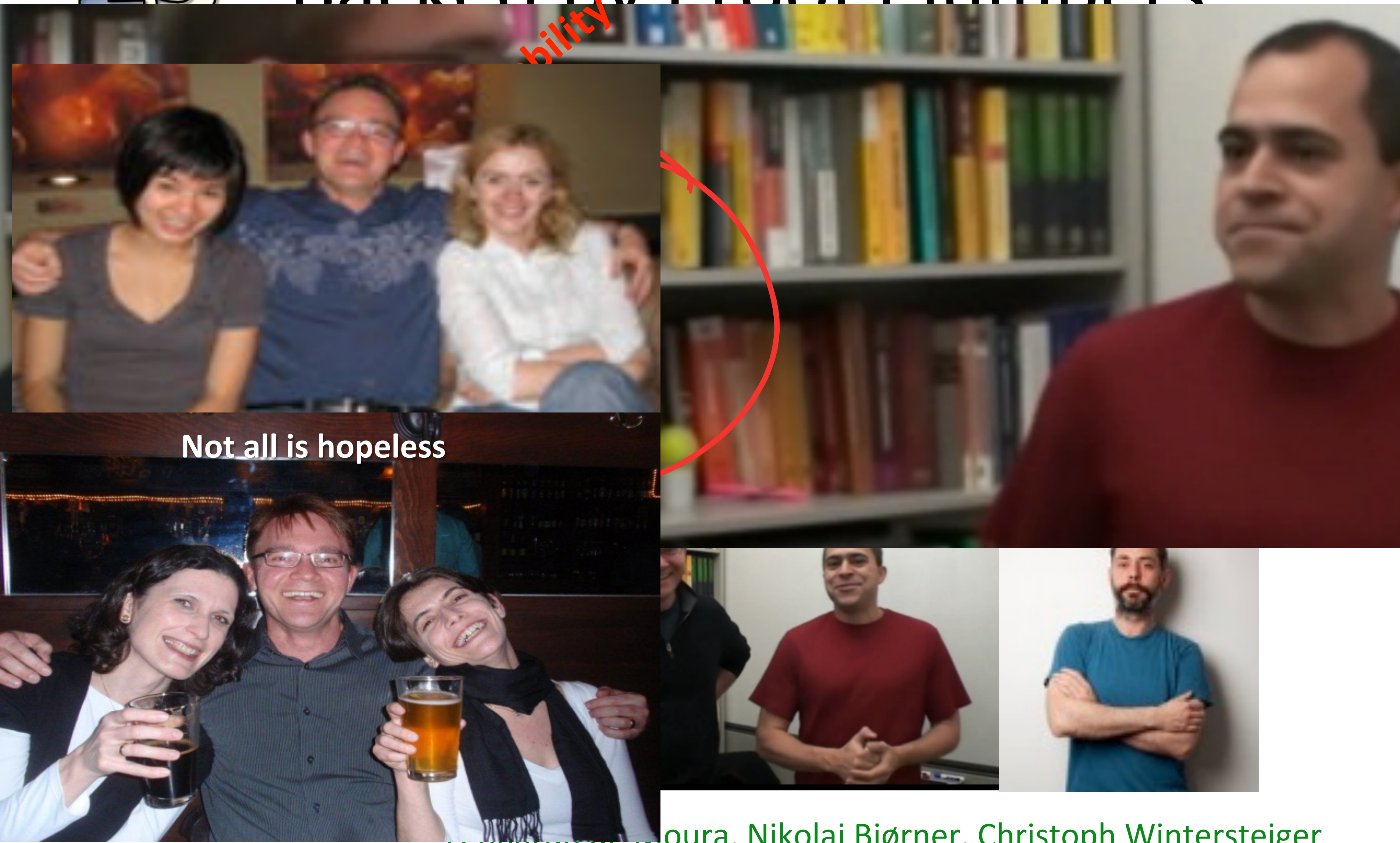
- Modern SMT solvers are a often good fit for program analysis tools.
  - Handle domains found in programs directly.

- The selected examples are intended to show instances where sub-tasks are reduced to SMT/Z3.



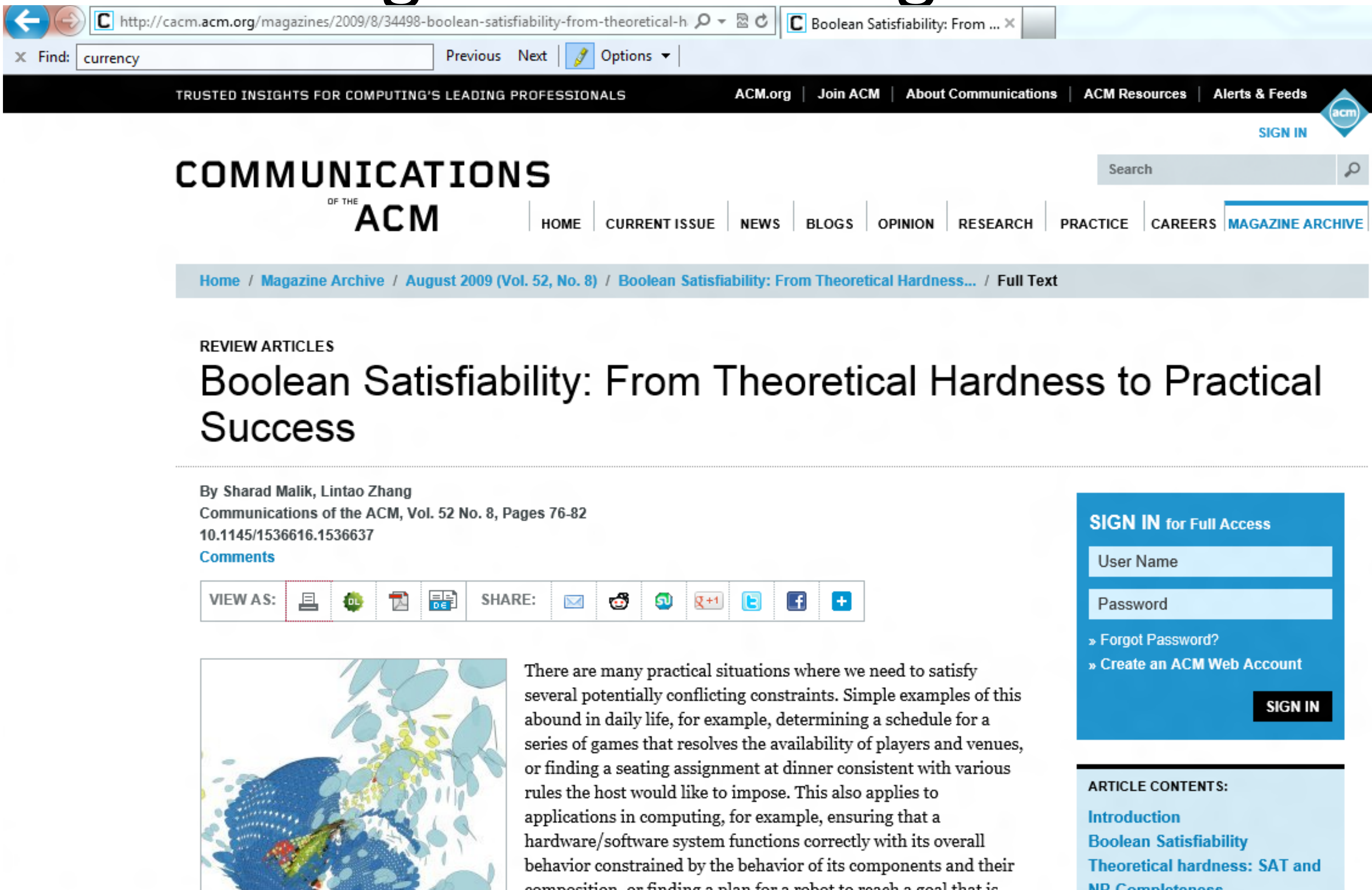
If you use Z3,
This could be you

Z3 – Backed by Proof Plumbers



Leonardo de Moura, Nikolaj Bjørner, Christoph Wintersteiger

Background Reading: SAT



http://cacm.acm.org/magazines/2009/8/34498-boolean-satisfiability-from-theoretical-h

Boolean Satisfiability: From ...

Find: currency Previous Next Options

TRUSTED INSIGHTS FOR COMPUTING'S LEADING PROFESSIONALS ACM.org | Join ACM | About Communications | ACM Resources | Alerts & Feeds

SIGN IN

COMMUNICATIONS OF THE ACM

Search

HOME | CURRENT ISSUE | NEWS | BLOGS | OPINION | RESEARCH | PRACTICE | CAREERS | MAGAZINE ARCHIVE

Home / Magazine Archive / August 2009 (Vol. 52, No. 8) / Boolean Satisfiability: From Theoretical Hardness... / Full Text

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

Comments

VIEW AS: SHARE:

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

SIGN IN for Full Access

User Name

Password

» Forgot Password?

» Create an ACM Web Account

SIGN IN

ARTICLE CONTENTS:

Introduction

Boolean Satisfiability

Theoretical hardness: SAT and

Background Reading: SMT

September 2011

...isfiability Modulo Theories: Introduction & Applications

Leonardo de Moura
Microsoft Research
One Microsoft Way
Redmond, WA 98052
leonardo@microsoft.com

Nikolaj Bjørner
Microsoft Research
One Microsoft Way
Redmond, WA 98052
nbjorner@microsoft.com

...RACT

...int satisfaction problems arise in many diverse ar-
...uding software and hardware verification, type infer-
...atic program analysis, test-case generation, schedul-
...anning and graph problems. These areas share a
...n trait, they include a core component using logical
...s for describing states and transformations between
...The most well-known constraint satisfaction problem
...ositional satisfiability, SAT, where the goal is to de-
...ether a formula over Boolean variables, formed using
...connectives can be made *true* by choosing *true*/*false*
...for its variables. Some problems are more naturally
...ed using richer languages, such as arithmetic. A sup-
... *theory* (of arithmetic) is then required to capture
...ning of these formulas. Solvers for such formulations
...monly called *Satisfiability Modulo Theories* (SMT)

SMT 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.

key driving factor [4]. An important ingredient is a common interchange format for benchmarks, called SMT-LIB [33], and the classification of benchmarks into various categories depending on which theories are required. Conversely, a growing number of applications are able to generate benchmarks in the SMT-LIB format to further inspire improving SMT solvers.

There is a relatively long tradition of using SMT solvers in select and specialized contexts. One prolific case is theorem proving systems such as ACL2 [26] and PVS [32]. These use decision procedures to discharge lemmas encountered during interactive proofs. SMT solvers have also been used for a long time in the context of program verification and *extended static checking* [21], where verification is focused on assertion checking. Recent progress in SMT solvers, however, has enabled their use in a set of diverse applications, including interactive theorem provers and extended static checkers, but also in the context of scheduling, planning, test-case generation, model-based testing and program development, static program analysis, program synthesis, and run-time analysis, among several others.

We begin by introducing a motivating application and a simple instance of it that we will use as a running example.

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

SAT IN A NUTSHELL

SAT in a nutshell

[$(Tie \lor Shirt) \land (\neg Tie \lor \neg Shirt) \land (\neg Tie \lor Shirt)$]

SMT IN A NUTSHELL

Satisfiability Modulo Theories (SMT)

Is formula φ satisfiable modulo theory *T* ?

SMT solvers have specialized algorithms for *T*

Satisfiability Modulo Theories (SMT)

$$x+2=y \Rightarrow f(select(store(a,x,3),y-2))=f(y-x+1)$$

Array Theory

Arithmetic

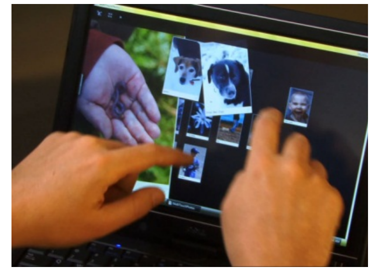
Uninterpreted Functions

$$select(store(a,i,v),i)=v$$
$$i \neq j \Rightarrow select(store(a,i,v),j)=select(a,j)$$

SMT SOLVING IN A NUTSHELL

Job Shop Scheduling

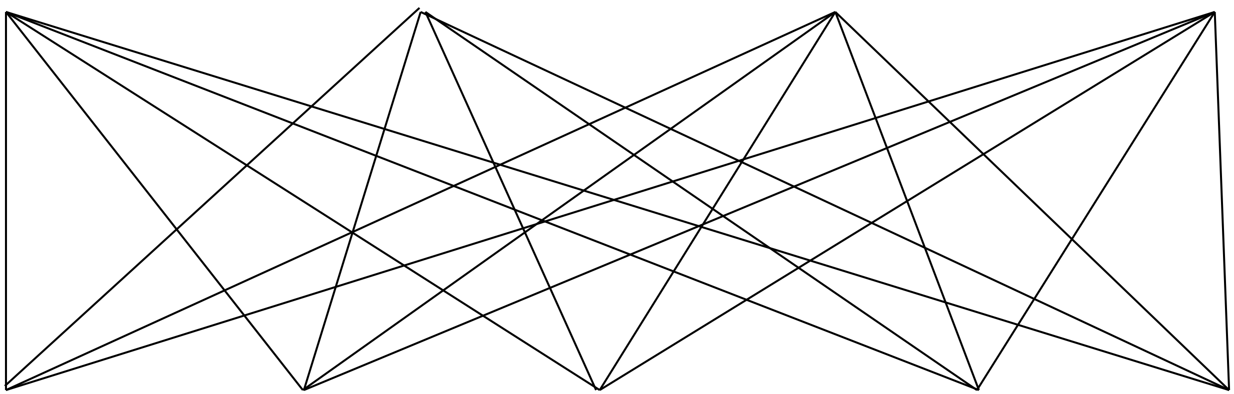
Job Shop Scheduling



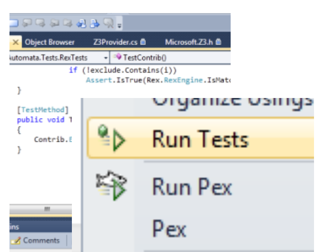
Machines

Tasks

Jobs



P = NP?

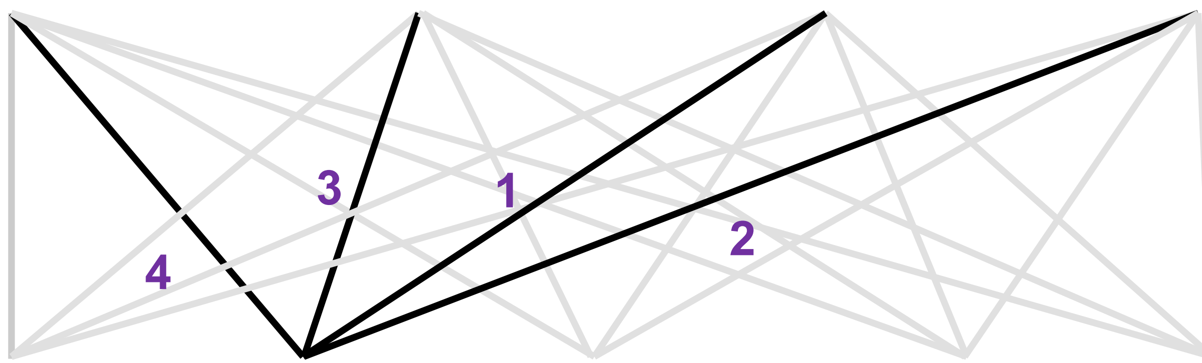


$\zeta(s)=0 \Rightarrow s=1/2+ir$

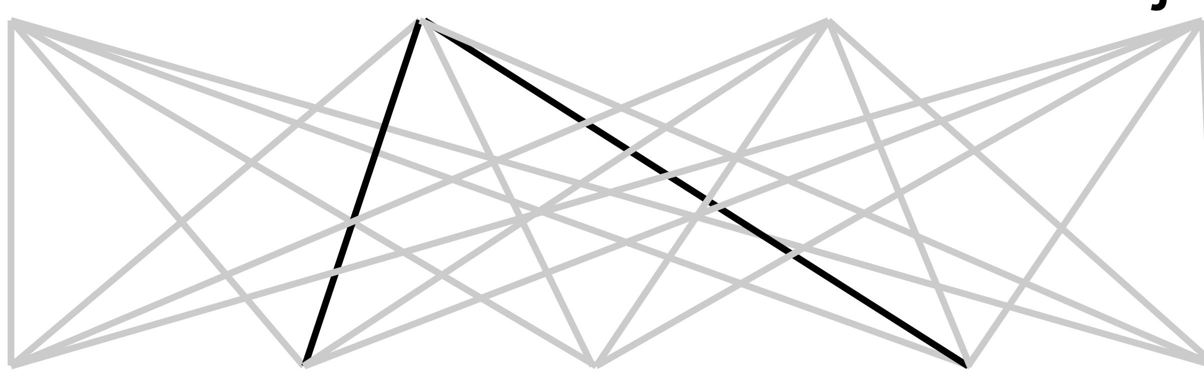
Job Shop Scheduling

Constraints:

Precedence: between two tasks of the same job



Resource: Machines execute at most one job at a time

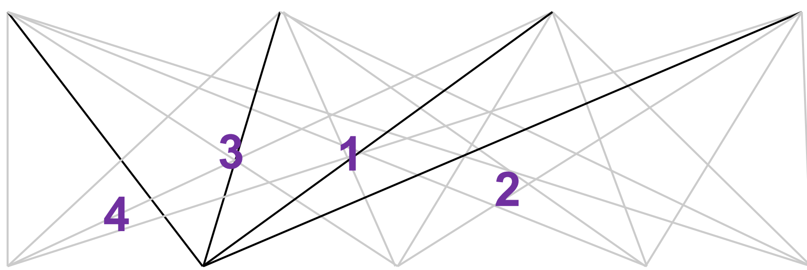


$$[start_{2,2} \,..\, end_{2,2}] \cap [start_{4,2} \,..\, end_{4,2}] = \emptyset$$

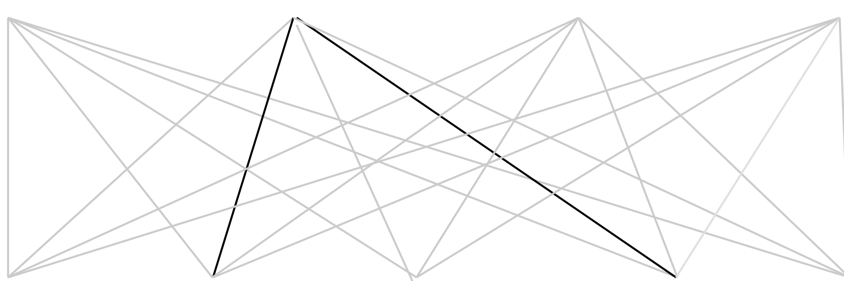
Job Shop Scheduling

Constraints:

Precedence:



Resource:



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

Encoding:

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

Not convex

$t_{2,2} + d_{2,2} \leq t_{4,2}$

$\vee$

$t_{4,2} + d_{4,2} \leq t_{2,2}$

Job Shop Scheduling

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

$max = 8$

Solution

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

Encoding

$$
\begin{aligned}
&(t_{1,1} \geq 0) \wedge (t_{1,2} \geq t_{1,1} + 2) \wedge (t_{1,2} + 1 \leq 8) \wedge \\
&(t_{2,1} \geq 0) \wedge (t_{2,2} \geq t_{2,1} + 3) \wedge (t_{2,2} + 1 \leq 8) \wedge \\
&(t_{3,1} \geq 0) \wedge (t_{3,2} \geq t_{3,1} + 2) \wedge (t_{3,2} + 3 \leq 8) \wedge \\
&((t_{1,1} \geq t_{2,1} + 3) \vee (t_{2,1} \geq t_{1,1} + 2)) \wedge \\
&((t_{1,1} \geq t_{3,1} + 2) \vee (t_{3,1} \geq t_{1,1} + 2)) \wedge \\
&((t_{2,1} \geq t_{3,1} + 2) \vee (t_{3,1} \geq t_{2,1} + 3)) \wedge \\
&((t_{1,2} \geq t_{2,2} + 1) \vee (t_{2,2} \geq t_{1,2} + 1)) \wedge \\
&((t_{1,2} \geq t_{3,2} + 3) \vee (t_{3,2} \geq t_{1,2} + 1)) \wedge \\
&((t_{2,2} \geq t_{3,2} + 3) \vee (t_{3,2} \geq t_{2,2} + 1))
\end{aligned}
$$

Job Shop Scheduling

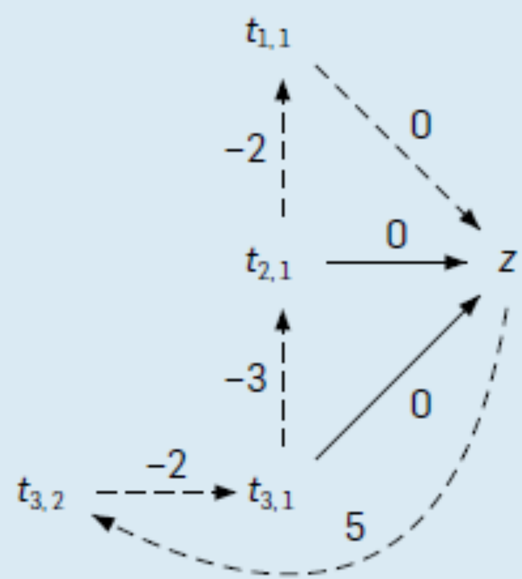
$$
\begin{array}{l}
(t_{1,1} \geq 0) \wedge (t_{1,2} \geq t_{1,1} + 2) \wedge (t_{1,2} + 1 \leq 8) \wedge \\
(t_{2,1} \geq 0) \wedge (t_{2,2} \geq t_{2,1} + 3) \wedge (t_{2,2} + 1 \leq 8) \wedge \\
(t_{3,1} \geq 0) \wedge (t_{3,2} \geq t_{3,1} + 2) \wedge (t_{3,2} + 3 \leq 8) \wedge \\
((t_{1,1} \geq t_{2,1} + 3) \vee (t_{2,1} \geq t_{1,1} + 2)) \wedge \\
((t_{1,1} \geq t_{3,1} + 2) \vee (t_{3,1} \geq t_{1,1} + 2)) \wedge \\
((t_{2,1} \geq t_{3,1} + 2) \vee (t_{3,1} \geq t_{2,1} + 3)) \wedge \\
((t_{1,2} \geq t_{2,2} + 1) \vee (t_{2,2} \geq t_{1,2} + 1)) \wedge \\
((t_{1,2} \geq t_{3,2} + 3) \vee (t_{3,2} \geq t_{1,2} + 1)) \wedge \\
((t_{2,2} \geq t_{3,2} + 3) \vee (t_{3,2} \geq t_{2,2} + 1))
\end{array}
$$

case split

case split

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

$$
\begin{array}{rcrcr}
z & - & t_{1,1} & \leq & 0 \\
z & - & t_{2,1} & \leq & 0 \\
z & - & t_{3,1} & \leq & 0 \\
t_{3,2} & - & z & \leq & 5 \\
t_{3,1} & - & t_{3,2} & \leq & -2 \\
t_{2,1} & - & t_{3,1} & \leq & -3 \\
t_{1,1} & - & t_{2,1} & \leq & -2
\end{array}
$$



$$z - z = 5 - 2 - 3 - 2 = -2 < 0$$

THEORIES

Theories

[Uninterpreted functions]

Microsoft Research

z3

Is this formula satisfiable? **Ask z3!**

```
(declare-sort () A)
(declare-fun f (A) A))
(declare-const a A)
(assert (= a (f (f a))))
(assert (= a (f (f (f a)))))
(check-sat)
(get-model)
(echo "Adding contradiction")
(assert (not (= a (f a))))
(check-sat)
```

ask z3

home tutorial video permal

Theorie

Uninterpreted funct

[Arithmetic (linear)]()

Microsoft Research

z3py

Explore the Z3 API using Python

```
t11, t12, t21, t22, t31, t32 = Ints('t11 t12 t21 t22 t31 t32')

s = Solver()

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

s.add(Or(t11 >= t21 + 3, t21 >= t11 + 2))
s.add(Or(t11 >= t31 + 2, t31 >= t11 + 2))
s.add(Or(t21 >= t31 + 2, t31 >= t21 + 3))
s.add(Or(t21 >= t22 + 1, t22 >= t12 + 1))
s.add(Or(t12 >= t32 + 3, t32 >= t12 + 1))
s.add(Or(t22 >= t32 + 3, t32 >= t22 + 1))

print ">>", s.check()
print ">>", s.model()
```

▶ tutorial home permalink

'▶' shortcut: Alt+B

```
>> sat
>> [t31 = 0, t21 = 4, t22 = 7, t32 = 2, t12 = 5, t11 = 2]
```

samples about Z3Py - Python interface for the Z3 Theorem P

Theories

Uninterpreted functions

Arithmetic (linear)

Bit-vectors

Microsoft Research

z3py

Explore the Z3 API using Python

```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)
```

▶ tutorial home permalink

'▶' shortcut: Alt+B

```
proved
buggy version...
counterexample
[x = 0]
```

Theories

Uninterpreted functions

Arithmetic (linear)

Bit-vectors

[Algebraic data-types]

Microsoft Research

z3py

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

print ">>", simplify(cdr(l1))

print ">>", simplify(car(l1))

print ">>", simplify(l1 == nil)


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

▶ tutorial home permalink

'▶' shortcut: Alt+B

Theories

Uninterpreted functions

Arithmetic (linear)

Bit-vectors

Algebraic data-types

[Arrays]()

```
; supported in Z3.
; This includes Combinatory Array Logic (de Moura &
;
(define-sort A () (Array Int Int))
(declare-fun x () Int)
(declare-fun y () Int)
(declare-fun z () Int)
(declare-fun a1 () A)
(declare-fun a2 () A)
(declare-fun a3 () A)
(push) ; illustrate select-store
(assert (= (select a1 x) x))
(assert (= (store a1 x y) a1))
(check-sat)
(get-model)
(assert (not (= x y)))
(check-sat)
(pop)
(define-fun all1_array () A ((as const A) 1))
(simplify (select all1_array x))
(define-sort IntSet () (Array Int Bool))
```

ask z3 | home | tutorial | video | permalink

```
sat
(model
  (define-fun y () Int
    1)
  (define-fun a1 () (Array Int Int)
    (_ as-array k!0))
  (define-fun x () Int
    1)
  (define-fun k!0 ((x!1 Int)) Int
    (ite (= x!1 1) 1
```

Theories

Uninterpreted functions

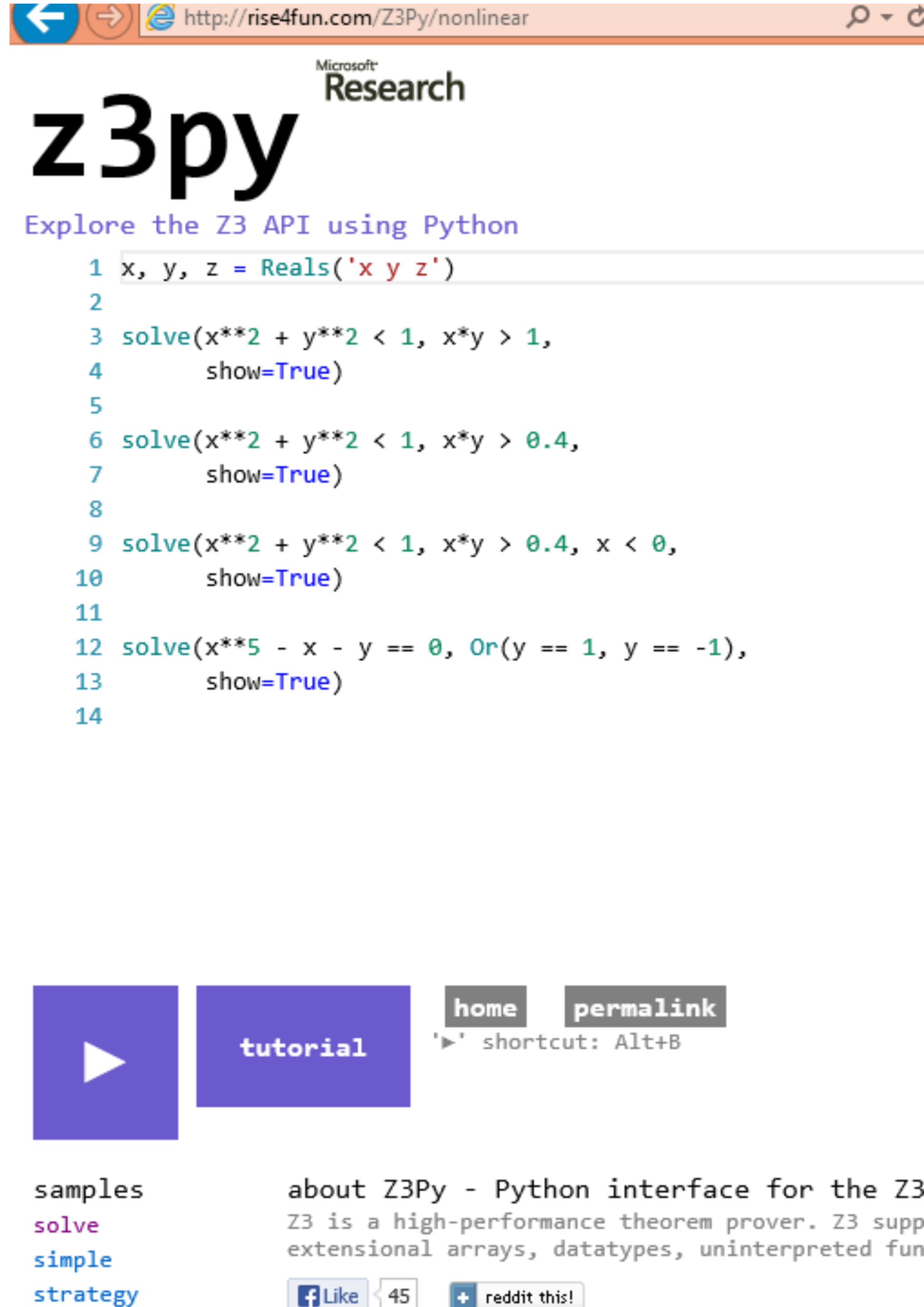
Arithmetic (linear)

Bit-vectors

Algebraic data-types

Arrays

[Polynomial Arithmetic]



QUANTIFIERS

Equality-Matching

■&p↓(∀…) @∧& $a=g(b,b)$@∧&$b=c$@∧&$f(a)\neq c$@∧&p↓(∀x …) → $f(g(c,b)$
∀x $f(g(c$

$g(c,x)$ matches $g(b,b)$
with substitution $[x\mapsto b]$
modulo $b=c$

[de Moura, B. CADE 2007]

Quantifier Elimination

```

(define-fun stamp () Bool
  (forall((x Int))
   (=>
     (>= x 8)
     (exists ((u Int) (v Int))
       (and (>= u 0) (>= v 0) (= x (+ (* 3 u) (* 5 v))))))))

(simplify stamp)

(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]

Superposition

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

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 if x > 100

mc(x) = mc(mc(x+11)) if x ≤ 100

assert (mc(x) ≥ 91)

$$\forall X.\ X > 100 \rightarrow \mathrm{mc}(X, X-10)$$

$$\forall X, Y, R.\ X \leq 100 \wedge \mathrm{mc}(X+11, Y) \wedge \mathrm{mc}(Y, R) \rightarrow \mathrm{mc}(X, R)$$

$$\forall X, R.\ \mathrm{mc}(X, R) \wedge X \leq 101 \rightarrow R = 91$$

***Solver finds solution for* mc**

[Hoder, B. SAT 2012]

MODELS, PROOFS, CORES & SIMPLIFICATION

Models

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
z3

```
(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)
```

Logical Formula

ask z3

*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))"))
```

Sat/Model

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

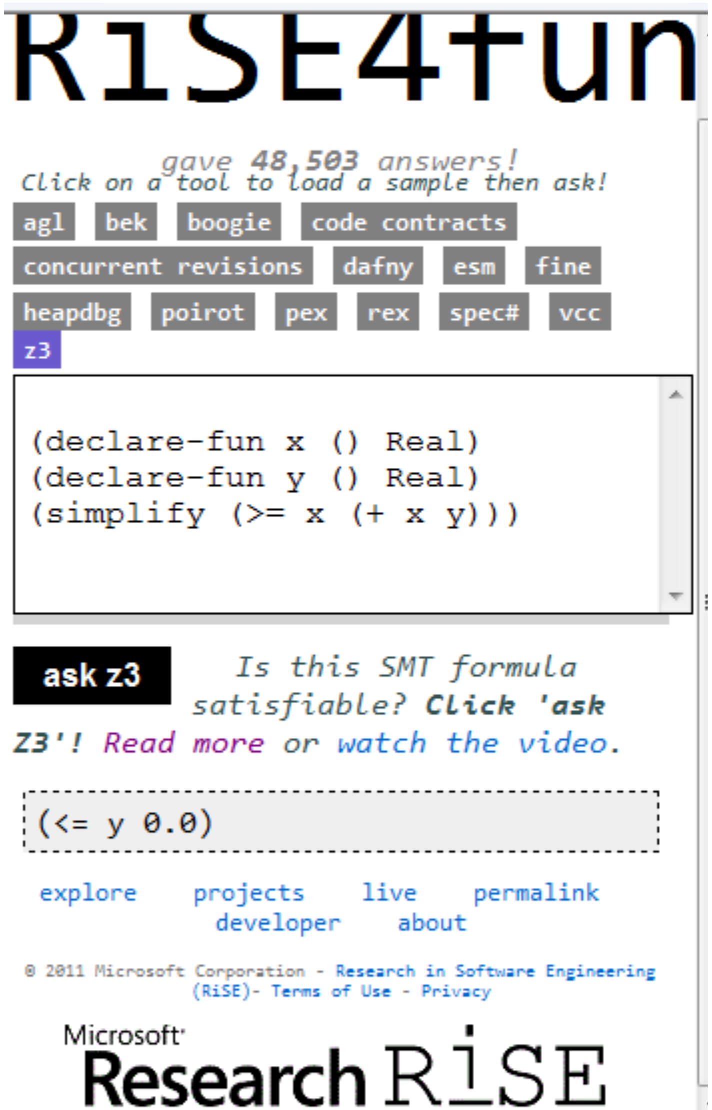
Unsat/Proof

```
proof.smt2 PROOF_MODE=2

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

erted (= (- (* x 3) (* y 3)) (- z u))]
ns
onotonicity
[trans
   [monotonicity
      [rewrite (= (* x 3) (* 3 x))]
      [rewrite (= (* y 3) (* 3 y))]
      (= (- (* x 3) (* y 3)) (- (* 3 x) (* 3 y)))]
   [rewrite (= (- (* 3 x) (* 3 y)) (+ (* 3 x) (* -3 y)))]
   (= (- (* x 3) (* y 3)) (+ (* 3 x) (* -3 y)))]
[rewrite (= (- z u) (+ z (* -1 u)))]
(iff (= (- (* x 3) (* y 3)) (- z u))
     (= (+ (* 3 x) (* -3 y)) (+ z (* -1 u))))]
 [rewrite
  (iff (= (+ (* 3 x) (* -3 y)) (+ z (* -1 u)))
       (= (+ (* 3 x) (+ (* -3 y) (+ (* -1 z) u))) 0))]
 (iff (= (- (* x 3) (* y 3)) (- z u))
      (= (+ (* 3 x) (+ (* -3 y) (+ (* -1 z) u))) 0))]
 (= (+ (* 3 x) (+ (* -3 y) (+ (* -1 z) u))) 0)]
[rewrite
 (iff (= (+ (* 3 x) (+ (* -3 y) (+ (* -1 z) u))) 0)
      (not (or (not ((<= (+ (* 3 x) (+ (* -3 y) (+ (* -1 z) u))) 0))
               (not ((>= (+ (* 3 x) (+ (* -3 y) (+ (* -1 z) u))) 0))))))]
(not (or (not ((<= (+ (* 3 x) (+ (* -3 y) (+ (* -1 z) u))) 0))
         (not ((>= (+ (* 3 x) (+ (* -3 y) (+ (* -1 z) u))) 0))))]
((<= (+ (* 3 x) (+ (* -3 y) (+ (* -1 z) u))) 0)]
[mp
 [asserted ((> x y)]
 [rewrite (iff ((> x y) (not ((<= (+ x (* -1 y)) 0)))]
 (not ((<= (+ x (* -1 y)) 0))]
[mp [asserted ((< z 3)] [rewrite (iff ((< z 3) (not ((>= z 3)))] (not ((>= z 3))]
false]
```

Simplification



Logical Formula

Simplify

Cores

```
(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)
```

Logical Formula

ask z3 *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)))
```

Unsat. Core

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:

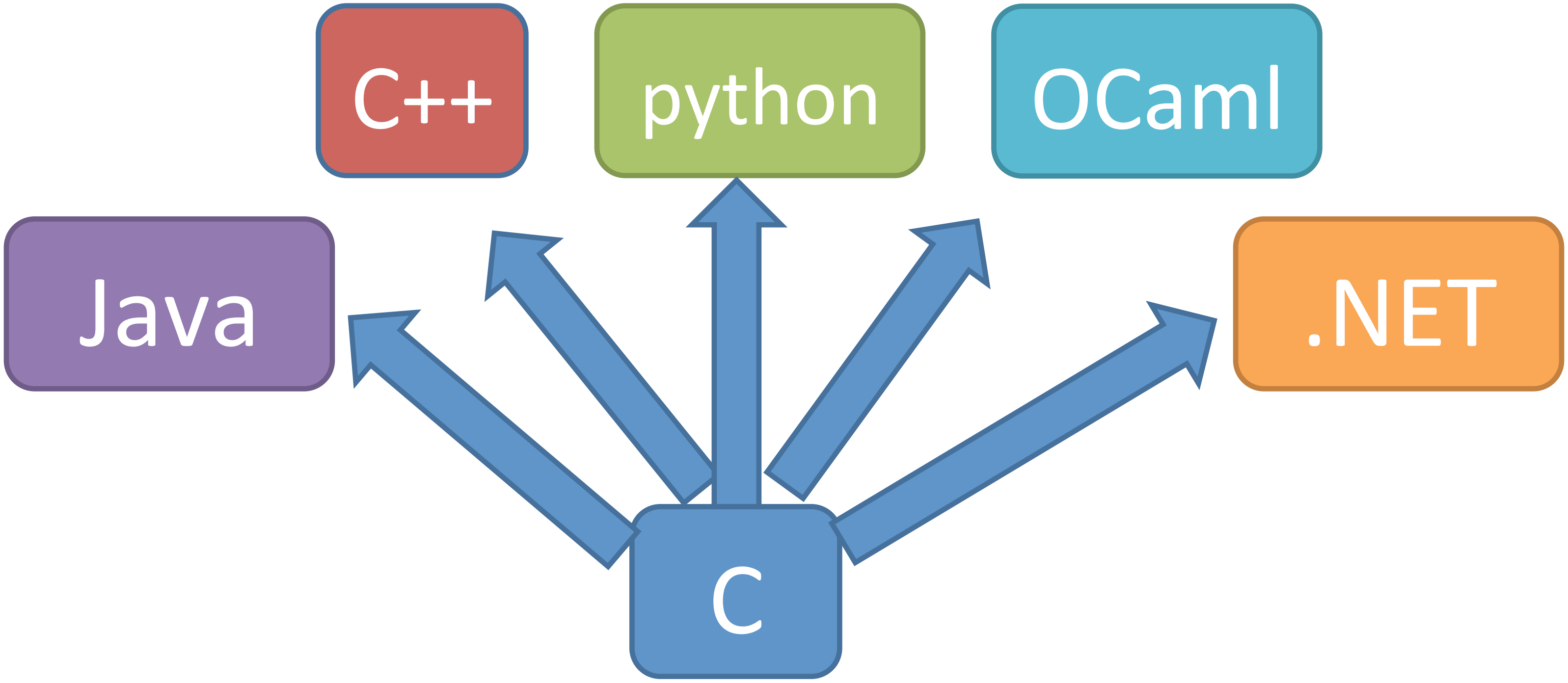
- (then t s)
- (par-then t s) applies t to the input goal and s to every subgoal produced by t in parallel.
- (or-else t s)
- (par-or t s) applies t and s in parallel until one of them succeed.
- (repeat t)
- (repeat t n)
- (try-for t ms)
- (using-params t params) Apply the given tactic using the given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

```
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