Software Verification with Satisfiability Modulo Theories

Nikolaj Bjørner
Microsoft Research
SSFT 2014, Menlo Park

Contents

A primer on SMT with Z3

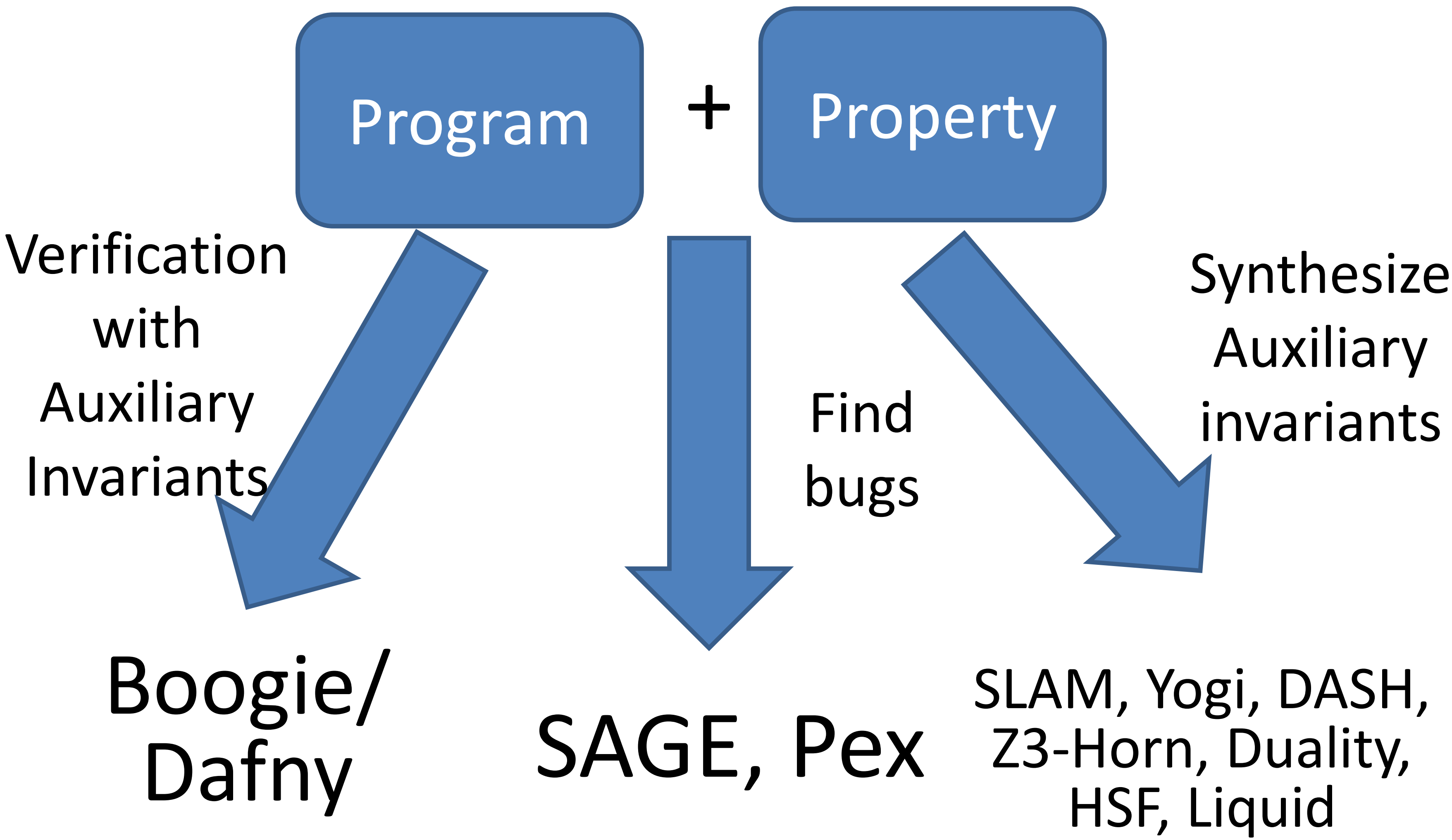
SMT & Verification by Assertion Checking

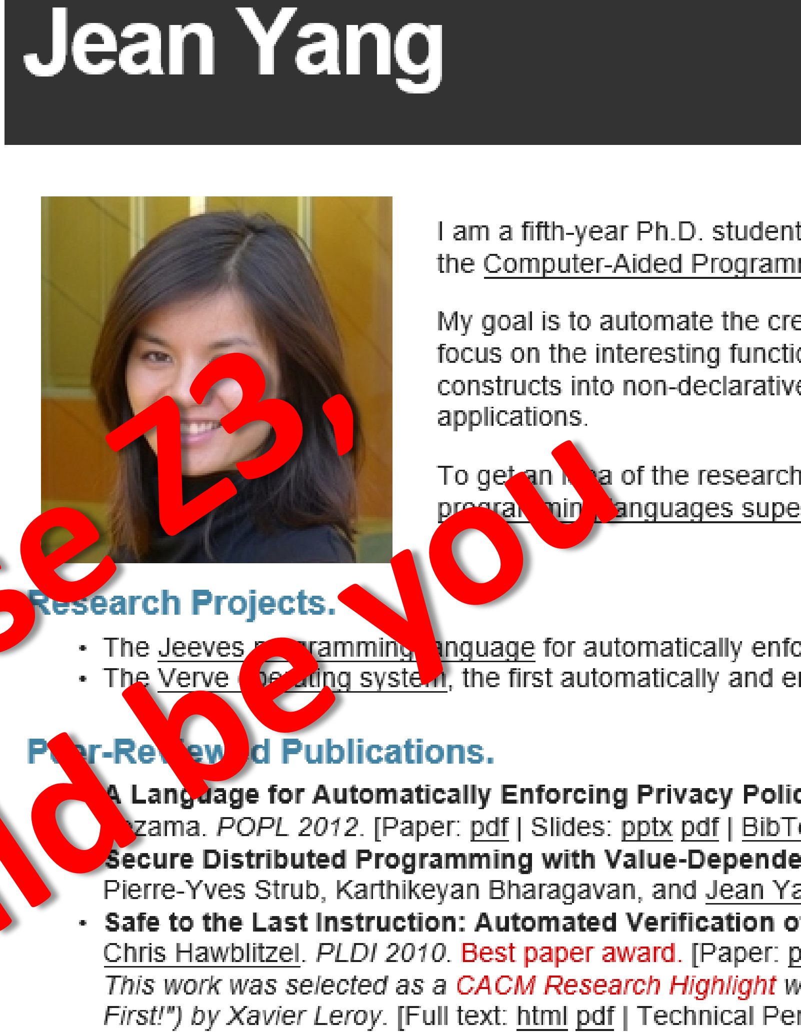
- Boogie GC, Quantifiers, Theories

SMT & Verification by Assertion Inference

- Symbolic Software Model Checking,
Horn Clauses

SMT & Verification





If you use Z3,
This could be you

Z3 – Backed by Proof Plumbers

Leonardo de Moura, Nikolaj Bjørner, Christoph Wintersteiger

A primer on SMT with Z3

Material

Satisfiability Modulo Theories: Introduction

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

Nikolaj Bjørner
Microsoft Resear
One Microsoft W
Redmond, WA 980
nbjorner@microsoft

ABSTRACT

Constraint satisfaction problems arise in many diverse areas including software and hardware verification, type inference, static program analysis, test-case generation, scheduling, planning and graph problems. These areas share a common trait, they include a core component using logical formulas for describing states and transformations between them. The most well-known constraint satisfaction problem is *propositional satisfiability*, SAT, where the goal is to decide whether a formula over Boolean variables, formed using logical connectives can be made *true* by choosing *true*/*false* values for its variables. Some problems are more naturally described using richer languages, such as arithmetic. A supporting *theory* (of arithmetic) is then required to capture the meaning of these formulas. Solvers for such formulations are commonly called *Satisfiability Modulo Theories* (SMT) solvers.

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 im interchange format for ben and the classification of ben depending on which theori growing number of applicati marks in the SMT-LIB form SMT solvers.

There is a relatively long t select and specialized contex proving systems such as ACI decision procedures to discha interactive proofs. SMT sol long time in the context of pr *static checking* [21], where ve checking. Recent progress 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

September 2011

Nikolaj Bjørner Leonardo de Moura

Applications of SMT solvers to Program Verification

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

May 16, 2014

Springer

Some Microsoft Tools based on Z3

Dafny

Program safely. With Dafny.

http://research.microsoft.com/dafny

Program Verification

VCC

HAVOC

Auditing

BOOGIE

Type Safety

SLAyer

SLAM

Over-Approximation

TERMINATOR

Yogi

Under-Approximation

SAGE

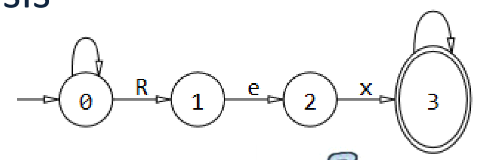
Pex

Spec Explorer

Testing

M3

Analysis



Synthesis

FORMULA

Modeling Foundations.

http://rise4fun.com/

rise4fun

your tool here

1214986 programs analyzed

rise4fun

a community of software engineering tools

all tutorial automata concurrency design encoders infrastructure languages security synthesis testing verification

new!

f*
A verification tool for higher-order stateful programs

fast
A domain specific language for writing and analyzing tree manipulating programs

iz3
Efficient Interpolating Theorem Prover

microsoft

agl
Automatic Graph Layout

bek
A domain specific language for writing and analyzing common string functions

bex
A domain specific language for writing and analyzing string encoders and decoders

boogie
Intermediate Verification Language

chalice
A language and program verifier for reasoning about concurrent programs.

code contracts
Language agnostic modular program verification and repair with abstract interpretation.

counterdog
Theorem-prover for Counterfactual Datalog

dafny
A language and program verifier for functional correctness

dkal
Distributed Knowledge Authorization Language

esm
Empirical Software Engineering and Measurement Group

fast
A domain specific language for writing and analyzing tree manipulating programs

formula
Formal Modeling Using Logic Programming and Analysis

formula2
Formal Modeling Using Logic Programming and Analysis

try f#
Programming language combining functional, object-oriented and scripting programming.

f*
A verification tool for higher-order stateful programs

heapdbg
Runtime heap abstraction

iz3
Efficient Interpolating Theorem Prover

koka
A function-oriented language with effect inference

pex
Automatic test generation using Dynamic Symbolic Execution for .NET

quickcode
Programming-by-example technology for learning string transformation programs

concurrent revisions
Parallel and Concurrent Programming with Snapshots

rex
Regular Expression Exploration

seal
Side-Effects AnalysiS

slayer
Automatic formal verification for programs with heaps.

spec#
A formal language for API contracts

touchdevelop
Program your phone on your phone.

vcc
A Verifier for Concurrent C

visual c++
The Visual C++ compiler

z3
Efficient Theorem Prover

z34bio
SMT-based Analysis of Biological Computation

z3py
Python interface for the Z3 Theorem Prover

albert-ludwigs-universität freiburg

gravy
The Gradual Verifier

joogie
Infeasible Code Detection for Java

eth zurich - chair of software engineering

autoproof
a Program Verifier for Eiffel

boogaloo
the Boogie Interpreter

javanni
a Verifier for JavaScript

qfis
a Program Verifier for Integer Sequences

ku leuven

verifast
Verifier for C and Java Programs

multicore programming group, imperial college london

gpuverify-cuda
A verifier for CUDA/OpenCL kernels

gpuverify-opencl
A verifier for CUDA/OpenCL kernels

university of utah and imdea software institute

smack
Verifier for C/C++ Programs

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

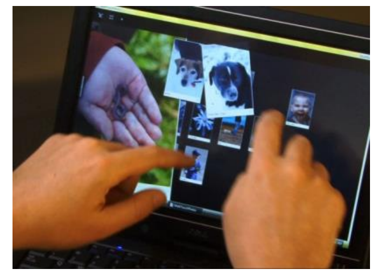
Uninterpreted Functions

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

SMT SOLVING IN A NUTSHELL

Job Shop Scheduling

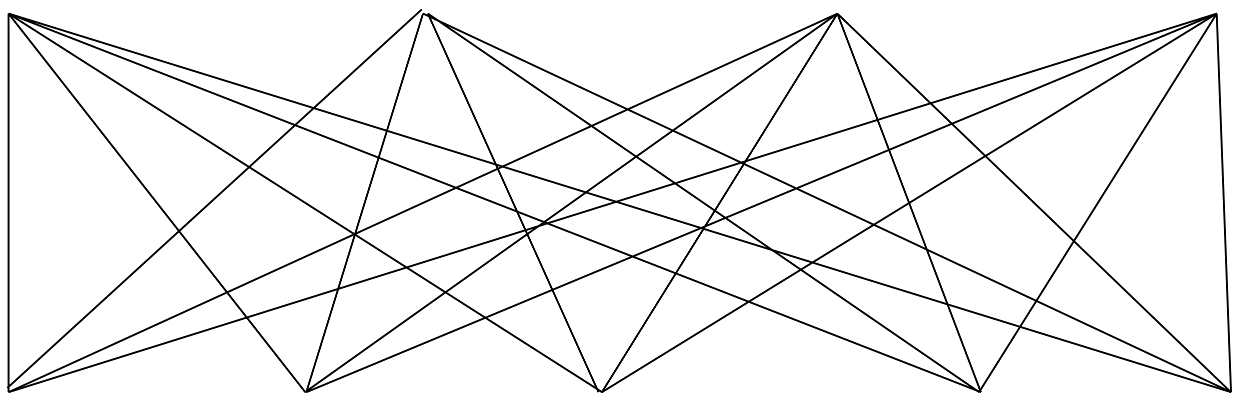
Job Shop Scheduling



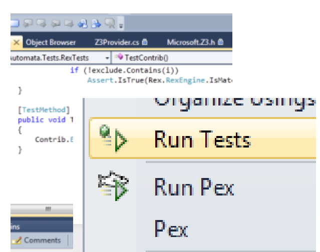
Machines

Tasks

Jobs



P = NP?

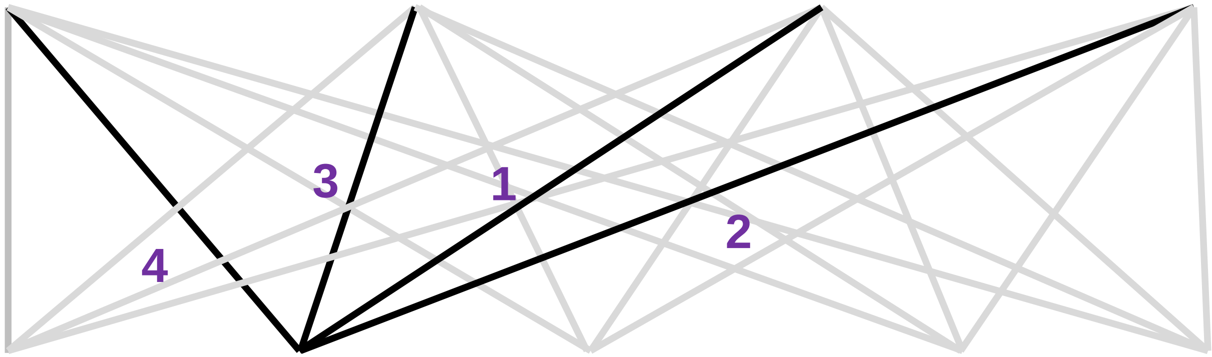


$\zeta(s) = 0 \Rightarrow s = \frac{1}{2} + ir$

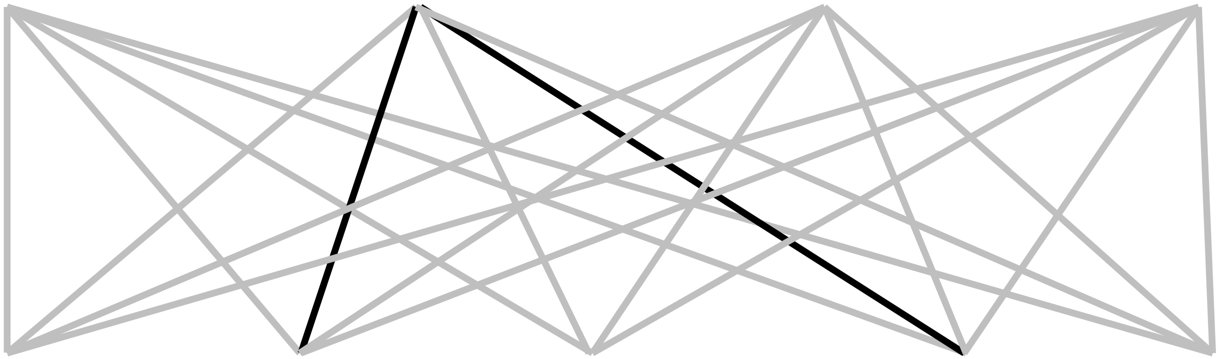
Job Shop Scheduling

Constraints:

Precedence: between two tasks of the same job



Resource: Machines execute at most one job at a time

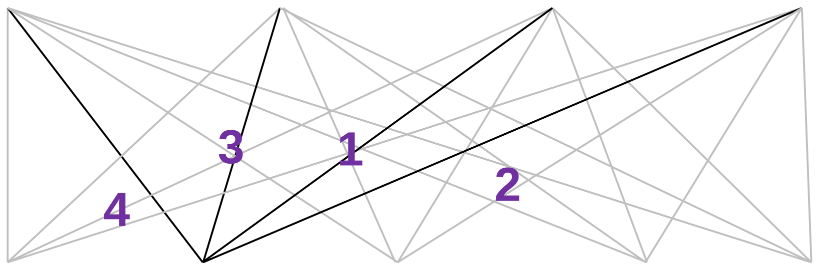


$$[start_{2,2}..end_{2,2}] \cap [start_{4,2}..end_{4,2}] = \emptyset$$

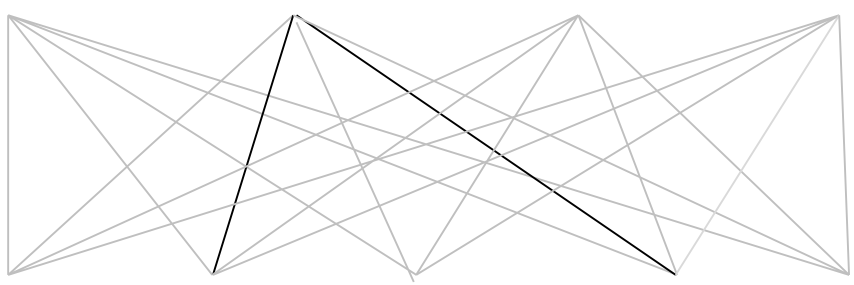
Job Shop Scheduling

Constraints:

Precedence:



Resource:



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

Encoding:

$t_{2,3}$ - 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

$$t_{2,2} + d_{2,2} \leq t_{4,2}$$
$$\vee$$
$$t_{4,2} + \mathrm{d}_{4,2} \leq t_{2,2}$$

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

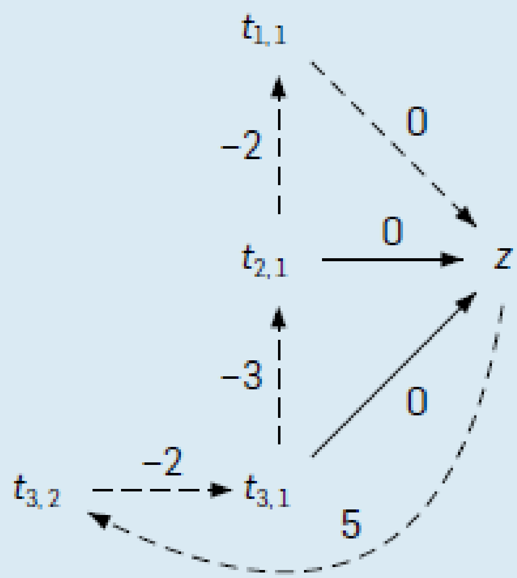
$$(t_{1,1} \geq 0) \wedge (t_{1,2} \geq t_{1,1} + 2) \wedge (t_{1,2} + 1 \leq 8) \wedge$$
$$(t_{2,1} \geq 0) \wedge (t_{2,2} \geq t_{2,1} + 3) \wedge (t_{2,2} + 1 \leq 8) \wedge$$
$$(t_{3,1} \geq 0) \wedge (t_{3,2} \geq t_{3,1} + 2) \wedge (t_{3,2} + 3 \leq 8) \wedge$$
$$((t_{1,1} \geq t_{2,1} + 3) \vee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

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

case split

case split

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

z3

Microsoft Research

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]

z3py

Explore the Z3 API using 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]

z3py

Microsoft Research

Explore the Z3 API using 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

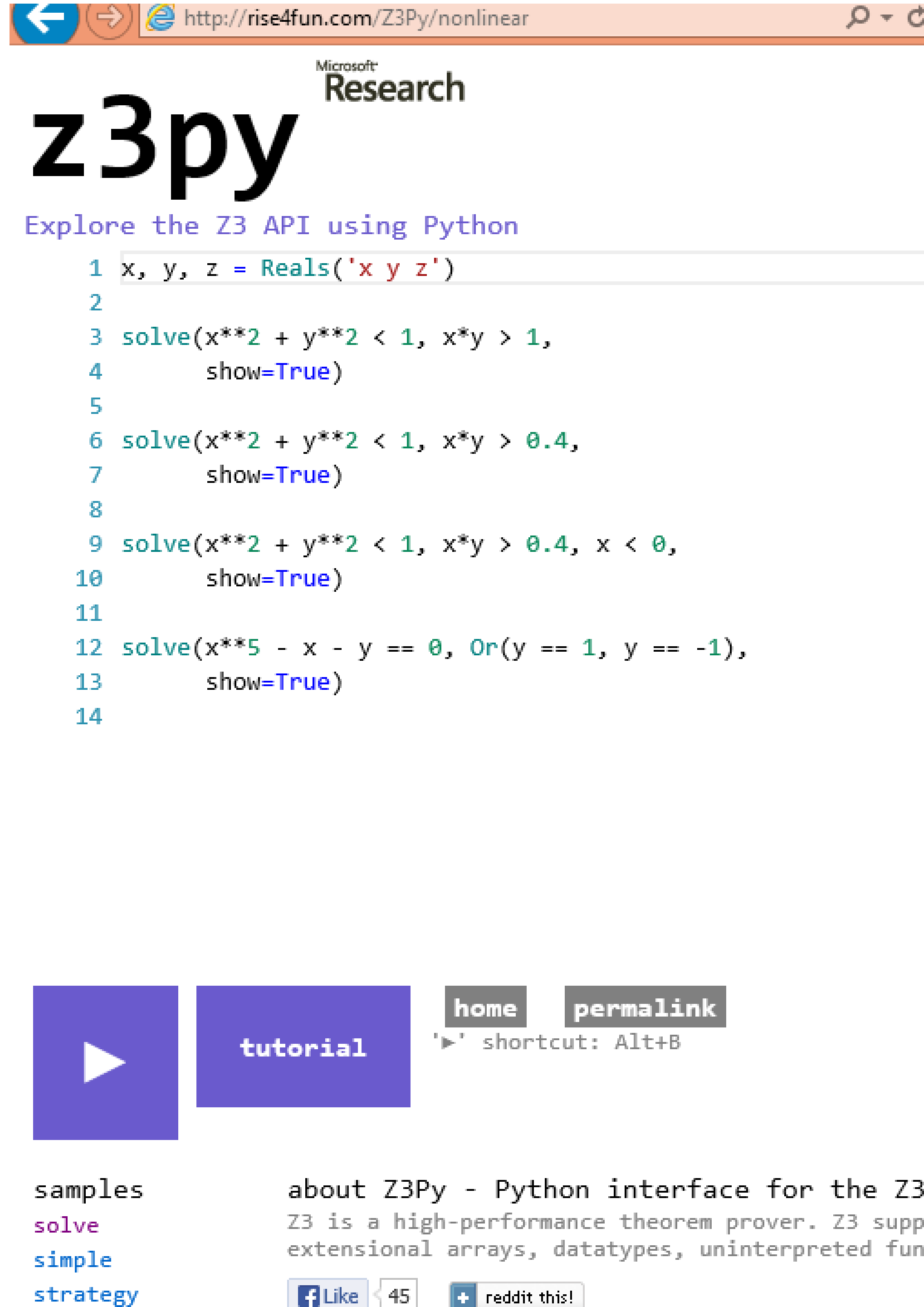
Arithmetic (linear)

Bit-vectors

Algebraic data-types

Arrays

Polynomial Arithmetic



QUANTIFIERS

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]

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

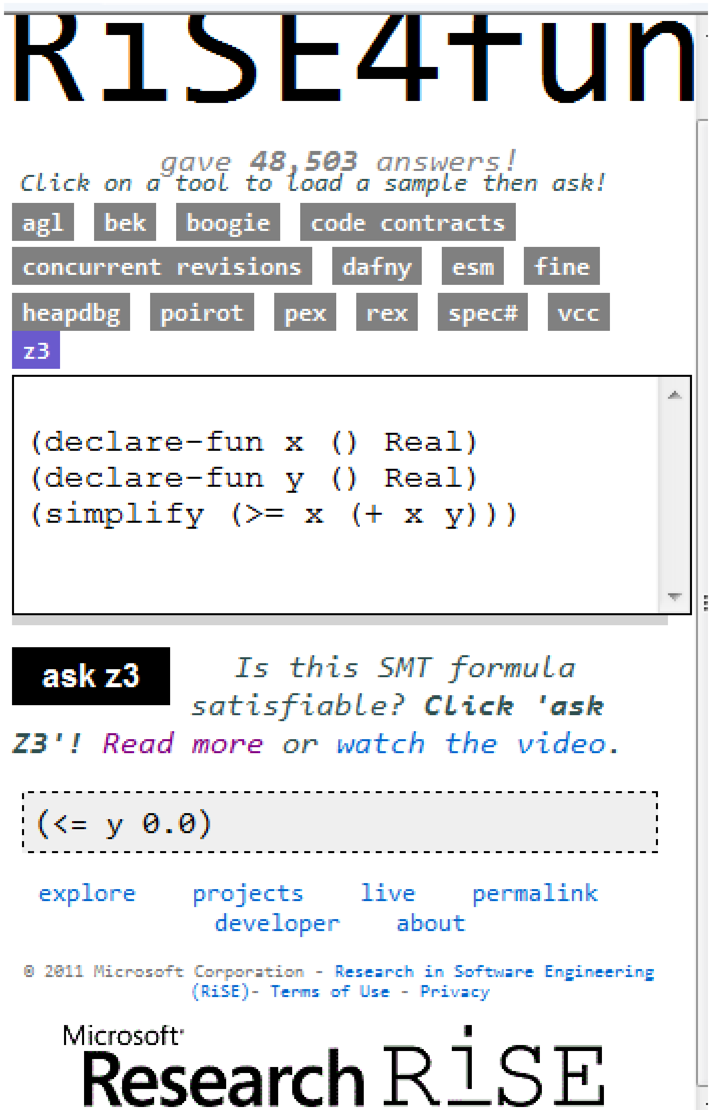
proof.smt2 PROOF_MODE=2

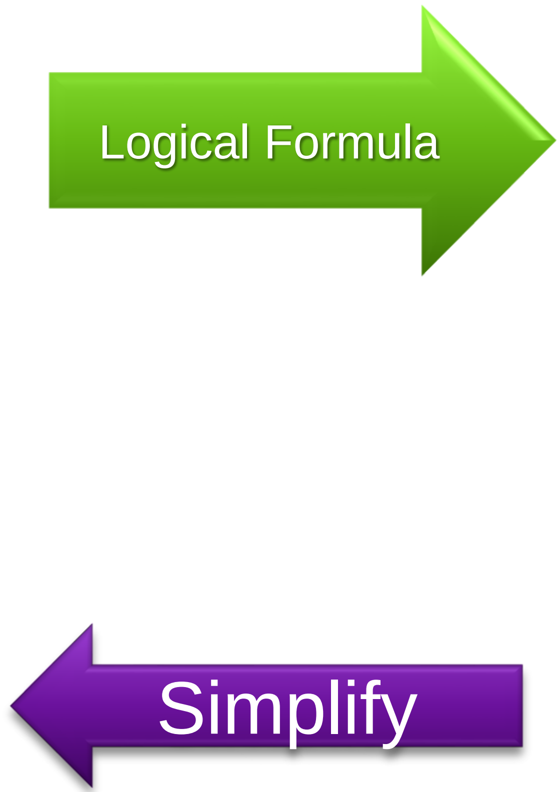
```
ted ((<= 0 u)] [rewrite (iff ((<= 0 u) (>= u 0))] (>= u 0)]
im

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]
```

Unsat/Proof

Simplification





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:

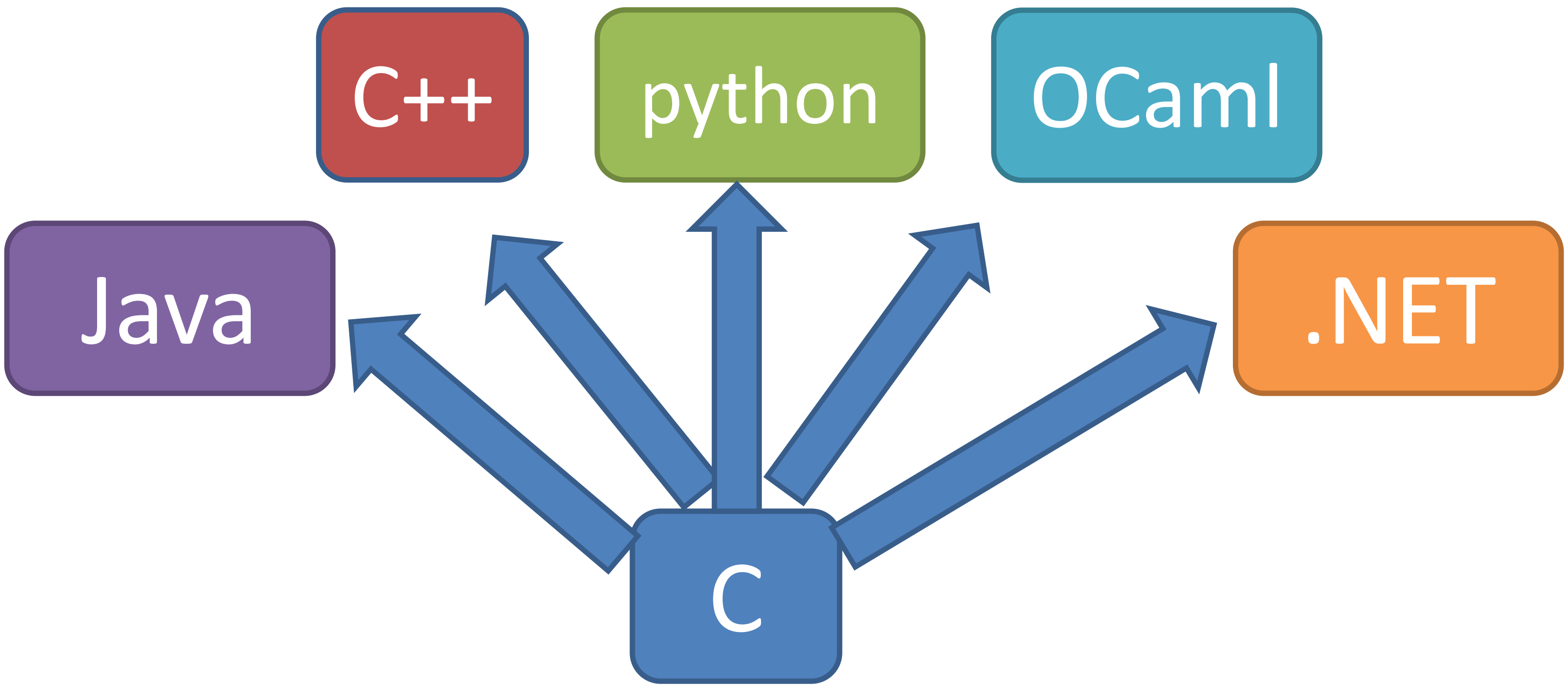
- (then t s)
- (par-then t s) applies t to the input goal and s to every subgoal produced by t in parallel.
- (or-else t s)
- (par-or t s) applies t and s in parallel until one of them succeed.
- (repeat t)
- (repeat t n)
- (try-for t ms)
- (using-params t params) Apply the given tactic using the given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