Engineering Theories with Z3

Nikolaj Bjørner
Microsoft Research
IWIL March 10th 2012

Not so Hidden Agenda

Blatant, Shameless Propaganda

Z3 – An Efficient SMT Solver

Tutorial style

Many techniques apply broadly to SMT solvers: Barcelogic, CVC, Ergo,Mathsat, OpenSMT, Yices, ..

Many tools already use techniques ….

.. But many more tools should really do it too.

Why Engineering Theories?

Support

Rich Theories *(and logics) with*
Efficient *Decision Procedures*

Auth | MSOL | Sequences | XDucers | Queues | ASP | DL

homomorphisms | Optimization | Orders | Objects | HOL | MultiSets | BAPA

Strings | Reg. Exprs. | NRA | NIA | Floats | f* | *

SAT | EUF | LRA | LIA | Arrays | Bit-Vectors | Alg. DT

We review three methods:

Theory Solver: Optimization,
Partial Orders

Reduction: Object Types

Saturation: HOL

Goal: Tools to make users happy & productive

http://icwww.epfl.ch/~piskac/softwa

MUNCH - Automated Reas...

The MUNCH Tool: automated reasoner for collections

This is the web page for the MUNCH tool.
Currently the following is available for download:

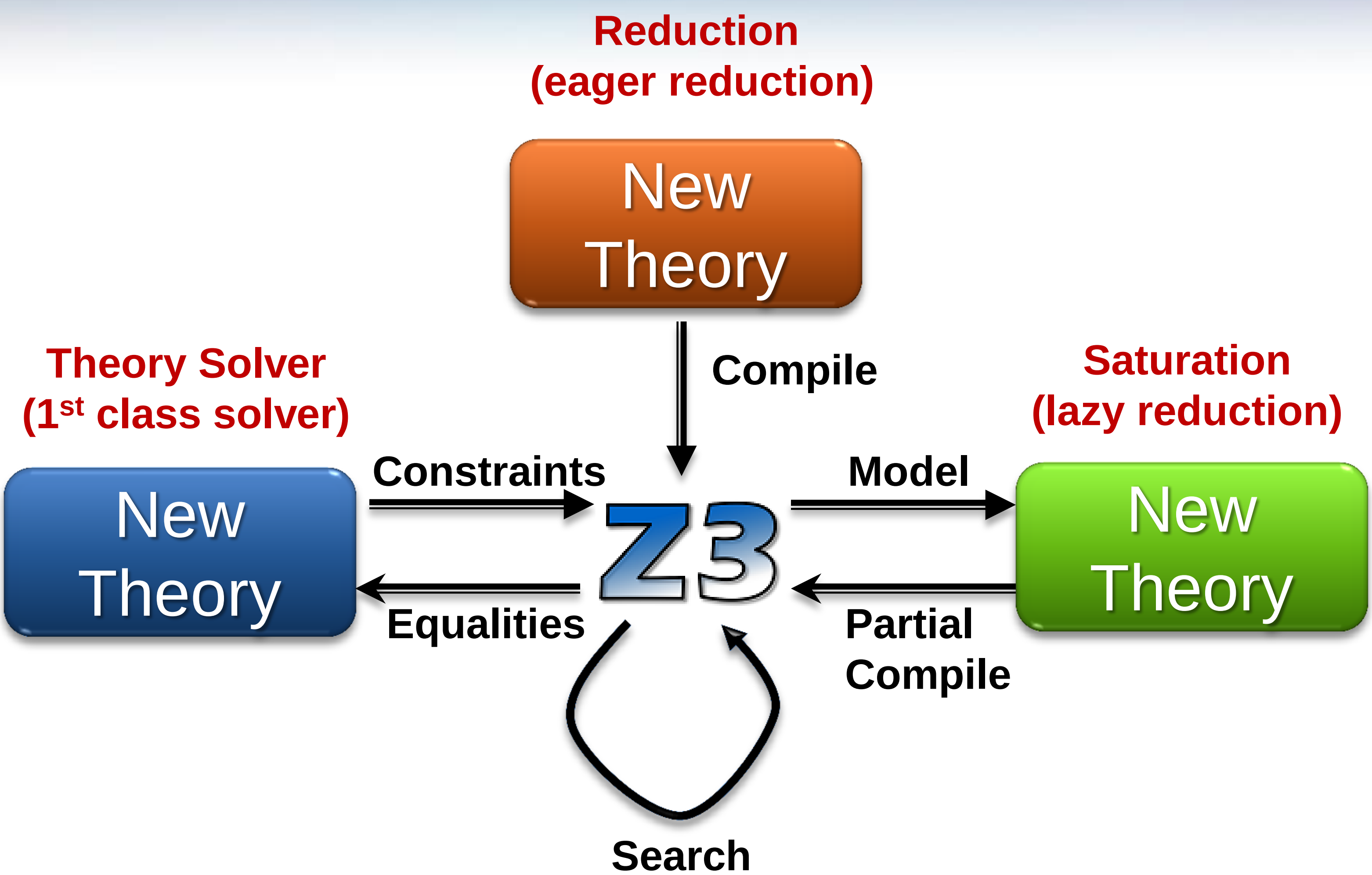
- **paper describing the tool**
- **implementation**
- **some examples and their output**

Examples are written in the separate file (**examples.txt**). The tool then parses this input into a language corresponding to the grammar described in the paper and in the file ASTMultisets.scala. MUNCH invokes z3 .

Playing with the MUNCH tool

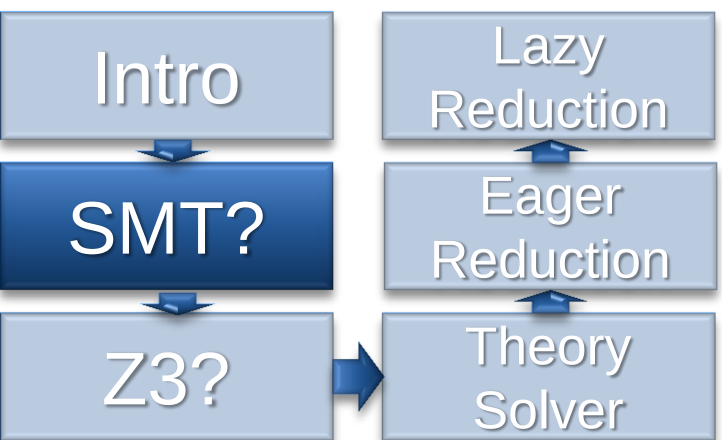
The MUNCH tool is written in **Scala** and for testing MUNCH you need to have Scala installed. To run MUNCH, on your machine, first download the source code and compile it

Overview of methods



SMT

What and Why



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

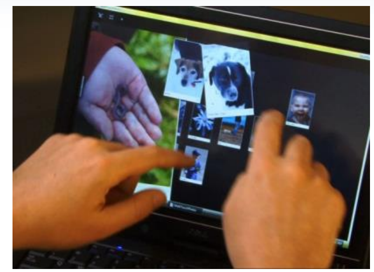
Arithmetic

Uninterpreted Functions

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

Job Shop Scheduling

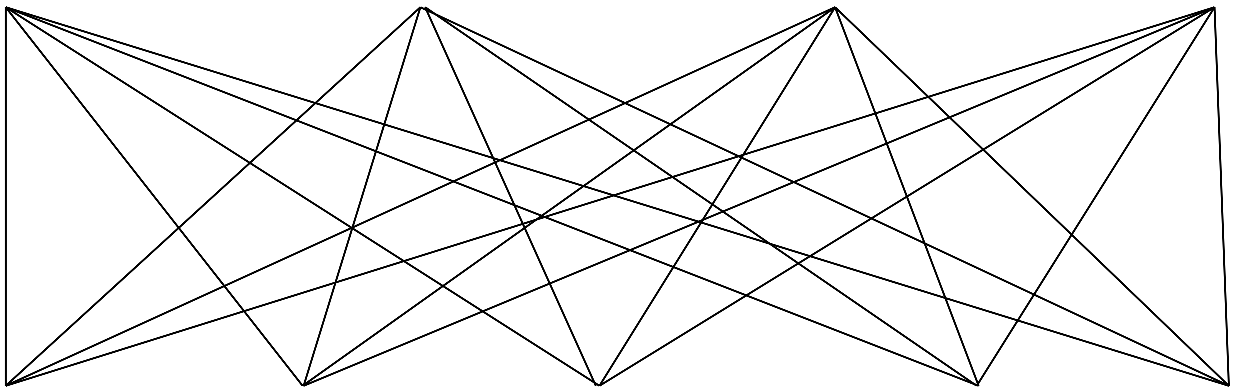




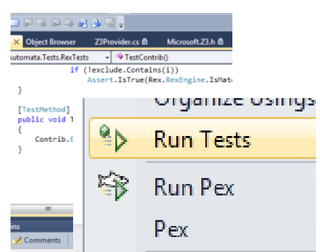

Machines

Tasks

Jobs



P = NP?

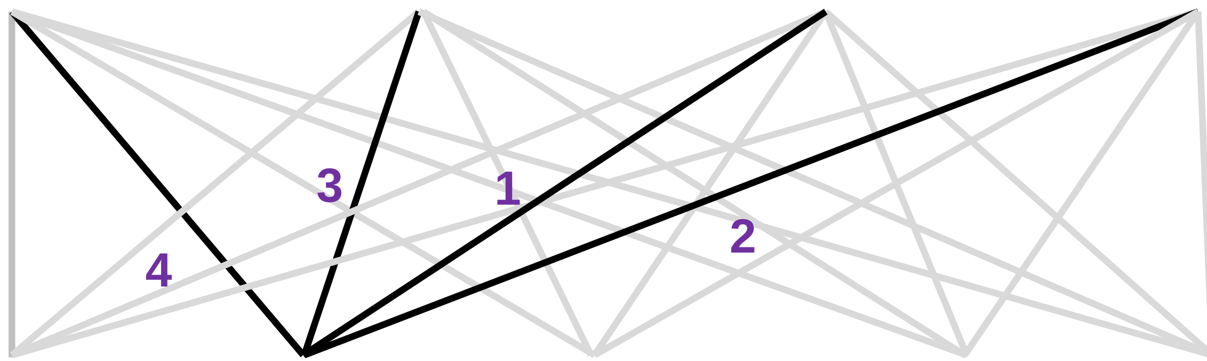



$$\zeta(s) = 0 \Rightarrow s = \frac{1}{2} + ir$$

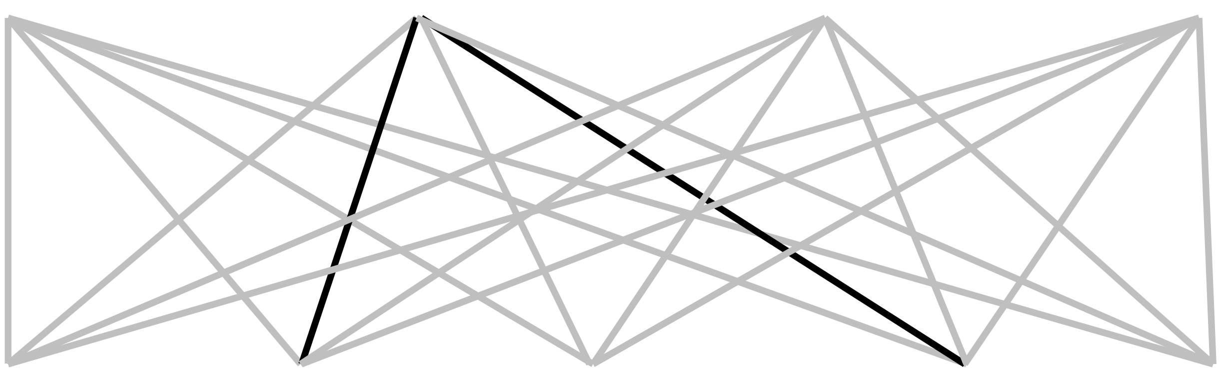
Job Shop Scheduling

Constraints:

Precedence: between two tasks of the same job



Resource: Machines execute at most one job at a time

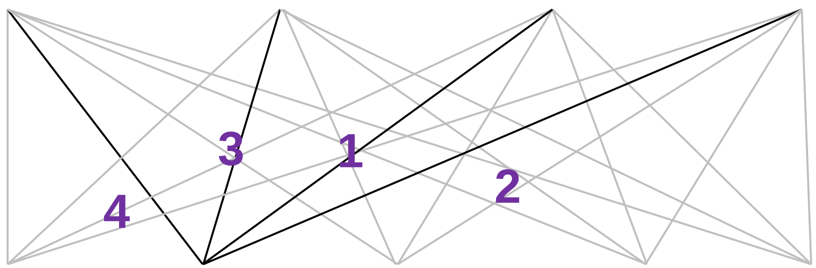


$$[start_{2,2}..end_{2,2}] \cap [start_{4,2}..end_{4,2}] = \emptyset$$

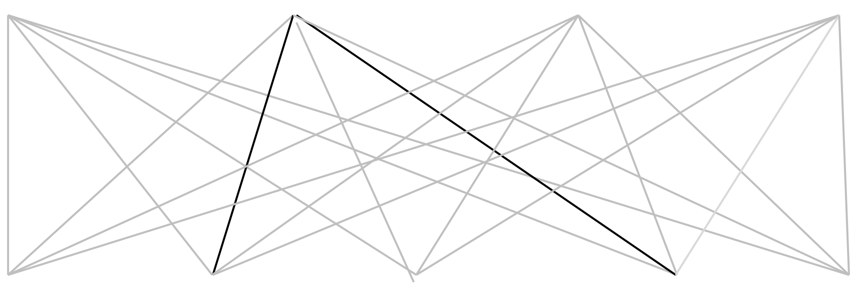
Job Shop Scheduling

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

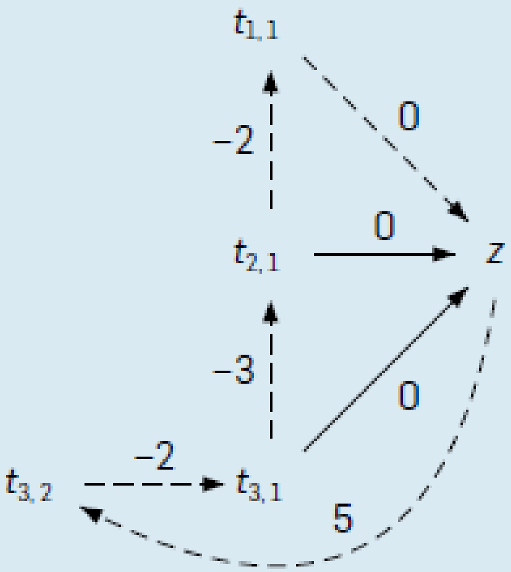
$$
\begin{aligned}
&(t_{1,1} \geq 0) \wedge (t_{1,2} \geq t_{1,1} + 2) \wedge (t_{1,2} + 1 \leq 8) \wedge \\
&(t_{2,1} \geq 0) \wedge (t_{2,2} \geq t_{2,1} + 3) \wedge (t_{2,2} + 1 \leq 8) \wedge \\
&(t_{3,1} \geq 0) \wedge (t_{3,2} \geq t_{3,1} + 2) \wedge (t_{3,2} + 3 \leq 8) \wedge \\
&((t_{1,1} \geq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

case split

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

case split

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



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

Symbolic Engines: SAT, FTP and SMT

- SAT: Propositional Satisfiability.

  $(\text{Tie} \lor \text{Shirt}) \land (\neg\text{Tie} \lor \neg\text{Shirt}) \land (\neg\text{Tie} \lor \text{Shirt})$

- FTP: First-order Theorem Proving.

  $\forall X,Y,Z\ [X*(Y*Z) = (X*Y)*Z]$
  $\forall X\ [X*inv(X) = e]\ \forall X\ [X*e = e]$

- SMT: Satisfiability Modulo background Theories
  $b + 2 = c \land A[3] \neq A[c-b+1]$

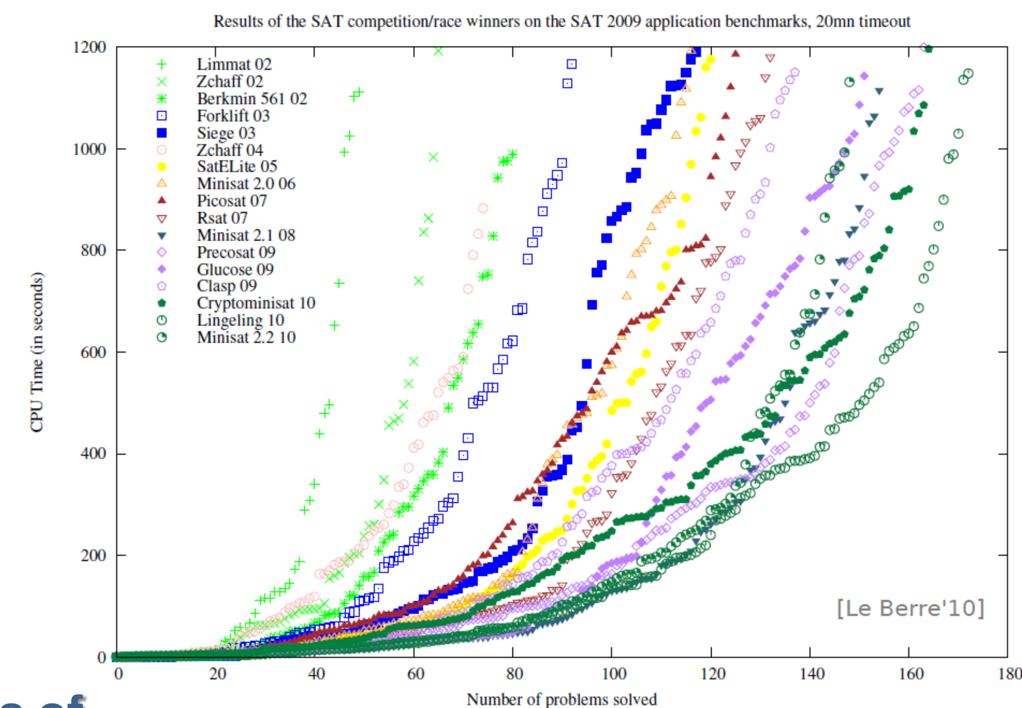
SAT - Milestones

year	Milestone
1960	Davis-Putnam procedure
1962	Davis-Logeman-Loveland
1984	Binary Decision Diagrams
1992	DIMACS SAT challenge
1994	SATO: clause indexing
1997	GRASP: conflict clause learning
1998	Search Restarts
2001	zChaff: 2-watch literal, VSIDS
2005	Preprocessing techniques
2007	Phase caching
2008	Cache optimized indexing
2009	In-processing, clause management
2010	Blocked clause elimination

Problems impossible 10 years ago are trivial today

Concept

2002 2010



Millions of variables from HW designs

Courtesy Daniel le Berre

Microsoft Research

FTP - Milestones

Year	Milestone	Who	Year	Milestone	Who
1930	Hebrand's theorem	Herbrand	1970	Completion and saturation procedures	many people and provers
1934	Sequent calculi	Gentzen	1970	Knuth-Bendix ordering	Knuth; Bendix
1934	Inverse method	Gentzen	1971	Selection function	Kowalski; Kuehner
1955	Semantic tableaux	Beth	1972	Built-in equational theories	Plotkin
1960	Herbrand-based theorem proving	Wang Hao	1972	Prolog	Colmerauer
1960	Ordered resolution	Davis; Putnam	1974	Saturation algorithms	Overbeek
1962	DLL	Davis; Logemann; Loveland	1975	Completeness of paramodulation	Brand
1963	First-order inverse method	Maslov	1975	AC-unification	Stickel
1965	Unification	J. Robinson	1976	Resolution as a decision procedure	Joyner
1965	First-order resolution	J. Robinson	1979	Basic paramodulation	Degtyarev
1965	Subsumption	J. Robinson	1980	Lexicographic path orderings	Kamin; Levy
1967	Orderings	Slagle	1985	Theory resolution	Stickel
1967	Demodulation or rewriting	Wos; G. Robinson; Carson; Shalla	1986	Definitional clause form transformation	Plaisted; Greenbaum
1968	Model elimination	Loveland	1988	Superposition	Zhang
1969	Paramodulation	G. Robinson; Wos	1988	Model construction	Zhang
			1989	Term indexing	Stickel; Overbeek
			1990	General theory of redundancy	Bachmair; Ganzinger
			1992	Basic superposition	Nieuwenhuis; Rubio
			1993	First instance-based methods	Billon; Plaisted
			1993	Discount saturation algorithm	Avenhaus; Denzinger
			1998	Finite model finding using SAT	McCune
			2000	First-order DPLL	Baumgartner
			2003	iProver method	Ganzinger; Korovin
			2008	Sine selection	Hoder

Some success stories:

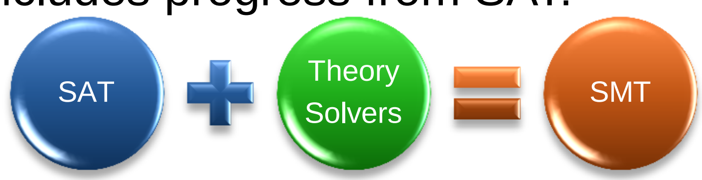
- Open Problems (of 25 years):
  XCB: $X \equiv ((X \equiv Y) \equiv (Z \equiv Y)) \equiv Z)$
  is a single axiom for equivalence
- Knowledge Ontologies
  GBs of formulas

Courtesy Andrei Voronkov, Manchester U

SMT - Milestones

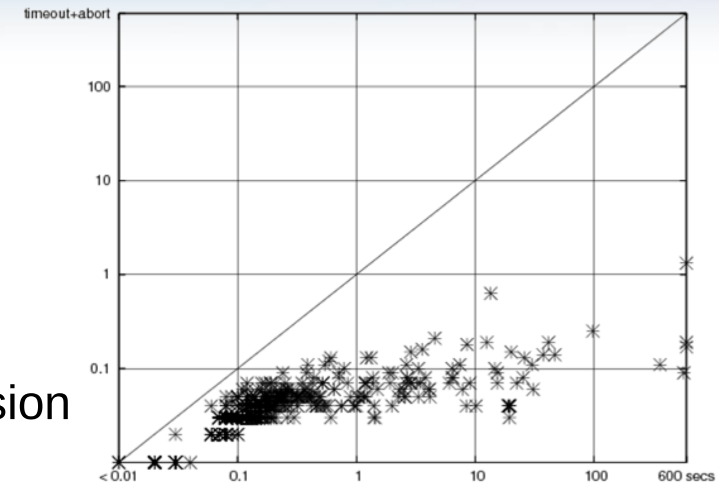
year	Milestone
1977	Efficient Equality Reasoning
1979	Theory Combination Foundations
1979	Arithmetic + Functions
1982	Combining Canonizing Solvers
1992-8	Systems: PVS, Simplify, STeP, SVC
2002	Theory Clause Learning
2005	SMT competition
2006	Efficient SAT + Simplex
2007	Efficient Equality Matching
2009	Combinatory Array Logic, …

Includes progress from SAT:



15KLOC + 285KLOC = Z3

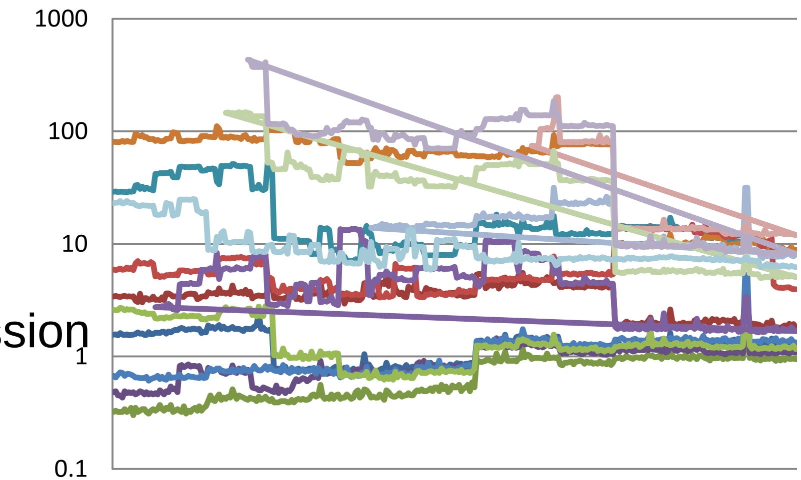
Z3 (of '07) Time On Boogie Regression



1sec

Simplify (of '01) time

Z3 Time On VCC Regression



Nov 08 March 09

Z3

An Efficient SMT Solver From Microsoft Research

Intro

SMT?

Z3?

Lazy Reduction

Eager Reduction

Theory Solver

By Leonardo de Moura, Nikolaj Bjørner, Christoph Wintersteiger

Z3: Little Engines of Proof

Freely available from http://research.microsoft.com/projects/z3

Microsoft Research

Research around Z3

Decision Procedures

Modular Difference Logic is Hard TR 08 B, Blass Gurevich, Muthuvathi.
Linear Functional Fixed-points. CAV 09 B. & Hendrix.
A Priori Reductions to Zero for Strategy-Independent Gröbner Bases SYNASC 09 M& Passmore.
Efficient, Generalized Array Decision Procedures FMCAD 09 M & B
Quantifier Elimination as an Abstract Decision Procedure IJCAR 10, B
Cutting to the Chase CADE 11, Jojanovich, M

Combining Decision Procedures

Model-based Theory Combination SMT 07 M & B. .
Accelerating Lemma learning using DPLL(U) LPAR 08 B, Dutetre & M
Proofs, Refutations and Z3 IWIL 08 M & B
On Locally Minimal Nullstellensatz Proofs. SMT 09 M & Passmore.
A Concurrent Portfolio Approach to SMT Solving CAV 09 Wintersteiger, Hamadi & M
Conflict Directed Theory Resolution Cambridge Univ. Press 12, M & B

Quantifiers, quantifiers, quantifiers

Efficient E-matching for SMT Solvers. CADE 07 M & B.
Relevancy Propagation. TR 07 M & B.
Deciding Effectively Propositional Logic using DPLL and substitution sets IJCAR 08 M & B.
Engineering DPLL(T) + saturation. IJCAR 08 M & B.
Complete instantiation for quantified SMT formulas CAV 09 Ge & M.
On deciding satisfiability by DPLL(Γ+ T) and unsound theorem proving.
CADE 09 Bonachina, M & Lynch

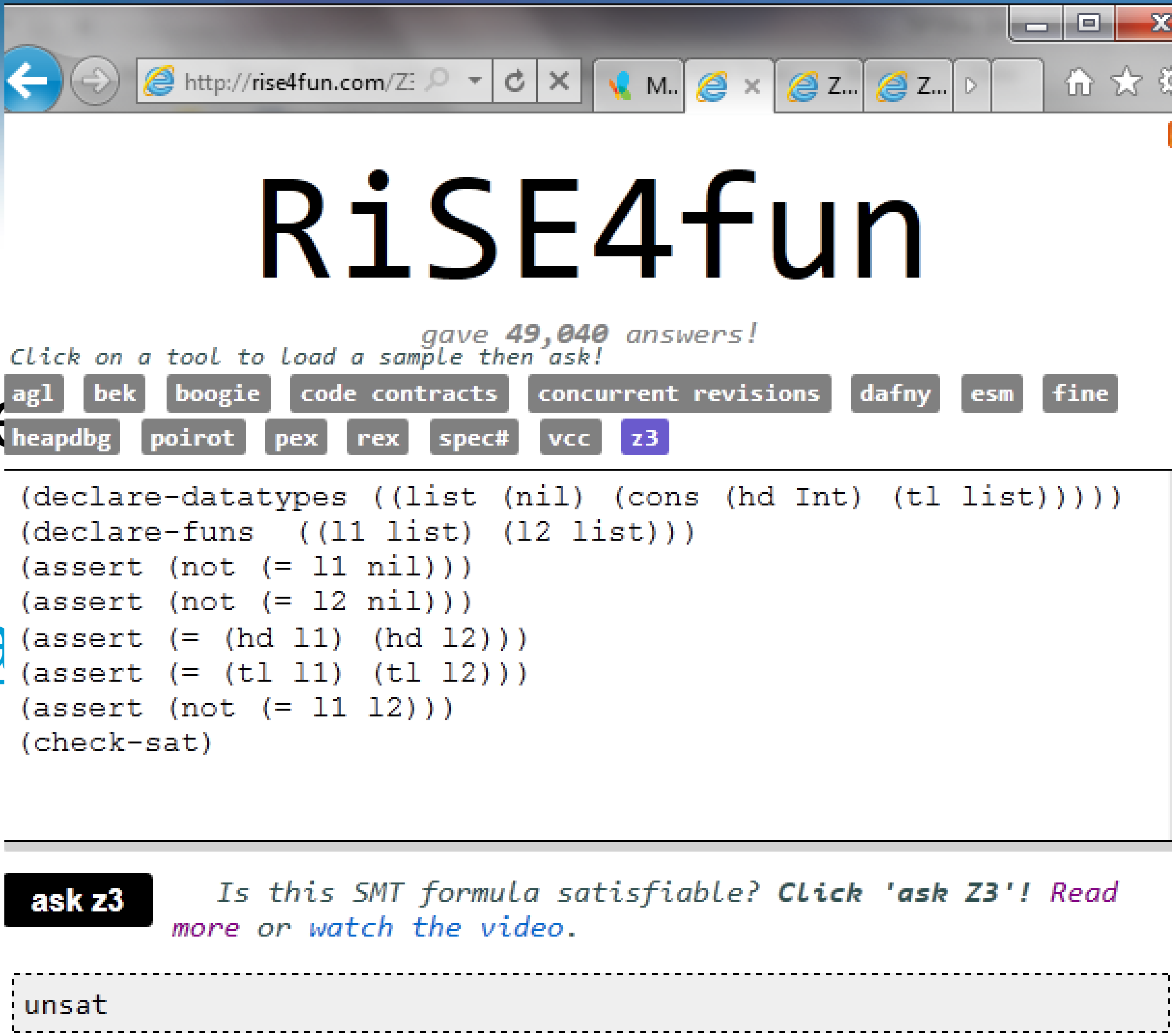
Fast and powerful

http://smtcomp.org

Microsoft Research

Theories

- Uninterpreted
- Arithmetic (line
- Bit-vectors
- Algebraic data
- Arrays
- User-defined



Some Microsoft tools using Z3

Property Driven

F7 1.0

Price: FREE

F7 is created to be a simple to use yet ideal typechecker for the F# programming language. F7 supports static checking of properties expressed with refinement types.

Filesize: 3.2 MB

Download now

Details

Screenshot

F7 1.0 simple to use to use Simple To programming language

F

Aud

HAVOC

SLAyer

Type Safety

BOOGIE

Execution Guided

SLAM

Over-Approximation

Yogi

Under-Approximation

SAGE

Pex

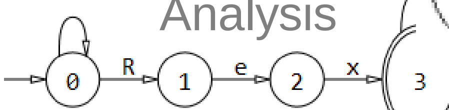
Model Based

The Spec# Programming System

Testing

M3

Analysis



BEK

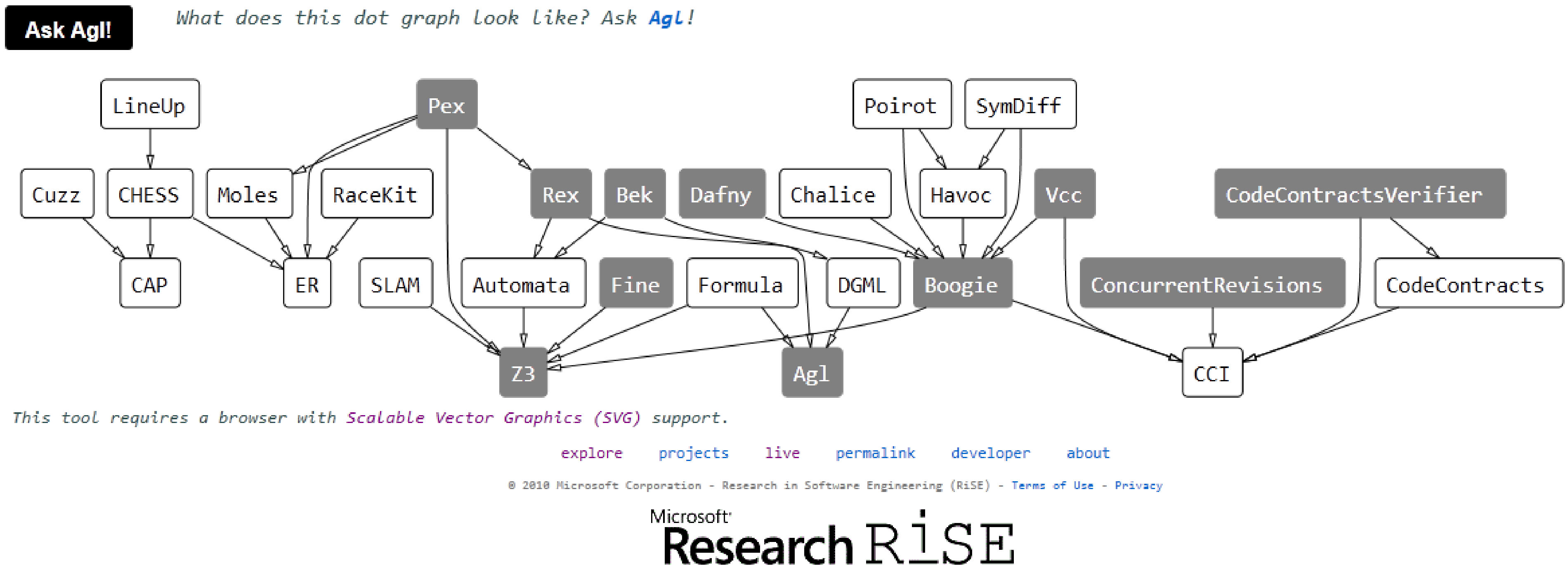
FORMULA

Modeling Foundations.

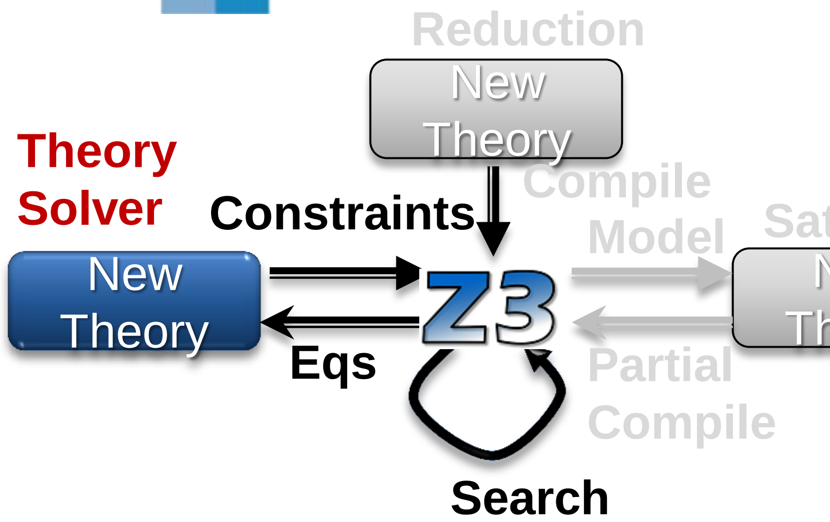
Synthesis

Microsoft Research

Some Microsoft tools using Z3

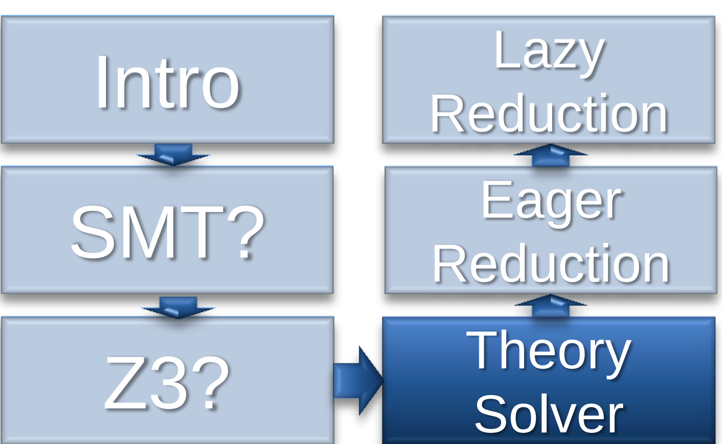


http://rise4fun.com



Optimization

Get More Satisfaction with SMT



Oliveras, Nieuenhuis, SAT 2006

Weighted MaxSMT

$Name$	$Formula$	$weight$
F_0	$a \vee b \vee x \geq 2$	∞
F_1	$\neg a \vee x \geq 3$	3
F_2	$\neg b \vee x \geq 3$	4
F_3	$x < 2$	5

Unsat

Weighted MaxSMT

$Name$	$Formula$	$weight$
F_0	$a \vee b \vee x \geq 2$	∞
F_1	$\neg a \vee x \geq 3$	3
F_2	$\neg b \vee x \geq 3$	4
F_3	$x < 2$	5

Penalty: ∞

Sat $\neg \boldsymbol{a} \wedge \neg \boldsymbol{b} \wedge \boldsymbol{x} < \mathbf{2}$

Weighted MaxSMT

$Name$	$Formula$	$weight$
F_0	$a \vee b \vee x \geq 2$	∞
F_1	$\neg a \vee x \geq 3$	3
F_2	$\neg b \vee x \geq 3$	4
F_3	$x < 2$	5

Sat $\neg \boldsymbol{a} \wedge \boldsymbol{b} \wedge \boldsymbol{x} = 2$

Penalty: 9 = 4 + 5

Weighted MaxSMT

$Name$	$Formula$	$weight$
F_0	$a \vee b \vee x \geq 2$	∞
F_1	$\neg a \vee x \geq 3$	3
F_2	$\neg b \vee x \geq 3$	4
F_3	$x < 2$	5

Sat $\neg \boldsymbol{a} \wedge \neg \boldsymbol{b} \wedge \boldsymbol{x} \geq \mathbf{2}$

Penalty: 5

Weighted MaxSMT

$Name$	$Formula$	$weight$
F_0	$a \vee b \vee x \geq 2$	∞
F_1	$\neg a \vee x \geq 3$	3
F_2	$\neg b \vee x \geq 3$	4
F_3	$x < 2$	5

Penalty: 3

Sat $\boldsymbol{a \wedge \neg b \wedge x < 2}$

Weighted MaxSMT

$Formula$	$weight$
$a \vee b \vee x \geq 2$	∞
$F_1 \vee \neg a \vee x \geq 3$	3
$F_2 \vee \neg b \vee x \geq 3$	4
$F_3 \vee x < 2$	5

Initially: All atoms are unassigned

$Cost = 0$

Assert $\neg a \wedge b \wedge x < 2$

Propagate: F_2: $Cost := Cost + 4 := 4$

Best so far: $MinCost = 4$

Add Axiom $\neg F_2$ *- backtrack*

Assert F_3 $Cost = 5 > MinCost$

Add Axiom $\neg F_3$ *- backtrack*

…. **Assert** $a \wedge \neg b \wedge x < 2 \wedge F_1$

```
let block() =
    let offender = optimize_cost ctx costs min_cost
    th.AssertTheoryAxiom(ctx.MkNot(ctx.MkAnd offender))
let Assign p vl =
    if vl then
       let w = weights.[p]
       cost <- cost + w;         trail.Add (fun () -> cost <- cost - w)
       costs <- (w,p)::costs;    trail.Add (fun () -> costs <- List.tail costs)
       if cost > min_cost then
          block()
let _ = th.NewAssignment <- (fun p vl -> Assign p vl)
```

What does it take to encode this in Z3??

Principles of Modern SMT solvers in two slides

Modern DPLL in a nutshell

Initialize	$\epsilon \mid F$	F *is a set of clauses*
Decide	$M \mid F \implies M, \ell \mid F$	ℓ *is unassigned*
Propagate	$M \mid F, C \vee \ell \implies M, \ell^{C \vee \ell} \mid F, C \vee \ell$	C *is false under* M
Conflict	$M \mid F, C \implies M \mid F, C \mid C$	C *is false under* M
Resolve	$M \mid F \mid C' \vee \neg\ell \implies M \mid F \mid C' \vee C$	$\ell^{C \vee \ell} \in M$
Learn	$M \mid F \mid C \implies M \mid F, C \mid C$	
Backjump	$M \neg\ell M' \mid F \mid C \vee \ell \implies M \ell^{C \vee \ell} \mid F$	C *has no literals in* M'
Unsat	$M \mid F \mid \emptyset \implies Unsat$	
Sat	$M \mid F \implies M$	F *true under* M
Restart	$M \mid F \implies \epsilon \mid F$	

Adapted and modified from [Nieuwenhuis, Oliveras, Tinelli J.ACM 06]

DPLL(*T*) solver interaction

T- Propagate $M \mid F, C \vee \ell \implies M, \ell^{C\vee\ell} \mid F, C \vee \ell$ $\quad C\ is\ false\ under\ T + M$

T- Conflict $M \mid F \implies M \mid F \mid \neg M'$ $\quad M' \subseteq M\ and\ M' is\ false\ under\ T$

T- Propagate $a > b, b > c \mid F, a \leq c \vee b \leq d \implies$

$$a > b, b > c, b \leq d^{a \leq c \vee b \leq d} \mid F, a \leq c \vee b \leq d$$

T- Conflict $M \mid F \implies M \mid F,\ a \leq b \vee b \leq c \vee c < a$

$$where\ a > b, b > c, a \leq c \subseteq M$$

How does Z3 enable *T* solvers?

DPLL(T) Solver Interaction

Calls into DPLL engine

Callbacks from DPLL engine with new assignment

T-Propagate

T-Propagate

T-Conflict

T-Conflict

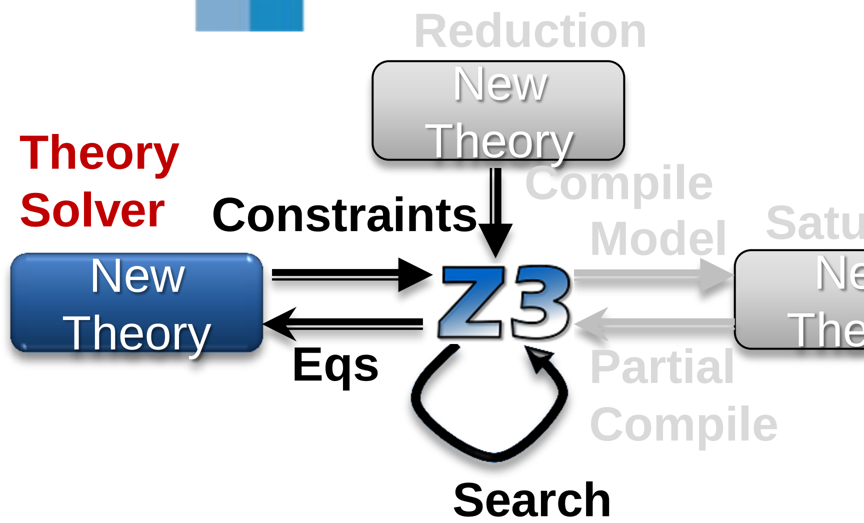
Calls into DPLL engine

Callbacks from DPLL engine

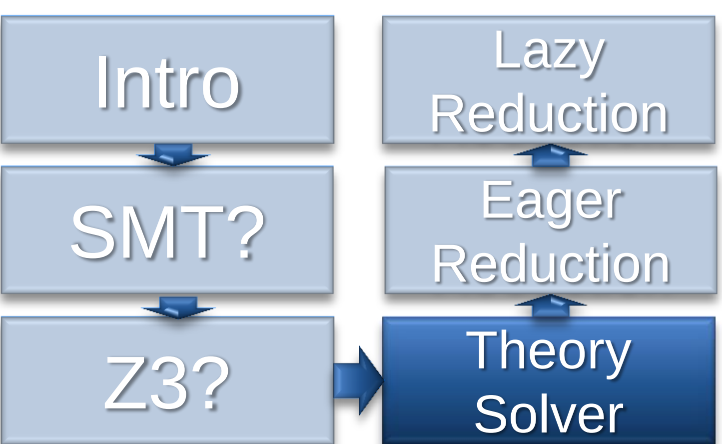
```
let mutable cost = 0
let mutable costs = []
let mutable min_cost = Int32.max_int
let mutable weights = Dictionary<Term,int>()
let th = ctx.MkTheory("opt")
let block() =
    let offender = optimize_cost ctx costs min_cost
    th.AssertTheoryAxiom(ctx.MkNot(ctx.MkAnd offender))
let Assign p vl =
    if vl then
       let w = weights.[p]
       cost <- cost + w;
       trail.Add (fun () -> cost <- cost - w)
       costs <- (w,p)::costs;
       trail.Add (fun () -> costs <- List.tail costs)
       if cost > min_cost then
          block()

let FinalCheck() =
    if cost < min_cost then
       min_cost <- cost
    block()
    true

let _ = th.NewAssignment <- (fun p vl -> Assign p vl)
let _ = th.FinalCheck <- (fun () -> FinalCheck())
let _ = th.Pop <- (fun () -> trail.Pop())
let _ = th.Push <- (fun () -> trail.Push())
```



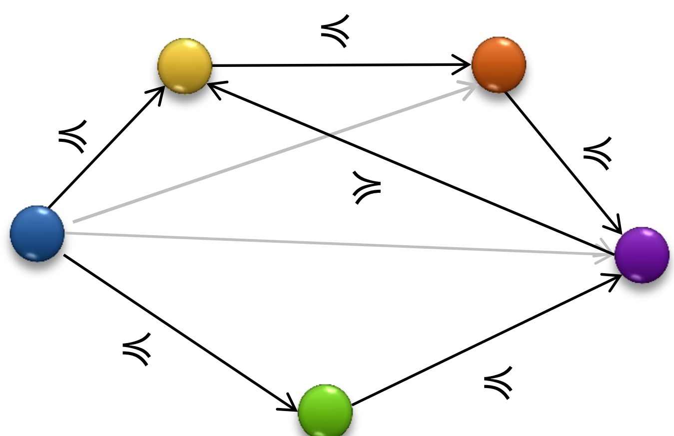
Partial Orders & Object Hierarcies



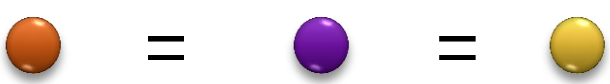
Acyclic graphs and SMT

Partial Orders as Acyclic Graphs

$$\forall x.\, x \preccurlyeq x$$
$$\forall x, y.\, x \preccurlyeq y \land y \preccurlyeq x \rightarrow x = y$$
$$\forall x, y, z\,.\, x \preccurlyeq y \land y \preccurlyeq z \rightarrow x \preccurlyeq z$$



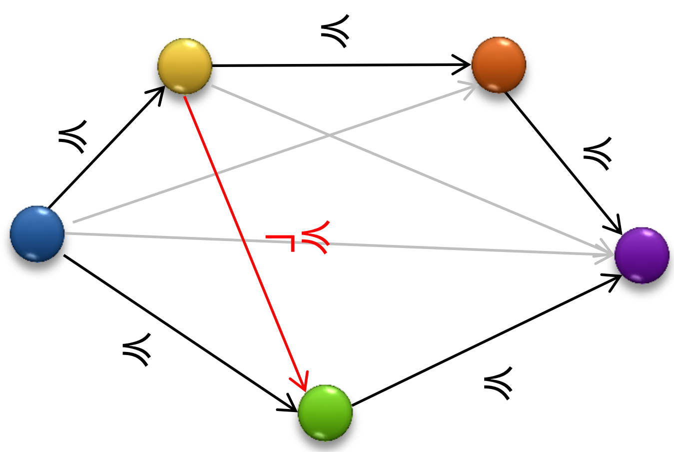
Elements are equal in strongly connected components



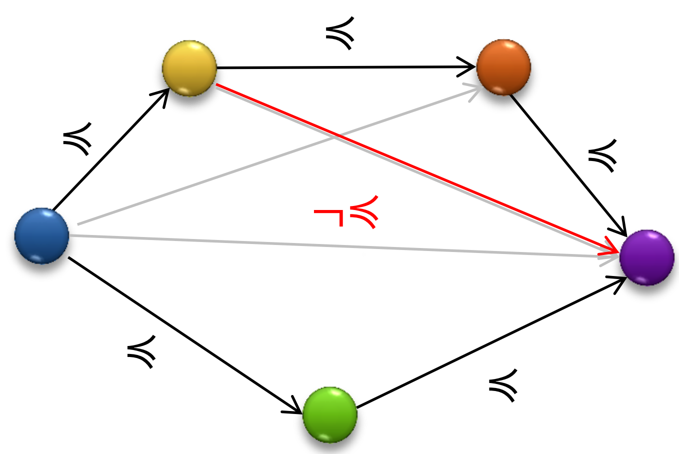
Partial Orders as Acyclic Graphs

Checking negations

$$\forall x.\, x \preccurlyeq x$$
$$\forall x, y.\, x \preccurlyeq y \wedge y \preccurlyeq x \rightarrow x = y$$
$$\forall x, y, z\,.\, x \preccurlyeq y \wedge y \preccurlyeq z \rightarrow x \preccurlyeq z$$



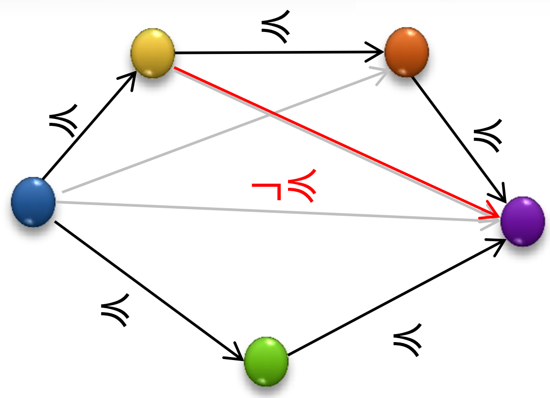
OK



Not OK

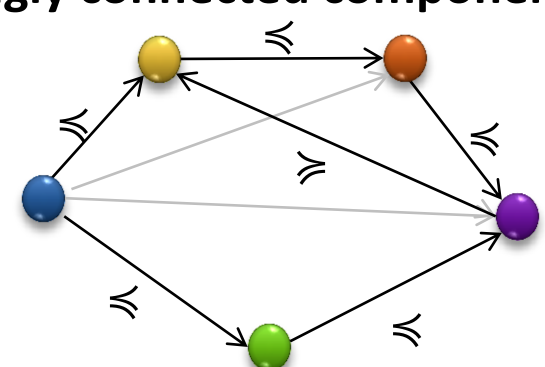
Partial Orders as Acyclic Graphs

Checking Consistency of $\neg(x \preccurlyeq y)$:



Is there is a $\preccurlyeq$ path from (yellow) to (purple) ?

Extracting Equalities from $\preccurlyeq$ using strongly connected components:

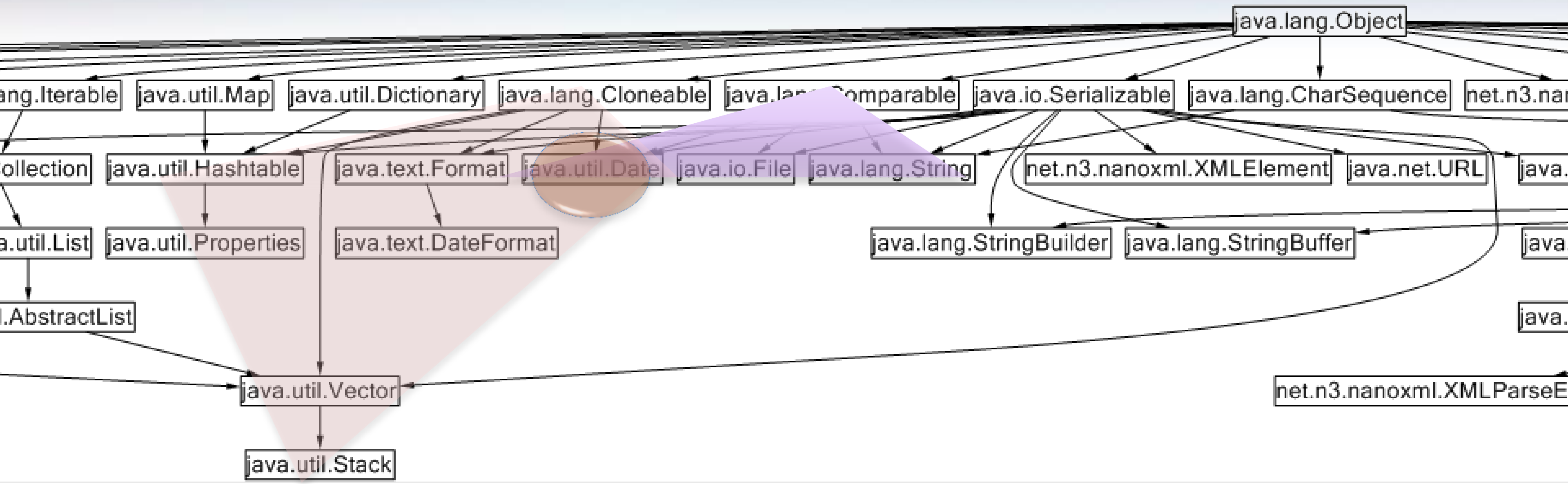



```
let FinalCheck() =
    for t in terms do     // connect equal terms.
        let tr = th.GetEqcRoot t
        if tr <> t then
           add_eq t tr
    done
    check_not_leqs()     // check negations
    add_implied_eqs()    // propagate equalities
    true


let NewAssignment (a:Term) v =
    assert (is_leq a)
    let args = a.GetAppArgs()
    let t1, t2 = args.[0], args.[1]
    add_term t1; add_term t2
    if v then
       g.AddEdge t1 t2
    else
       trail.Add (fun () -> not_leqs <- List.tail not_leqs)
       not_leqs <- (t1,t2)::not_leqs

let initialize() =
    th.FinalCheck <- (fun () -> FinalCheck())
    th.NewAssignment <- (fun a v -> NewAssignment a v)
    th.Push <- (fun () -> trail.Push())
    th.Pop <- (fun () -> trail.Pop())
```

Inheritance as table-lookup



$x \preccurlyeq java.lang.Comparable$

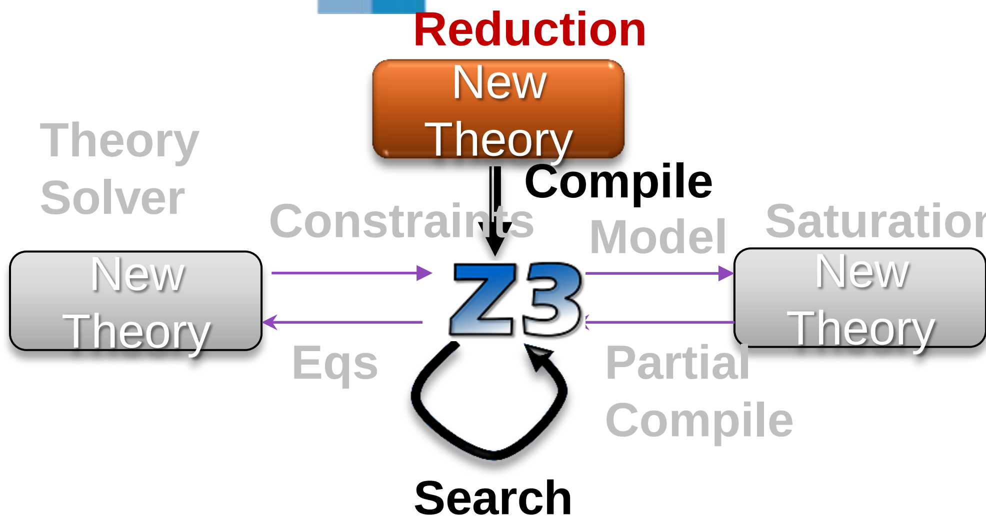
$x \preccurlyeq java.lang.Clonable$

$x = java.util.Date$

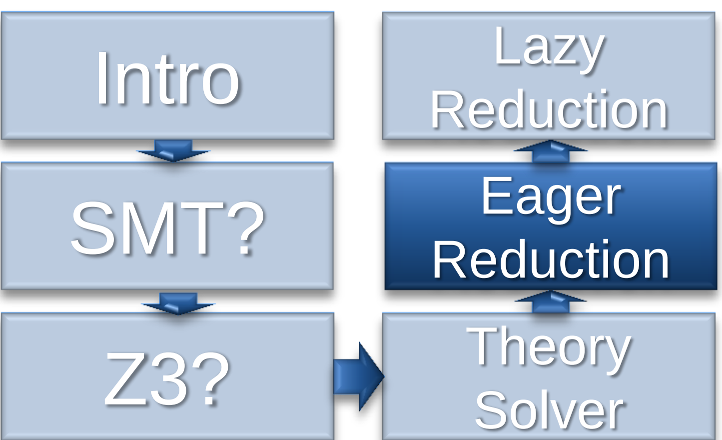
Efficient propagators using
Type Slicing algorithm
Leverages ordering of children
J. Gil and Y. Zibin.[TOPLAS 2007]

Available as F#/Z3 sample

Sherman, Garvin, Dwyer. IJCAR 2010



Object Graphs



To Cycle *and* not to Cycle

from Pex

A Theory of Objects

```
class O {

  public readonly D d;
  public readonly O left;
  public O right;

  public O(D data,
           O left,
           O right) {
    this.data = data;
    this.left = left;
    this.right = right;
  }
}
```

```
void f(O n0) {

  Assert (n0 == null ||
          n0.left != n0);

  O n1 = new O(1,null,null);
  O n2 = new O(2,n1,null);
  O n3 = new O(2,n1,null);

  Assert(n2 != n3);

  n1.right = n2;
  n2.right = n1;
...
```

Read-only fields

Heap can be updated

Objects are non-extensional

A Theory of Objects

$$
\begin{array}{ll}
\textit{sorts:} & O, \\
\textit{constructors:} & \mathit{null} : O,\ O : H \times D \times O \times O \to H \times O, \\
\textit{accessors:} & \mathit{data} : H \times O \to D,\ \mathit{left} : H \times O \to O,\ \mathit{right} : H \times O \to O, \\
\textit{modifiers:} & \mathit{update\_right} : H \times O \times O \to H
\end{array}
$$

$$(h', o) = O(h, d, l, r) \implies o \neq \mathit{null}$$

$$(h', o) = O(h, d, l, r) \implies \mathit{data}(h', o) = d$$

$$(h', o) = O(h, d, l, r) \implies \mathit{left}(h', o) = l$$

$$\boxed{o \neq \mathit{null} \implies \mathit{left}(h_1, \mathit{left}(h_2, \mathit{left}(\ldots \mathit{left}(h_n, o)))) \neq o}$$

$$h' = \mathit{update\_right}(h, o, r) \wedge o' \neq o \implies \mathit{right}(h', o') = \mathit{right}(h, o')$$

$$h' = \mathit{update\_right}(h, o, r) \implies \mathit{left}(h', o') = \mathit{left}(h, o')$$

$$h' = \mathit{update\_right}(h, o, r) \implies \mathit{data}(h', o') = \mathit{data}(h, o')$$

Encoding: Heaps as Arrays

Domains: objects are Natural numbers, left child is a smaller number

$$O = N$$
$$H = \langle data : O \Rightarrow D, left : O \Rightarrow O, right : O \Rightarrow O, clock : N \rangle$$

Most axioms follow by function definitions.

$$right(\langle data, left, right, clock\rangle, o) = select(right, o)$$
$$update\_right(\langle data, left, right, clock\rangle, o, r) = \langle data, left, store(right, o, r), clock\rangle$$
$$\ldots$$

Only Axiom: Instantiate for every occurrence of *left(h,o)*

$$\forall h : H, o : O \ . \ o \neq null \Longrightarrow 0 \leq left(h, o) < o$$

Encoding: Heaps as Arrays+Data-Types

Domains: read-only fields use algebraic data-types

$$O = null \mid O(id : N, data : D, left : O)$$
$$H = \langle right : O \Rightarrow O, clock : N \rangle$$

Most axioms follow by function definitions.

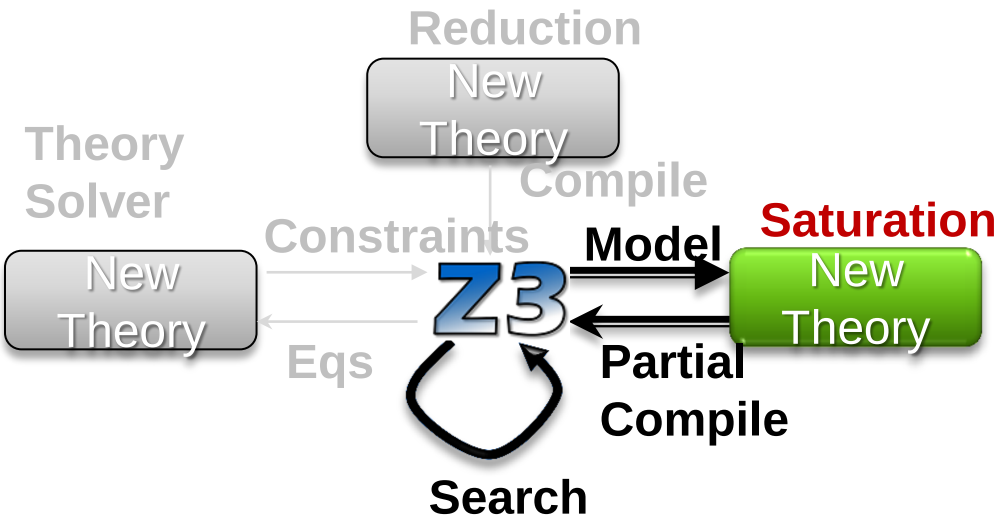
$$left(h, O(id, d, l)) = l$$
$$left(h, null) = null$$
$$right(\langle right, clock \rangle, o) = select(right, o)$$
$$update\_right(\langle right, clock \rangle, o, r) = \langle store(right, o, r), clock \rangle$$

No Extra Axiom: Data-type theory enforces acyclicity over *left*

⇒ More efficient search

Microsoft Research

HOL



Z3 at the service of

Γ,Π,Σ,α,β,λ,η,κ,*,□

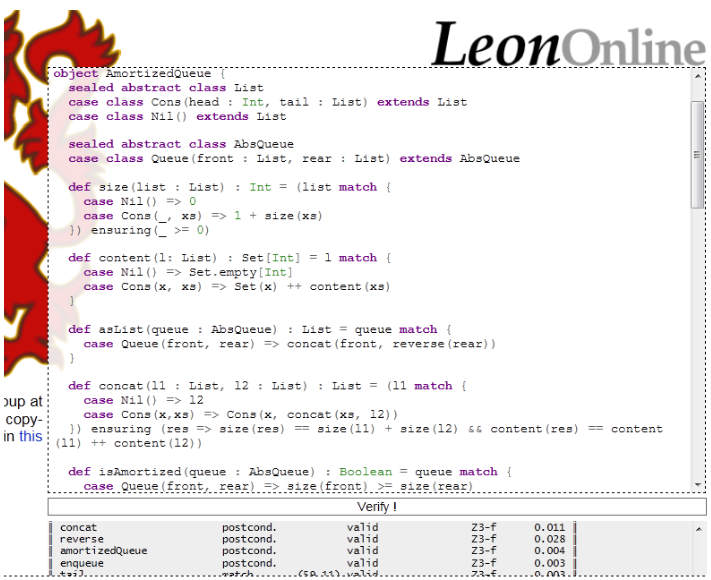
Intro
SMT?
Z3?
Lazy Reduction
Eager Reduction
Theory Solver

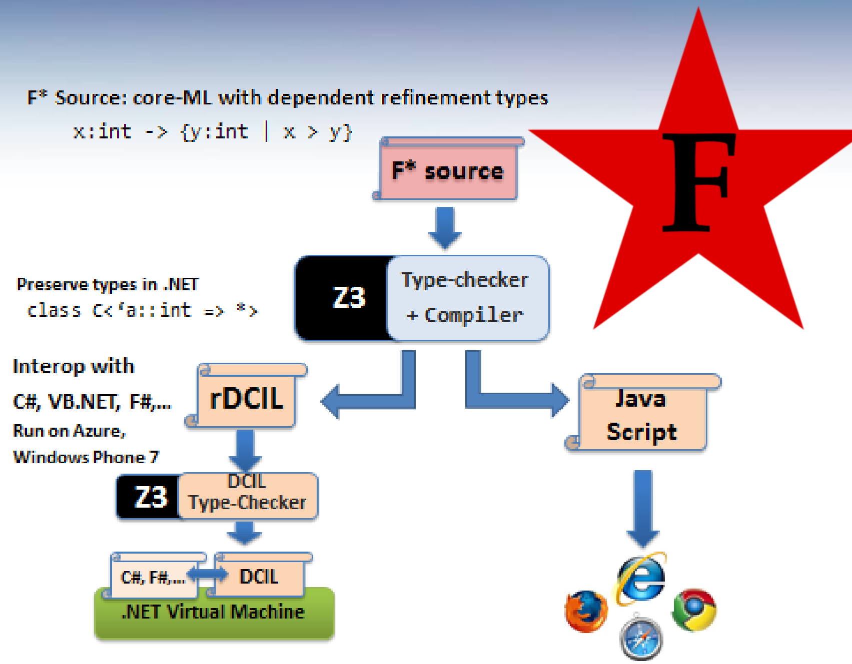
SMT version of Satalax, Brown, CADE 2011

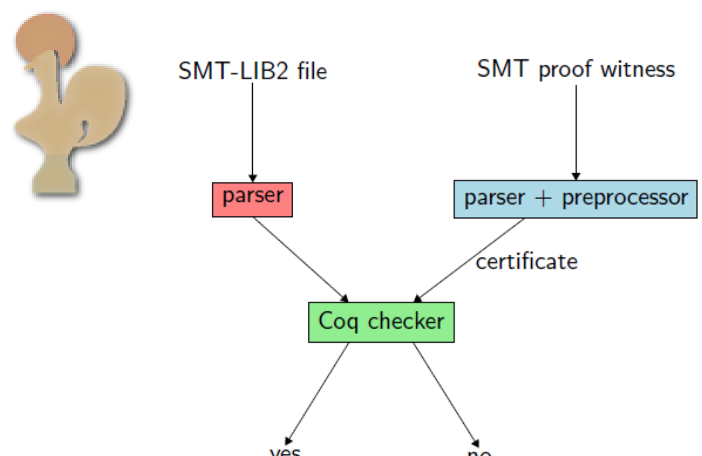
Types and Z3 do mingle

Sledge Hammer

Liquid Types



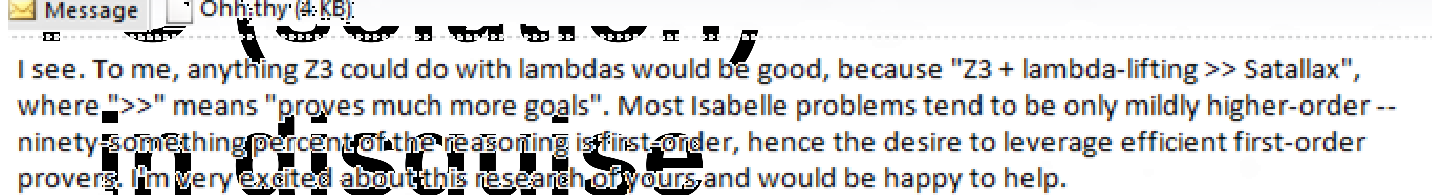




Armand, Grégoire, Keller, Théry, Werner

But
Used for First-Order Theorems

Sure, often HOL (problem) is just FO (solution) in disguise



Re: Higher-order problem

Jasmin Blanchette <jasmin.blanchette@gmail.com>

This message is part of a tracked conversation. Click here to find all related messages or to open th
Extra line breaks in this message were removed.

ent: Wed 2/22/2012 9:25 AM

o: Nikolaj Bjorner

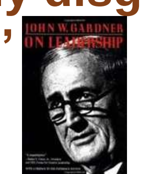
c: Philippe Suter

Message | Ohh.thy (4 KB)

I see. To me, anything Z3 could do with lambdas would be good, because "Z3 + lambda-lifting >> Satallax", where ">>" means "proves much more goals". Most Isabelle problems tend to be only mildly higher-order -- ninety-something percent of the reasoning is first-order, hence the desire to leverage efficient first-order provers. I'm very excited about this research of yours and would be happy to help.

"We are all faced with a series of great opportunities brilliantly disguised as unsolvable problems."

John W. Gardner



"For every problem there is a solution which is simple, clean and wrong."

Henry Louis Mencken

Digression: CAL

CAL – Combinatory Array Logic

$$store(a, i, v) = \lambda j.\, \boldsymbol{if}\ i = j\ \boldsymbol{then}\ v\ \boldsymbol{else}\ a[j]$$

$$K(v) = \lambda j\,.\, v$$

$$map_f(a, b) = \lambda j\,.\, f(a[j], b[j])$$

Existential fragment is in NP by reduction to congruence closure using polynomial set of instances.

but can we do something more HOLish?

e.g.,

$$\forall f.\,(\forall x, y.\, f(x) = f(y) \rightarrow x = y)$$
$$\rightarrow \exists g\,.\,\forall x\,.\,x = g(f(x))$$

Idea: Saturate for Henkin Models

Types $\sigma ::= i \mid o \qquad \tau ::= \sigma \mid \tau \to \tau$

Terms $M, N ::= \lambda x{:}\,\tau.\, M \mid (M\ N) \mid x$

Constants

$$false : o \qquad \Rightarrow : o \to o \to o$$

$$\epsilon : (\tau \to o) \to \tau, \qquad \forall : (\tau \to o) \to o,$$

$$= : \tau \to \tau \to o$$

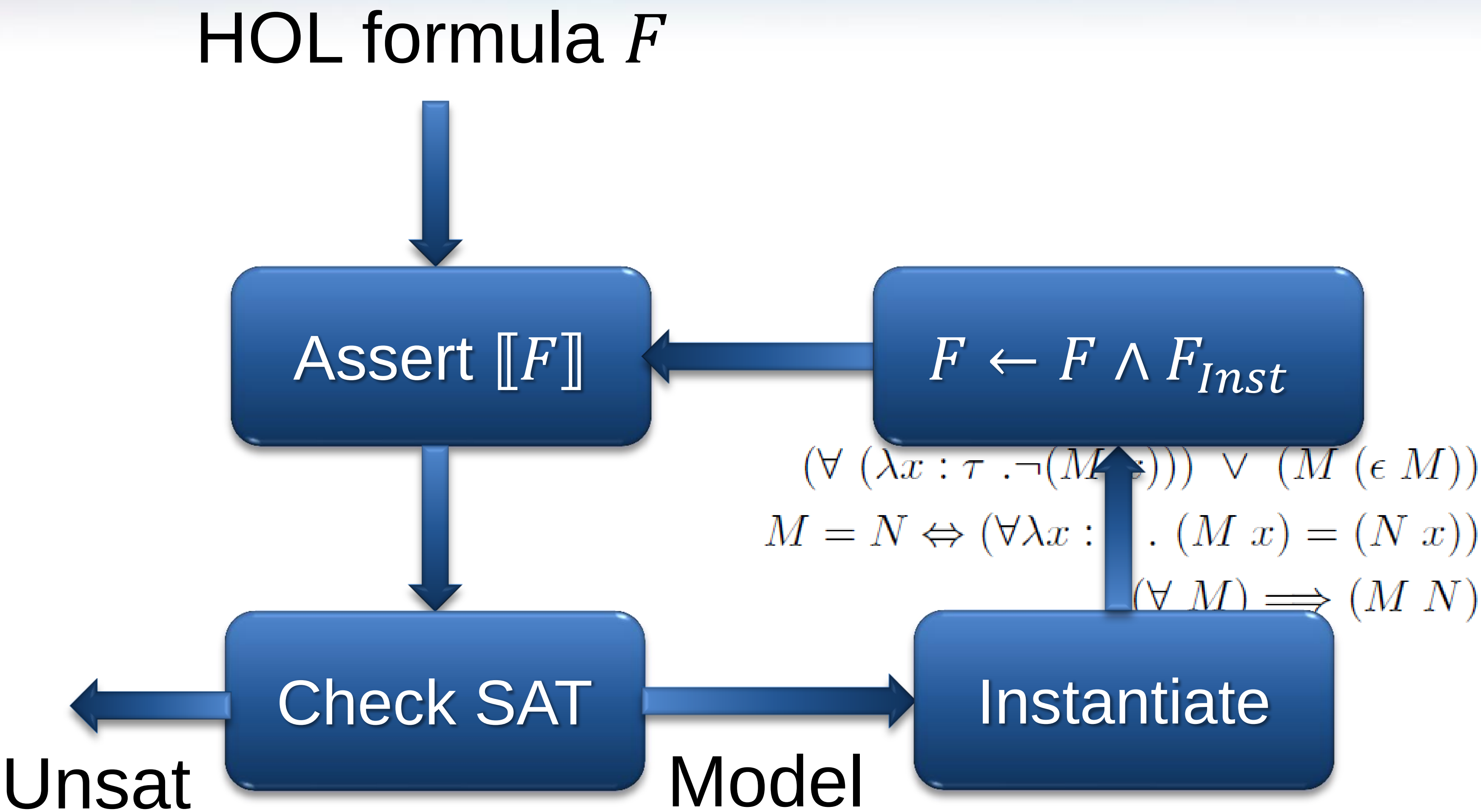
Axioms

$$(\forall\ (\lambda x : \tau\ .\neg(M\ x)))\ \vee\ (M\ (\epsilon\ M)) \qquad \text{for every } M : \tau \to o$$

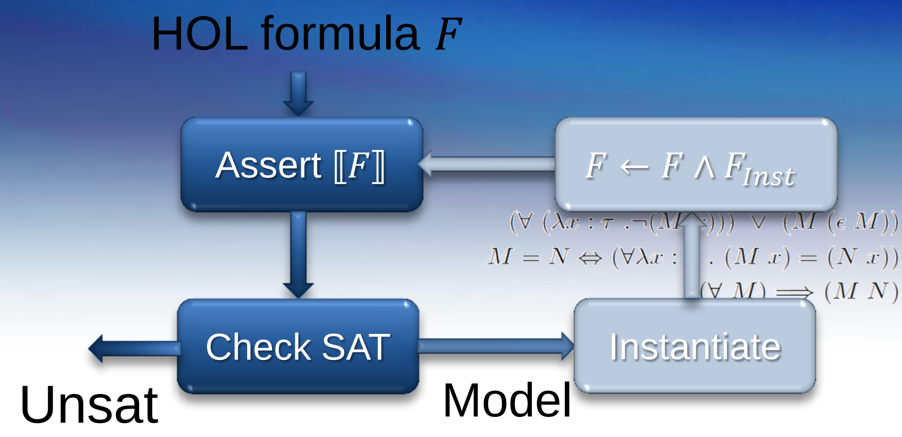
$$M = N \Leftrightarrow (\forall \lambda x : \tau\ .\ (M\ x) = (N\ x)) \qquad \text{for every } M, N : \tau \to \tau'$$

$$(\forall\ M) \Longrightarrow (M\ N) \qquad \text{for every } M : \tau \to o, N : \tau$$

Lazy Saturation loop



HOL ⇒SMT



$[\![\_]\!]: HOL \rightarrow SMT$

$$[\![(\forall\ M)]\!] = \lceil(\forall\ M)\rceil$$
$$[\![(\epsilon\ M)]\!] = \lceil(\epsilon\ M)\rceil$$
$$[\![M \implies N]\!] = [\![M]\!] \implies [\![N]\!]$$
$$[\![M = N]\!] = [\![M]\!] = [\![N]\!]$$
$$[\![(M\ N)]\!] = \mathsf{select}([\![M]\!], [\![N]\!])$$
$$[\![\lambda x : \tau\ .\ M]\!] = \lceil\lambda x : \tau\ .\ M\rceil$$
$$[\![f]\!] = f \quad \text{for constant } f$$

Propositional reasoning

SAT

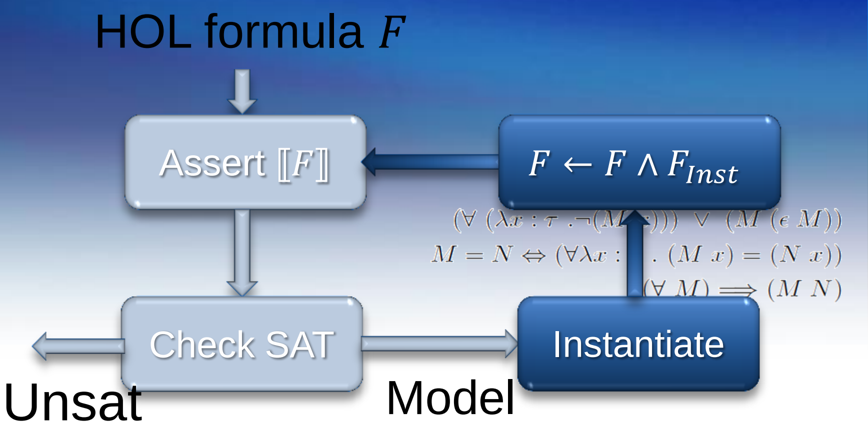
Equalities

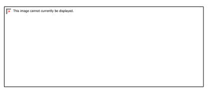
Congruence Closure

SMT

Extensional arrays

$\beta\eta$ long NF





Set of $\beta\eta$ long NF terms with free variables from Γ of type τ

Enumerate $T[\Gamma;\tau]$ by depth:

$$
\begin{array}{lll}
(\lambda x : \tau \,.\, M) \in \mathcal{T}[\Gamma; \tau \to \tau'] & \text{if} & M \in \mathcal{T}[\Gamma, x : \tau; \tau'] \\
(x\ M_1\ \ldots\ M_k) \in \mathcal{T}[\Gamma; \sigma] & \text{if} & (x : \overline{\tau} \to \sigma) \in \Gamma,\ M_i \in \mathcal{T}[\Gamma; \tau_i]
\end{array}
$$

Many more algorithms (matching, unification)/optimizations required for anything viable…

… but main task of Boolean search, equalities, functions is delegated

Conclusions

We surveyed three methods for adding new theories (logics) to Z3:

- **As 1st class Theory Solver**
- **Eager reduction: embed theory in Z3**
- **Lazy reduction: add facts on demand**

Choose one that fits your theory!

[Zvonimir Rakamaric, Roberto Bruttomesso, Alan J. Hu, Alessandro Cimatti: Verifying Heap-Manipulating Programs in an SMT Framework. ATVA 2007: 237-252]

[Stan Rosenberg, Anindya Banerjee and David Naumann. Decision Procedures for Region Logic. VMCAI 2012]

Theories vs. Problem Classes

Applications often generate problems with particular characteristics (many ground clauses/bit-vectors + predicates/arithmetic + transendentals/..)

New Z3 feature by de Moura & Passmore:

- Compose strategies using tactical interface.