Logic: FOL and SMT

Testing, Quality Assurance, and Maintenance
Winter 2017

Prof. Arie Gurfinkel

based on slides by Prof. Ruzica Piskac, Nikolaj Bjorner, and others

UNIVERSITY OF WATERLOO

Conflict Directed Clause Learning

Lemma learning

$\neg t, p, q, s \mid t \vee \neg p \vee q, \neg q \vee s, \neg p \vee \neg s$

$\neg t, p, q, s \mid t \vee \neg p \vee q, \neg q \vee s, \neg p \vee \neg s \mid \neg p \vee \neg s$

$\neg t, p, q, s \mid t \vee \neg p \vee q, \neg q \vee s, \neg p \vee \neg s \mid \neg p \vee \neg q$

$\neg t, p, q, s \mid t \vee \neg p \vee q, \neg q \vee s, \neg p \vee \neg s \mid \neg p \vee t$

Learned Clause by Resolution

$$\dfrac{\dfrac{t \vee \neg p \vee q \qquad \neg q \vee s}{t \vee \neg p \vee s} \qquad \neg p \vee \neg s}{\neg p \vee t}$$

Modern CDCL

Initialize	$\epsilon \mid F$	F *is a set of clauses*
Decide	$M \mid F \Longrightarrow M, \ell \mid F$	ℓ *is unassigned*
Propagate	$M \mid F, C \vee \ell \Longrightarrow M, \ell^{C \vee \ell} \mid F, C \vee \ell$	C *is false under* M
Sat	$M \mid F \Longrightarrow M$	F *true under* M
Conflict	$M \mid F, C \Longrightarrow M \mid F, C \mid C$	C *is false under* M
Learn	$M \mid F \mid C \Longrightarrow M \mid F, C \mid C$	
Unsat	$M \mid F \mid \emptyset \Longrightarrow Unsat$	
Backjump	$MM' \mid F \mid C \vee \ell \Longrightarrow M\ell^{C \vee \ell} \mid F$	$\bar{C} \subseteq M, \neg\ell \in M'$
Resolve	$M \mid F \mid C' \vee \neg\ell \Longrightarrow M \mid F \mid C' \vee C$	$\ell^{C \vee \ell} \in M$
Forget	$M \mid F, C \Longrightarrow M \mid F$	C is a learned clause
Restart	$M \mid F \Longrightarrow \epsilon \mid F$	

Model

Proof

Conflict Resolution

[Nieuwenhuis, Oliveras, Tinelli J.ACM 06] customized

FIRST ORDER LOGIC

The language of First Order Logic

Functions , Variables, Predicates

- *f, g,…* *x, y, z, …* P, Q, =, <, …

Atomic formulas, Literals

- P(x,f(y)), ¬Q(y,z)

Quantifier free formulas

- P(f(a), b) ∧ c = g(d)

Formulas, sentences

- ∀x . ∀y . [P(x, f(x)) ∨ g(y,x) = h(y)]

Language: Signatures

A *signature* Σ is a finite set of:

- Function symbols:

$$\Sigma_F = \{ f, g, +, \ldots \}$$

- Predicate symbols:

$$\Sigma_P = \{ P, Q, =, \text{true}, \text{false}, \ldots \}$$

- And an *arity* function:

$$\Sigma \rightarrow \mathbb{N}$$

Function symbols with arity 0 are *constants*

- notation: $f_{/2}$ means a symbol with arity 2

A countable set *V* of *variables*

- disjoint from Σ

Language: Terms

The set of *terms* $T(\Sigma_F, V)$ is the smallest set formed by the syntax rules:

- $t \in T$ ::= v $\quad v \in V$
  | $f(t_1, \ldots, t_n)$ $\quad f \in \Sigma_F, t_1, \ldots, t_n \in T$

Ground terms are given by $T(\Sigma_F, \varnothing)$

Language: Atomic Formulas

$a \in Atoms \quad ::= P(t_1, \ldots, t_n)$

$P \in \Sigma_P \; t_1, \ldots, t_n \in T$

An atom is *ground* if $t_1, \ldots, t_n \in T(\Sigma_F, \varnothing)$

Literals are (negated) atoms:

$l \in Literals \quad ::= a \mid \neg a \qquad\qquad a \in Atoms$

Language: Quantifier free formulas

The set QFF(Σ,V) of *quantifier free formulas* is the smallest set such that:

$\varphi \in$ QFF	::= $a \in$ *Atoms*	*atoms*
	$\mid \neg\, \varphi$	*negations*
	$\mid \varphi \leftrightarrow \varphi'$	*bi-implications*
	$\mid \varphi \wedge \varphi'$	*conjunction*
	$\mid \varphi \vee \varphi'$	*disjunction*
	$\mid \varphi \rightarrow \varphi'$	*implication*

Language: Formulas

The set of *first-order formulas* are obtained by adding the formation rules:

$$\varphi ::= \ldots$$

$$\mid \quad \forall x \,.\, \varphi \qquad \textit{universal quant.}$$

$$\mid \quad \exists x \,.\, \varphi \qquad \textit{existential quant.}$$

Free (occurrences) of *variables* in a formula are theose not bound by a quantifier.

A sentence is a first-order formula with no free variables.

Dreadbury Mansion Mystery

Someone who lived in Dreadbury Mansion kill Aunt Agatha. Agatha, the Butler and Charles were the only people who lived in Dreadbury Mansion. A killer always hates his victim, and is never richer than his victim. Charles hates no one that aunt Agatha hates. Agatha hates everyone except the butler. The butler hates everyone not richer than Aunt Agatha. The butler also hates everyone Agatha hates. No one hates everyone. Agatha is not the butler.

Who killed Aunt Agatha?

Dreadbury Mansion Mystery

$killed/_2, hates/_2, richer/_2, a/_0, b/_0, c/_0$

$$\exists x \cdot killed(x, a) \tag{1}$$

$$\forall x \cdot \forall y \cdot killed(x, y) \implies (hates(x, y) \wedge \neg richer(x, y)) \tag{2}$$

$$\forall x \cdot hates(a, x) \implies \neg hates(c, x) \tag{3}$$

$$hates(a, a) \wedge hates(a, c) \tag{4}$$

$$\forall x \cdot \neg richer(x, a) \implies hates(b, x) \tag{5}$$

$$\forall x \cdot hates(a, x) \implies hates(b, x) \tag{6}$$

$$\forall x \cdot \exists y \cdot \neg hates(x, y) \tag{7}$$

$$a \neq b \tag{8}$$

Models (Semantics)

A model *M* is defined as:

- Domain *S;* set of elements.
- Interpretation, $f^M : S^n \rightarrow S$ for each $f \in \Sigma_F$ with arity(f) = n
- Interpretation $P^M \subseteq S^n$ for each $P \in \Sigma_P$ with arity(P) = n
- Assignment $x^M \in S$ for every variable $x \in V$

A formula φ is true in a model *M* if it evaluates to true under the given interpretations over the domain *S.*

M is a *model for the theory T* if all sentences of *T* are true in *M.*

Models (Semantics)

A term t in a model M is interpreted as:

- Variable $x \in V$ is interpreted as x^M
- $f(t_1, \ldots, t_n)$ is interpreted as $f^M(a_1, \ldots, a_n)$,
  - where t_i is interpreted as a_i

An $P(t_1, \ldots, t_n)$ atom in a model M is interpreted as b, where

- $b \leftrightarrow (a_1, \ldots, a_n) \in P^M$
- t_i is interpreted as a_i

Models (Semantics)

A formula φ in a model M is interpreted as:

- $M \models \neg \varphi$ iff $M \not\models \varphi$ (M is not a model for φ)
- $M \models \varphi \leftrightarrow \varphi'$ iff $M \models \varphi$ is equivalent to $M \models \varphi'$
- $M \models \varphi \wedge \varphi'$ iff $M \models \varphi$ and $M \models \varphi'$
- $M \models \varphi \vee \varphi'$ iff $M \models \varphi$ or $M \models \varphi'$
- $M \models \varphi \rightarrow \varphi'$ iff $M \models \varphi$ implies $M \models \varphi'$
- $M \models \forall x.\varphi$ iff for all $s \in S$, $M[x:=s] \models \varphi$
- $M \models \exists x.\varphi$ iff exists $s \in S$, $M[x:=s] \models \varphi$

Interpretation Example

$$
\begin{aligned}
\Sigma &= \{0, +, <\}, \text{ and } M \text{ such that } |M| = \{a, b, c\} \\
M(0) &= a, \\
M(+) &= \{\langle a, a \mapsto a\rangle, \langle a, b \mapsto b\rangle, \langle a, c \mapsto c\rangle, \langle b, a \mapsto b\rangle, \langle b, b \mapsto c\rangle, \\
&\quad \langle b, c \mapsto a\rangle, \langle c, a \mapsto c\rangle, \langle c, b \mapsto a\rangle, \langle c, c \mapsto b\rangle\} \\
M(<) &= \{\langle a, b\rangle, \langle a, c\rangle, \langle b, c\rangle\}
\end{aligned}
$$

If $M(x) = a, M(y) = b, M(z) = c$, then

$M[\![+(+(x, y), z)]\!] =$

$M(+)(M(+)(M(x), M(y)), M(z)) = M(+)(M(+)(a, b), c) =$

$M(+)(b, c) = a$

Interpretation Example

$$
\begin{aligned}
\Sigma &= \{0, +, <\}, \text{ and } M \text{ such that } |M| = \{a, b, c\} \\
M(0) &= a, \\
M(+) &= \{\langle a, a \mapsto a\rangle, \langle a, b \mapsto b\rangle, \langle a, c \mapsto c\rangle, \langle b, a \mapsto b\rangle, \langle b, b \mapsto c\rangle, \\
&\quad \langle b, c \mapsto a\rangle, \langle c, a \mapsto c\rangle, \langle c, b \mapsto a\rangle, \langle c, c \mapsto b\rangle\} \\
M(<) &= \{\langle a, b\rangle, \langle a, c\rangle, \langle b, c\rangle\}
\end{aligned}
$$

$$M \models (\forall x : (\exists y : +(x, y) = 0))$$

$$M \not\models (\forall x : (\exists y : x < y))$$

$$M \models (\forall x : (\exists y : +(x, y) = x))$$

Dreadbury Mansion Mystery

$killed/_2, hates/_2, richer/_2, a/_0, b/_0, c/_0$

$$\exists x \cdot killed(x, a) \tag{1}$$

$$\forall x \cdot \forall y \cdot killed(x, y) \implies (hates(x, y) \wedge \neg richer(x, y)) \tag{2}$$

$$\forall x \cdot hates(a, x) \implies \neg hates(c, x) \tag{3}$$

$$hates(a, a) \wedge hates(a, c) \tag{4}$$

$$\forall x \cdot \neg richer(x, a) \implies hates(b, x) \tag{5}$$

$$\forall x \cdot hates(a, x) \implies hates(b, x) \tag{6}$$

$$\forall x \cdot \exists y \cdot \neg hates(x, y) \tag{7}$$

$$a \neq b \tag{8}$$

Dreadbury Mansion Mystery: Model

$killed/_2, hates/_2, richer/_2, a/_0, b/_0, c/_0$

$$S = \{a, b, c\}$$

$$M(a) = a \qquad M(b) = b$$

$$M(c) = c \qquad M(killed) = \{(a, a)\}$$

$$M(richer) = \{(b, a)\}$$

$$M(hates) = \{(a, a), (a, c)(b, a), (b, c)\}$$

Semantics: Exercise

Drinker's paradox:

There is someone in the pub such that, if he is drinking, everyone in the pub is drinking.

- $\exists x.\ (D(x) \rightarrow \forall y.\ D(y))$

Is this logical formula valid?

Or unsatisfiable?

Or satisfiable but not valid?

Theories

A (first-order) *theory T (over signature Σ) is a set of (deductively closed) sentences (over Σ and V) - axioms*

Let *DC*(Γ) be the deductive closure of a set of sentences Γ.

- For every theory T, *DC*(T) = T

A theory T is *constistent* if *false* $\notin$ *T*

We can view a (first-order) theory *T* as the class of all *models* of *T* (due to completeness of first-order logic).

Theory of Equality T_E

Signature: $\Sigma_E = \{ =, a, b, c, \ldots, f, g, h, \ldots, P, Q, R, \ldots. \}$

=, a binary predicate, interpreted by axioms

all constant, function, and predicate symbols.

Axioms:

1. $\forall x \,.\, x = x$ (reflexivity)
2. $\forall x, y \,.\, x = y \rightarrow y = x$ (symmetry)
3. $\forall x, y, z \,.\, x = y \wedge y = z \rightarrow x = z$ (transitivity)

Theory of Equality T_E

Signature: $\Sigma_E = \{ =, a, b, c, \ldots, f, g, h, \ldots, P, Q, R, \ldots. \}$
=, a binary predicate, interpreted by axioms
all constant, function, and predicate symbols.
Axioms:

4. for each positive integer *n* and *n*-ary function symbol *f*,

$\forall x_1, \ldots, x_n, y_1, \ldots, y_n \,.\, \bigwedge_i x_i = y_i \rightarrow f(x_1, \ldots, x_n) = f(y_1, \ldots, y_n)$ (congruence)

5. for each positive integer *n* and *n*-ary predicate symbol *P*

$\forall x_1, \ldots, x_n, y_1, \ldots, y_n \,.\, \bigwedge_i x_i = y_i \rightarrow (P(x_1, \ldots, x_n) \leftrightarrow P(y_1, \ldots, y_n))$ (equivalence)

Peano Arithmetic (Natural Number) – An Example for a Theory

Signature: $\Sigma_{PA} = \{ 0, 1, +, *, = \}$

Axioms of T_{PA} : axioms for theory of equality, T_E , plus:

1. $\forall x.\ \neg (x+ 1 = 0)$ (zero)
2. $\forall x, y.\ x + 1 = y + 1 \rightarrow x = y$ (successor)
3. $F[0] \ \wedge\ (\forall x.F[x] \rightarrow F[x+ 1]) \rightarrow \forall x.F[x]$ (induction)
4. $\forall x.\ x + 0 = x$ (plus zero)
5. $\forall x, y.\ x + (y + 1) = (x + y) + 1$ (plus successor)
6. $\forall x.\ x * 0 = 0$ (times zero)
7. $\forall x, y.\ x * (y + 1) = x * y + x$ (times successor)

Line 3 is an axiom schema.

Theory of Arrays T_A

Signature: Σ_A = { read, write, = }

read (a, i) is a binary function:

- reads an array a at the index I
- alternative notations:
  - (select a i), a[i]

write (a, i, v) is a ternary function:

- writes a value v to the index i of array a
- alternative notations:
  - (store a i v) , a[i:=v]

Axioms of T_A

Array congruence

- $\forall a, i, j . i = j \rightarrow read(a, i) = read(a, j)$

Read-Over-Write 1

- $\forall a, v, i, j. i = j \rightarrow read(write(a, i, v), j) = v$

Read-Over-Write 2

- $\forall a, v, i, j. i \neq j \rightarrow read(write(a, i, v), j) = read(a, j)$

Extensionality

- $a = b \leftrightarrow \forall i . read(a, i) = read(b, i)$

T-Satisfiability

A formula $\varphi(x)$ is T-satisfiable in a theory T if there is a model of $DC^*(T \cup \exists x.\varphi(x))$. That is, there is a model M for T in which $\varphi(x)$ evaluates to true.

Notation:

$$M \models_T \varphi(x)$$

**DC = deductive closure*

T-Validity

A formula $\varphi(x)$ is T-*valid* in a theory T if $\forall x.\varphi(x) \in T$.

That is, $\forall x.\varphi(x)$ evaluates to *true* in every model M of T.

T-validity:

$$\models_T \varphi(x)$$

Fragment of a Theory

Fragment of a theory *T* is a syntactically restricted subset of formulae of the theory

Example:

- Quantifier-free fragment of theory T is the set of formulae without quantifiers that are valid in T

Often decidable fragments for undecidable theories

Theory *T* is *decidable* if *T*-validity is decidable for every formula *F* of *T*

- There is an algorithm that always terminates with "yes" if *F* is *T*-valid, and "no" if *F* is *T*-unsatisfiable

Exercises (1/2)

Find a model for $P(f(x,y)) \Rightarrow P(g(x,y,x))$

Write an axiom that will restrict that every model has to have exactly three different elements.

Write a FOL formula stating that *i* is the position of the minimal element of an integer array *A*

Write a FOL formula stating that *v* is the minimal element of an integer array *A*

Exercises (1/2)

Find a model for P(f(x,y)) ⇒ P(g(x,y,x))

Write an axiom that will restrict that every model has to have exactly three different elements.

$$(\exists x, y, z \cdot x \neq y \wedge x \neq z \wedge y \neq z) \wedge (\forall a_0, a_1, a_2, a_3 \cdot \bigvee_{0 \leq i < j \leq 3} a_i = a_j)$$

Write a FOL formula stating that *i* is the position of the minimal element of an integer array *A*

$$isIntArray(A) \wedge isInt(i) \wedge 0 \leq i < len(A)$$
$$\forall j \cdot 0 \leq j < len(A) \wedge i \neq j \implies A[i] \leq A[j]$$

Write a FOL formula stating that *v* is the minimal element of an integer array *A*

$$isIntArray(A) \wedge isInt(v)$$
$$\exists i \cdot 0 \leq i < len(A) \wedge A[i] = v$$
$$\forall i \cdot 0 \leq i < len(A) \implies A[i] \leq v$$

Exercises (2/2)

Show whether the following sentence is valid or not

$$(\exists x \cdot P(x) \vee Q(x)) \iff (\exists x \cdot P(x)) \vee (\exists x \cdot Q(x))$$

Show whether the following FOL sentence is valid or not

$$(\exists x \cdot P(x) \wedge Q(x)) \iff (\exists x \cdot P(x)) \wedge (\exists x \cdot Q(x))$$

Exercises (2/2)

Show whether the following sentence is valid or not

$$(\exists x \cdot P(x) \vee Q(x)) \iff (\exists x \cdot P(x)) \vee (\exists x \cdot Q(x))$$

- Valid. Prove by contradiction that every model M of the LHS is a model of the RHS and vice versa.

Show whether the following FOL sentence is valid or not

$$(\exists x \cdot P(x) \wedge Q(x)) \iff (\exists x \cdot P(x)) \wedge (\exists x \cdot Q(x))$$

- Not valid. Prove by constructing a model M of the RHS that is not a model of the LHS. For example, S = {0,1}, M(P) = { 0 }, and M(Q) = { 1 }

Completeness, Compactness, Incompleteness

Gödel Completeness Theorem of FOL

- any (first-order) formula which is true in *all* models of a theory, must be logically deducible from that theory, and vice versa

Corollary: Compactness Theorem

- A FOL theory G is SAT iff every finite subset G' of G is SAT
- A set G of FOL sentences is UNSAT iff exists a finite subset G' of G that is UNSAT

Incompleteness of FOL Theories

- A theory is *consistent* if it is impossible to prove both *p* and ~*p* for any sentence *p* in the signature of the theory
- A theory is *complete* if for every sentence *p* it includes either *p* or ~*p*
- There are FOL theories that are consistent but incomplete

https://terrytao.wordpress.com/2009/04/10/the-completeness-and-compactness-theorems-of-first-order-logic/

https://plato.stanford.edu/entries/goedel-incompleteness/

SMT SOLVERS

Satisfiability Modulo Theory (SMT)

Satisfiability is the problem of determining wither a formula F has a model

- if F is ***propositional***, a model is a truth assignment to Boolean variables
- if F is ***first-order formula***, a model assigns values to variables and interpretation to all the function and predicate symbols

SAT Solvers

- check satisfiability of propositional formulas

SMT Solvers

- check satisfiability of formulas in a ***decidable*** first-order theory (e.g., linear arithmetic, uninterpreted functions, array theory, bit-vectors)

Background Reading: SMT

September 2011

UNIVERSITY OF WATERLOO

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-
nning 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
sitional 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*
or its variables. Some problems are more naturally
d 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

Example

$$b + 2 = c \wedge f(\texttt{read}(\texttt{write}(a, b, 3), c - 2)) \neq f(c - b + 1)$$

Example

$$b + 2 = c \wedge f(\texttt{read}(\texttt{write}(a, b, 3), c - 2)) \neq f(c - b + 1)$$

Arithmetic

Example

$$b + 2 = c \wedge f(\texttt{read}(\texttt{write}(a, b, 3), c - 2)) \neq f(c - b + 1)$$

Array theory

Example

$$b + 2 = c \wedge f(\texttt{read}(\texttt{write}(a, b, 3), c - 2)) \neq f(c - b + 1)$$

Uninterpreted function

Example

$$b + 2 = c \wedge f(\texttt{read}(\texttt{write}(a, b, 3), c - 2)) \neq f(c - b + 1)$$

Example

$$b + 2 = c \wedge f(\texttt{read}(\texttt{write}(a, b, 3), c - 2)) \neq f(c - b + 1)$$

By arithmetic, this is equivalent to

$$b + 2 = c \wedge f(\texttt{read}(\texttt{write}(a, b, 3), b)) \neq f(3)$$

Example

$$b + 2 = c \wedge f(\texttt{read}(\texttt{write}(a, b, 3), c - 2)) \neq f(c - b + 1)$$

By arithmetic, this is equivalent to

$$b + 2 = c \wedge f(\texttt{read}(\texttt{write}(a, b, 3), b)) \neq f(3)$$

then, by the array theory axiom: $\texttt{read}(\texttt{write}(v, i, x), i) = x$

$$b + 2 = c \wedge f(3) \neq f(3)$$

Example

$$b + 2 = c \wedge f(\texttt{read}(\texttt{write}(a, b, 3), c - 2)) \neq f(c - b + 1)$$

By arithmetic, this is equivalent to

$$b + 2 = c \wedge f(\texttt{read}(\texttt{write}(a, b, 3), b)) \neq f(3)$$

then, by the array theory axiom: $\texttt{read}(\texttt{write}(v, i, x), i) = x$

$$b + 2 = c \wedge f(3) \neq f(3)$$

then, the formula is unsatisfiable

Example 2

$$x \geq 0 \wedge f(x) \geq 0 \wedge y \geq 0 \wedge f(y) \geq 0 \wedge x \neq y$$

Example 2

$$x \geq 0 \wedge f(x) \geq 0 \wedge y \geq 0 \wedge f(y) \geq 0 \wedge x \neq y$$

This formula is satisfiable

Example 2

$$x \geq 0 \wedge f(x) \geq 0 \wedge y \geq 0 \wedge f(y) \geq 0 \wedge x \neq y$$

This formula is satisfiable:

Example model:

$$x \rightarrow 1$$

$$y \rightarrow 2$$

$$f(1) \rightarrow 0$$

$$f(2) \rightarrow 1$$

$$f(\ldots) \rightarrow 0$$

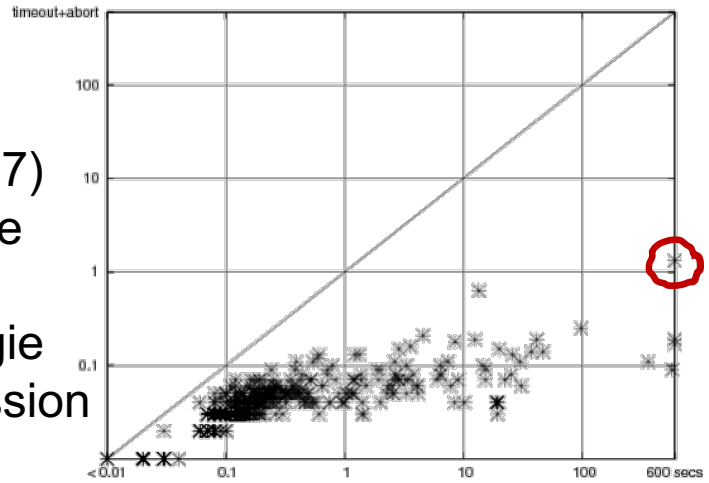
SMT - Milestones

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:

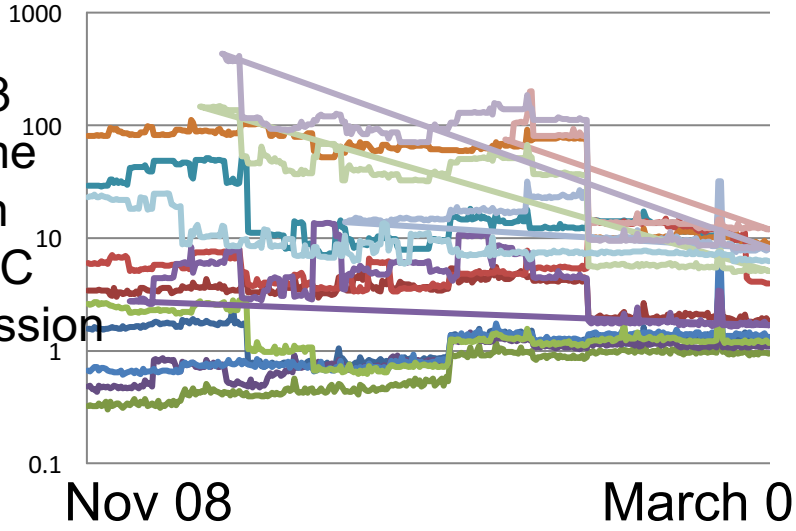
15KLOC + 285KLOC = Z3

Z3 (of '07) Time On Boogie Regression



Simplify (of '01) time

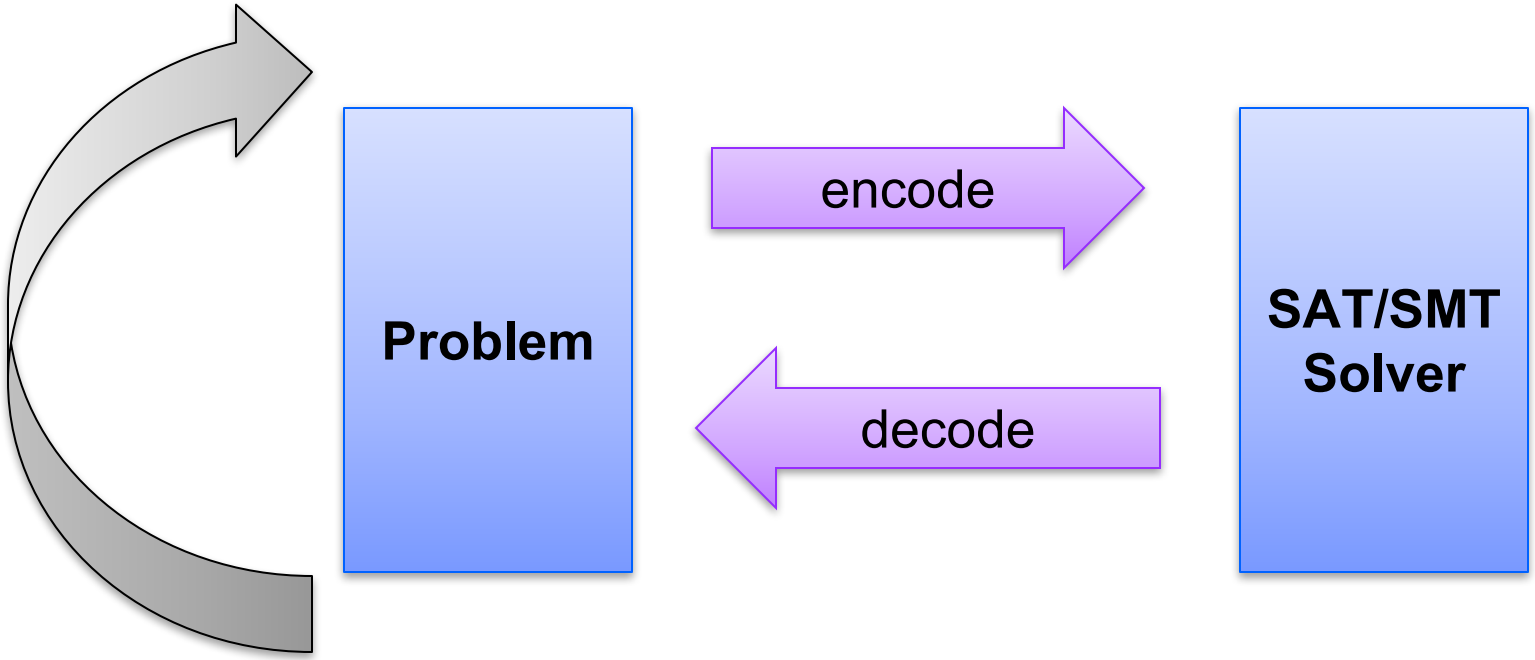
Z3 Time On VCC Regression



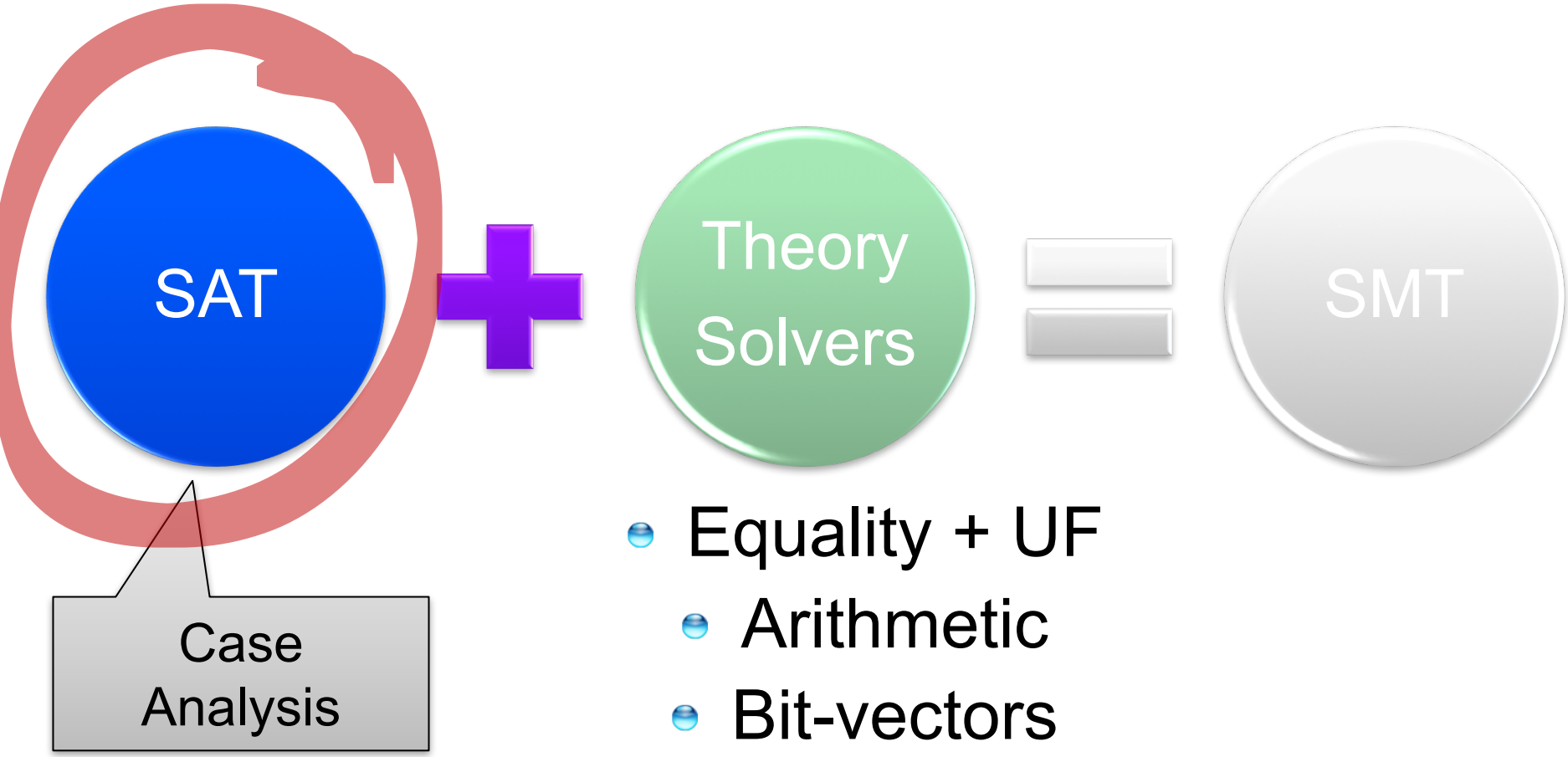
SAT/SMT Revolution

Solve any computational problem by effective reduction to SAT/SMT

- iterate as necessary



SMT : Basic Architecture



- Equality + UF
  - Arithmetic
  - Bit-vectors
    - …

SAT + Theory solvers

Basic Idea

$x \geq 0,\ y = x + 1,\ (y > 2 \vee y < 1)$

Abstract (aka “naming” atoms)

$p_1,\ p_2,\ (p_3 \vee p_4)$ $\quad p_1 \equiv (x \geq 0),\ p_2 \equiv (y = x + 1),$
$p_3 \equiv (y > 2),\ p_4 \equiv (y < 1)$

SAT + Theory solvers

Basic Idea

$x \geq 0,\ y = x + 1,\ (y > 2 \vee y < 1)$

Abstract (aka "naming" atoms)

$p_1,\ p_2,\ (p_3 \vee p_4)$ $\quad p_1 \equiv (x \geq 0),\ p_2 \equiv (y = x + 1),$
$p_3 \equiv (y > 2),\ p_4 \equiv (y < 1)$

SAT Solver

SAT + Theory solvers

Basic Idea

$x \geq 0, y = x + 1, (y > 2 \vee y < 1)$

Abstract (aka "naming" atoms)

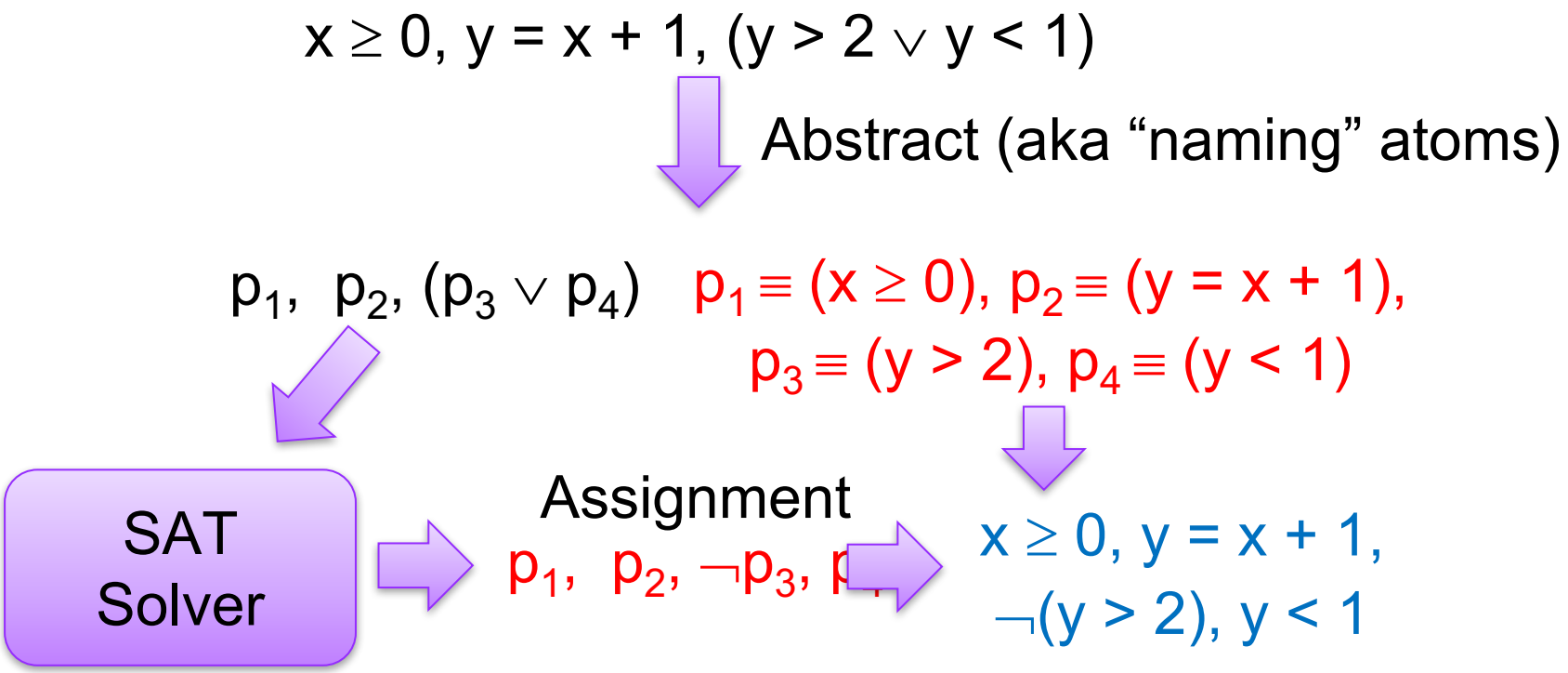
$p_1, p_2, (p_3 \vee p_4)$ $p_1 \equiv (x \geq 0), p_2 \equiv (y = x + 1),$
$p_3 \equiv (y > 2), p_4 \equiv (y < 1)$

SAT Solver

Assignment
$p_1, p_2, \neg p_3, p_4$

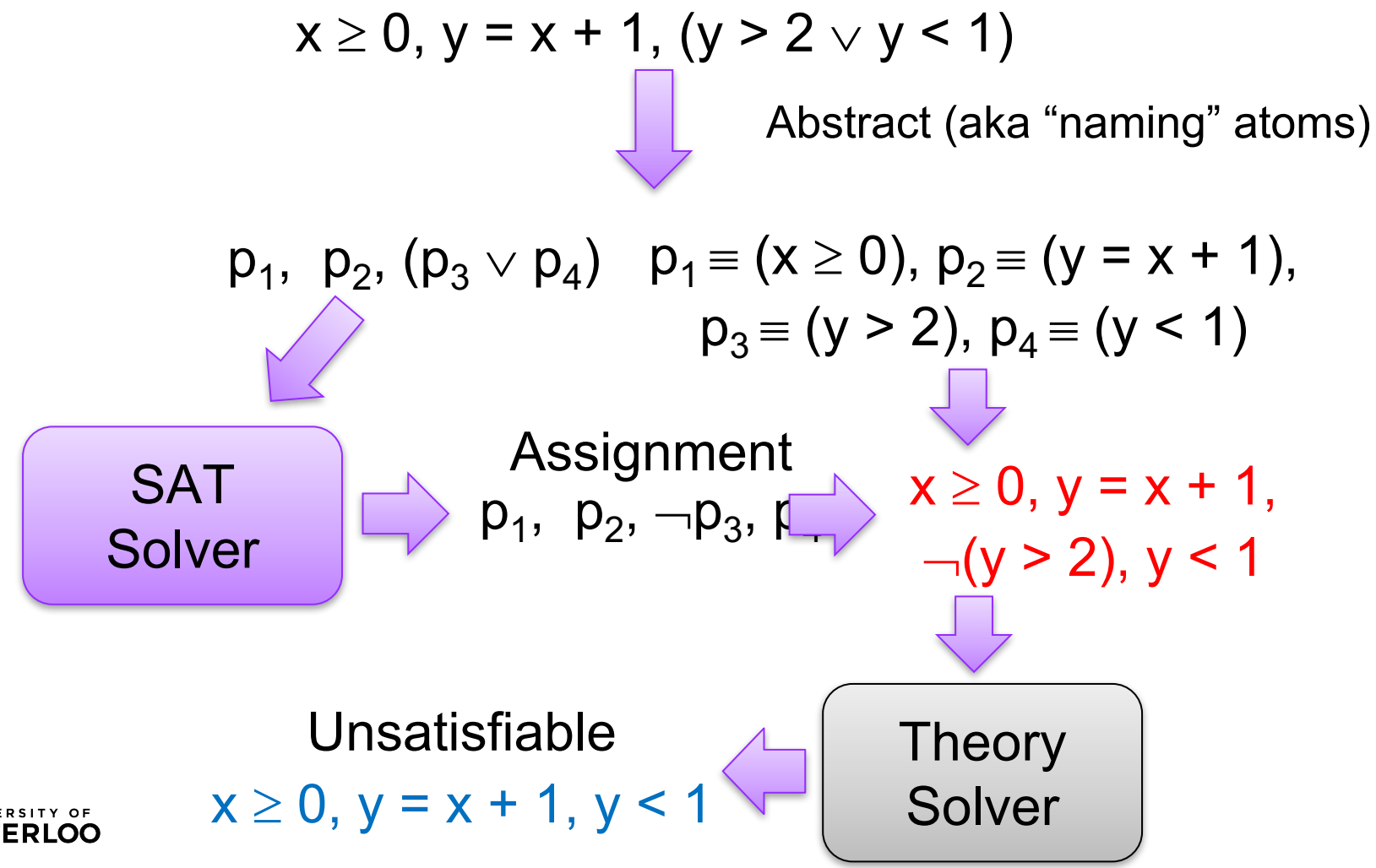
SAT + Theory solvers

Basic Idea



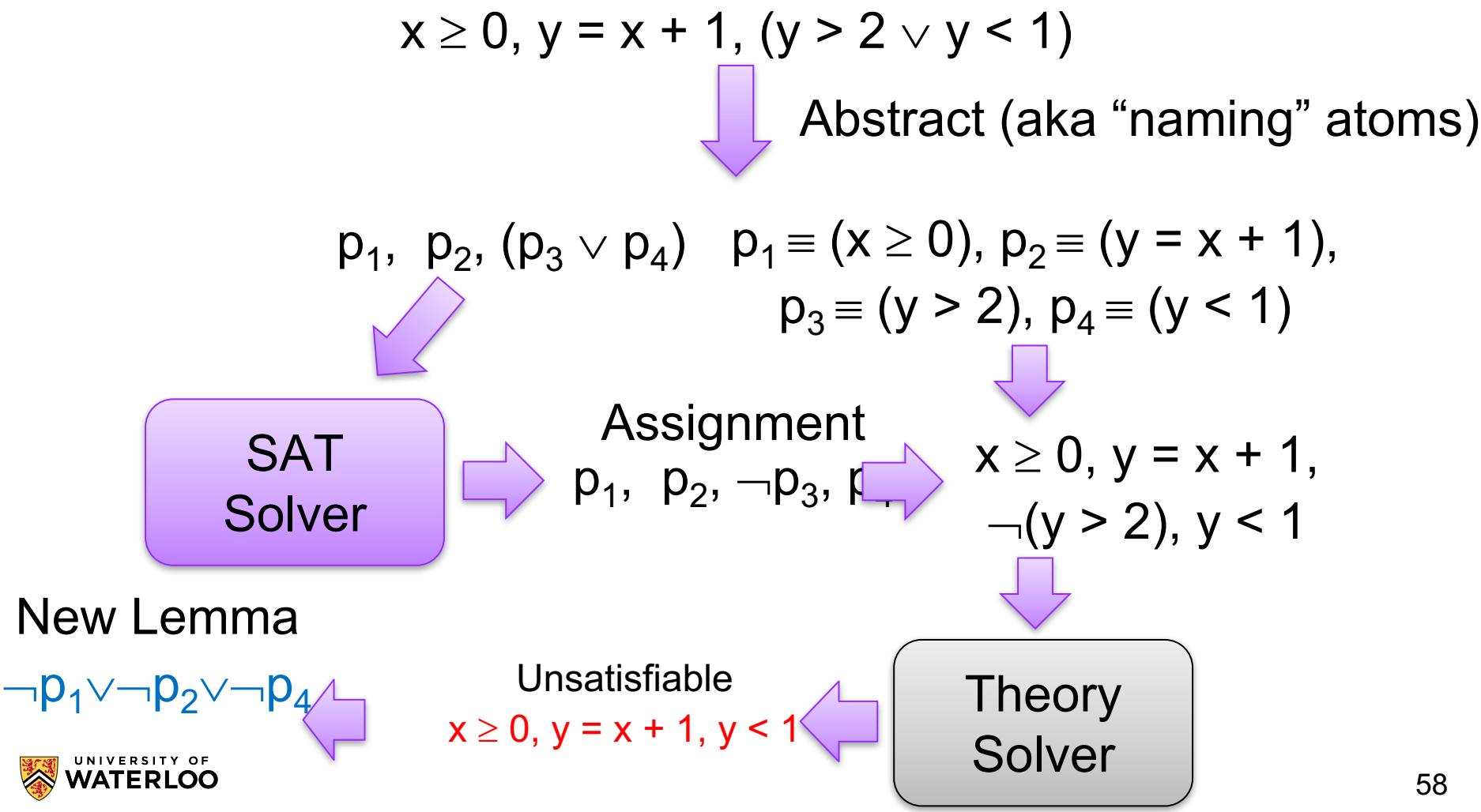
SAT + Theory solvers

Basic Idea

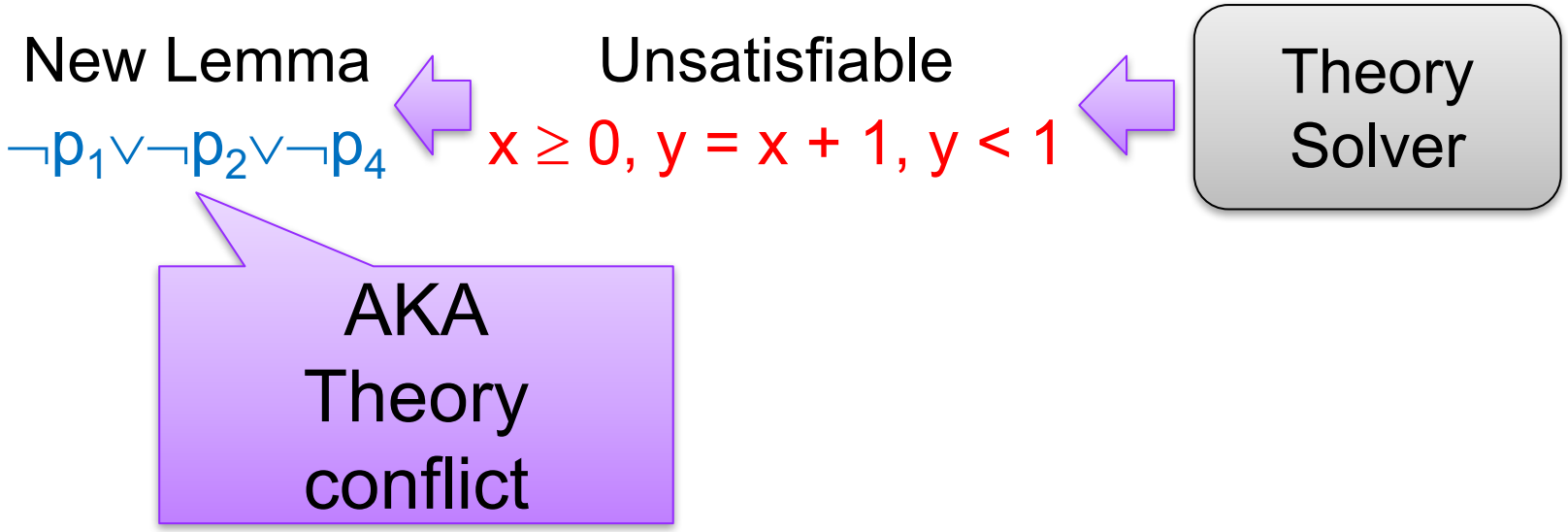


SAT + Theory solvers

Basic Idea



SAT + Theory solvers



USING Z3 AND Z3PY

SMT-LIB: http://smt-lib.org

International initiative for facilitating research and development in SMT

Provides rigorous definition of syntax and semantics for theories

SMT-LIB syntax

- based on s-expressions (LISP-like)
- common syntax for interpreted functions of different theories
  - e.g. (and (= x y) (<= (* 2 x) z))
- commands to interact with the solver
  - (declare-fun …) declares a constant/function symbol
  - (assert p) conjoins formula p to the curent context
  - (check-sat) checks satisfiability of the current context
  - (get-model) prints current model (if the context is satisfiable)

- see examples at http://rise4fun.com/z3

SMT-LIB Syntax

```
(declare-fun x () Int)
(declare-fun y () Int)
(declare-fun z () Int)
(assert (>= (* 2 x) (+ y z)))
(declare-fun f (Int) Int)
(declare-fun g (Int Int) Int)
(assert (< (f x) (g x x)))
(assert (> (f y) (g x x)))
(check-sat)
(get-model)
```

SMT Example

z3 Microsoft Research

Is this formula satisfiable?

```
1  ; This example illustrates basic arithmetic and
2  ; uninterpreted functions
3
4  (declare-fun x () Int)
5  (declare-fun y () Int)
6  (declare-fun z () Int)
7  (assert (>= (* 2 x) (+ y z)))
8  (declare-fun f (Int) Int)
9  (declare-fun g (Int Int) Int)
10 (assert (< (f x) (g x x)))
11 (assert (> (f y) (g x x)))
12 (check-sat)
13 (get-model)
14 (push)
15 (assert (= x y))
16 (check-sat)
17 (pop)
18 (exit)
19
```

http://rise4fun.com/z3

Example

$$b + 2 = c \wedge f(\texttt{read}(\texttt{write}(a, b, 3), c - 2)) \neq f(c - b + 1)$$

Z3

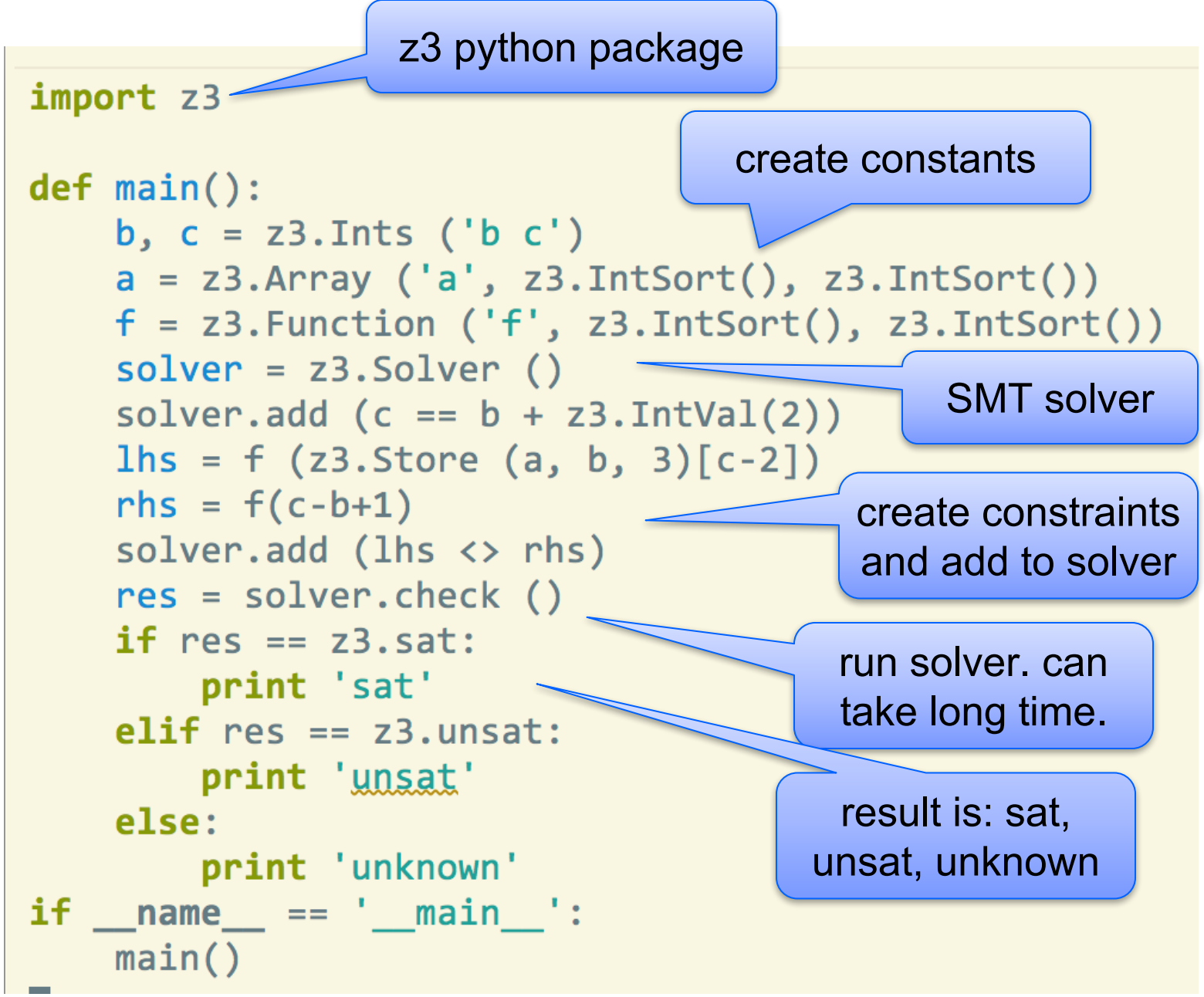
Microsoft Research

Is this formula satisfiable?

```
1 ;; Is this formula satisfiable?
2 (declare-fun b () Int)
3 (declare-fun c () Int)
4 (declare-fun a () (Array Int Int))
5 (declare-fun f (Int) Int)
6 (assert (= (+ b 2) c))
7 (assert (not (= (f (select (store a b 3) (- c 2))) (f (+ (- c b) 1)))))
8 (check-sat)
```

```
import z3

def main():
    b, c = z3.Ints ('b c')
    a = z3.Array ('a', z3.IntSort(), z3.IntSort())
    f = z3.Function ('f', z3.IntSort(), z3.IntSort())
    solver = z3.Solver ()
    solver.add (c == b + z3.IntVal(2))
    lhs = f (z3.Store (a, b, 3)[c-2])
    rhs = f(c-b+1)
    solver.add (lhs <> rhs)
    res = solver.check ()
    if res == z3.sat:
        print 'sat'
    elif res == z3.unsat:
        print 'unsat'
    else:
        print 'unknown'
if __name__ == '__main__':
    main()
```




```
import z3

def main():
    b, c = z3.Ints ('b c')
    a = z3.Array ('a', z3.IntSort(), z3.IntSort())
    f = z3.Function ('f', z3.IntSort(), z3.IntSort())
    solver = z3.Solver ()
    solver.add (c == b + z3.IntVal(2))
    lhs = f (z3.Store (a, b, 3)[c-2])
    rhs = f(c-b+1)
    solver.add (lhs <> rhs)
    res = solver.check ()
    if res == z3.sat:
        print 'sat'
    elif res == z3.unsat:
        print 'unsat'
    else:
        print 'unknown'
if __name__ == '__main__':
    main()
```

Useful Z3Py Functions

All these functions are under python package `z3`

Create constants and values

- `Int(name)` – an integer constant with a given name
- `FreshInt(name)` – unique constant starting with name
- `IntVal(v), BoolVal(v)` – integer and boolean values

Arithmetic functions and predicates

- `+,-,/,<,<=,>,>=,==,` etc.
- Distinct(a, b, …) – the arugments are distinct (expands to many disequalities)

Propositional operators

- `And, Or, Not`

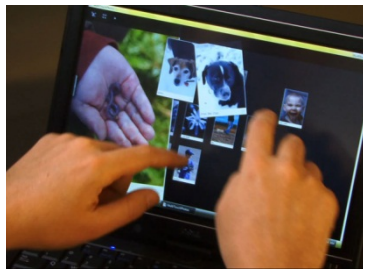
Methods of the `z3.Solver` class

- `add(phi)` – add formula phi to the solver
- `check()` – returns z3.sat, z3.unsat, or z3.unknown (on failure to solve)
- `model()` – model if the result is sat

Methods of `z3.Model` class

- `eval(phi)` – returns the value of phi in the model

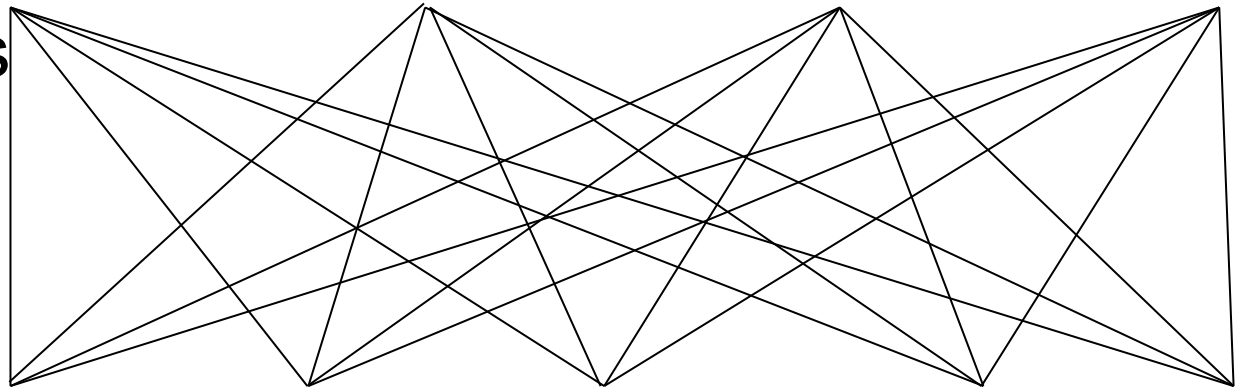
Job Shop Scheduling



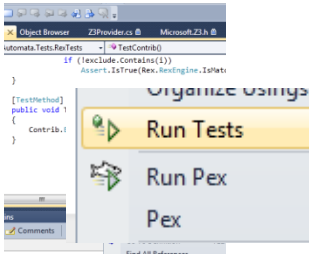
Machines

Tasks

Jobs



P = NP?

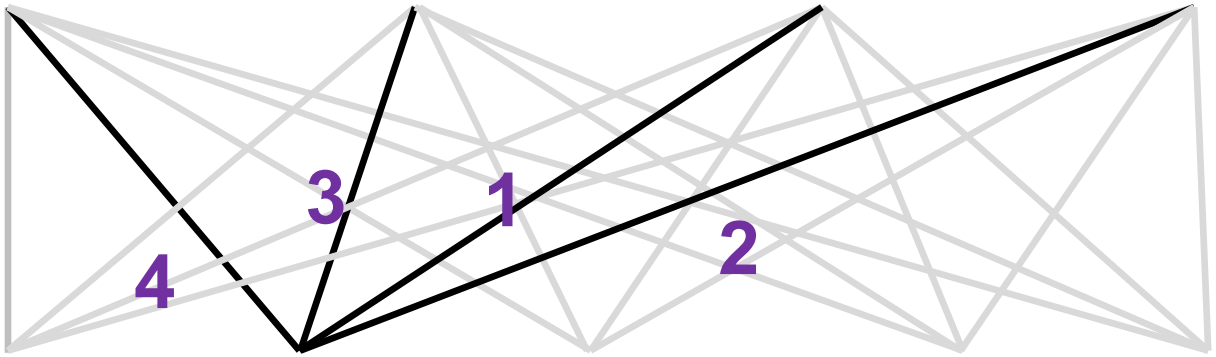


$$\zeta(s) = 0 \Rightarrow s = \frac{1}{2} + ir$$

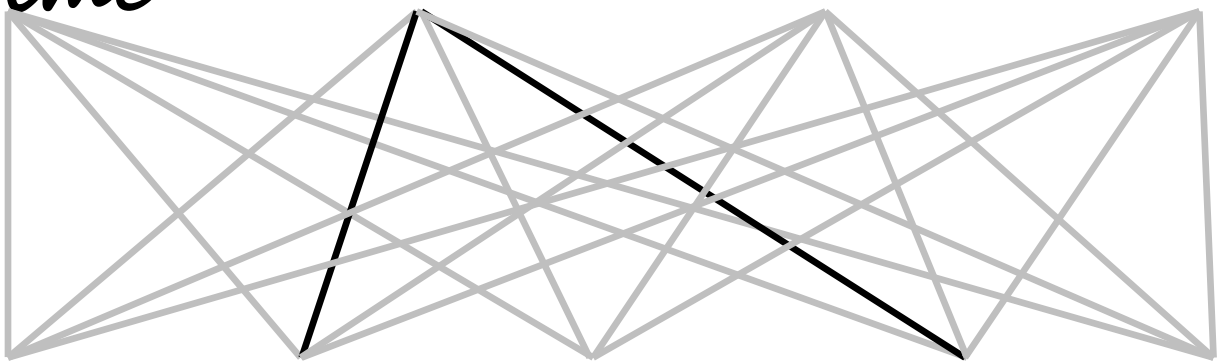
Job Shop Scheduling

Constraints:

Precedence: between two tasks of the same job



Resource: Machines execute at most one job at a time

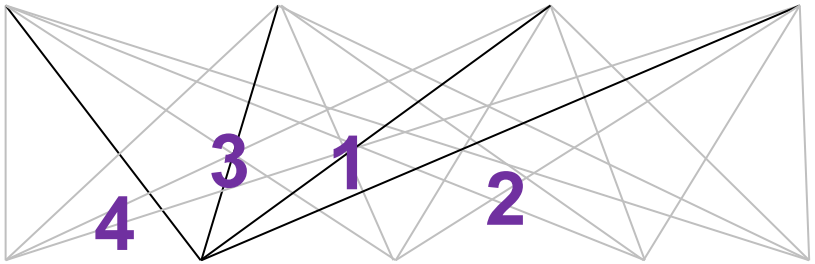


$$[start_{2,2}..end_{2,2}] \cap [start_{4,2}..end_{4,2}] = \emptyset$$

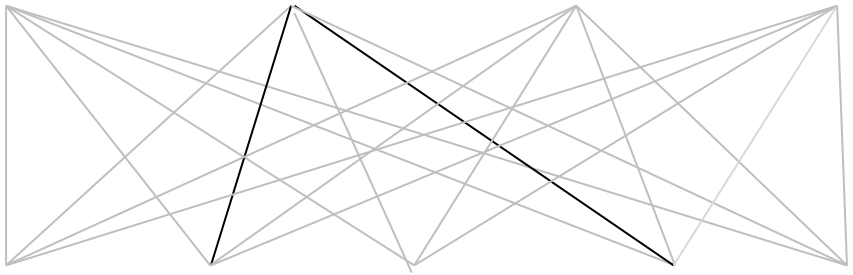
Job Shop Scheduling

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

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

Bit Tricks

Let `x, y` be a 32 bit machine integers (a bit-vector)

Show that `x!=0 && !(x & (x-1))` is true iff x is a power of 2

Show that x and y have different signs iff `x^y < 0`

Dog, Cat, Mouse

Spend exactly 100 dollars and buy exactly 100 animals.

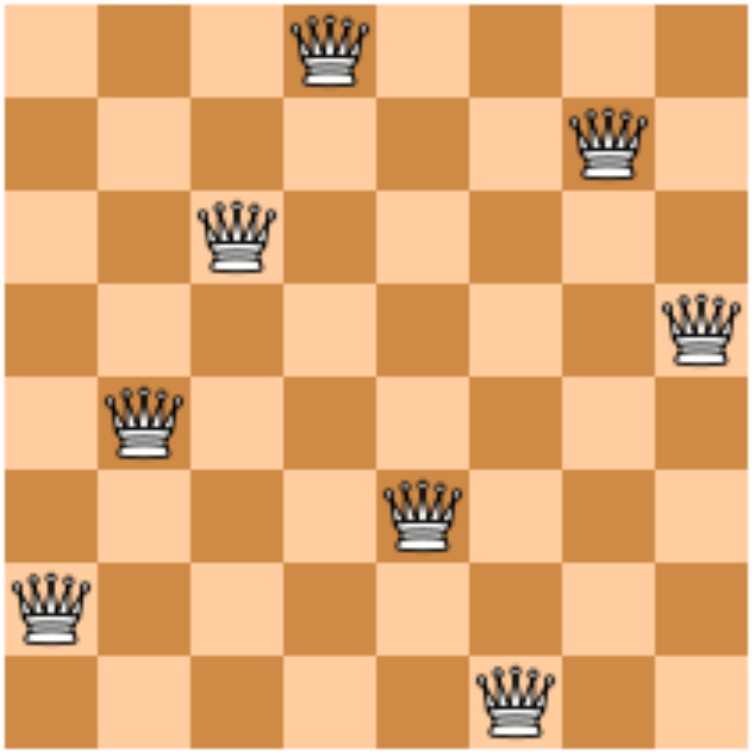
- Dogs cost 15 dollars,
- cats cost 1 dollar,
- and mice cost 25 cents each.

You have to buy at least one of each.

How many of each should you buy?

Eight Queens Problem

Place 8 queens on an 8x8 chess board so that no two queen attacks one another



Incremental Interface

Z3 provides two interfaces for incremental solving that allow for adding and removing constraints

- push/pop, and assumptions

Constraints can be added at any time. This is not called incremental ☺

Push/Pop Interface

- Store current solver state by a call to push
  - `s.push()` in Python, and `(push)` in SMT-LIB
- Restore previous state by a call to pop
  - `s.pop()` in Python and `(pop)` in SMT-LIB

Incremental Interface: Assumptions

Requires two steps, but much more flexible than push/pop

1. tag constraints by fresh Boolean constants
   - e.g., use `(assert (=> p phi))` instead of `(assert phi)`
2. during check-sat, enable constraints by forcing tags to be true
   - e.g., use (check-sat p)

For example,

```
(assert (=> a0 c0))
(assert (=> a1 c1))
(assert (=> a2 c2))
(check-sat a0)          ; check whether c0 is sat
(check-sat a0 a2)       ; check whether c0 and c2 are sat
(check-set a1 a2)       ; check whether c1 and c3 are sat
```

Assumptions in Python Interface

Methods of `z3.Solver` class

- `check(self, *assumptions)` – check with assumptions
- `unsat_core(self)` – if the last call to `check` was `unsat`, returns the subset of assumptions that were actually used to show `unsat`