Satisfiability with and without Theories

KR 2010, Toronto

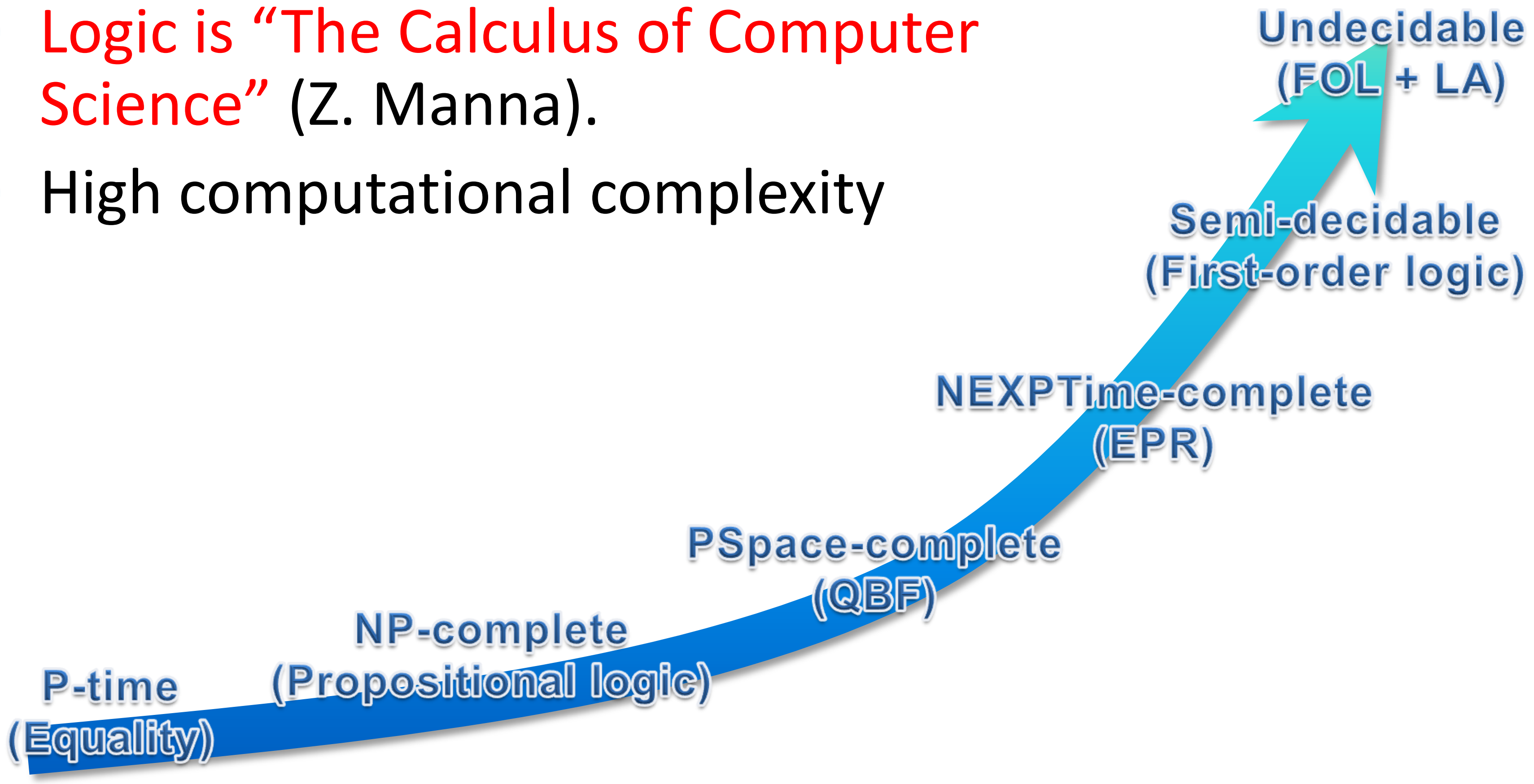
Leonardo de Moura
Microsoft Research

Symbolic Reasoning

Verification/Analysis tools need some form of **Symbolic Reasoning**

Symbolic Reasoning

- Logic is "The Calculus of Computer Science" (Z. Manna).
- High computational complexity



Applications

Test case generation

Verifying Compilers

Predicate Abstraction

Invariant Generation

Type Checking

Model Based Testing

Some Applications @ Microsoft

The Spec# Programming System

HAVOC

FormuLa

Hyper-V
Microsoft | Virtualization

Terminator T-2

VCC

SLAM
i!=node->(); i ++ visit procs.end() node){

Yogi

NModel

Vigilante

SpecExplorer

Pex

F7

SAGE

Microsoft Research

Test case generation

```
unsigned GCD(x, y) {
  requires(y > 0);
  while (true) {
     unsigned m = x % y;
      if (m == 0) return y;
      x = y;
      y = m;
  }
}
```

SSA

$(y_0 > 0)$ and

$(m_0 = x_0 \% y_0)$ and

not $(m_0 = 0)$ and

$(x_1 = y_0)$ and

$(y_1 = m_0)$ and

$(m_1 = x_1 \% y_1)$ and

$(m_1 = 0)$

Solver

$x_0 = 2$

$y_0 = 4$

$m_0 = 2$

$x_1 = 4$

$y_1 = 2$

$m_1 = 0$

We want a trace where the loop is executed twice.

Microsoft Research

Type checking

Signature:

div : int, { x : int | x $\neq$ 0 } $\rightarrow$ int

Subtype

Call site:

if a $\leq$ 1 and a $\leq$ b then

      return div(a, b)

Verification condition

a $\leq$ 1 and a $\leq$ b implies b $\neq$ 0

Microsoft Research

What is logic?

- Logic is the art and science of effective reasoning.
- How can we draw general and reliable conclusions from a collection of facts?
- Formal logic: Precise, syntactic characterizations of well-formed expressions and valid deductions.
- Formal logic makes it possible to calculate consequences at the symbolic level.

- Computers can be used to automate such symbolic calculations.

What is logic?

- Logic studies the relationship between language, meaning, and (proof) method.
- A logic consists of a language in which (well-formed) sentences are expressed.
- A semantic that distinguishes the valid sentences from the refutable ones.
- A proof system for constructing arguments justifying valid sentences.
- Examples of logics include propositional logic, equational logic, first-order logic, higher-order logic, and modal logics.

What is logical language?

- A language consists of logical symbols whose interpretations are fixed, and non-logical ones whose interpretations vary.
- These symbols are combined together to form well-formed formulas.
- In propositional logic PL, the connectives $\wedge$, $\vee$, and $\neg$ have a fixed interpretation, whereas the constants p, q, r may be interpreted at will.

Propositional Logic

Formulas: $\varphi := p \mid \varphi_1 \vee \varphi_2 \mid \varphi_1 \wedge \varphi_2 \mid \neg \varphi_1 \mid \varphi_1 \Rightarrow \varphi_2$

Examples:

$p \vee q \Rightarrow q \vee p$

$p \wedge \neg q \wedge (\neg p \vee q)$

We say p and q are propositional variables.

Exercise: Using a programming language, define a representation for formulas and a checker for well-formed formulas.

Microsoft Research

Interpretation

An interpretation $\mathcal{M}$ assigns truth values $\{\top, \bot\}$ to propositional variables.

Let A and B range over PL formulas.

$\mathcal{M}[\![\phi]\!]$ is the meaning of ϕ in $\mathcal{M}$ and is computed using *truth tables*:

ϕ	A	B	$\neg A$	$A \vee B$	$A \wedge \neg A$	$A \Rightarrow B$	$A \Rightarrow (B \vee A)$
$\mathcal{M}_1(\phi)$	$\bot$	$\bot$	$\top$	$\bot$	$\bot$	$\top$	$\top$
$\mathcal{M}_2(\phi)$	$\bot$	$\top$	$\top$	$\top$	$\bot$	$\top$	$\top$
$\mathcal{M}_3(\phi)$	$\top$	$\bot$	$\bot$	$\top$	$\bot$	$\bot$	$\top$
$\mathcal{M}_4(\phi)$	$\top$	$\top$	$\bot$	$\top$	$\bot$	$\top$	$\top$

Satisfiability & Validity

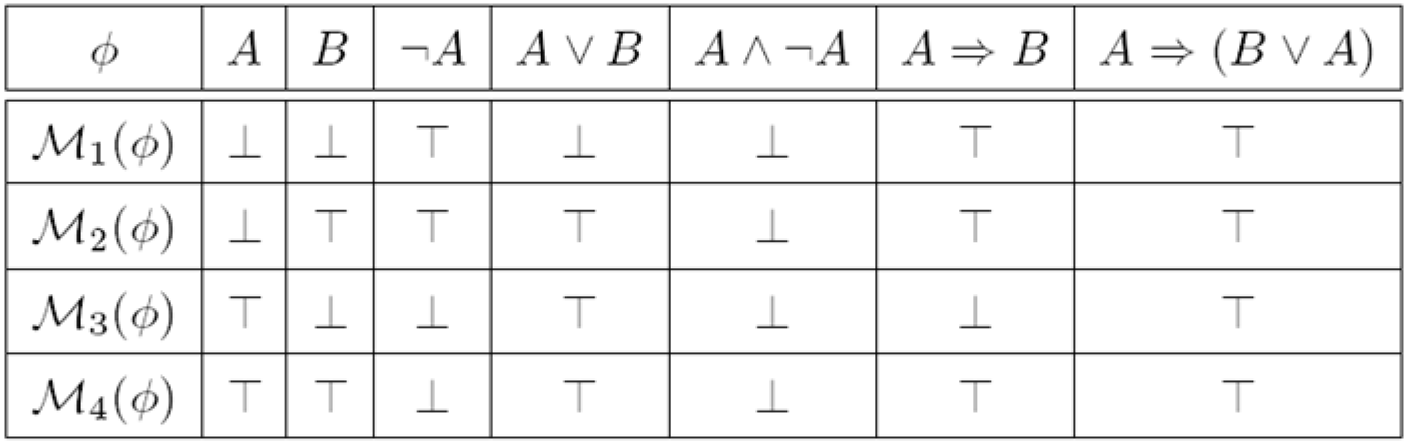
- A formula is satisfiable if it has an interpretation that makes it logically true.
- In this case, we say the interpretation is a model.
- A formula is unsatisfiable if it does not have any model.
- A formula is valid if it is logically true in any interpretation.
- A propositional formula is valid if and only if its negation is unsatisfiable.

Satisfiability & Validity: examples

$p \lor q \Rightarrow q \lor p$

$p \lor q \Rightarrow q$

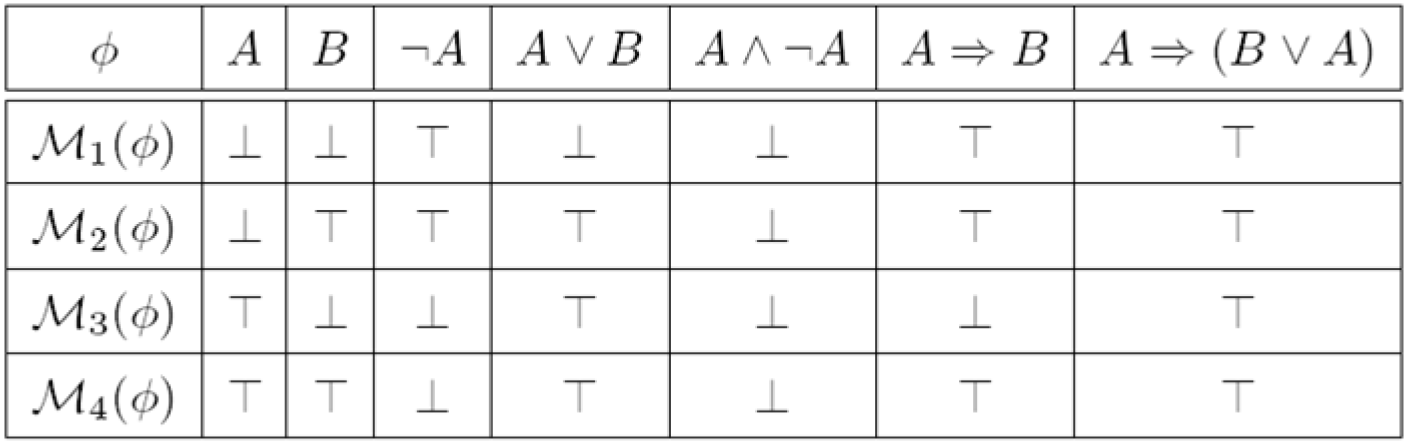
$p \land \neg q \;\; \land (\neg p \lor q)$



ϕ	A	B	$\neg A$	$A \lor B$	$A \land \neg A$	$A \Rightarrow B$	$A \Rightarrow (B \lor A)$
$\mathcal{M}_1(\phi)$	$\bot$	$\bot$	$\top$	$\bot$	$\bot$	$\top$	$\top$
$\mathcal{M}_2(\phi)$	$\bot$	$\top$	$\top$	$\top$	$\bot$	$\top$	$\top$
$\mathcal{M}_3(\phi)$	$\top$	$\bot$	$\bot$	$\top$	$\bot$	$\bot$	$\top$
$\mathcal{M}_4(\phi)$	$\top$	$\top$	$\bot$	$\top$	$\bot$	$\top$	$\top$

Satisfiability & Validity: examples

$p \vee q \Rightarrow q \vee p$	VALID
$p \vee q \Rightarrow q$	SATISFIABLE
$p \wedge \neg q \;\; \wedge (\neg p \vee q)$	UNSATISFIABLE



ϕ	A	B	$\neg A$	$A \vee B$	$A \wedge \neg A$	$A \Rightarrow B$	$A \Rightarrow (B \vee A)$
$\mathcal{M}_1(\phi)$	$\bot$	$\bot$	$\top$	$\bot$	$\bot$	$\top$	$\top$
$\mathcal{M}_2(\phi)$	$\bot$	$\top$	$\top$	$\top$	$\bot$	$\top$	$\top$
$\mathcal{M}_3(\phi)$	$\top$	$\bot$	$\bot$	$\top$	$\bot$	$\bot$	$\top$
$\mathcal{M}_4(\phi)$	$\top$	$\top$	$\bot$	$\top$	$\bot$	$\top$	$\top$

Equivalence

Two formulas A and B are equivalent, $A \iff B$, if their truth values agree in each interpretation.

Exercise 2 *Prove that the following are equivalent*

1. $\neg\neg A \iff A$
2. $A \Rightarrow B \iff \neg A \vee B$
3. $\neg(A \wedge B) \iff \neg A \vee \neg B$
4. $\neg(A \vee B) \iff \neg A \wedge \neg B$
5. $\neg A \Rightarrow B \iff \neg B \Rightarrow A$

Equisatisfiable

We say formulas *A* and *B* are equisatisfiable if and only if *A* is satisfiable if and only if *B* is.

During this course, we will describe transformations that preserve equivalence and equisatisfiability.

Normal Forms

A formula where negation is applied only to propositional atoms is said to be in negation normal form (NNF).

A literal is either a propositional atom or its negation.

A formula that is a multiary conjunction of multiary disjunctions of literals is in conjunctive normal form (CNF).

A formula that is a multiary disjunction of multiary conjunctions of literals is in disjunctive normal form (DNF).

Exercise 3 *Show that every propositional formula is equivalent to one in NNF, CNF, and DNF.*

Exercise 4 *Show that every n-ary Boolean function can be expressed using just $\neg$ and $\vee$.*

Normal Forms

NNF?

$(p \lor \neg q) \land (q \lor \neg(r \land \neg p))$

Normal Forms

NNF? NO

$(p \vee \neg q) \wedge (q \vee \neg(r \wedge \neg p))$

Normal Forms

NNF? NO

$(p \vee \neg q) \wedge (q \vee \neg(r \wedge \neg p))$

1. $\neg\neg A \iff A$
2. $A \Rightarrow B \iff \neg A \vee B$
3. $\neg(A \wedge B) \iff \neg A \vee \neg B$
4. $\neg(A \vee B) \iff \neg A \wedge \neg B$

Normal Forms

NNF? NO

$(p \lor \neg q) \land (q \lor \neg(r \land \neg p))$

$\Leftrightarrow$

$(p \lor \neg q) \land (q \lor (\neg r \lor \neg\neg p))$

1. $\neg\neg A \iff A$
2. $A \Rightarrow B \iff \neg A \lor B$
3. $\neg(A \land B) \iff \neg A \lor \neg B$
4. $\neg(A \lor B) \iff \neg A \land \neg B$

Normal Forms

NNF? NO

$(p \vee \neg q) \wedge (q \vee \neg(r \wedge \neg p))$

$\Leftrightarrow$

$(p \vee \neg q) \wedge (q \vee (\neg r \vee \neg\neg p))$

$\Leftrightarrow$

$(p \vee \neg q) \wedge (q \vee (\neg r \vee p))$

1. $\neg\neg A \iff A$
2. $A \Rightarrow B \iff \neg A \vee B$
3. $\neg(A \wedge B) \iff \neg A \vee \neg B$
4. $\neg(A \vee B) \iff \neg A \wedge \neg B$

Normal Forms

CNF?

$((p \wedge s) \vee (\neg q \wedge r)) \wedge (q \vee \neg p \vee s) \wedge (\neg r \vee s)$

Normal Forms

CNF? NO

$((p \wedge s) \vee (\neg q \wedge r)) \wedge (q \vee \neg p \vee s) \wedge (\neg r \vee s)$

Normal Forms

CNF? NO

$((p \wedge s) \vee (\neg q \wedge r)) \wedge (q \vee \neg p \vee s) \wedge (\neg r \vee s)$

Distributivity

1. $A \vee (B \wedge C) \Leftrightarrow (A \vee B) \wedge (A \vee C)$
2. $A \wedge (B \vee C) \Leftrightarrow (A \wedge B) \vee (A \wedge C)$

Normal Forms

CNF? NO

$((p \wedge s) \vee (\neg q \wedge r)) \wedge (q \vee \neg p \vee s) \wedge (\neg r \vee s)$

$\Leftrightarrow$

$((p \wedge s) \vee \neg q)) \wedge ((p \wedge s) \vee r)) \wedge (q \vee \neg p \vee s) \wedge (\neg r \vee s)$

Distributivity

1. $A \vee (B \wedge C) \Leftrightarrow (A \vee B) \wedge (A \vee C)$
2. $A \wedge (B \vee C) \Leftrightarrow (A \wedge B) \vee (A \wedge C)$

Normal Forms

CNF? NO

$((p \land s) \lor (\neg q \land r)) \land (q \lor \neg p \lor s) \land (\neg r \lor s)$

$\Leftrightarrow$

$((p \land s) \lor \neg q)) \land ((p \land s) \lor r)) \land (q \lor \neg p \lor s) \land (\neg r \lor s)$

$\Leftrightarrow$

$(p \lor \neg q) \land (s \lor \neg q) \land ((p \land s) \lor r)) \land (q \lor \neg p \lor s) \land (\neg r \lor s)$

Distributivity

1. $A \lor (B \land C) \Leftrightarrow (A \lor B) \land (A \lor C)$
2. $A \land (B \lor C) \Leftrightarrow (A \land B) \lor (A \land C)$

Normal Forms

CNF? NO

$((p \land s) \lor (\neg q \land r)) \land (q \lor \neg p \lor s) \land (\neg r \lor s)$

$\Leftrightarrow$

$((p \land s) \lor \neg q)) \land ((p \land s) \lor r)) \land (q \lor \neg p \lor s) \land (\neg r \lor s)$

$\Leftrightarrow$

$(p \lor \neg q) \land (s \lor \neg q) \land ((p \land s) \lor r)) \land (q \lor \neg p \lor s) \land (\neg r \lor s)$

$\Leftrightarrow$

$(p \lor \neg q) \land (s \lor \neg q) \land (\mathrm{p} \lor \mathrm{r}) \land (\mathrm{s} \lor \mathrm{r}) \land (q \lor \neg p \lor s) \land (\neg r \lor s)$

Normal Forms

DNF?

$p \wedge (\neg p \vee q) \wedge (\neg q \vee r)$

Normal Forms

DNF? NO, actually this formula is in CNF

$p \wedge (\neg p \vee q) \wedge (\neg q \vee r)$

Normal Forms

DNF? NO, actually this formula is in CNF

$p \wedge (\neg p \vee q) \wedge (\neg q \vee r)$

Distributivity

1. $A \vee (B \wedge C) \Leftrightarrow (A \vee B) \wedge (A \vee C)$
2. $A \wedge (B \vee C) \Leftrightarrow (A \wedge B) \vee (A \wedge C)$

Normal Forms

DNF? NO, actually this formula is in CNF

$p \wedge (\neg p \vee q) \wedge (\neg q \vee r)$

$\Leftrightarrow$

$((p \wedge \neg p) \vee (p \vee q)) \wedge (\neg q \vee r)$

Distributivity

1. $A \vee (B \wedge C) \Leftrightarrow (A \vee B) \wedge (A \vee C)$
2. $A \wedge (B \vee C) \Leftrightarrow (A \wedge B) \vee (A \wedge C)$

Normal Forms

DNF? NO, actually this formula is in CNF

$p \wedge (\neg p \vee q) \wedge (\neg q \vee r)$

$\Leftrightarrow$

$((p \wedge \neg p) \vee (p \vee q)) \wedge (\neg q \vee r)$

$\Leftrightarrow$

$(p \vee q) \wedge (\neg q \vee r)$

Distributivity

1. $A \vee (B \wedge C) \Leftrightarrow (A \vee B) \wedge (A \vee C)$
2. $A \wedge (B \vee C) \Leftrightarrow (A \wedge B) \vee (A \wedge C)$

Other Rules

1. $A \wedge \neg A \Leftrightarrow \bot$
2. $A \vee \bot \Leftrightarrow A$

Normal Forms

DNF? NO, actually this formula is in CNF

$p \wedge (\neg p \vee q) \wedge (\neg q \vee r)$

$\Leftrightarrow$

$((p \wedge \neg p) \vee (p \vee q)) \wedge (\neg q \vee r)$

$\Leftrightarrow$

$(p \vee q) \wedge (\neg q \vee r)$

$\Leftrightarrow$

$((p \vee q) \wedge \neg q) \vee ((p \vee q) \wedge r)$

Distributivity

1. $A \vee (B \wedge C) \Leftrightarrow (A \vee B) \wedge (A \vee C)$
2. $A \wedge (B \vee C) \Leftrightarrow (A \wedge B) \vee (A \wedge C)$

Other Rules

1. $A \wedge \neg A \Leftrightarrow \bot$
2. $A \vee \bot \Leftrightarrow A$

Normal Forms

DNF? NO, actually this formula is in CNF

$p \land (\neg p \lor q) \land (\neg q \lor r)$

$\Leftrightarrow$

$((p \land \neg p) \lor (p \lor q)) \land (\neg q \lor r)$

$\Leftrightarrow$

$(p \lor q) \land (\neg q \lor r)$

$\Leftrightarrow$

$((p \lor q) \land \neg q) \lor ((p \lor q) \land r)$

$\Leftrightarrow$

$(p \land \neg q) \lor (q \land \neg q) \lor ((p \lor q) \land r)$

$\Leftrightarrow$

$(p \land \neg q) \lor (p \land r) \lor (q \land \mathrm{r})$

CNF (again)

A *CNF* formula is a conjunction of *clauses*. A *clause* is a disjunction of *literals*.

Ex: Implement a linear-time decision procedure for 2CNF (each clause has at most 2 literals).

A clause is *trivial* if it contains a *complementary* pair of literals.

Since the *order* of the *literals* in a clause is *irrelevant*, the clause can be treated as a *set*.

A set of clauses is *trivial* if it contains the *empty clause* (false).

CNF (again)

Equivalence rules can be used to translate any formula to CNF.

eliminate $\Rightarrow$	$A \Rightarrow B \equiv \neg A \vee B$
reduce the scope of $\neg$	$\neg(A \vee B) \equiv \neg A \wedge \neg B$, $\neg(A \wedge B) \equiv \neg A \vee \neg B$
apply distributivity	$A \vee (B \wedge C) \equiv (A \vee B) \wedge (A \vee C)$, $A \wedge (B \vee C) \equiv (A \wedge B) \vee (A \wedge C)$

CNF (again)

The CNF translation described in the previous slide is too *expensive* (distributivity rule).

However, there is a *linear time* translation to CNF that produces an *equisatisfiable* formula. Replace the distributivity rules by the following rules:

$$\frac{F[l_i \ op \ l_j]}{F[x], x \Leftrightarrow l_i \ op \ l_j}{}^{*}$$

$$\frac{x \Leftrightarrow l_i \vee l_j}{\neg x \vee l_i \vee l_j, \neg l_i \vee x, \neg l_j \vee x}$$

$$\frac{x \Leftrightarrow l_i \wedge l_j}{\neg x \vee l_i, \neg x \vee l_j, \neg l_i \vee \neg l_j \vee x}$$

(*) x must be a fresh variable.

Ex: Show that the rules preserve equisatisfiability.

CNF translation (example)

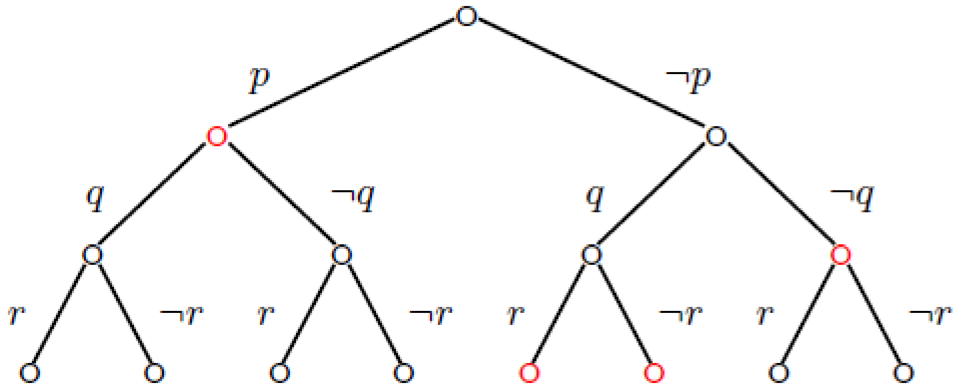
Translation of $(p \wedge (q \vee r)) \vee t$:

$$\frac{\frac{\frac{\frac{(p \wedge (q \vee r)) \vee t}{(p \wedge x_1) \vee t, x_1 \Leftrightarrow q \vee r}}{x_2 \vee t, x_2 \Leftrightarrow p \wedge x_1, x_1 \Leftrightarrow q \vee r}}{x_2 \vee t, \neg x_2 \vee p, \neg x_2 \vee x_1, \neg p \vee \neg x_1 \vee x_2, x_1 \Leftrightarrow q \vee r}}{x_2 \vee t, \neg x_2 \vee p, \neg x_2 \vee x_1, \neg p \vee \neg x_1 \vee x_2, \neg x_1 \vee q \vee r, \neg q \vee x_1, \neg r \vee x_1}$$

Ex: Implement a CNF translator.

Semantic Trees

A *semantic tree* represents the set of partial interpretations for a set of clauses. A semantic tree for $\{p \vee \neg q \vee \neg r, p \vee r, p \vee q, \neg p\}$:



A node N is a failure node if its associated interpretation *falsifies* a clause, but its ancestor doesn't.

Ex: Show that the semantic tree for an unsatisfiable (non-trivial) set of clauses must contain a non failure node such that its descendants are failure nodes.

Resolution

Formula must be in *CNF*.

Resolution procedure uses only *one rule*:

$$\frac{C_1 \vee p, C_2 \vee \neg p}{C_1 \vee p, C_2 \vee \neg p, C_1 \vee C_2}\, res$$

The result of the resolution rule is also a clause, it is called the *resolvent*. *Duplicate literals* in a clause and *trivial clauses* are *eliminated*.

There is no *branching* in the resolution procedure.

Example: The resolvent of $p \vee q \vee r$, and $\neg p \vee r \vee t$ is $q \vee r \vee t$.

Termination argument: there is a *finite* number of distinct clauses over n propositional variables.

Ex: Show that the resolution rule is sound.

Resolution (example)

A refutation of $\neg p \vee \neg q \vee r$, $p \vee r$, $q \vee r$, $\neg r$:

$$\begin{tikzcd}
 & & & \neg p \vee \neg q \vee r \arrow[dr] & & \neg r \arrow[dl] & & q \vee r \arrow[dr] & & \neg r \arrow[dl] \\
p \vee r \arrow[dr] & & \neg r \arrow[dl] & & \neg p \vee \neg q \arrow[drr] & & & & q \arrow[dll] & \\
 & p \arrow[drr] & & & & & \neg p \arrow[dlll] & & & \\
 & & & \perp & & & & & &
\end{tikzcd}$$

Ex: Implement a naïve resolution procedure.

Completeness of Resolution

Let $Res(S)$ be the closure of S under the resolution rule.

Completeness: S is unsatisfiable iff $Res(S)$ contains the *empty clause*.

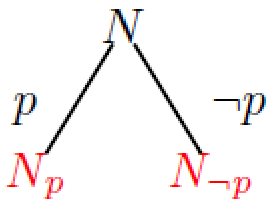
Proof ($\Rightarrow$):

Assume that S is unsatisfiable, and $Res(S)$ does not contain the *empty clause*.

Key points: $Res(S)$ is unsatisfiable, and $Res(S)$ is a non trivial set of clauses.

The semantic tree of $Res(S)$ must contain a non failure node N such that its descendants (N_p, $N_{\neg p}$) are failure nodes.

Completeness of Resolution



There is $C_1 \vee \neg p$ which is falsified by N_p, but not by N.

There is $C_2 \vee p$ which is falsified by $N_{\neg p}$, but not by N.

$C_1 \vee C_2$ is the resolvent of $C_1 \vee \neg p$ and $C_2 \vee p$.

$C_1 \vee C_2$ is in $Res(S)$, and it is falsified by N (*contradiction*).

Proof ($\Leftarrow$): $Res(S)$ is unsatisfiable, and equivalent to S. So, S is unsatisifiable.

Subsumption

The *resolution* procedure may generate several *irrelevant* and *redundant clauses*.

Subsumption is a clause *deletion strategy* for the resolution procedure.

$$\frac{C_1, C_1 \vee C_2}{C_1}\, sub$$

Example: $p \vee \neg q$ subsumes $p \vee \neg q \vee r \vee t$.

Deletion strategy: Remove the subsumed clauses.

Unit & Input Resolution

Unit resolution: one of the clauses is a unit clause.

$$\frac{C \vee \bar{l}, l}{C, l}\,unit$$

Unit resolution always *decreases* the configuration *size* ($C \vee \bar{l}$ is subsumed by C).

Input resolution: one of the clauses is in S.

Ex: Show that the unit and input resolution procedures are not complete.

Ex: Show that a set of clauses S has an unit refutation iff it has an input refutation (hint: induction on the number of propositions).

Horn Clauses

Each clause has at most on positive literal.

Rule base systems ($\neg p_1 \vee \ldots \vee \neg p_n \vee q \;\equiv\; p_1 \wedge \ldots \wedge p_n \Rightarrow q$).

Positive unit rule:

$$\frac{C \vee \neg p, p}{C, p}\, unit^+$$

Horn clauses are the basis of programming languages as *Prolog*.

Ex: Show that the positive unit rule is a complete procedure for Horn clauses.

Ex: Implement a linear time algorithm for Horn clauses.

DPLL

DPLL $=$ Unit resolution $+$ Split rule.

$$\frac{\Gamma}{\Gamma, p \mid \Gamma, \neg p}\,split \qquad p \text{ and } \neg p \text{ are not in } \Gamma.$$

$$\frac{C \vee \bar{l}, l}{C, l}\,unit$$

Used in the most efficient SAT solvers.

Pure Literals

A literal is pure if only occurs positively or negatively.

Example :

$\varphi = (\neg x_1 \vee x_2) \wedge (x_3 \vee \neg x_2) \wedge (x_4 \vee \neg x_5) \wedge (x_5 \vee \neg x_4)$

$\neg x_1$ and x_3 are pure literals

Pure literal rule :

Clauses containing pure literals can be removed from the formula (i.e. just satisfy those pure literals)

$$\varphi_{\neg x_1, x_3} = (x_4 \vee \neg x_5) \wedge (x_5 \vee \neg x_4)$$

Preserve satisfiability, not logical equivalency !

Pure Literals

A literal is pure if only occurs positively or negatively.

Example :

$\varphi = (\neg x_1 \vee x_2) \wedge (x_3 \vee \neg x_2) \wedge (x_4 \vee \neg x_5) \wedge (x_5 \vee \neg x_4)$

$\neg x_1$ and x_3 are pure literals

Pure literal rule :

Clauses containing pure literals can be removed from the formula (i.e. just satisfy those pure literals)

$$\varphi_{\neg x_1, x_3} = (x_4 \vee \neg x_5) \wedge (x_5 \vee \neg x_4)$$

Preserve satisfiability, not logical equivalency !

DPLL (as a procedure)

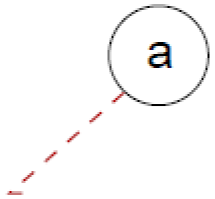
- Standard backtrack search
- DPLL(F) :
    - Apply unit propagation
    - If conflict identified, return UNSAT
    - Apply the pure literal rule
    - If F is satisfied (empty), return SAT
    - Select decision variable x
        - If DPLL($F \wedge x$)=SAT return SAT
        - return DPLL($F \wedge \neg x$)

DPLL (example)

$$\begin{aligned}\varphi \quad = \quad & (a \vee \neg b \vee d) \wedge (a \vee \neg b \vee e) \wedge \\ & (\neg b \vee \neg d \vee \neg e) \wedge \\ & (a \vee b \vee c \vee d) \wedge (a \vee b \vee c \vee \neg d) \wedge \\ & (a \vee b \vee \neg c \vee e) \wedge (a \vee b \vee \neg c \vee \neg e)\end{aligned}$$

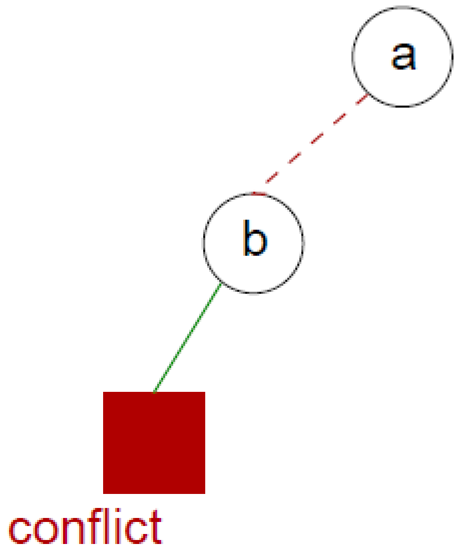
DPLL (example)

$$\begin{aligned}\varphi \quad = \quad & (a \vee \neg b \vee d) \wedge (a \vee \neg b \vee e) \wedge \\ & (\neg b \vee \neg d \vee \neg e) \wedge \\ & (a \vee b \vee c \vee d) \wedge (a \vee b \vee c \vee \neg d) \wedge \\ & (a \vee b \vee \neg c \vee e) \wedge (a \vee b \vee \neg c \vee \neg e)\end{aligned}$$



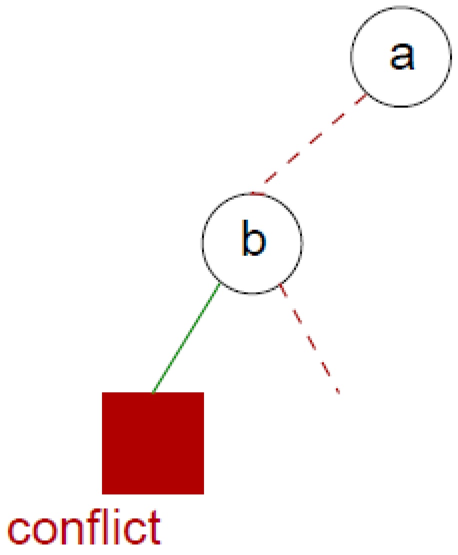
DPLL (example)

$$\begin{aligned}\varphi \;=\; & (a \vee \neg b \vee d) \wedge (a \vee \neg b \vee e) \wedge \\ & (\neg b \vee \neg d \vee \neg e) \wedge \\ & (a \vee b \vee c \vee d) \wedge (a \vee b \vee c \vee \neg d) \wedge \\ & (a \vee b \vee \neg c \vee e) \wedge (a \vee b \vee \neg c \vee \neg e)\end{aligned}$$



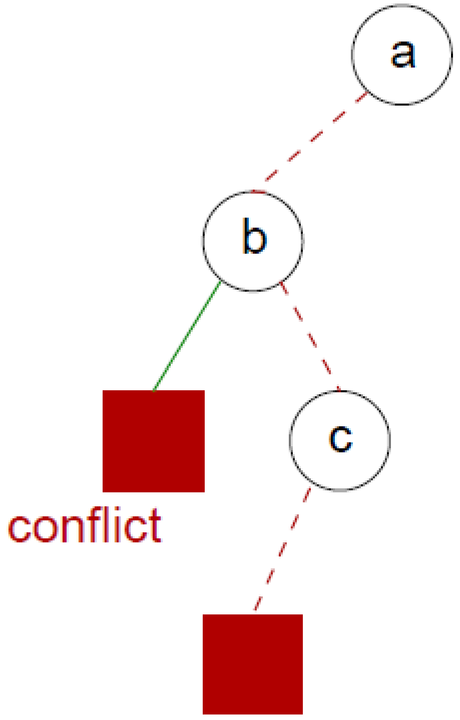
DPLL (example)

$$\begin{aligned}\varphi \quad = \quad & (a \vee \neg b \vee d) \wedge (a \vee \neg b \vee e) \wedge \\ & (\neg b \vee \neg d \vee \neg e) \wedge \\ & (a \vee b \vee c \vee d) \wedge (a \vee b \vee c \vee \neg d) \wedge \\ & (a \vee b \vee \neg c \vee e) \wedge (a \vee b \vee \neg c \vee \neg e)\end{aligned}$$



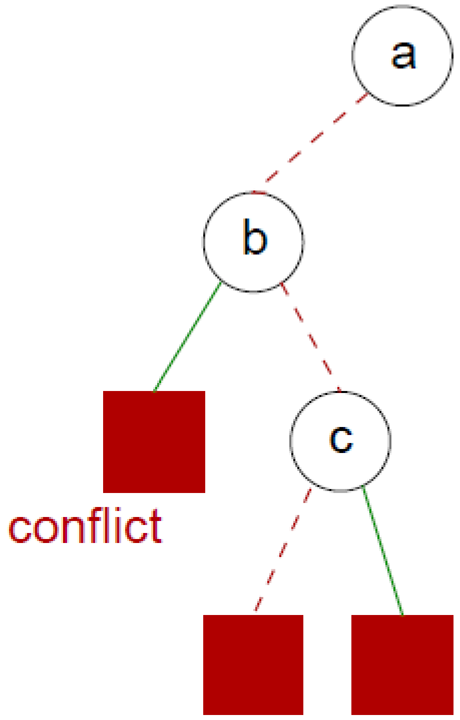
DPLL (example)

$$\begin{aligned}\varphi \;=\; & (a \vee \neg b \vee d) \wedge (a \vee \neg b \vee e) \wedge \\ & (\neg b \vee \neg d \vee \neg e) \wedge \\ & (a \vee b \vee c \vee d) \wedge (a \vee b \vee c \vee \neg d) \wedge \\ & (a \vee b \vee \neg c \vee e) \wedge (a \vee b \vee \neg c \vee \neg e)\end{aligned}$$



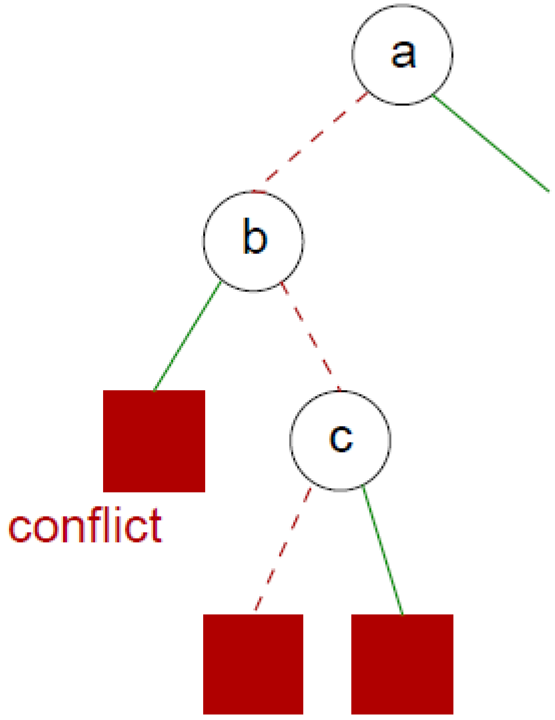
DPLL (example)

$$\begin{aligned}\varphi \;=\; & (a \lor \neg b \lor d) \land (a \lor \neg b \lor e) \land \\ & (\neg b \lor \neg d \lor \neg e) \land \\ & (a \lor b \lor c \lor d) \land (a \lor b \lor c \lor \neg d) \land \\ & (a \lor b \lor \neg c \lor e) \land (a \lor b \lor \neg c \lor \neg e)\end{aligned}$$



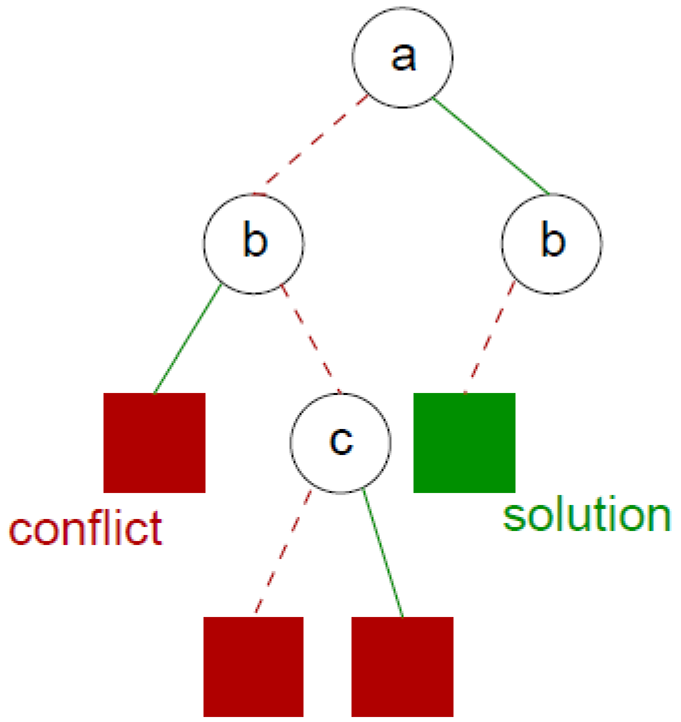
DPLL (example)

$$\begin{aligned}\varphi \quad = \quad & (a \vee \neg b \vee d) \wedge (a \vee \neg b \vee e) \wedge \\ & (\neg b \vee \neg d \vee \neg e) \wedge \\ & (a \vee b \vee c \vee d) \wedge (a \vee b \vee c \vee \neg d) \wedge \\ & (a \vee b \vee \neg c \vee e) \wedge (a \vee b \vee \neg c \vee \neg e)\end{aligned}$$



DPLL (example)

$$\begin{aligned}\varphi \;=\; & (a \vee \neg b \vee d) \wedge (a \vee \neg b \vee e) \wedge \\ & (\neg b \vee \neg d \vee \neg e) \wedge \\ & (a \vee b \vee c \vee d) \wedge (a \vee b \vee c \vee \neg d) \wedge \\ & (a \vee b \vee \neg c \vee e) \wedge (a \vee b \vee \neg c \vee \neg e)\end{aligned}$$



Some Applications

Bit-vector / Machine arithmetic

Let x, y and z be 8-bit (unsigned) integers.

Is $x > 0 \wedge y > 0 \wedge z = x + y \Rightarrow z > 0$ valid?

Is $x > 0 \wedge y > 0 \wedge z = x + y \wedge \neg(z > 0)$ satisfiable?

Bit-vector / Machine arithmetic

We can encode bit-vector satisfiability problems in propositional logic.

Idea 1:

Use *n* propositional variables to encode *n*-bit integers.

$$x \rightarrow (x_1, \ldots, x_n)$$

Idea 2:

Encode arithmetic operations using hardware circuits.

Encoding equality

$p \Leftrightarrow q$ is equivalent to $(\neg p \vee q) \wedge (\neg q \vee p)$

The bit-vector equation x = y is encoded as:

$(x_1 \Leftrightarrow y_1) \wedge \ldots \wedge (x_n \Leftrightarrow y_n)$

Encoding addition

We use $(r_1, \ldots, r_n)$ to store the result of x + y

p xor q is defined as $\neg(p \Leftrightarrow q)$

xor is the 1-bit adder

p	***q***	***p* xor *q***	***p* ∧ *q***
0	0	0	0
1	0	1	0
0	1	1	0
1	1	0	1

carry

Encoding 1-bit full adder

1-bit full adder

Three inputs: x, y, c_{in}

Two outputs: r, c_{out}

x	y	c_{in}	$r = x$ xor y xor c_{in}	$c_{out} = (x \wedge y) \vee (x \wedge c_{in}) \vee (y \wedge c_{in})$
0	0	0	0	0
1	0	0	1	0
0	1	0	1	0
1	1	0	0	1
0	0	1	1	0
1	0	1	0	1
0	1	1	0	1
1	1	1	1	1

Encoding n-bit adder

We use $(r_1, \ldots, r_n)$ to store the result of $x + y$, and $(c_1, \ldots, c_n)$

$r_1 \Leftrightarrow (x_1 \text{ xor } y_1)$

$c_1 \Leftrightarrow (x_1 \wedge y_1)$

$r_2 \Leftrightarrow (x_2 \text{ xor } y_2 \text{ xor } c_1)$

$c_2 \Leftrightarrow (x_2 \wedge y_2) \vee (x_2 \wedge c_1) \vee (y_2 \wedge c_1)$

…

$r_n \Leftrightarrow (x_n \text{ xor } y_n \text{ xor } c_{n-1})$

$c_n \Leftrightarrow (x_n \wedge y_n) \vee (x_n \wedge c_{n-1}) \vee (y_n \wedge c_{n-1})$

Test case generation (again)

```
unsigned GCD(x, y) {
  requires(y > 0);
  while (true) {
     unsigned m = x % y;
      if (m == 0) return y;
      x = y;
      y = m;
  }
}
```

SSA →

$(y_0 > 0)$ and

$(m_0 = x_0 \% y_0)$ and

not $(m_0 = 0)$ and

$(x_1 = y_0)$ and

$(y_1 = m_0)$ and

$(m_1 = x_1 \% y_1)$ and

$(m_1 = 0)$

Solver →

$x_0 = 2$

$y_0 = 4$

$m_0 = 2$

$x_1 = 4$

$y_1 = 2$

$m_1 = 0$

We want a trace where the loop is executed twice.

Microsoft Research

Experimental Exercises

- The first step is to pick up a SAT solver.
- Play with simple examples
- Translate your problem into SAT
- Experiment

Available SAT Solvers

Several open source SAT solvers exist :

Minisat (C++) `www.minisat.se` Presumably the most widely used within the SAT community. Used to be the best general purpose SAT solver. A large community around the solver.

Picosat (C)/Precosat (C++) `http://fmv.jku.at/software/index.html` Award winner in 2007 and 2009 of the SAT competition, industrial category.

SAT4J (Java) `http://www.sat4j.org`. For Java users. Far less efficient than the two others.

UBCSAT (C) `http://www.satlib.org/ubcsat/` Very efficient stochastic local search for SAT.

`http://www.satcompetition.org` Both the binaries and the source code of the solvers are made available for research purposes.

Available Examples

- Satisfiability library: http://www.satlib.org
- The SAT competion: http://www.satcompetition.org
- Search the WEB: “SAT benchmarks”

Using SAT solvers

All SAT solvers support the very simple DIMACS CNF input format :

$$(a \vee b \vee \neg c) \wedge (\neg b \vee \neg c)$$

will be translated into

```
p cnf 3 2
1 2 -3 0
-2 -3 0
```

The first line is of the form
`p cnf <maxVarId> <numberOfClauses>`
Each variable is represented by an integer, negative literals as negative integers, 0 is the clause separator.

Satisfiability Modulo Theories (SMT)

Is formula *F* satisfiable modulo theory *T* ?

SMT solvers have specialized algorithms for *T*

Microsoft Research

Satisfiability Modulo Theories (SMT)

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

Satisfiability Modulo Theories (SMT)

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

Arithmetic

Satisfiability Modulo Theories (SMT)

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

Array Theory

Microsoft Research

Satisfiability Modulo Theories (SMT)

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

Uninterpreted Functions

Microsoft Research

Satisfiability Modulo Theories (SMT)

$b + 2 = c$ and $f(read(write(a,b,3), c-2)) \neq f(c-b+1)$

Substituting c by b+2

Microsoft Research

Satisfiability Modulo Theories (SMT)

b + 2 = c and f(read(write(a,b,3), b+2-2)) ≠ f(b+2-b+1)

Simplifying

Satisfiability Modulo Theories (SMT)

$b + 2 = c$ and $f(read(write(a,b,3), b)) \neq f(3)$

Satisfiability Modulo Theories (SMT)

b + 2 = c and f(read(write(a,b,3), b)) ≠ f(3)

Applying array theory axiom

forall a,i,v: read(write(a,i,v), i) = v

Microsoft Research

Satisfiability Modulo Theories (SMT)

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

Inconsistent/Unsatisfiable

Microsoft Research

SMT-Lib

- Repository of Benchmarks
- http://www.smtlib.org
- Benchmarks are divided in “logics”:
  - QF_UF: unquantified formulas built over a signature of uninterpreted sort, function and predicate symbols.
  - QF_UFLIA: unquantified linear integer arithmetic with uninterpreted sort, function, and predicate symbols.
  - AUFLIA: closed linear formulas over the theory of integer arrays with free sort, function and predicate symbols.

Ground formulas

For most SMT solvers: ***F is a set of ground formulas***

Many Applications

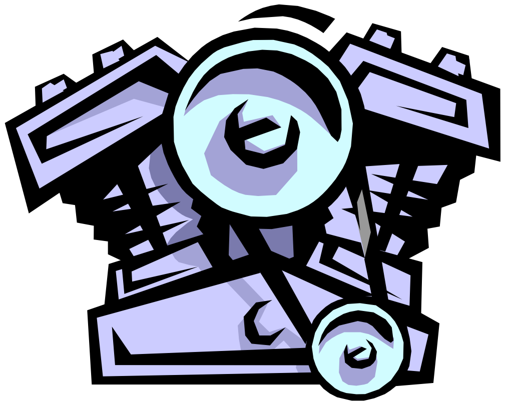
Bounded Model Checking

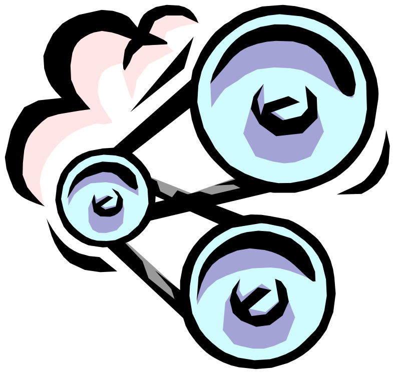
Test-Case Generation

Little Engines of Proof

An SMT Solver is a collection of

Little Engines of Proof



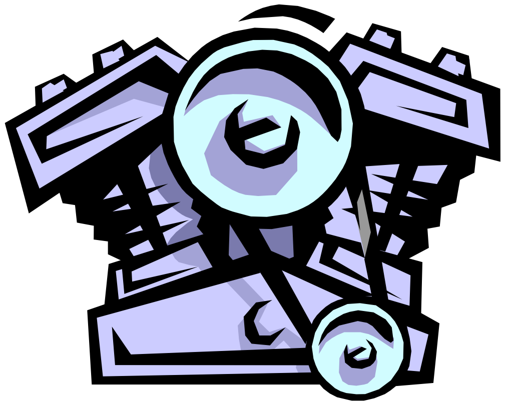


Microsoft Research

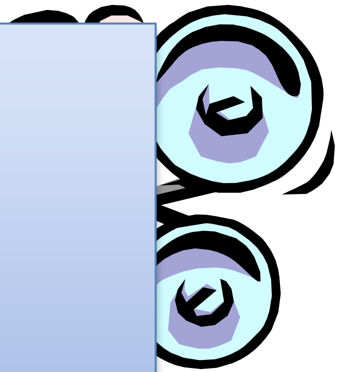
Little Engines of Proof

An SMT Solver is a collection of

Little Engines of Proof



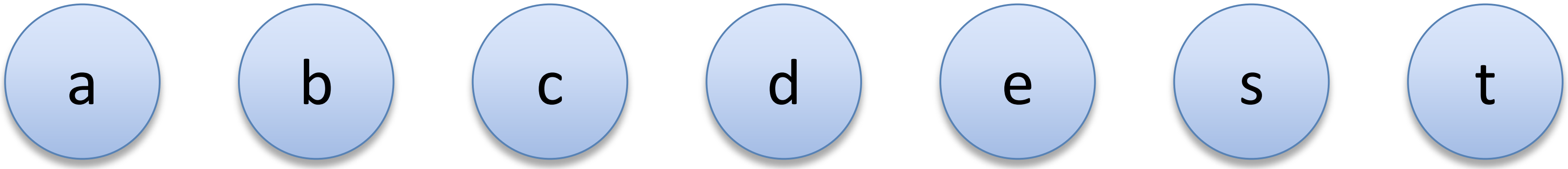
Examples:
SAT Solver
Equality solver



Microsoft Research

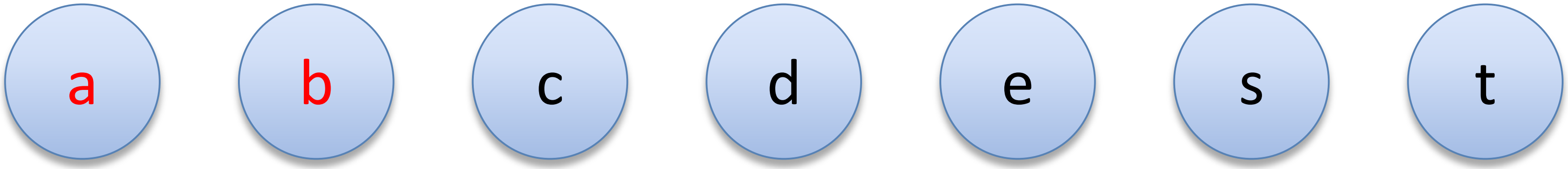
Deciding Equality

$a = b, b = c, d = e, b = s, d = t, a \neq e, a \neq s$



Deciding Equality

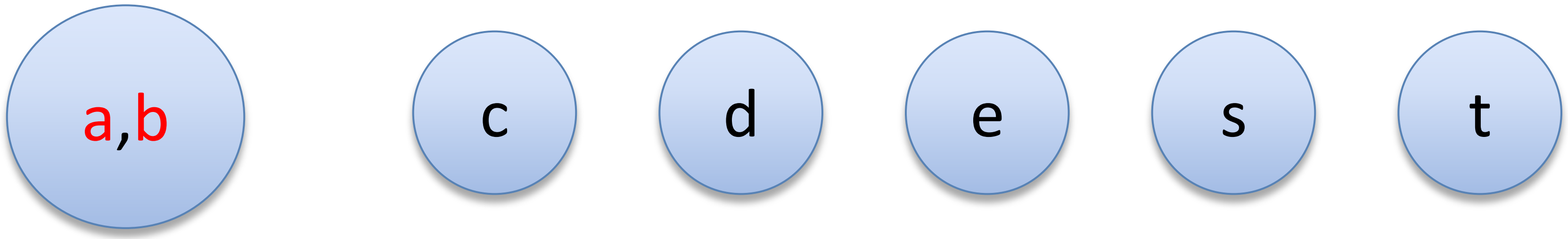
$a = b, b = c, d = e, b = s, d = t, a \neq e, a \neq s$



Microsoft Research

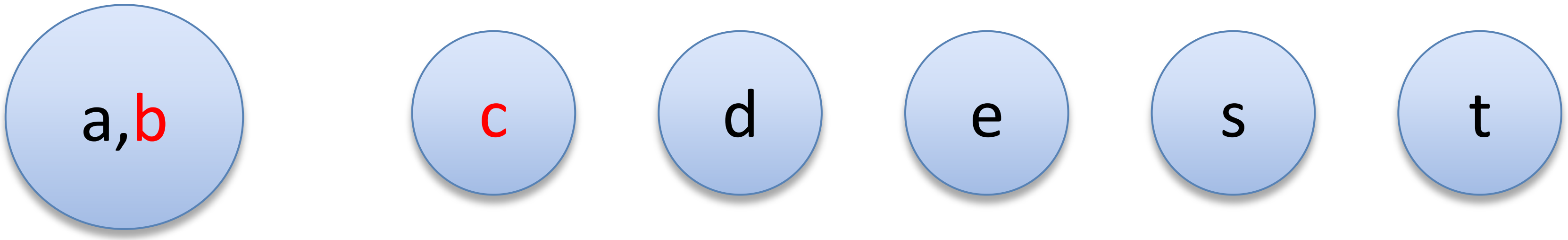
Deciding Equality

a = b, b = c, d = e, b = s, d = t, a ≠ e, a ≠ s



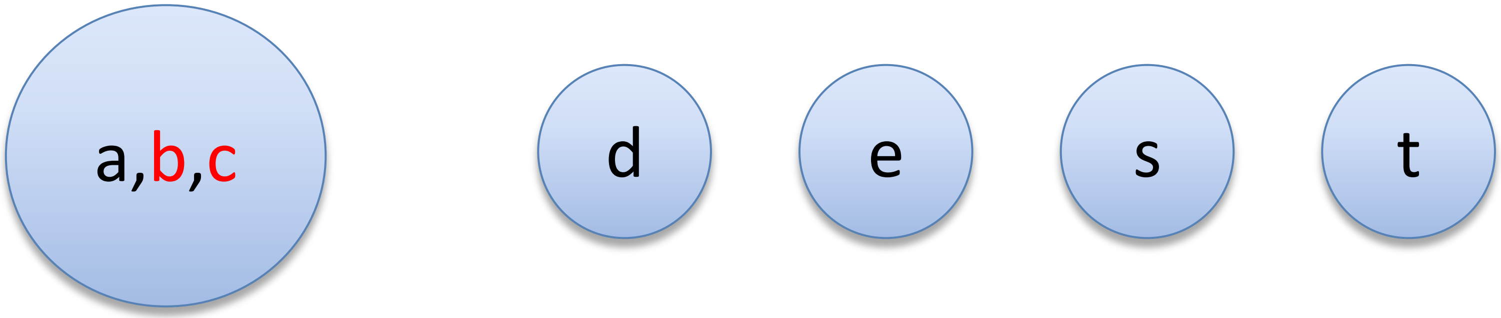
Deciding Equality

a = b, b = c, d = e, b = s, d = t, a≠ e, a≠ s



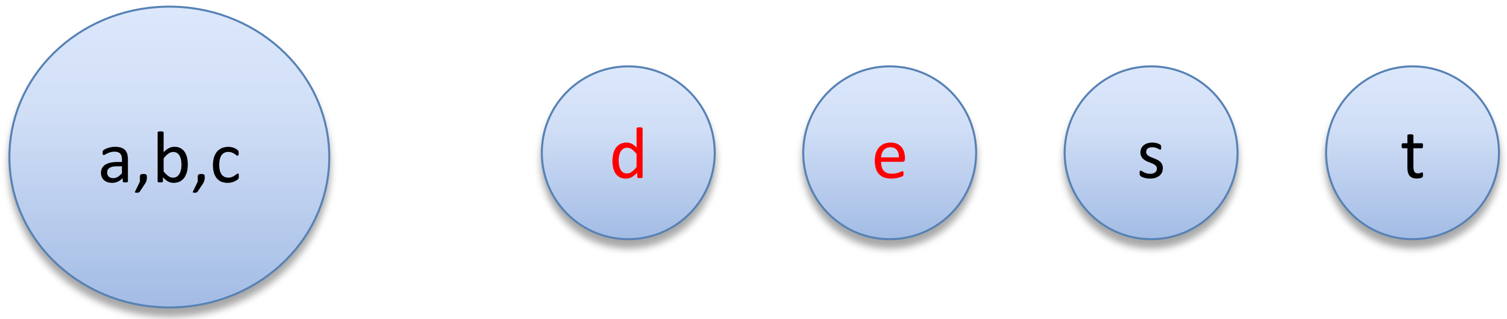
Deciding Equality

$a = b, b = c, d = e, b = s, d = t, a \neq e, a \neq s$



Deciding Equality

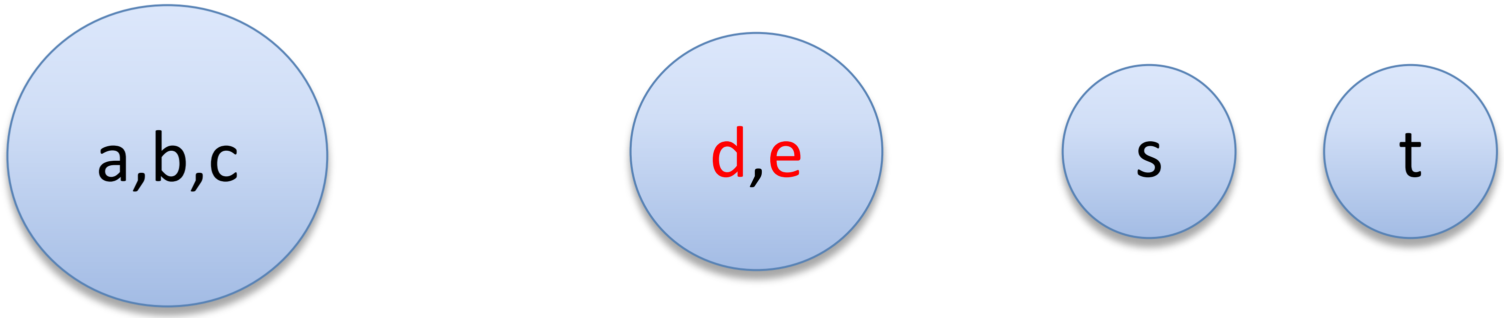
a = b, b = c, d = e, b = s, d = t, a ≠ e, a ≠ s



Microsoft Research

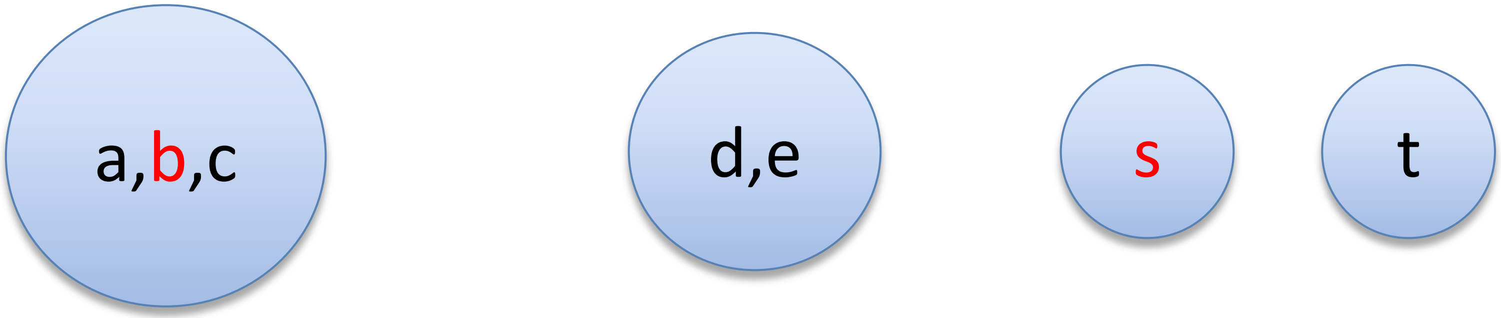
Deciding Equality

$a = b,\ b = c,\ d = e,\ b = s,\ d = t,\ a \neq e,\ a \neq s$



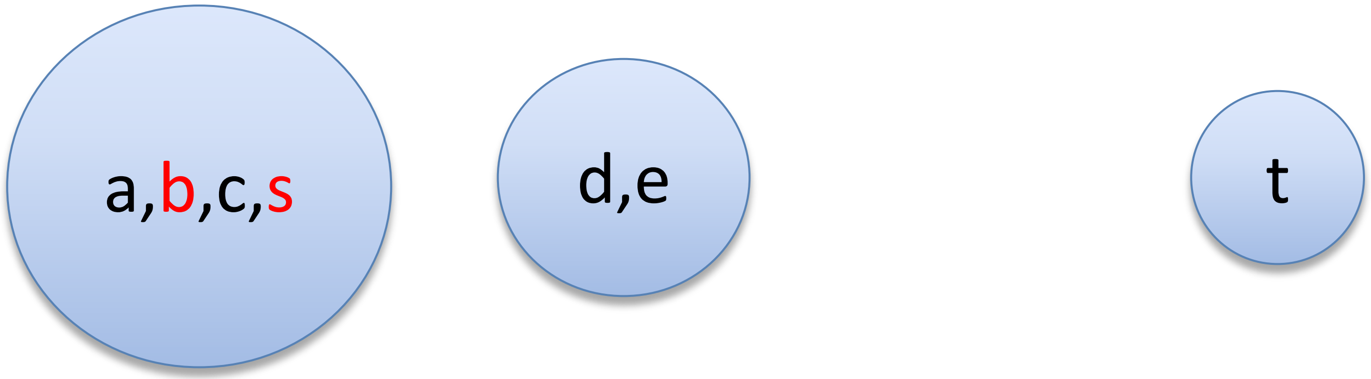
Deciding Equality

$a = b,\ b = c,\ d = e,\ b = s,\ d = t,\ a \neq e,\ a \neq s$



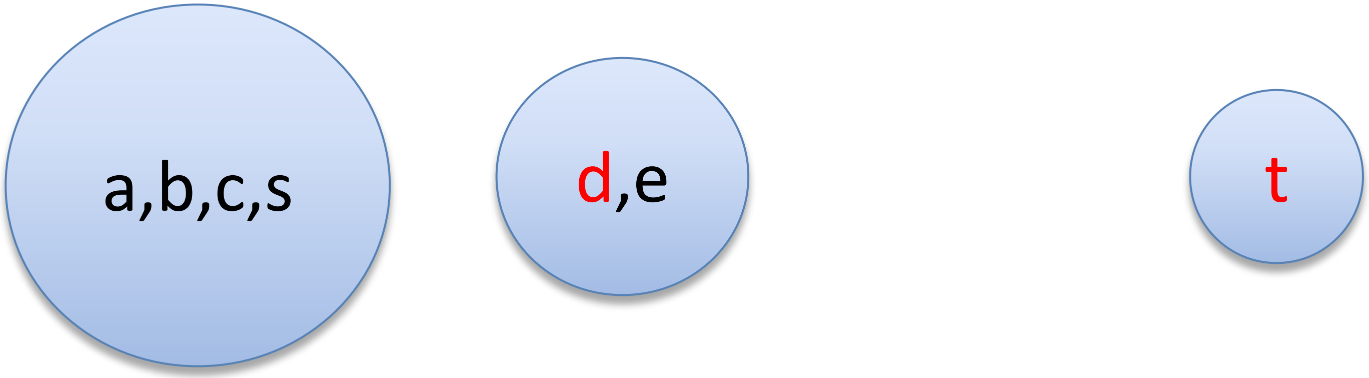
Deciding Equality

$a = b,\ b = c,\ d = e,\ b = s,\ d = t,\ a \neq e,\ a \neq s$



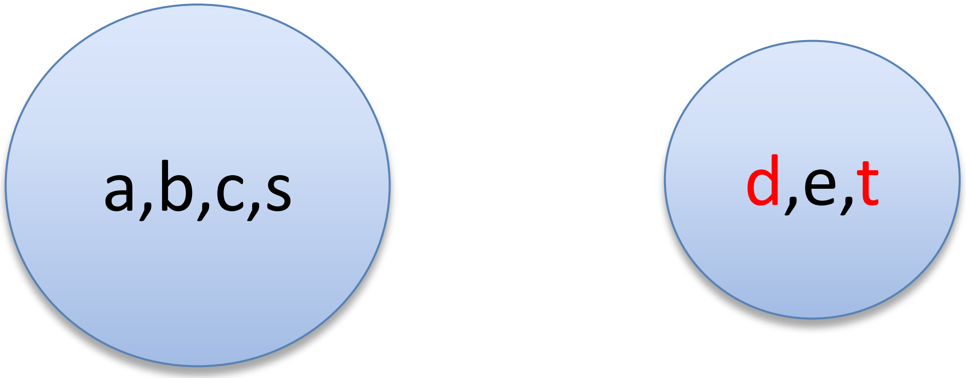
Deciding Equality

$a = b,\ b = c,\ d = e,\ b = s,\ d = t,\ a \neq e,\ a \neq s$



Deciding Equality

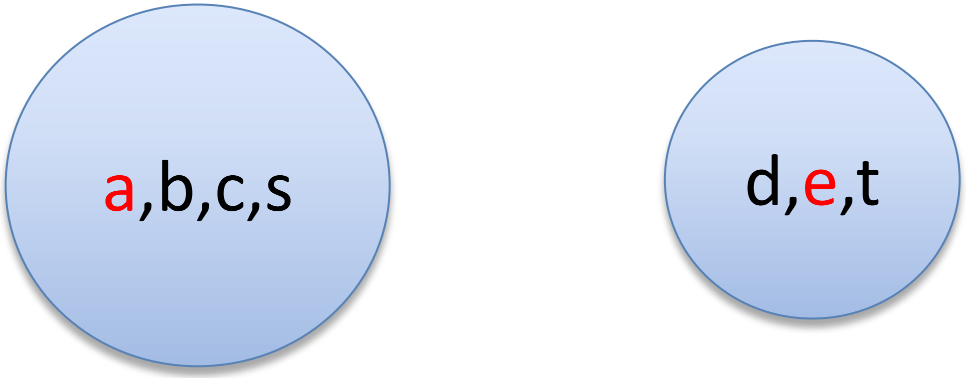
$a = b, b = c, d = e, b = s, d = t, a \neq e, a \neq s$



Microsoft Research

Deciding Equality

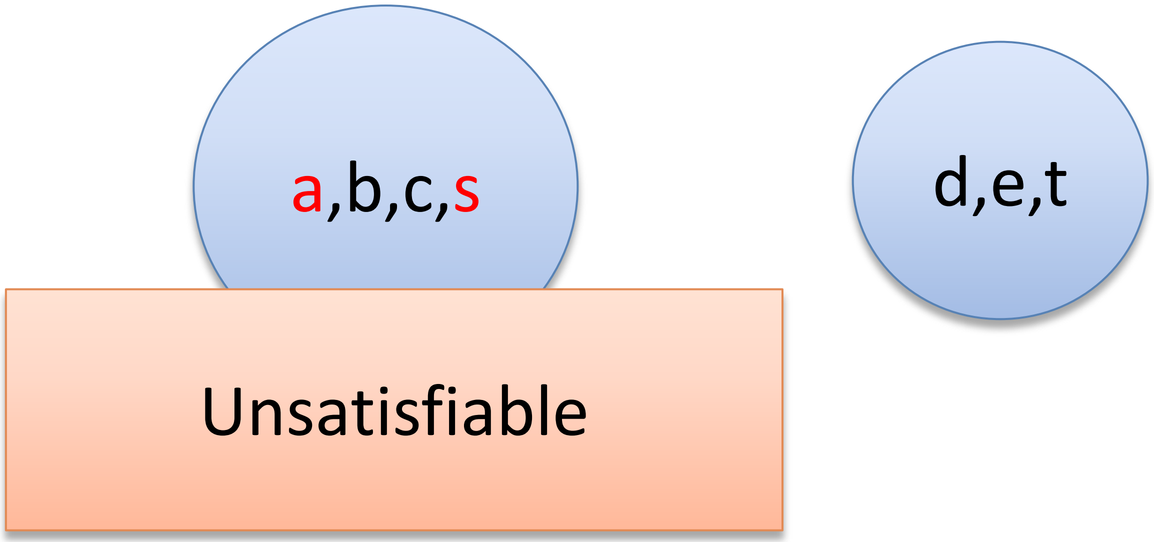
$a = b,\ b = c,\ d = e,\ b = s,\ d = t,\ a \neq e,\ a \neq s$



Microsoft Research

Deciding Equality

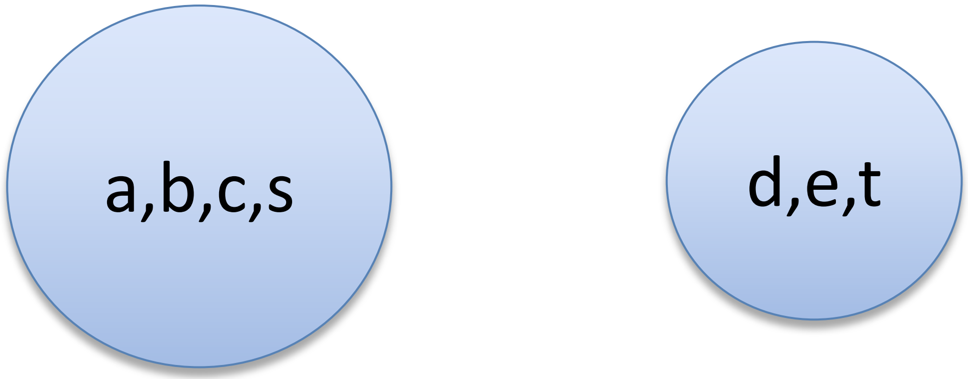
$a = b,\ b = c,\ d = e,\ b = s,\ d = t,\ a \neq e,\ a \neq s$



Microsoft Research

Deciding Equality

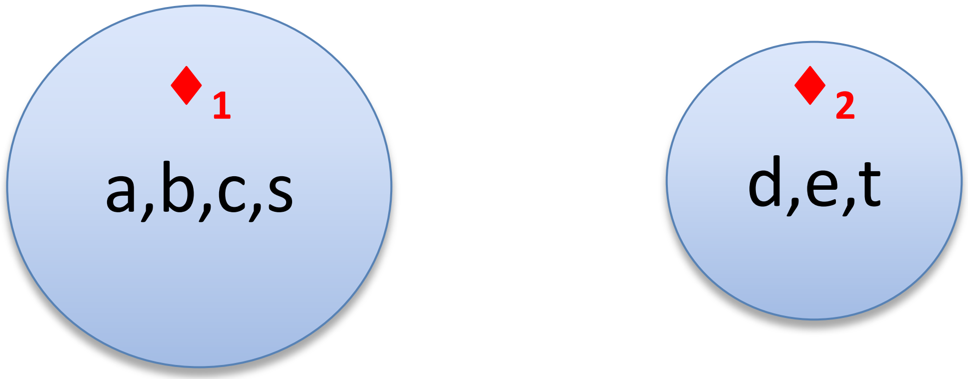
$a = b,\ b = c,\ d = e,\ b = s,\ d = t,\ a \neq e$



Model construction

Deciding Equality

$a = b, b = c, d = e, b = s, d = t, a \neq e$

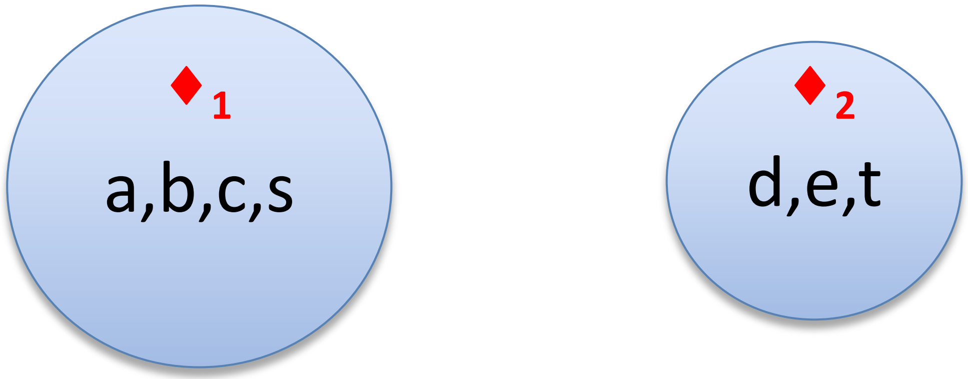


Model construction

$|M| = \{♦_1, ♦_2\}$ (universe, aka domain)

Deciding Equality

a = b, b = c, d = e, b = s, d = t, a$\neq$ e



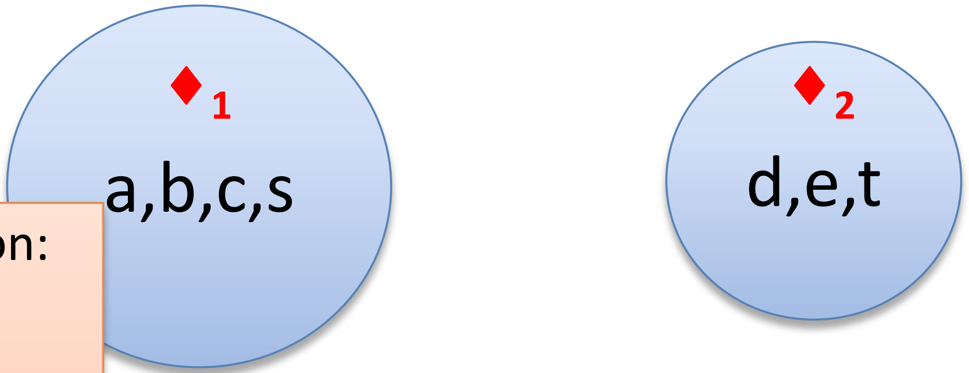
Model construction

|M| = {$♦_1$,$♦_2$} (universe, aka domain)

M(a) = $♦_1$ (assignment)

Deciding Equality

a = b, b = c, d = e, b = s, d = t, a $\neq$ e



Alternative notation:
a^M = ♦$_1$

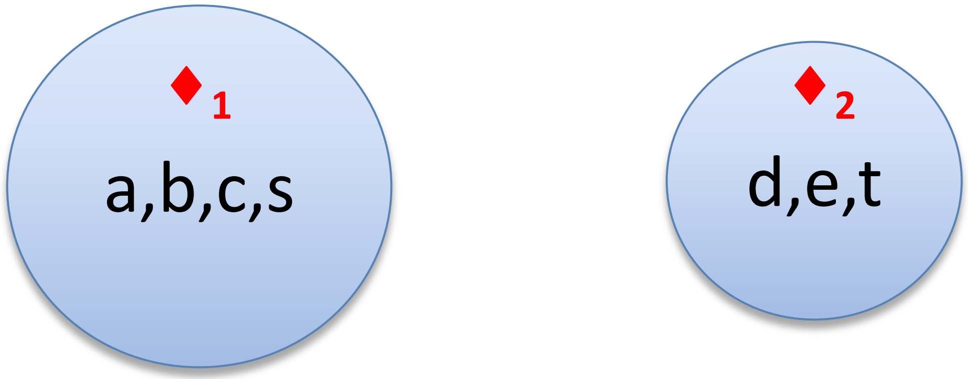
Model construction

|M| = {♦$_1$, ♦$_2$} (universe, aka domain)

M(a) = ♦$_1$ (assignment)

Deciding Equality

$a = b, b = c, d = e, b = s, d = t, a \neq e$



Model construction

$|M| = \{♦_1, ♦_2\}$ (universe, aka domain)

$M(a) = M(b) = M(c) = M(s) = ♦_1$

$M(d) = M(e) = M(t) = ♦_2$

Microsoft Research

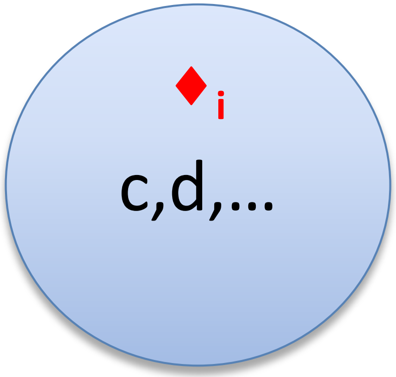
Deciding Equality: Termination, Soundness, Completeness

- Termination: easy
- Soundness
  - Invariant: all constants in a "ball" are known to be equal.
  - The "ball" merge operation is justified by:
    - Transitivity and Symmetry rules.
- Completeness
  - We can build a model if an inconsistency was not detected.
  - Proof template (by contradiction):
    - Build a candidate model.
    - Assume a literal was not satisfied.
    - Find contradiction.

Microsoft Research

Deciding Equality: Termination, Soundness, Completeness

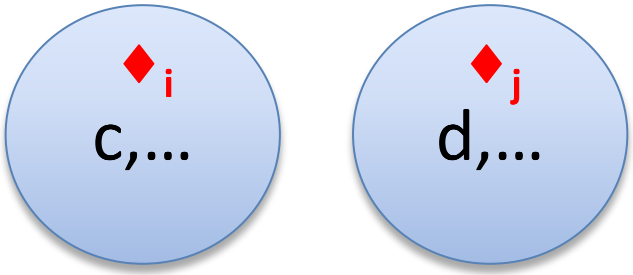
- Completeness
  - We can build a model if an inconsistency was not detected.
  - Instantiating the template for our procedure:
    - Assume some literal c = d is not satisfied by our model.
    - That is, $M(c) \neq M(d)$.
    - This is impossible, c and d must be in the same "ball".



$M(c) = M(d) = \blacklozenge_i$

Deciding Equality: Termination, Soundness, Completeness

- Completeness
  - We can build a model if an inconsistency was not detected.
  - Instantiating the template for our procedure:
    - Assume some literal $c \neq d$ is not satisfied by our model.
    - That is, $M(c) = M(d)$.
    - Key property: we only check the disequalities after we processed all equalities.
    - This is impossible, c and d must be in the different "balls"



$M(c) = \blacklozenge_i$

$M(d) = \blacklozenge_j$

Deciding Equality + (uninterpreted) Functions

$a = b, b = c, d = e, b = s, d = t, f(a, g(d)) \neq f(b, g(e))$

Congruence Rule:

$x_1 = y_1, \ldots, x_n = y_n$ implies $f(x_1, \ldots, x_n) = f(y_1, \ldots, y_n)$

Microsoft Research

Deciding Equality + (uninterpreted) Functions

$a = b, b = c, d = e, b = s, d = t, f(a, g(d)) \neq f(b, g(e))$

First Step: "Naming" subterms

Congruence Rule:

$x_1 = y_1, \ldots, x_n = y_n$ implies $f(x_1, \ldots, x_n) = f(y_1, \ldots, y_n)$

Microsoft Research

Deciding Equality + (uninterpreted) Functions

$a = b, b = c, d = e, b = s, d = t, f(a, v_1) \neq f(b, g(e))$

$v_1 \equiv g(d)$

First Step: “Naming” subterms

Congruence Rule:

$x_1 = y_1, \ldots, x_n = y_n$ implies $f(x_1, \ldots, x_n) = f(y_1, \ldots, y_n)$

Microsoft Research

Deciding Equality + (uninterpreted) Functions

$$a = b,\ b = c,\ d = e,\ b = s,\ d = t,\ f(a, v_1) \neq f(b, g(e))$$

$$v_1 \equiv g(d)$$

First Step: “Naming” subterms

Congruence Rule:

$x_1 = y_1, \ldots, x_n = y_n$ implies $f(x_1, \ldots, x_n) = f(y_1, \ldots, y_n)$

Microsoft Research

Deciding Equality + (uninterpreted) Functions

$a = b, b = c, d = e, b = s, d = t, f(a, v_1) \neq f(b, v_2)$

$v_1 \equiv g(d), v_2 \equiv g(e)$

First Step: “Naming” subterms

Congruence Rule:

$x_1 = y_1, \ldots, x_n = y_n$ implies $f(x_1, \ldots, x_n) = f(y_1, \ldots, y_n)$

Deciding Equality + (uninterpreted) Functions

$a = b, b = c, d = e, b = s, d = t, f(a, v_1) \neq f(b, v_2)$

$v_1 \equiv g(d), v_2 \equiv g(e)$

First Step: "Naming" subterms

Congruence Rule:

$x_1 = y_1, \ldots, x_n = y_n$ implies $f(x_1, \ldots, x_n) = f(y_1, \ldots, y_n)$

Microsoft Research

Deciding Equality + (uninterpreted) Functions

$a = b, b = c, d = e, b = s, d = t, v_3 \neq f(b, v_2)$

$v_1 \equiv g(d), v_2 \equiv g(e), v_3 \equiv f(a, v_1)$

First Step: “Naming” subterms

Congruence Rule:

$x_1 = y_1, \ldots, x_n = y_n$ implies $f(x_1, \ldots, x_n) = f(y_1, \ldots, y_n)$

Microsoft Research

Deciding Equality + (uninterpreted) Functions

$a = b, b = c, d = e, b = s, d = t, v_3 \neq f(b, v_2)$

$v_1 \equiv g(d), v_2 \equiv g(e), v_3 \equiv f(a, v_1)$

First Step: “Naming” subterms

Congruence Rule:

$x_1 = y_1, \ldots, x_n = y_n$ implies $f(x_1, \ldots, x_n) = f(y_1, \ldots, y_n)$

Deciding Equality + (uninterpreted) Functions

$$a = b,\ b = c,\ d = e,\ b = s,\ d = t,\ v_3 \neq v_4$$

$$v_1 \equiv g(d),\ v_2 \equiv g(e),\ v_3 \equiv f(a, v_1),\ v_4 \equiv f(b, v_2)$$

First Step: "Naming" subterms

Congruence Rule:

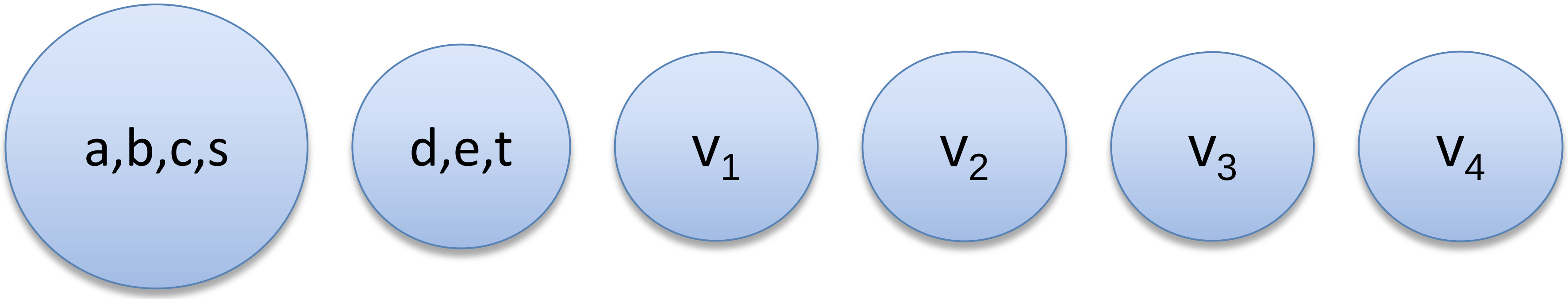
$$x_1 = y_1, \ldots, x_n = y_n \text{ implies } f(x_1, \ldots, x_n) = f(y_1, \ldots, y_n)$$

Microsoft Research

Deciding Equality + (uninterpreted) Functions

$$a = b,\ b = c,\ d = e,\ b = s,\ d = t,\ v_3 \neq v_4$$

$$v_1 \equiv g(d),\ v_2 \equiv g(e),\ v_3 \equiv f(a, v_1)\ ,\ v_4 \equiv f(b, v_2)$$



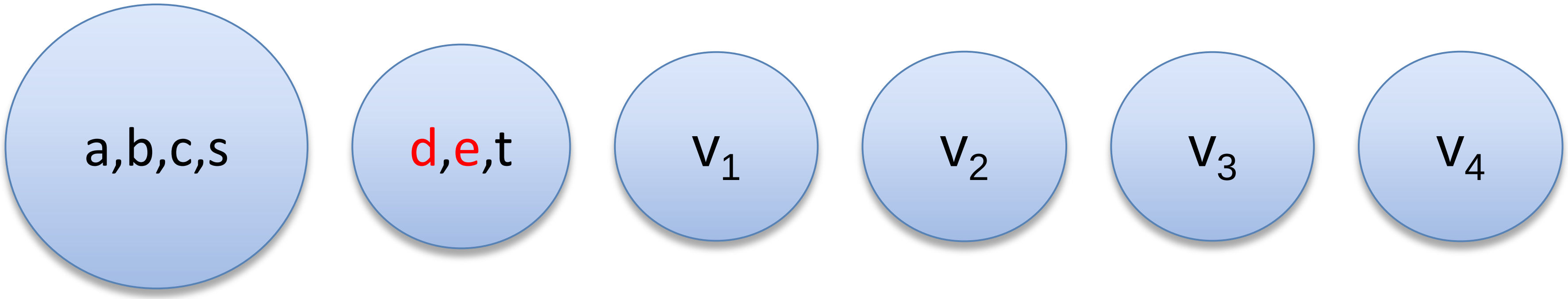
Congruence Rule:

$$x_1 = y_1, \ldots, x_n = y_n \text{ implies } f(x_1, \ldots, x_n) = f(y_1, \ldots, y_n)$$

Deciding Equality + (uninterpreted) Functions

$a = b,\ b = c,\ d = e,\ b = s,\ d = t,\ v_3 \neq v_4$

$v_1 \equiv g(d),\ v_2 \equiv g(e),\ v_3 \equiv f(a, v_1)\ ,\ v_4 \equiv f(b, v_2)$



Congruence Rule:

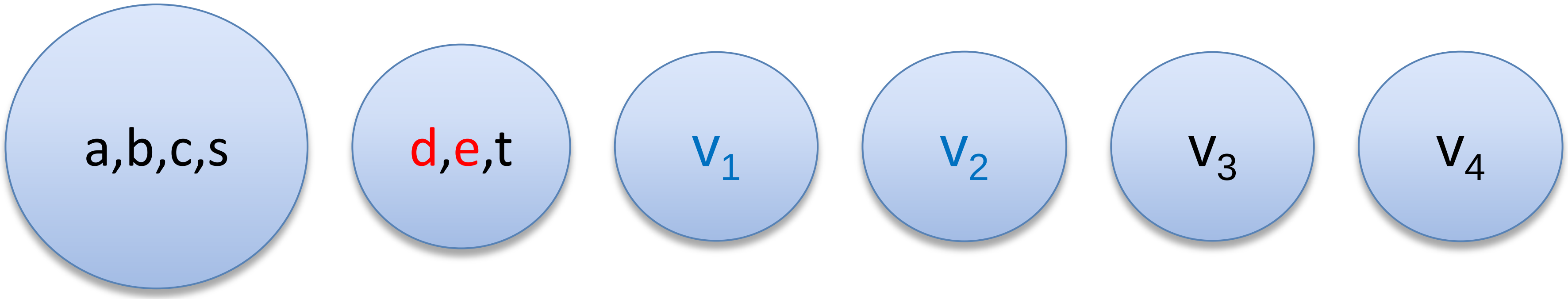
$x_1 = y_1, \ldots, x_n = y_n$ implies $f(x_1, \ldots, x_n) = f(y_1, \ldots, y_n)$

$d = e$ implies $g(d) = g(e)$

Deciding Equality + (uninterpreted) Functions

$a = b,\ b = c,\ d = e,\ b = s,\ d = t,\ v_3 \neq v_4$

$v_1 \equiv g(d),\ v_2 \equiv g(e),\ v_3 \equiv f(a, v_1)\ ,\ v_4 \equiv f(b, v_2)$



Congruence Rule:

$x_1 = y_1, \ldots, x_n = y_n$ implies $f(x_1, \ldots, x_n) = f(y_1, \ldots, y_n)$

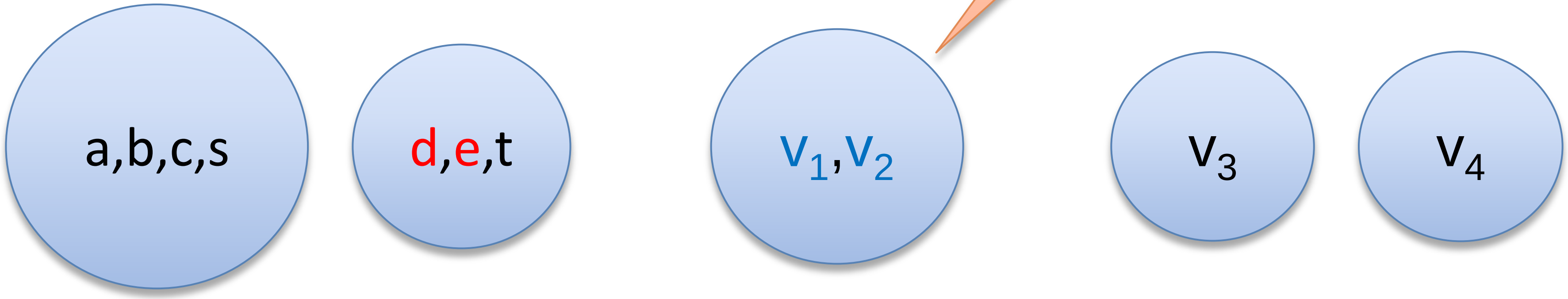
$d = e$ implies $v_1 = v_2$

Deciding Equality + (uninterpreted) Functions

We say:
v_1 and v_2 are congruent.

$a = b, b = c, d = e, b = s, d = t,$ … v_4

$v_1 \equiv g(d), v_2 \equiv g(e), v_3 \equiv f(a, v_1)$ … $_4 \equiv f(b, v_2)$



Congruence Rule:

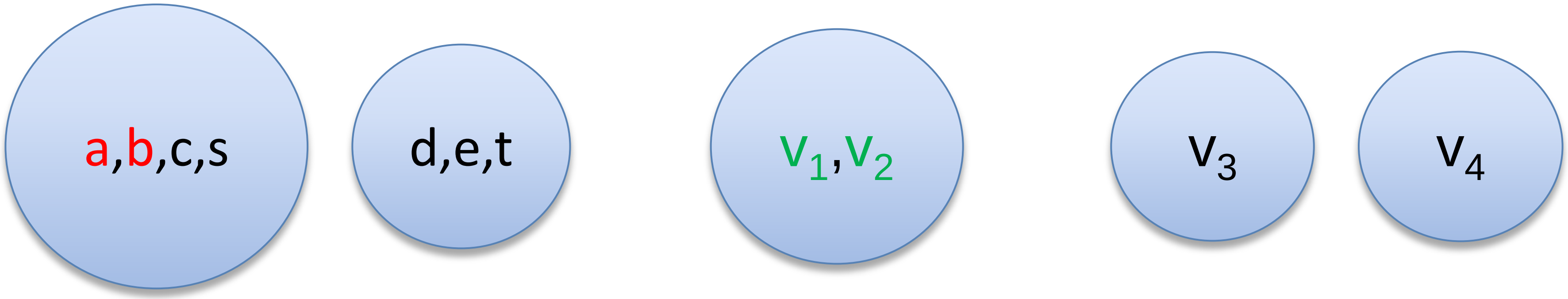
$x_1 = y_1, \ldots, x_n = y_n$ implies $f(x_1, \ldots, x_n) = f(y_1, \ldots, y_n)$

$d = e$ implies $v_1 = v_2$

Deciding Equality + (uninterpreted) Functions

$a = b, b = c, d = e, b = s, d = t, v_3 \neq v_4$

$v_1 \equiv g(d), v_2 \equiv g(e), v_3 \equiv f(a, v_1), v_4 \equiv f(b, v_2)$



Congruence Rule:

$x_1 = y_1, \ldots, x_n = y_n$ implies $f(x_1, \ldots, x_n) = f(y_1, \ldots, y_n)$

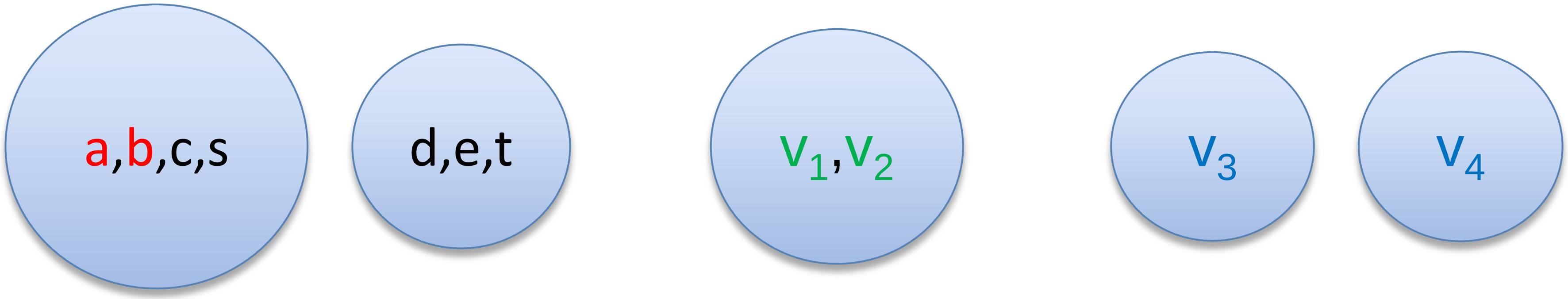
$a = b, v_1 = v_2$ implies $f(a, v_1) = f(b, v_2)$

Microsoft Research

Deciding Equality + (uninterpreted) Functions

$a = b,\ b = c,\ d = e,\ b = s,\ d = t,\ v_3 \neq v_4$

$v_1 \equiv g(d),\ v_2 \equiv g(e),\ v_3 \equiv f(a, v_1)\ ,\ v_4 \equiv f(b, v_2)$



Congruence Rule:

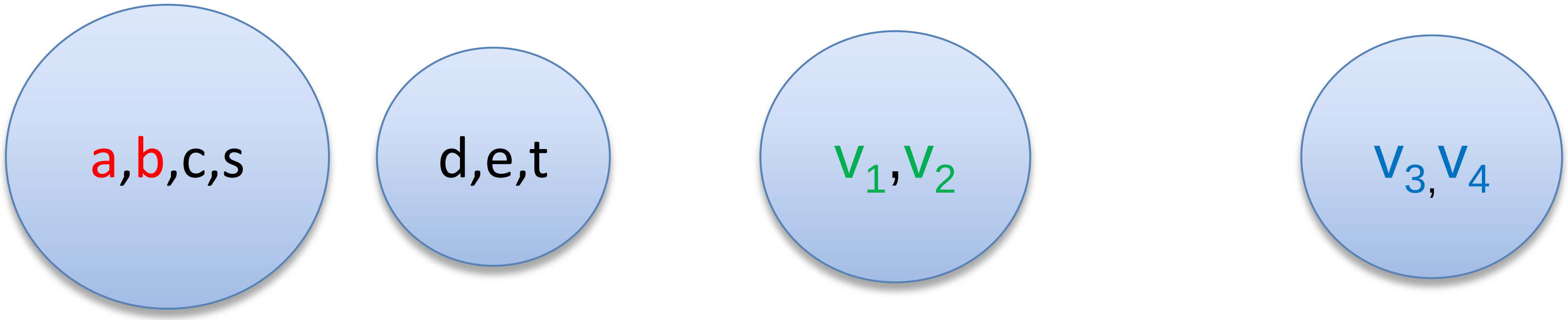
$x_1 = y_1, \ldots, x_n = y_n$ implies $f(x_1, \ldots, x_n) = f(y_1, \ldots, y_n)$

$a = b,\ v_1 = v_2$ implies $v_3 = v_4$

Deciding Equality + (uninterpreted) Functions

$a = b,\ b = c,\ d = e,\ b = s,\ d = t,\ v_3 \neq v_4$

$v_1 \equiv g(d),\ v_2 \equiv g(e),\ v_3 \equiv f(a, v_1),\ v_4 \equiv f(b, v_2)$



Congruence Rule:

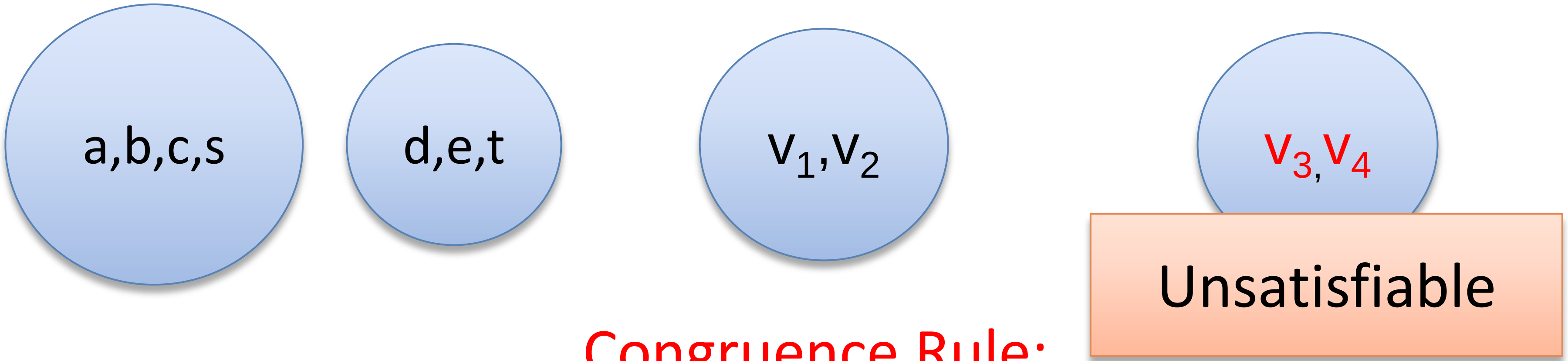
$x_1 = y_1, \ldots, x_n = y_n$ implies $f(x_1, \ldots, x_n) = f(y_1, \ldots, y_n)$

$a = b,\ v_1 = v_2$ implies $v_3 = v_4$

Deciding Equality + (uninterpreted) Functions

$a = b,\ b = c,\ d = e,\ b = s,\ d = t,\ v_3 \neq v_4$

$v_1 \equiv g(d),\ v_2 \equiv g(e),\ v_3 \equiv f(a, v_1)\ ,\ v_4 \equiv f(b, v_2)$



Congruence Rule:

$x_1 = y_1, \ldots, x_n = y_n$ implies $f(x_1, \ldots, x_n) = f(y_1, \ldots, y_n)$

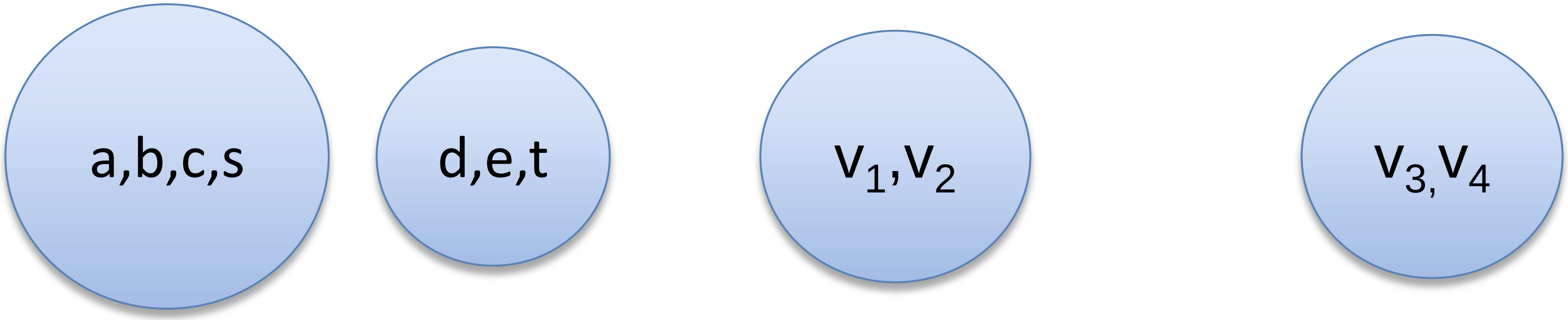
Microsoft Research

Deciding Equality + (uninterpreted) Functions

$a = b, b = c, d = e, b = s, d = t, a \neq v_4, v_2 \neq v_3$

$v_1 \equiv g(d), v_2 \equiv g(e), v_3 \equiv f(a, v_1), v_4 \equiv f(b, v_2)$

Changing the problem



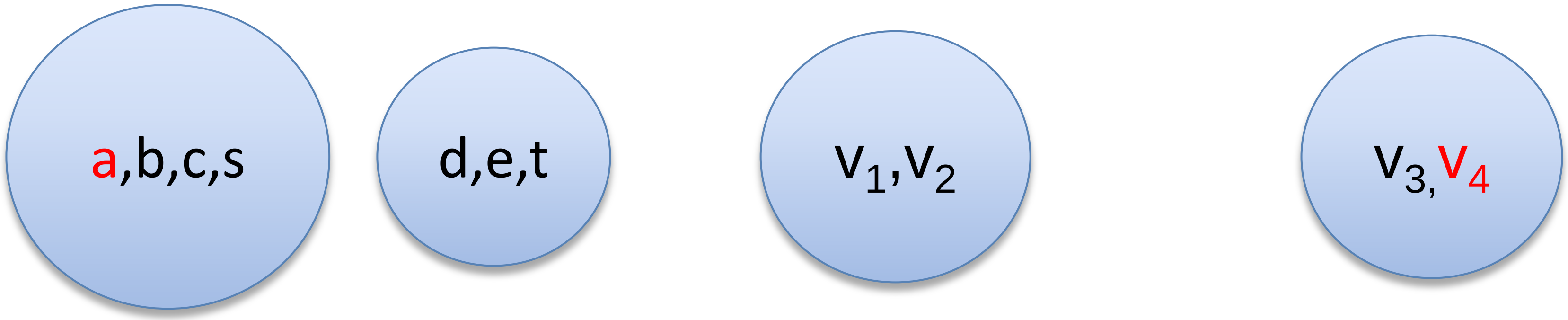
Congruence Rule:

$x_1 = y_1, \ldots, x_n = y_n$ implies $f(x_1, \ldots, x_n) = f(y_1, \ldots, y_n)$

Deciding Equality + (uninterpreted) Functions

$a = b,\ b = c,\ d = e,\ b = s,\ d = t,\ a \neq v_4,\ v_2 \neq v_3$

$v_1 \equiv g(d),\ v_2 \equiv g(e),\ v_3 \equiv f(a, v_1)\ ,\ v_4 \equiv f(b, v_2)$



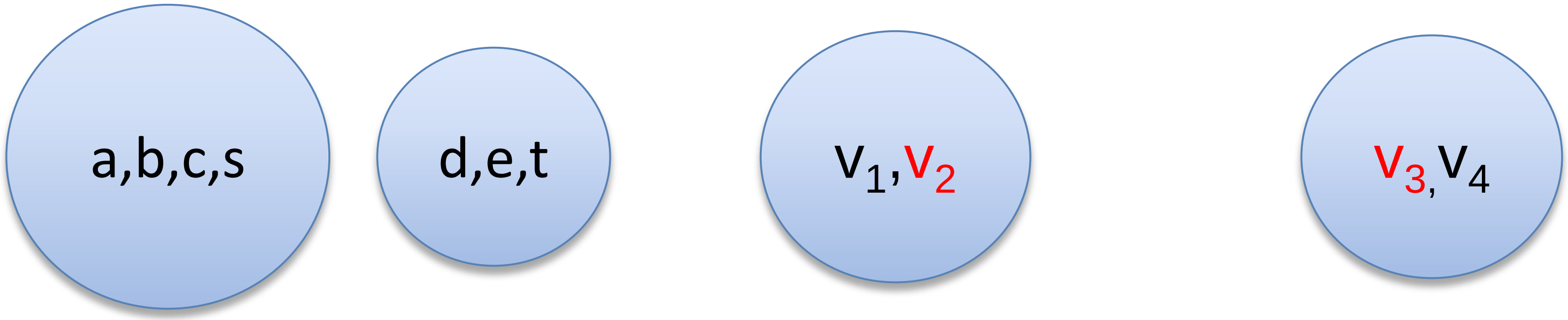
Congruence Rule:

$x_1 = y_1, \ldots, x_n = y_n$ implies $f(x_1, \ldots, x_n) = f(y_1, \ldots, y_n)$

Deciding Equality + (uninterpreted) Functions

$a = b,\ b = c,\ d = e,\ b = s,\ d = t,\ a \neq v_4,\ v_2 \neq v_3$

$v_1 \equiv g(d),\ v_2 \equiv g(e),\ v_3 \equiv f(a, v_1),\ v_4 \equiv f(b, v_2)$



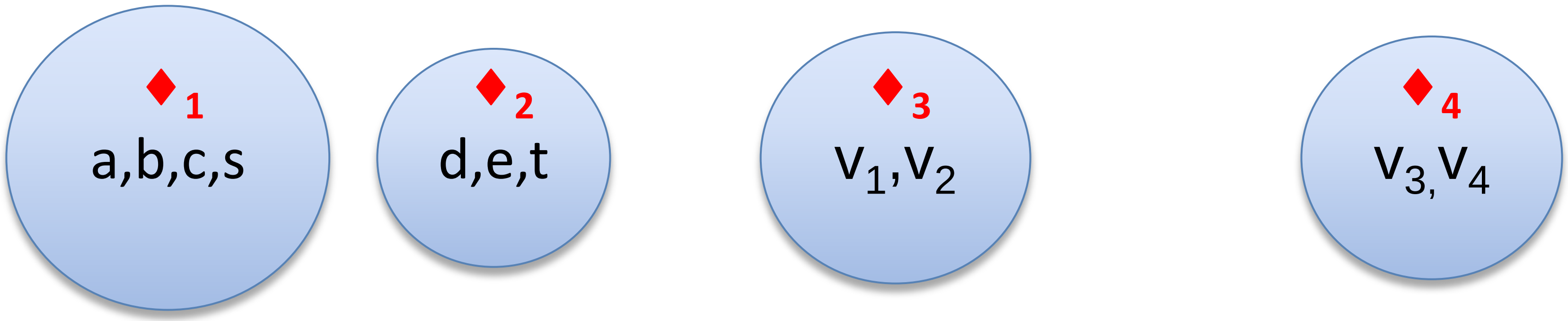
Congruence Rule:

$x_1 = y_1, \ldots, x_n = y_n$ implies $f(x_1, \ldots, x_n) = f(y_1, \ldots, y_n)$

Deciding Equality + (uninterpreted) Functions

$$a = b,\ b = c,\ d = e,\ b = s,\ d = t,\ a \neq v_4,\ v_2 \neq v_3$$

$$v_1 \equiv g(d),\ v_2 \equiv g(e),\ v_3 \equiv f(a, v_1)\ ,\ v_4 \equiv f(b, v_2)$$



Model construction:

$$|M| = \{\blacklozenge_1, \blacklozenge_2, \blacklozenge_3, \blacklozenge_4\}$$

$$M(a) = M(b) = M(c) = M(s) = \blacklozenge_1$$

$$M(d) = M(e) = M(t) = \blacklozenge_2$$

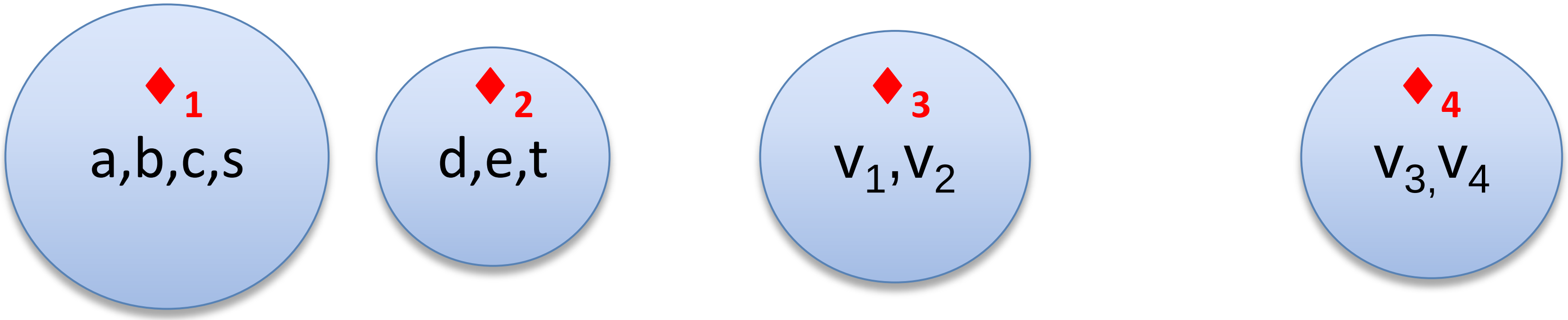
$$M(v_1) = M(v_2) = \blacklozenge_3$$

$$M(v_3) = M(v_4) = \blacklozenge_4$$

Deciding Equality + (uninterpreted) Functions

$$a = b,\ b = c,\ d = e,\ b = s,\ d = t,\ a \neq v_4,\ v_2 \neq v_3$$

$$v_1 \equiv g(d),\ v_2 \equiv g(e),\ v_3 \equiv f(a, v_1),\ v_4 \equiv f(b, v_2)$$



$\blacklozenge_1$ a,b,c,s
$\blacklozenge_2$ d,e,t
$\blacklozenge_3$ v_1,v_2
$\blacklozenge_4$ v_3,v_4

Model construction:

$$|M| = \{\blacklozenge_1, \blacklozenge_2, \blacklozenge_3, \blacklozenge_4\}$$

$$M(a) = M(b) = M(c) = M(s) = \blacklozenge_1$$

$$M(d) = M(e) = M(t) = \blacklozenge_2$$

$$M(v_1) = M(v_2) = \blacklozenge_3$$

$$M(v_3) = M(v_4) = \blacklozenge_4$$

Missing: Interpretation for f and g.

Microsoft Research

Deciding Equality + (uninterpreted) Functions

- Building the interpretation for function symbols
  - M(g) is a mapping from |M| to |M|
  - Defined as:
    $M(g)(♦_i) = ♦_j$ if there is $v \equiv g(a)$ s.t.
    $M(a) = ♦_i$
    $M(v) = ♦_j$
    $= ♦_k$, otherwise ($♦_k$ is an arbitrary element)
  - Is M(g) well-defined?

Deciding Equality + (uninterpreted) Functions

- Building the interpretation for function symbols
  - M(g) is a mapping from |M| to |M|
  - Defined as:

    $M(g)(\blacklozenge_i) = \blacklozenge_j$ if there is $v \equiv g(a)$ s.t.

    $M(a) = \blacklozenge_i$

    $M(v) = \blacklozenge_j$

    $= \blacklozenge_k$, otherwise ($\blacklozenge_k$ is an arbitrary element)
  - Is M(g) well-defined?
    - Problem: we may have

      $v \equiv g(a)$ and $w \equiv g(b)$ s.t.

      $M(a) = M(b) = \blacklozenge_1$ and $M(v) = \blacklozenge_2 \neq \blacklozenge_3 = M(w)$

      So, is $M(g)(\blacklozenge_1) = \blacklozenge_2$ or $M(g)(\blacklozenge_1) = \blacklozenge_3$?

Microsoft Research

Deciding Equality + (uninterpreted) Functions

- Building the interpretation for function symbols
  - M(g) is a mapping from |M| to |M
  - Defined as:

    M(g)($\blacklozenge_i$) = $\blacklozenge_j$ if there is v ≡ g

    M(a) = $\blacklozenge_i$

    M(v) = $\blacklozenge_j$

    = $\blacklozenge_k$, otherwise ($\blacklozenge_k$ i a ary element)
  - Is M(g) well-defined?
    - Problem: we may have

      v ≡ g(a) and w ≡ g(b) s.t.

      M(a) = M(b) = $\blacklozenge_1$ and M(v) = $\blacklozenge_2 \neq \blacklozenge_3$ = M(w)

      So, is M(g)($\blacklozenge_1$) = $\blacklozenge_2$ or M(g)($\blacklozenge_1$) = $\blacklozenge_3$?

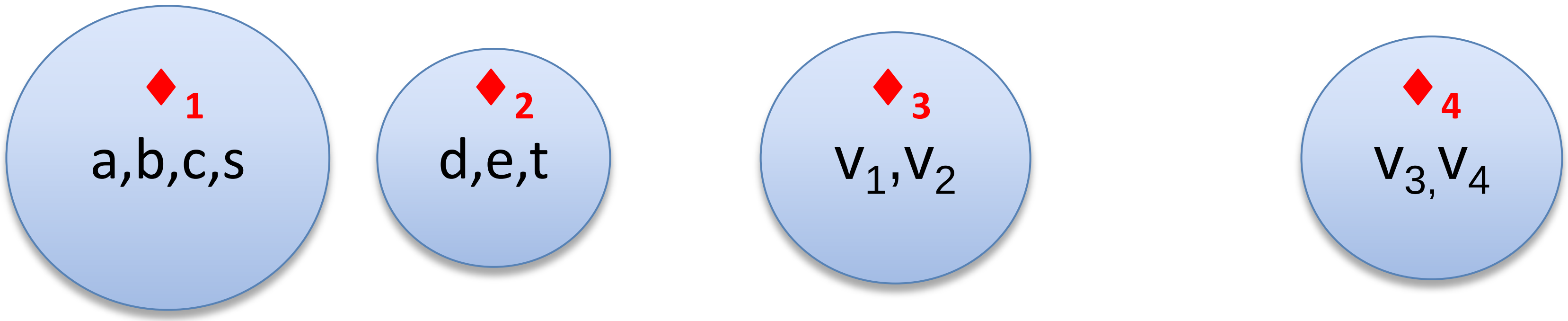
This is impossible because of the congruence rule!

a and b are in the same "ball", then so are v and w

Deciding Equality + (uninterpreted) Functions

$$a = b,\ b = c,\ d = e,\ b = s,\ d = t,\ a \neq v_4,\ v_2 \neq v_3$$

$$v_1 \equiv g(d),\ v_2 \equiv g(e),\ v_3 \equiv f(a, v_1)\ ,\ v_4 \equiv f(b, v_2)$$



$\blacklozenge_1$ a,b,c,s
$\blacklozenge_2$ d,e,t
$\blacklozenge_3$ v_1, v_2
$\blacklozenge_4$ v_3, v_4

Model construction:

$$|M| = \{\blacklozenge_1, \blacklozenge_2, \blacklozenge_3, \blacklozenge_4\}$$

$$M(a) = M(b) = M(c) = M(s) = \blacklozenge_1$$

$$M(d) = M(e) = M(t) = \blacklozenge_2$$

$$M(v_1) = M(v_2) = \blacklozenge_3$$

$$M(v_3) = M(v_4) = \blacklozenge_4$$

Deciding Equality + (uninterpreted) Functions

$a = b,\ b = c,\ d = e,\ b = s,\ d = t,\ a \neq v_4,\ v_2 \neq v_3$

$v_1 \equiv g(d),\ v_2 \equiv g(e),\ v_3 \equiv f(a, v_1)\ ,\ v_4 \equiv f(b, v_2)$

Model construction:

$|M| = \{\blacklozenge_1, \blacklozenge_2, \blacklozenge_3, \blacklozenge_4\}$

$M(a) = M(b) = M(c) = M(s) = \blacklozenge_1$

$M(d) = M(e) = M(t) = \blacklozenge_2$

$M(v_1) = M(v_2) = \blacklozenge_3$

$M(v_3) = M(v_4) = \blacklozenge_4$

$M(g)(\blacklozenge_i) = \blacklozenge_j$ if there is $v \equiv g(a)$ s.t.

$M(a) = \blacklozenge_i$

$M(v) = \blacklozenge_j$

$= \blacklozenge_k$, otherwise

Deciding Equality + (uninterpreted) Functions

$$a = b,\ b = c,\ d = e,\ b = s,\ d = t,\ a \neq v_4,\ v_2 \neq v_3$$

$$v_1 \equiv g(d),\ v_2 \equiv g(e),\ v_3 \equiv f(a, v_1),\ v_4 \equiv f(b, v_2)$$

Model construction:

$$|M| = \{\blacklozenge_1, \blacklozenge_2, \blacklozenge_3, \blacklozenge_4\}$$

$$M(a) = M(b) = M(c) = M(s) = \blacklozenge_1$$

$$M(d) = M(e) = M(t) = \blacklozenge_2$$

$$M(v_1) = M(v_2) = \blacklozenge_3$$

$$M(v_3) = M(v_4) = \blacklozenge_4$$

$$M(g) = \{\blacklozenge_2 \rightarrow \blacklozenge_3\}$$

$M(g)(\blacklozenge_i) = \blacklozenge_j$ if there is $v \equiv g(a)$ s.t.

$M(a) = \blacklozenge_i$

$M(v) = \blacklozenge_j$

$= \blacklozenge_k$, otherwise

Microsoft Research

Deciding Equality + (uninterpreted) Functions

$$a = b,\ b = c,\ d = e,\ b = s,\ d = t,\ a \neq v_4,\ v_2 \neq v_3$$

$$v_1 \equiv g(d),\ v_2 \equiv g(e),\ v_3 \equiv f(a, v_1),\ v_4 \equiv f(b, v_2)$$

Model construction:

$$|M| = \{\blacklozenge_1, \blacklozenge_2, \blacklozenge_3, \blacklozenge_4\}$$

$$M(a) = M(b) = M(c) = M(s) = \blacklozenge_1$$

$$M(d) = M(e) = M(t) = \blacklozenge_2$$

$$M(v_1) = M(v_2) = \blacklozenge_3$$

$$M(v_3) = M(v_4) = \blacklozenge_4$$

$$M(g) = \{\blacklozenge_2 \to \blacklozenge_3\}$$

$M(g)(\blacklozenge_i) = \blacklozenge_j$ if there is $v \equiv g(a)$ s.t.

$M(a) = \blacklozenge_i$

$M(v) = \blacklozenge_j$

$= \blacklozenge_k$, otherwise

Microsoft Research

Deciding Equality + (uninterpreted) Functions

$$a = b,\ b = c,\ d = e,\ b = s,\ d = t,\ a \neq v_4,\ v_2 \neq v_3$$

$$v_1 \equiv g(d),\ v_2 \equiv g(e),\ v_3 \equiv f(a, v_1),\ v_4 \equiv f(b, v_2)$$

Model construction:

$$|M| = \{♦_1, ♦_2, ♦_3, ♦_4\}$$

$$M(a) = M(b) = M(c) = M(s) = ♦_1$$

$$M(d) = M(e) = M(t) = ♦_2$$

$$M(v_1) = M(v_2) = ♦_3$$

$$M(v_3) = M(v_4) = ♦_4$$

$$M(g) = \{♦_2 \rightarrow ♦_3,\ \text{else} \rightarrow ♦_1\}$$

$M(g)(♦_i) = ♦_j$ if there is $v \equiv g(a)$ s.t.

$M(a) = ♦_i$

$M(v) = ♦_j$

$= ♦_k$, otherwise

Deciding Equality + (uninterpreted) Functions

$$a = b,\ b = c,\ d = e,\ b = s,\ d = t,\ a \neq v_4,\ v_2 \neq v_3$$

$$v_1 \equiv g(d),\ v_2 \equiv g(e),\ v_3 \equiv f(a, v_1),\ v_4 \equiv f(b, v_2)$$

Model construction:

$$|M| = \{\blacklozenge_1, \blacklozenge_2, \blacklozenge_3, \blacklozenge_4\}$$

$$M(a) = M(b) = M(c) = M(s) = \blacklozenge_1$$

$$M(d) = M(e) = M(t) = \blacklozenge_2$$

$$M(v_1) = M(v_2) = \blacklozenge_3$$

$$M(v_3) = M(v_4) = \blacklozenge_4$$

$$M(g) = \{\blacklozenge_2 \rightarrow \blacklozenge_3, \text{else} \rightarrow \blacklozenge_1\}$$

$$M(f) = \{ (\blacklozenge_1, \blacklozenge_3) \rightarrow \blacklozenge_4, \text{else} \rightarrow \blacklozenge_1\}$$

$M(g)(\blacklozenge_i) = \blacklozenge_j$ if there is $v \equiv g(a)$ s.t.

$M(a) = \blacklozenge_i$

$M(v) = \blacklozenge_j$

$= \blacklozenge_k$, otherwise

Microsoft Research

Deciding Equality + (uninterpreted) Functions

What about predicates?

$p(a, b), \quad \neg p(c, b)$

Deciding Equality + (uninterpreted) Functions

What about predicates?

$p(a, b), \quad \neg p(c, b)$

$\Downarrow$

$f_p(a, b) = T, \quad f_p(c, b) \neq T$

Microsoft Research

Ackermannization

It is possible to eliminate function symbols using a method called **Ackermannization**.

$$a = b,\ b = c,\ d = e,\ b = s,\ d = t,\ a \neq v_4,\ v_2 \neq v_3$$
$$v_1 \equiv g(d),\ v_2 \equiv g(e),\ v_3 \equiv f(a, v_1)\ ,\ v_4 \equiv f(b, v_2)$$

$$\Downarrow$$

$$a = b,\ b = c,\ d = e,\ b = s,\ d = t,\ a \neq v_4,\ v_2 \neq v_3$$
$$d \neq e \vee v_1 = v_2,$$
$$a \neq v_1 \vee b \neq v_2 \vee v_3 = v_4$$

Microsoft Research

Ackermannization

It is possible to eliminate function symbols using a method called **Ackermannization**.

$$a = b,\ b = c,\ d = e,\ b = s,\ d = t,\ a \neq v_4,\ v_2 \neq v_3$$

$$v_1 \equiv g(d),\ v_2 \equiv g(e),\ v_3 \equiv f(a, v_1)\ ,\ v_4 \equiv f(b, v_2)$$

$$\Downarrow$$

$$a = b,\ b = c,\ d = e,\ b = s,\ d = t,\ a \neq v_4,\ v_2 \neq v_3$$

$$d \neq e \vee v_1 = v_2,$$

$$a \neq v_1 \vee b \neq v_2 \vee v_3 = v_4$$

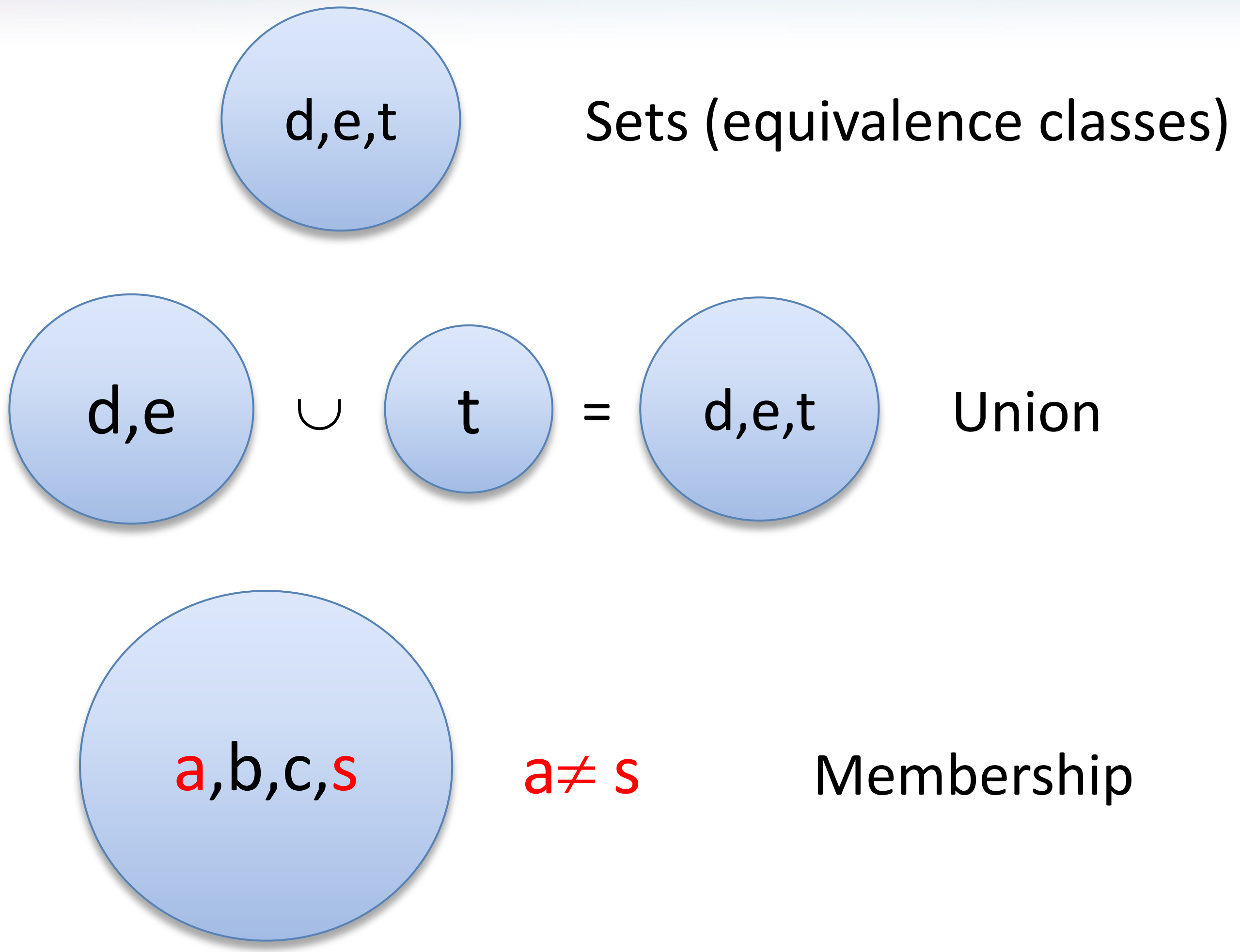
Main Problem: quadratic blowup

Microsoft Research

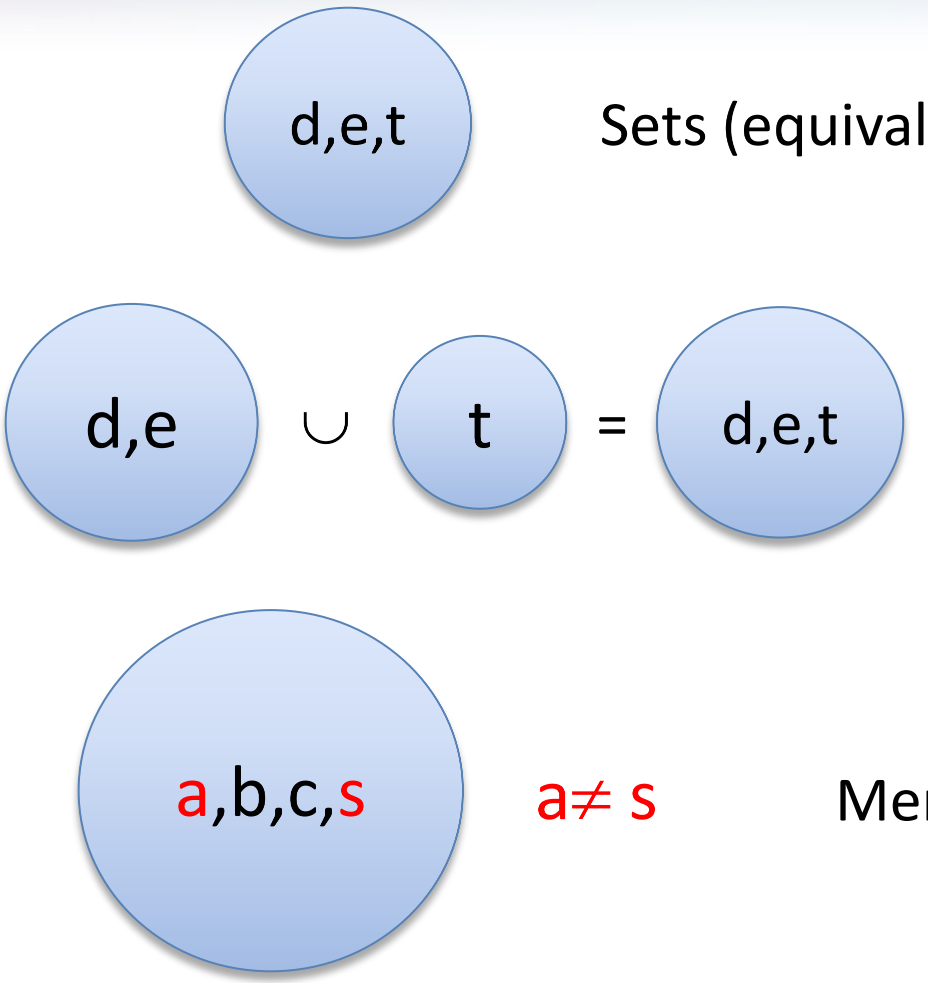
Deciding Equality + (uninterpreted) Functions

It is possible to implement our procedure in O(n log n)

Deciding Equality + (uninterpreted) Functions



Deciding Equality + (uninterpreted) Functions



Sets (equivale

Key observation:
The sets are disjoint!

Union

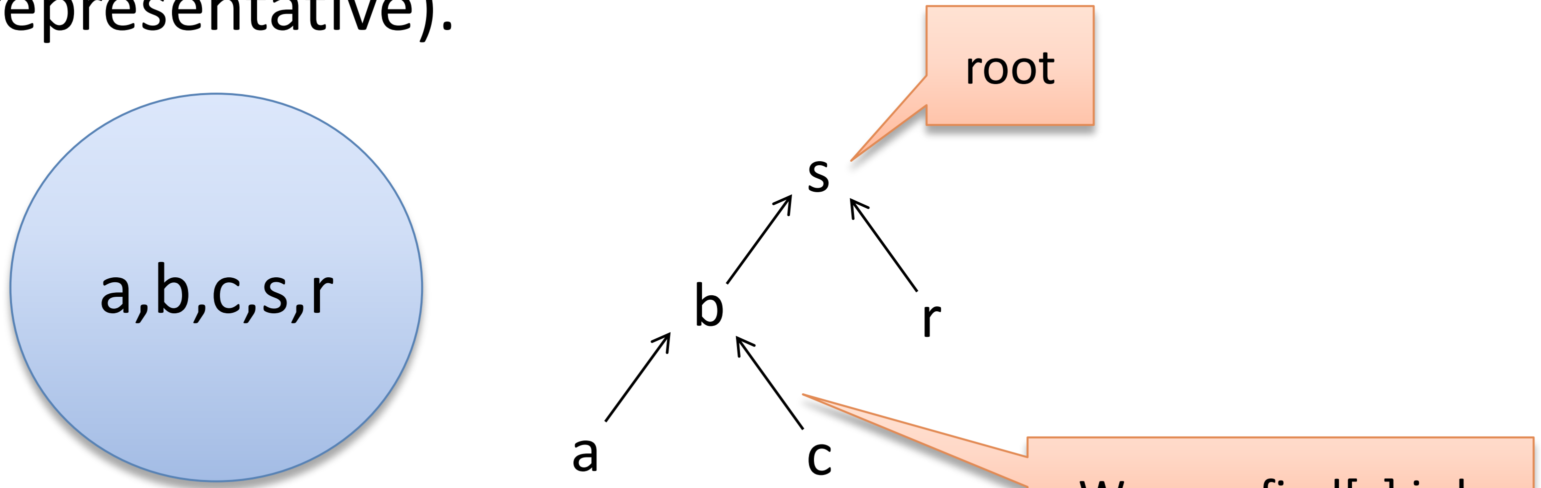
a≠ s

Membership

Deciding Equality + (uninterpreted) Functions

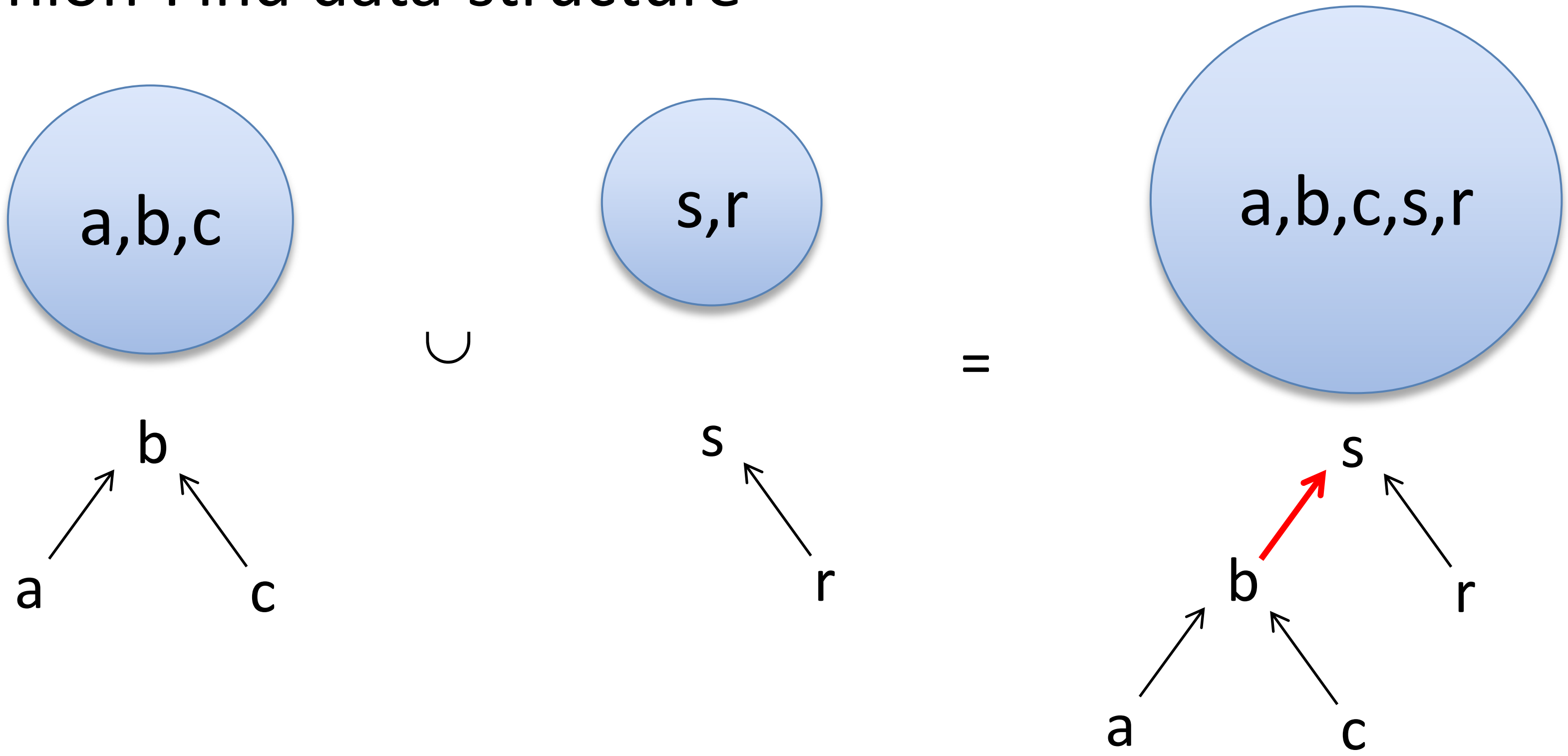
Union-Find data-structure

Every set (equivalence class) has a root element (representative).



Deciding Equality + (uninterpreted) Functions

Union-Find data-structure



Deciding Equality + (uninterpreted) Functions

Tracking the equivalence classes size is important!

$$a_1 \longrightarrow a_2 \quad \cup \quad a_3 \quad = \quad a_1 \longrightarrow a_2 \longrightarrow a_3$$

$$a_1 \longrightarrow a_2 \longrightarrow a_3 \quad \cup \quad a_4 \quad = \quad a_1 \longrightarrow a_2 \longrightarrow a_3 \longrightarrow a_4$$

…

$$a_1 \longrightarrow a_2 \longrightarrow a_3 \longrightarrow \ldots \longrightarrow a_{n-1} \quad \cup \quad a_n \quad =$$

$$a_1 \longrightarrow a_2 \longrightarrow a_3 \longrightarrow \ldots \longrightarrow a_{n-1} \longrightarrow a_n$$

Deciding Equality + (uninterpreted) Functions

Tracking the equivalence classes size is important!

$$\begin{tikzcd}
a_1 \arrow[r] & a_2 & \cup & a_3 & = & a_1 \arrow[r] & a_2 & a_3 \arrow[l]
\end{tikzcd}$$

$$\begin{tikzcd}
a_1 \arrow[r] & a_2 & a_3 \arrow[l] & \cup & a_4 & = & a_1 \arrow[r] & a_2 & a_3 \arrow[l] \\
& & & & & & & & a_4 \arrow[ul]
\end{tikzcd}$$

…

$$\begin{tikzcd}
& a_2 & & & \cup & a_n & = & & a_2 & & & a_n \arrow[lll] \\
a_1 \arrow[ur] & a_3 \arrow[u] & \cdots & a_{n-1} \arrow[ull] & & & & a_1 \arrow[ur] & a_3 \arrow[u] & \cdots & a_{n-1} \arrow[ull]
\end{tikzcd}$$

Deciding Equality + (uninterpreted) Functions

Tracking the equivalence classes size is important!

We can do n merges in O(n log n)

$$\begin{tikzcd} a_1 \arrow[r] & a_2 & \cup & a_3 & = & a_1 \arrow[r] & a_2 & a_3 \arrow[l] \end{tikzcd}$$

$$\begin{tikzcd} a_1 \arrow[r] & a_2 & a_3 \arrow[l] & \cup & a_4 & = & a_1 \arrow[r] & a_2 & a_3 \arrow[l] \\ & & & & & & & & a_4 \arrow[ul] \end{tikzcd}$$

…

$$\begin{tikzcd} & a_2 & & & & & & & a_2 & a_n \arrow[l] \\ a_1 \arrow[ur] & a_3 \arrow[u] & \ldots & a_{n-1} \arrow[ull] & \cup & a_n & = & a_1 \arrow[ur] & a_3 \arrow[u] & \ldots & a_{n-1} \arrow[ull] \end{tikzcd}$$

Each constant has two fields: find and size.

Microsoft Research

Deciding Equality + (uninterpreted) Functions

Implementing the congruence rule.

Occurrences of a constant: we say a occurs in v iff $v \equiv f(\ldots,a,\ldots)$

When we “merge” two equivalence classes we can traverse these occurrences to find new congruences.

$$\begin{tikzcd}
 & b & & & s & \\
a \arrow[ur] & & c \arrow[ul] & \cup & & r \arrow[ul]
\end{tikzcd}$$

occurrences[b] = $\{ v_1 \equiv g(b), v_2 \equiv f(a) \}$
occurrences[s] = $\{ v_3 \equiv f(r) \}$

Deciding Equality + (uninterpreted) Functions

Implementing the congruence rule.

Occurrences of a constant: we say a occurs in v iff $v \equiv f(\ldots,a,\ldots)$

When we "merge" two equivalence classes we can traverse these occurrences to find new congruences.

$$\begin{tikzcd} & b & & \cup & s & \\ a \arrow[ur] & & c \arrow[ul] & & & r \arrow[ul] \end{tikzcd}$$

occurrences(b) = { $v_1 \equiv g(b)$, $v_2 \equiv f(a)$ }
occurrences(s) = { $v_3 \equiv f(r)$ }

Inefficient version:

```
for each v in occurrences(b)
    for each w in occurrences(s)
        if v and w are congruent
            add (v,w) to todo queue
```

A queue of pairs that need to be merged.

Deciding Equality + (uninterpreted) Functions

$$\begin{tikzcd}
& b & & & s \\
a \arrow[ur] & & c \arrow[ul] & \cup & & r \arrow[ul]
\end{tikzcd}$$

occurrences[b] = { $v_1 \equiv g(b)$, $v_2 \equiv f(a)$ }
occurrences[s] = { $v_3 \equiv f(r)$ }

We also need to merge occurrences[b] with occurrences[s].
This can be done in constant time:
Use circular lists to represent the occurrences. (More later)

$$\begin{tikzcd}
v_1 \arrow[dd, bend left] & & & & v_1 \arrow[dr, bend left] & \\
& \cup & v_3 \arrow[loop above] & = & & v_3 \arrow[dl, bend left] \\
v_2 \arrow[uu, bend left] & & & & v_2 \arrow[uu, bend left] &
\end{tikzcd}$$

Deciding Equality + (uninterpreted) Functions

Avoiding the nested loop:

```
for each v in occurrences[b]
    for each w in occurrences[s]
        …
```

Use a hash table to store the elements $v_1 \equiv f(a_1, \ldots, a_n)$.
Each constant has an identifier (e.g., natural number).
Compute hash code using the identifier of the (equivalence class) roots of the arguments.

$\text{hash}(v_1) = \text{hash-tuple}(\text{id}(f), \text{id}(\text{root}(a_1)), \ldots, \text{id}(\text{root}(a_n)))$

Deciding Equality + (uninterpreted) Functions

Avoiding the nested loop:

```
for each v in occurrences(b)
    for each w in occurrences(s)
        …
```

Use a hash table to , …, a_n).
Each constant has a mber).
Compute hash code equivalence
class) roots of the argu .

hash-tuple can be the Jenkin's hash function for strings.
Just adding the ids produces a very bad hash-code!

$hash(v_1) = hash\text{-}tuple(id(f), id(root(a_1)), \ldots, id(root(a_n)))$

Microsoft Research

Deciding Equality + (uninterpreted) Functions

Efficient implementation of the congruence rule.

Merging the equivalences classes with roots: a_1 and a_2

Assume a_2 is smaller than a_1

Before merging the equivalence classes: a_1 and a_2

```
for each v in occurrences[$a_2$]
    remove v from the hash table   (its hashcode will change)
```

After merging the equivalence classes: a_1 and a_2

```
for each v in occurrences[$a_2$]
    if there is w congruent to v in the hash-table
        add (v,w) to todo queue
    else add v to hash-table
```

Deciding Equality + (uninterpreted) Functi

Trick:
Use dynamic arrays to represent the occurrences

Efficient implementation of the congru

Merging the equivalences classes with roo $_1$ and a$_2$

Assume a$_2$ is smaller than a$_1$

Before merging the equivalence classes: a$_1$ and a$_2$

```
for each v in occurrences[a_2]
     remove v from the hash table   (its hashcode will change)
```

After merging the equivalence classes: a$_1$ and a$_2$

```
for each v in occurrences[a_2]
     if there is w congruent to v in the hash-table
          add (v,w) to todo queue
     else add v to hash-table
          add v to occurrences(a_1)
```

Microsoft Research

Deciding Equality + (uninterpreted) Functions

The efficient version is not optimal (in theory).

Problem: we may have $v \equiv f(a_1, \ldots, a_n)$ with "huge" n.

Solution: currying

Use only binary functions, and represent $f(a_1, a_2, a_3, a_4)$ as

$f(a_1, h(a_2, h(a_3, a_4)))$

This is not necessary in practice, since the n above is small.

Deciding Equality + (uninterpreted) Functions

Each constant has now three fields:
find, size, and occurrences.

We also has use a hash-table for implementing the congruence rule.

We will need many more improvements!

Case Analysis

Many verification/analysis problems require:

case-analysis

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

Microsoft Research

Case Analysis

Many verification/analysis problems require:

case-analysis

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

Naïve Solution: Convert to DNF

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

Case Analysis

Many verification/analysis problems require:
case-analysis

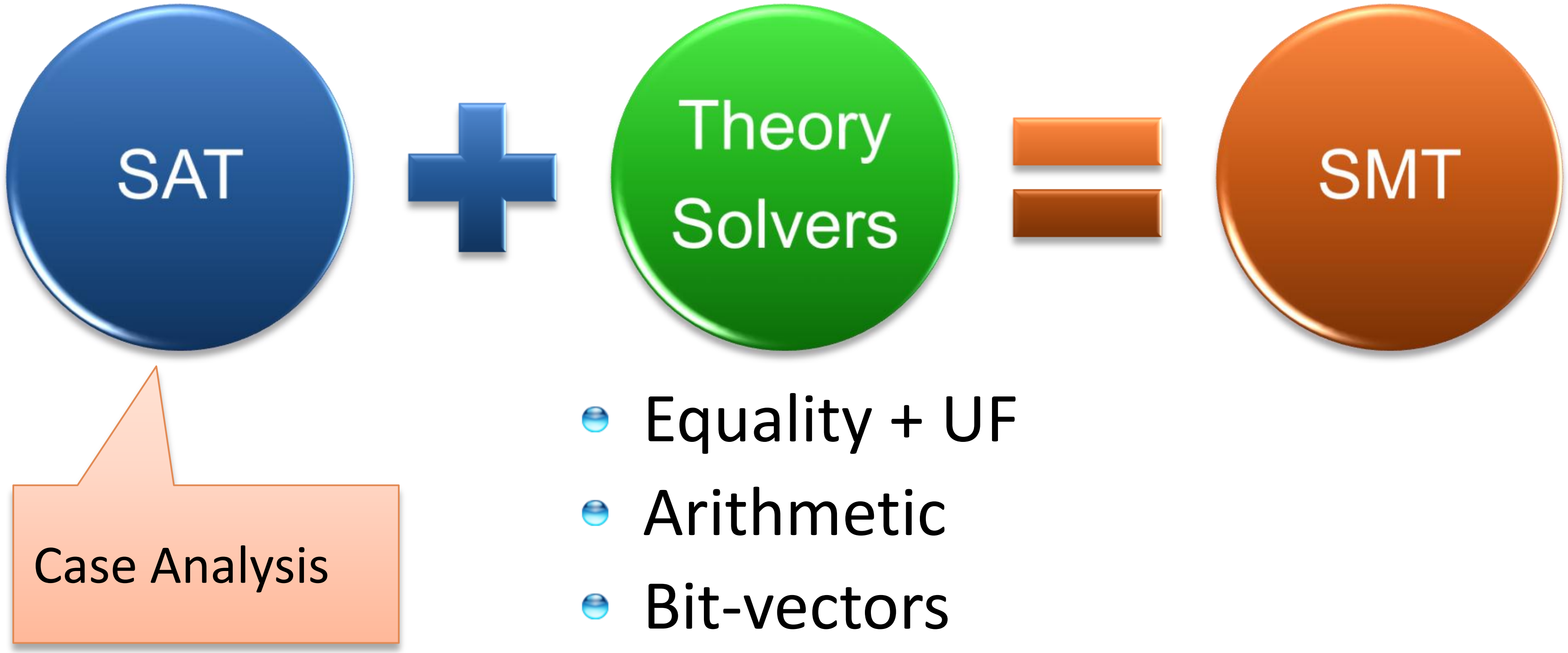
$x \geq 0, y = x + 1, (y > 2 \vee y < 1)$

Naïve Solution: Convert to DNF

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

Too Inefficient!
(exponential blowup)

SMT : Basic Architecture



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

Microsoft Research

DPLL

M | F

Partial model

Set of clauses

Microsoft Research

DPLL

Guessing

$$p \mid p \lor q, \neg q \lor r$$

$$\Downarrow$$

$$p, \neg q \mid p \lor q, \neg q \lor r$$

Microsoft Research

DPLL

Deducing

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

$$\Downarrow$$

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

Microsoft Research

DPLL

Backtracking

$$\begin{tikzcd}
p, \neg s, q \mid p \vee q, s \vee q, \neg p \vee \neg q \arrow[d] \\
p, s \mid p \vee q, s \vee q, \neg p \vee \neg q
\end{tikzcd}$$

Modern DPLL

- Efficient indexing (two-watch literal)
- Non-chronological backtracking (backjumping)
- Lemma learning

SAT + Theory solvers

Basic Idea

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

$\Downarrow$ Abstract (aka “naming” atoms)

$p_1,\ p_2,\ (p_3 \lor p_4)$ $p_1 \equiv (x \geq 0),\ p_2 \equiv (y = x + 1),$

$p_3 \equiv (y > 2),\ p_4 \equiv (y < 1)$

SAT + Theory solvers

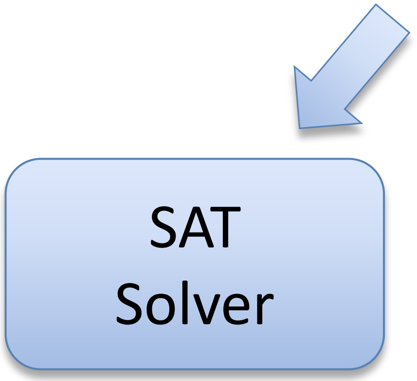
Basic Idea

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

Abstract (aka "naming" atoms)

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

$p_1,\ p_2,\ (p_3 \vee p_4)$



SAT + Theory solvers

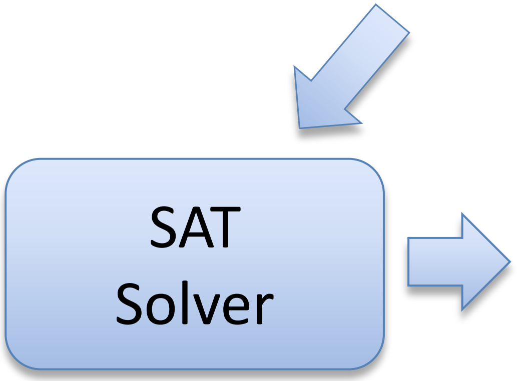
Basic Idea

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

Abstract (aka "naming" atoms)

$p_1,\ p_2,\ (p_3 \vee p_4)$

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



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

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

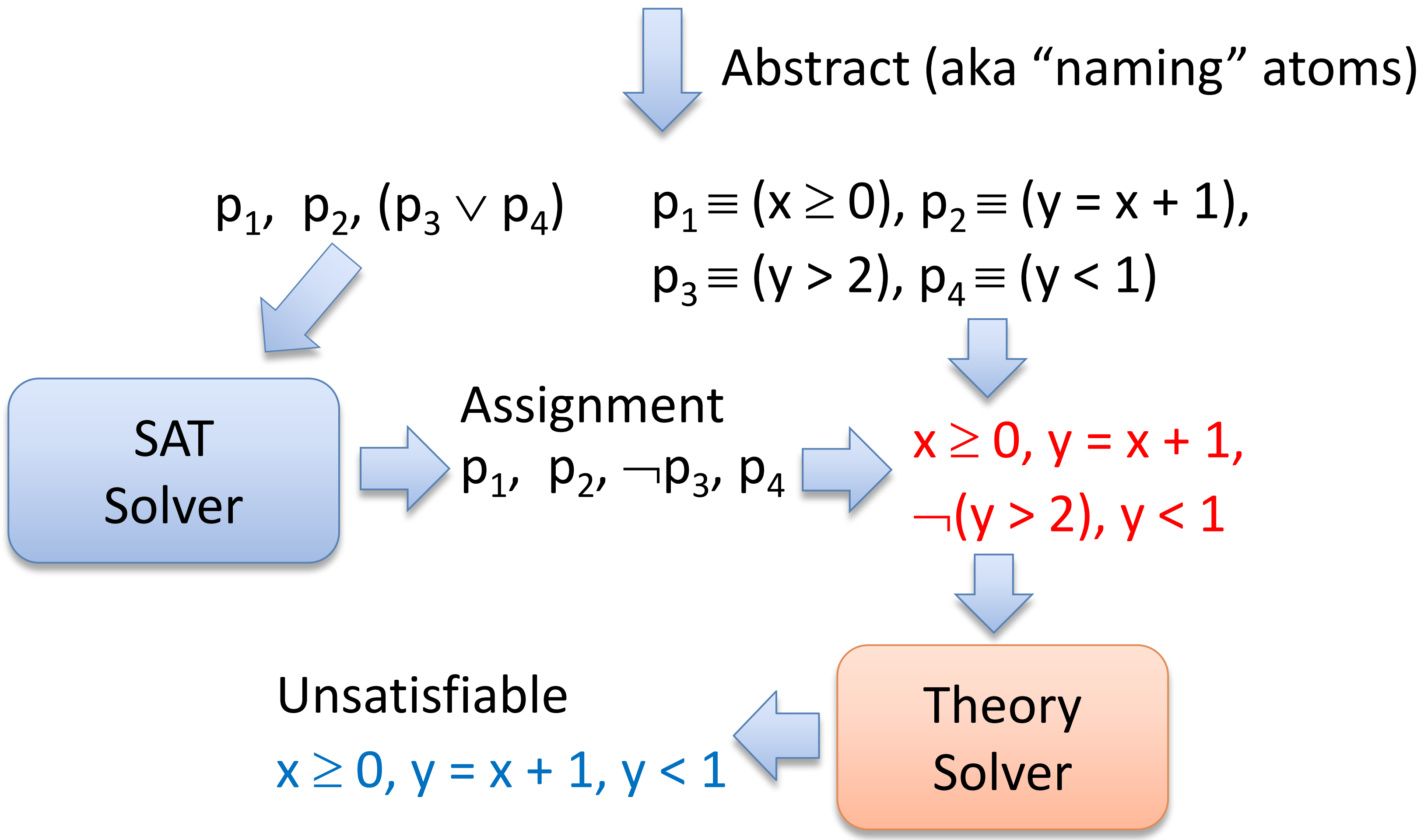
Assignment
$p_1,\ p_2,\ \neg p_3,\ p_4$

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

SAT + Theory solvers

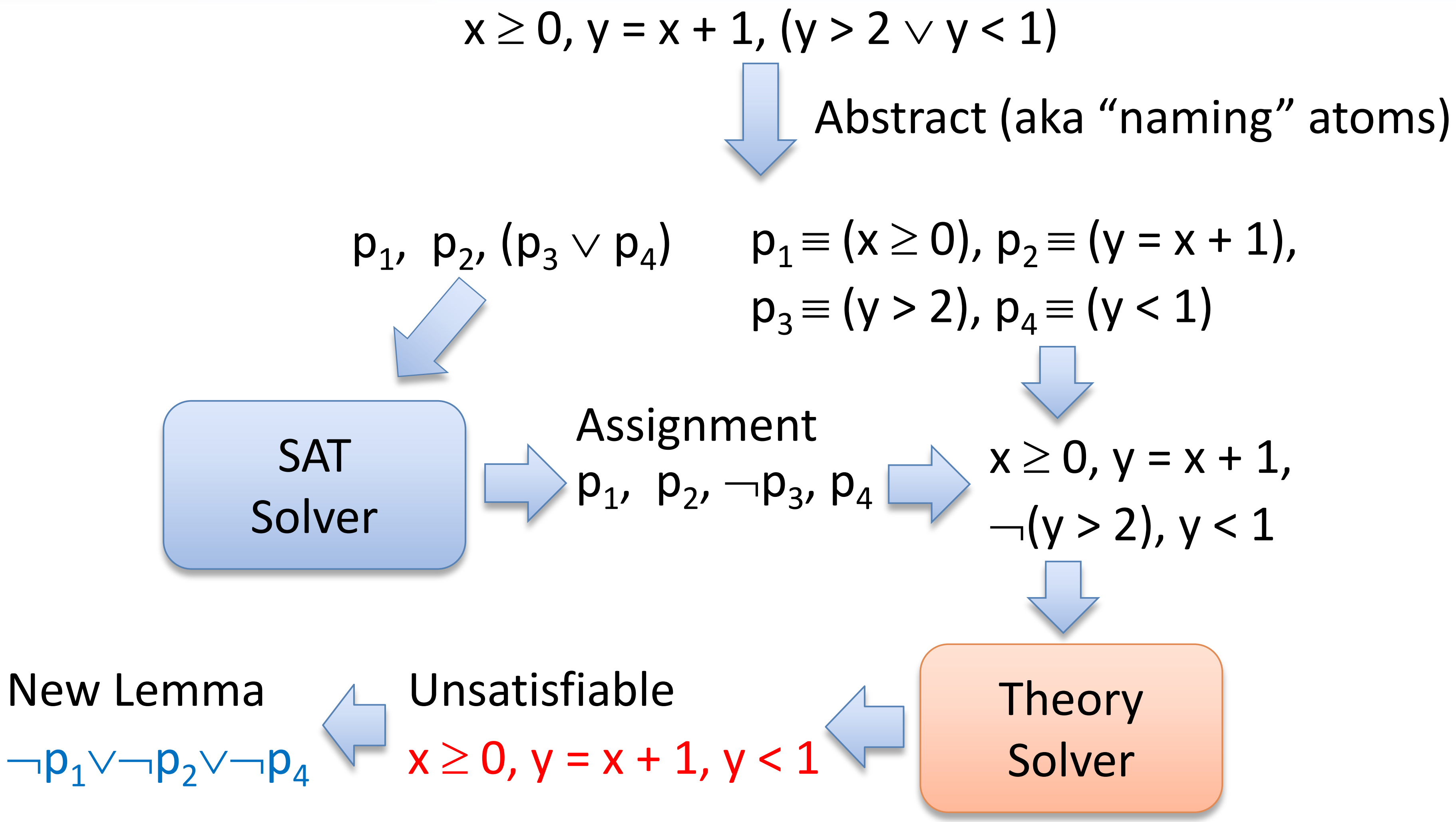
Basic Idea

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


Abstract (aka "naming" atoms)
$p_1,\ p_2,\ (p_3 \vee p_4)$
$p_1 \equiv (x \geq 0),\ p_2 \equiv (y = x + 1),$
$p_3 \equiv (y > 2),\ p_4 \equiv (y < 1)$
SAT Solver
Assignment
$p_1,\ p_2,\ \neg p_3,\ p_4$
$x \geq 0,\ y = x + 1,$
$\neg(y > 2),\ y < 1$
Theory Solver
Unsatisfiable
$x \geq 0,\ y = x + 1,\ y < 1$

SAT + Theory solvers

Basic Idea



$x \geq 0, y = x + 1, (y > 2 \lor y < 1)$
Abstract (aka "naming" atoms)
$p_1, p_2, (p_3 \lor p_4)$
$p_1 \equiv (x \geq 0), p_2 \equiv (y = x + 1),$
$p_3 \equiv (y > 2), p_4 \equiv (y < 1)$
SAT Solver
Assignment
$p_1, p_2, \neg p_3, p_4$
$x \geq 0, y = x + 1,$
$\neg(y > 2), y < 1$
Theory Solver
Unsatisfiable
$x \geq 0, y = x + 1, y < 1$
New Lemma
$\neg p_1 \lor \neg p_2 \lor \neg p_4$

SAT + Theory solvers

New Lemma

$\neg p_1 \vee \neg p_2 \vee \neg p_4$

Unsatisfiable

$x \geq 0,\ y = x + 1,\ y < 1$

Theory Solver

AKA

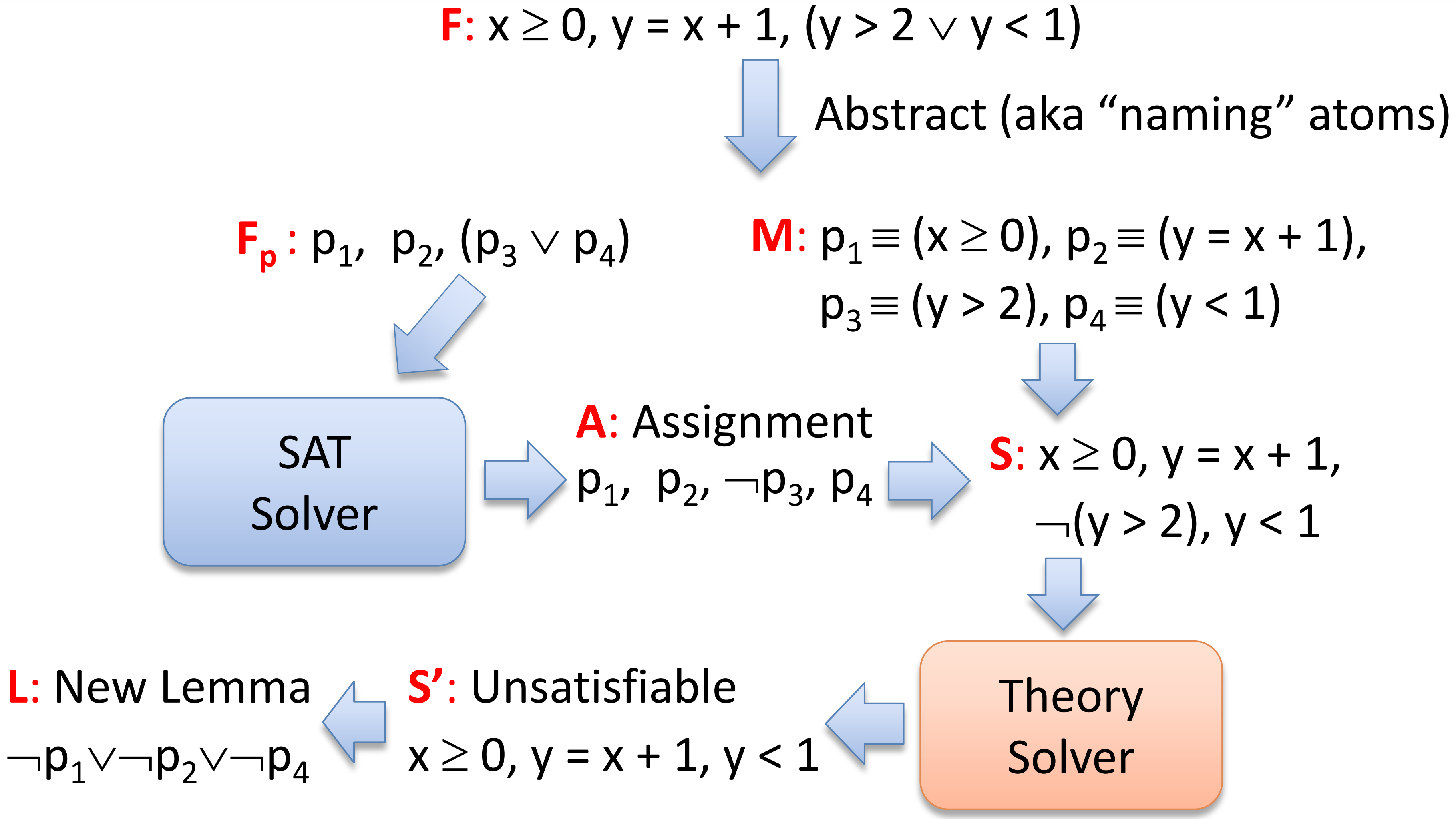
Theory conflict

SAT + Theory solvers: Main loop

```
procedure SmtSolver(F)
    ($F_p$, M) := Abstract(F)
    loop
        (R, A) := SAT_solver($F_p$)
        if R = UNSAT then return UNSAT
        S := Concretize(A, M)
        (R, S’) := Theory_solver(S)
        if R = SAT then return SAT
        L := New_Lemma(S’, M)
        Add L to $F_p$
```

SAT + Theory solvers

Basic Idea



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

F$_p$: $p_1,\ p_2,\ (p_3 \vee p_4)$

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

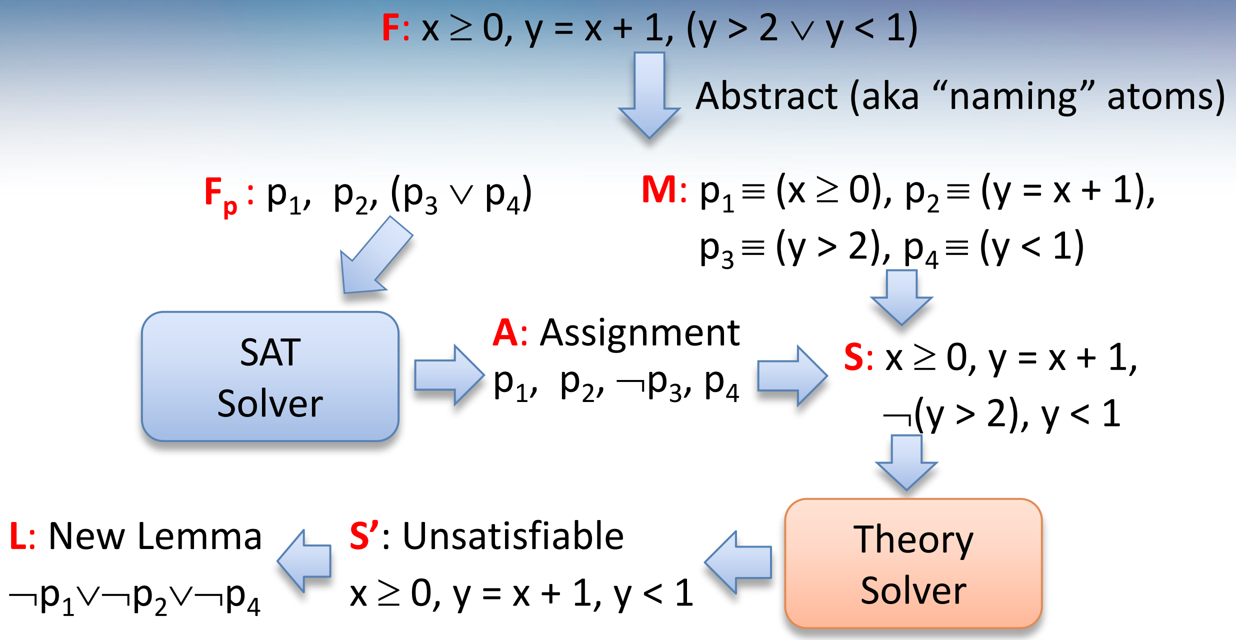
A: Assignment
$p_1,\ p_2,\ \neg p_3,\ p_4$

S: $x \geq 0,\ y = x + 1,$
$\neg(y > 2),\ y < 1$

S’: Unsatisfiable
$x \geq 0,\ y = x + 1,\ y < 1$

L: New Lemma
$\neg p_1 \vee \neg p_2 \vee \neg p_4$

SAT + Theory solvers



F: $x \geq 0, y = x + 1, (y > 2 \vee y < 1)$
Abstract (aka "naming" atoms)
F_p : $p_1, p_2, (p_3 \vee p_4)$
M: $p_1 \equiv (x \geq 0), p_2 \equiv (y = x + 1),$
$p_3 \equiv (y > 2), p_4 \equiv (y < 1)$
SAT Solver
A: Assignment
$p_1, p_2, \neg p_3, p_4$
S: $x \geq 0, y = x + 1,$
$\neg(y > 2), y < 1$
Theory Solver
S': Unsatisfiable
$x \geq 0, y = x + 1, y < 1$
L: New Lemma
$\neg p_1 \vee \neg p_2 \vee \neg p_4$

```
procedure SMT_Solver(F)
    (F_p, M) := Abstract(F)
    loop
        (R, A) := SAT_solver(F_p)
        if R = UNSAT then return UNSAT
        S = Concretize(A, M)
        (R, S') := Theory_solver(S)
        if R = SAT then return SAT
        L := New_Lemma(S, M)
        Add L to F_p
```

"Lazy translation"
to
DNF

SAT + Theory solvers

State-of-the-art SMT solvers implement many improvements.

SAT + Theory solvers

Incrementality

Send the literals to the Theory solver as they are assigned by the SAT solver

$p_1 \equiv (x \geq 0)$, $p_2 \equiv (y = x + 1)$,

$p_3 \equiv (y > 2)$, $p_4 \equiv (y < 1)$, $p_5 \equiv (x < 2)$,

$p_1, \ p_2, \ p_4 \ \ | \ \ p_1, \ p_2, (p_3 \vee p_4), (p_5 \vee \neg p_4)$

Partial assignment is already Theory inconsistent.

SAT + Theory solvers

Efficient Backtracking

We don’t want to restart from scratch after each backtracking operation.

SAT + Theory solvers

Efficient Lemma Generation (computing a small S')

Avoid lemmas containing redundant literals.

$p_1 \equiv (x \geq 0)$, $p_2 \equiv (y = x + 1)$,

$p_3 \equiv (y > 2)$, $p_4 \equiv (y < 1)$, $p_5 \equiv (x < 2)$,

$p_1, \ p_2, p_3, p_4 \ \ | \ \ p_1, \ p_2, (p_3 \vee p_4), (p_5 \vee \neg p_4)$

$\neg p_1 \vee \neg p_2 \vee \neg p_3 \vee \neg p_4$

Imprecise Lemma

SAT + Theory solvers

Theory Propagation

It is the SMT equivalent of unit propagation.

$p_1 \equiv (x \geq 0)$, $p_2 \equiv (y = x + 1)$,

$p_3 \equiv (y > 2)$, $p_4 \equiv (y < 1)$, $p_5 \equiv (x < 2)$,

$p_1,\ p_2 \mid\ p_1,\ p_2, (p_3 \vee p_4), (p_5 \vee \neg p_4)$

p_1, p_2 imply $\neg p_4$ by theory propagation

$p_1,\ p_2,\ \neg p_4 \mid\ p_1,\ p_2, (p_3 \vee p_4), (p_5 \vee \neg p_4)$

SAT + Theory solvers

Theory Propagation

It is the SMT equivalent of unit propagation.

$p_1 \equiv (x \geq 0)$, $p_2 \equiv (y = x + 1)$,

$p_3 \equiv (y > 2)$, $p_4 \equiv (y < 1)$, $p_5 \equiv (x < 2)$,

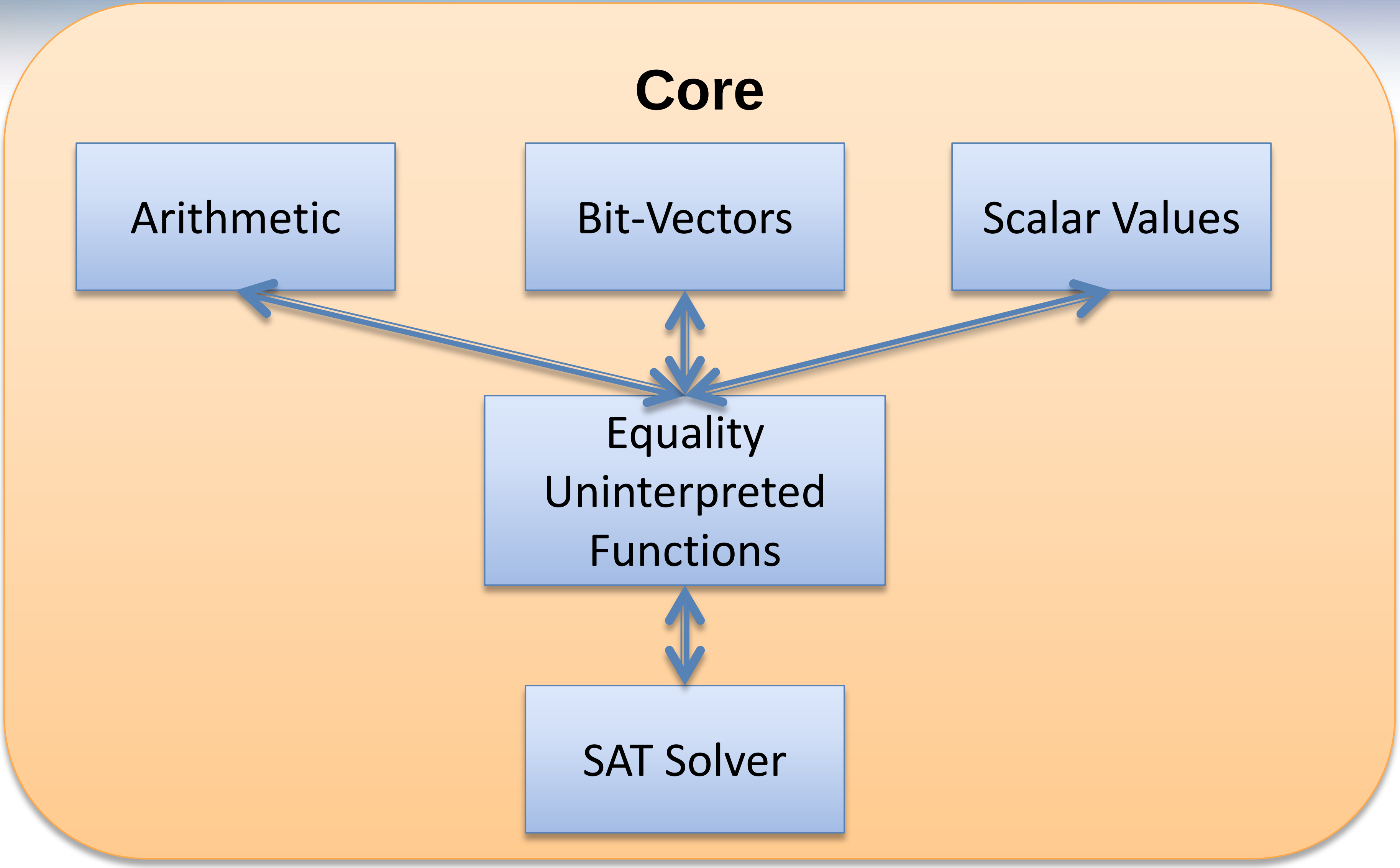
$p_1,\ p_2 \mid\ p_1,\ p_2, (p_3 \vee p_4), (p_5 \vee \neg p_4)$

p_1, p_2 imply $\neg p_4$ by theory propagation

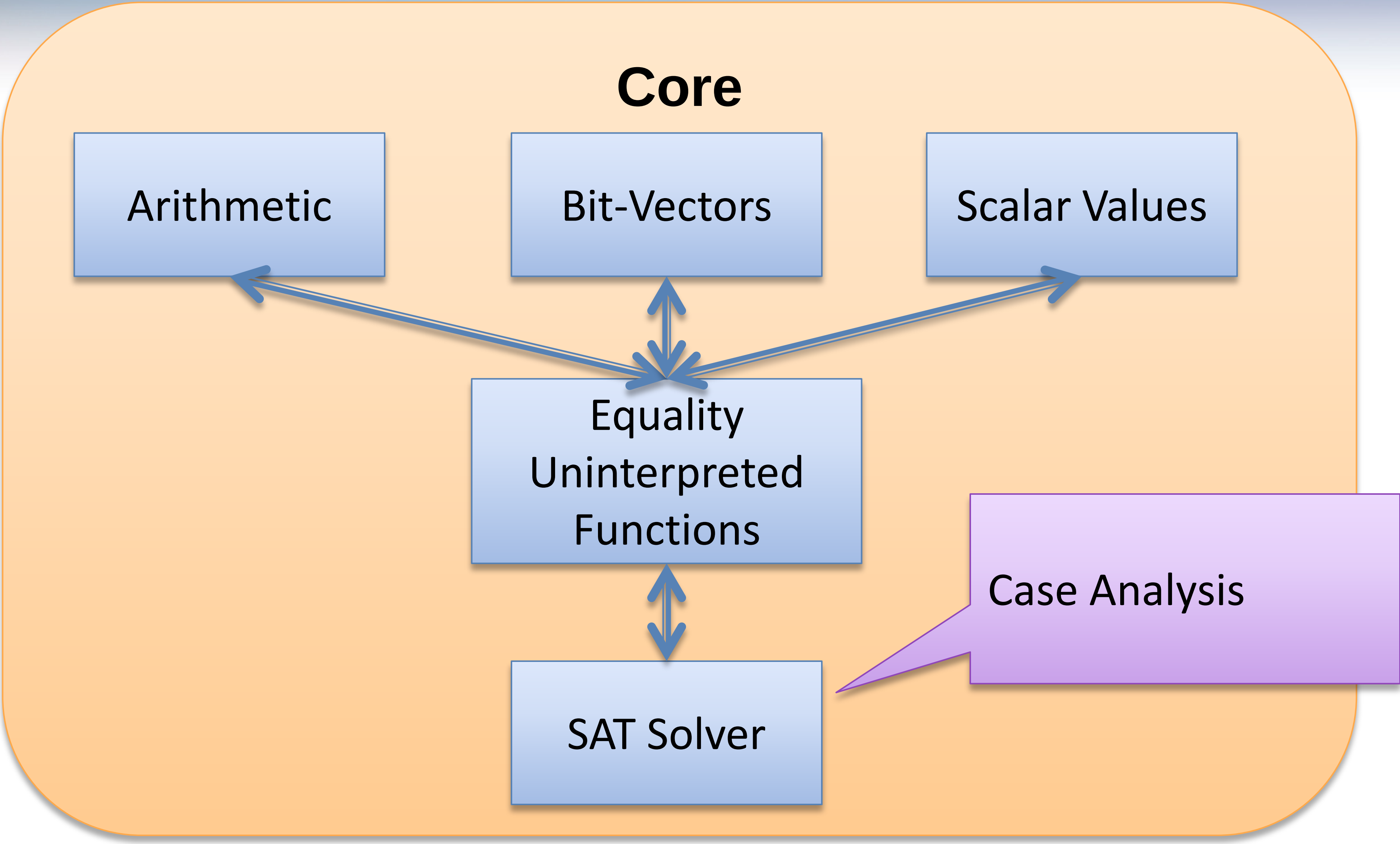
$p_1,\ p_2, \neg p_4 \mid\ p_1,\ p_2, (p_3 \vee p_4), (p_5 \vee \neg p_4)$

Tradeoff between precision × performance.

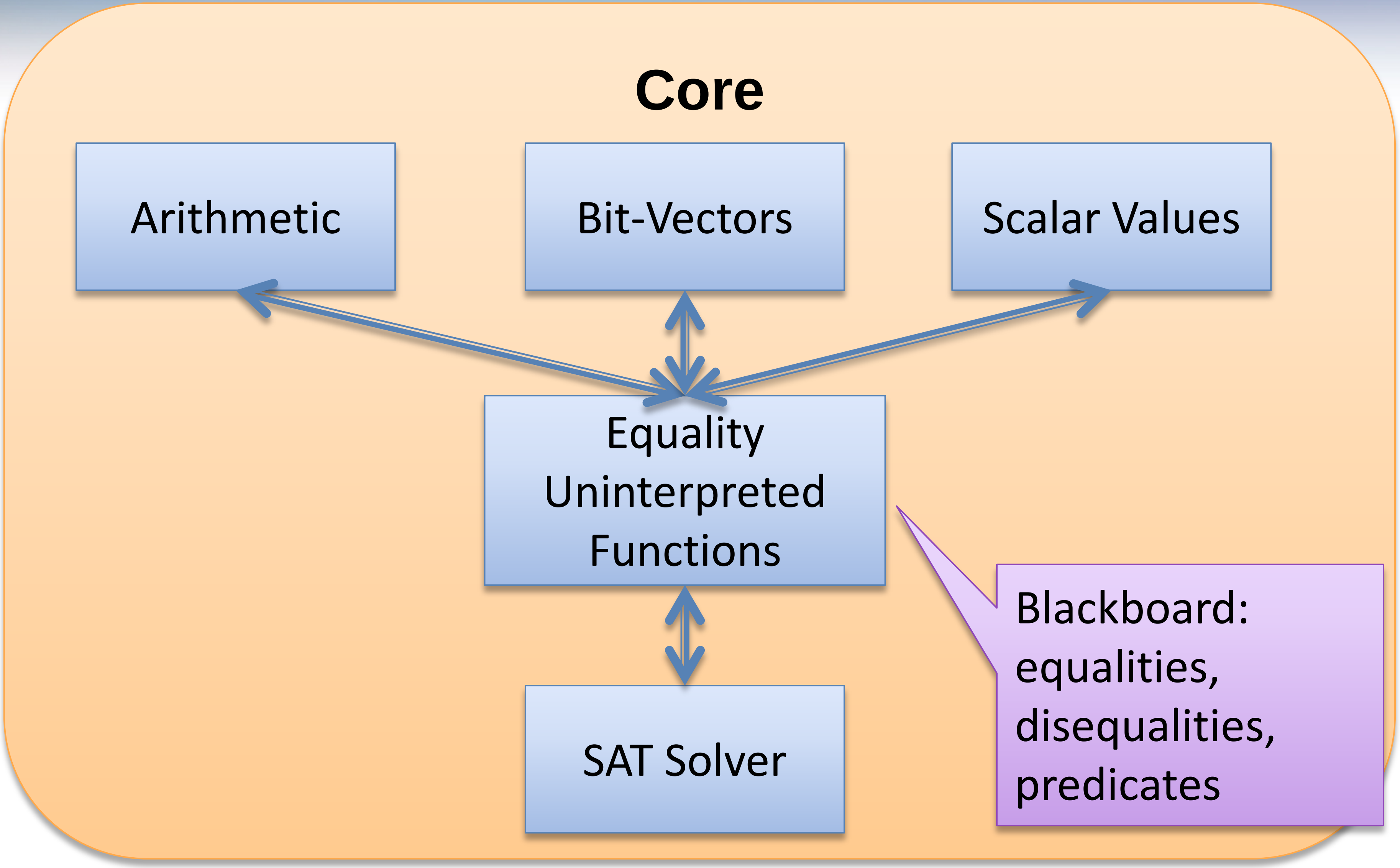
An Architecture: the core



An Architecture: the core



An Architecture: the core



Deciding Equality + (uninterpreted) Functions

Problem: our procedure for Equality + UF does not support:

- Incrementality
- Efficient Backtracking
- Theory Propagation
- Lemma Learning

Deciding Equality + (uninterpreted) Functions

Incrementality (main problem):

We were processing the disequalities after we processed **all** equalities.

$p_1 \equiv a = b$, $p_2 \equiv b = c$,
$p_3 \equiv d = e$, $p_4 \equiv a = c$

$p_1, \neg p_4, p_2 \mid p_1, p_3 \vee \neg p_4, p_2 \vee p_4$

$$\Downarrow$$

$a = b, a \neq c, b = c,$

Microsoft Research

Deciding Equality + (uninterpreted) Functions

Incrementality (main problem):

We were processing the disequalities after we processed **all** equalities.

$p_1 \equiv a = b,\ p_2 \equiv b = c,$
$p_3 \equiv d = e,\ p_4 \equiv a = c$

$p_1, \neg p_4, p_2 \ \mid\ p_1, p_3 \vee \neg p_4, p_2 \vee p_4$

$a = b,\ a \neq c,\ b = c,$

Microsoft Research

Deciding Equality + (uninterpreted) Functions

Incrementality

Store the disequalities of a constant.

Very similar to the structure occurrences.

$a = b$, $a \neq c$

$$\begin{tikzcd}
& b & c \\
a \arrow[ur] & &
\end{tikzcd}$$

diseqs[b] = { $a \neq c$ }

diseqs[c] = { $a \neq c$ }

Microsoft Research

Deciding Equality + (uninterpreted) Functions

Incrementality

Store the disequalities of a constant.

Very similar to the structure occurrences.

$a = b, a \neq c$

$$\begin{tikzcd} & b & c \\ a \arrow[ur] & & \end{tikzcd}$$

diseqs[b] = { a ≠ c }

diseqs[c] = { a ≠ c }

When we merge two equivalence classes, we must merge the sets diseqs. (circular lists again!)

Microsoft Research

Deciding Equality + (uninterpreted) Functions

Incrementality

Store the disequalities of a constant.

Very similar to the structure occurrences.

$a = b$, $a \neq c$

$$\begin{tikzcd}
 & b & c \\
a \arrow[ur] & &
\end{tikzcd}$$

diseqs(b) = { a ≠ c }

diseqs(c) = { a ≠ c }

When we merge two equivalence classes, we must merge the sets diseqs. (circular lists again!)

Before merging two equivalence classes, traverse one (the smallest) set of diseqs. (track the size of diseqs!)

Deciding Equality + (uninterpreted) Functions

Backtracking

- Option 1: functional data-structures (too slow).
- Option 2: trail stack (aka undo stack, fine grain backtracking)
  - Associate an undo operation to each update operation.
  - “Log” all update operations in a stack.
  - During backtracking execute the associated undo operations.

Deciding Equality + (uninterpreted) Functions

Backtracking

- We can do better: coarse grain backtracking.
  - Minimize the size of the undo stack.
  - Do not track each small update, but a big operation (merge).

Deciding Equality + (uninterpreted) Functions

Backtracking

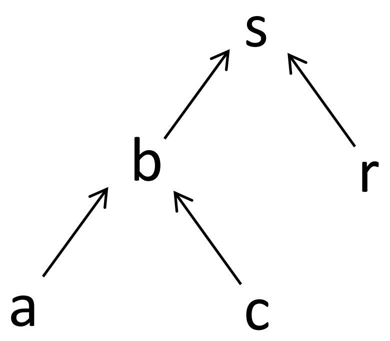
We can do better: coarse grain backtracking.

Minimize the size of the undo stack.

Do not track each small update, but a big operation (merge).

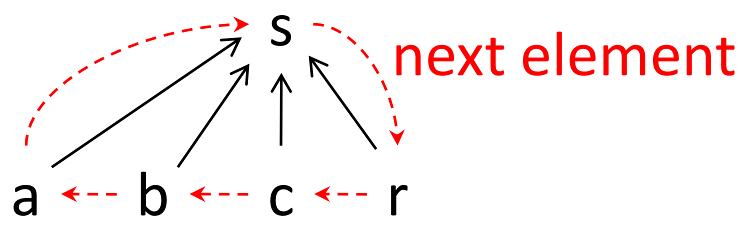
Let us change the union-find data-structure a little bit.

Before:



Fields: find, size

After:



Fields: root, next, size

Microsoft Research

Deciding Equality + (uninterpreted) Functions

Backtracking

We can do b

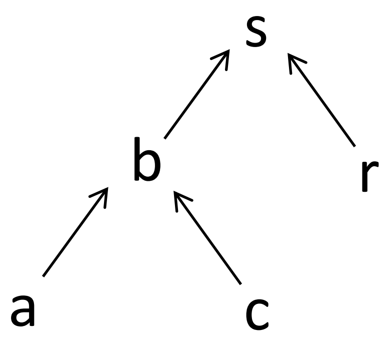
Minimiz

Do not t

New design possibility:
We do not need to merge occurrences and diseqs.
We can access all occurrences and diseqs by traversing the next fields.

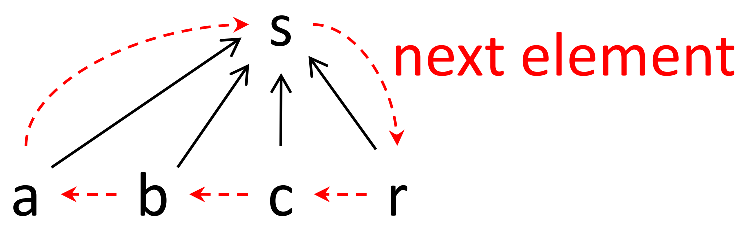
Let us change the union-find d ructure a little bit.

Before:



Fields: find, size

After:

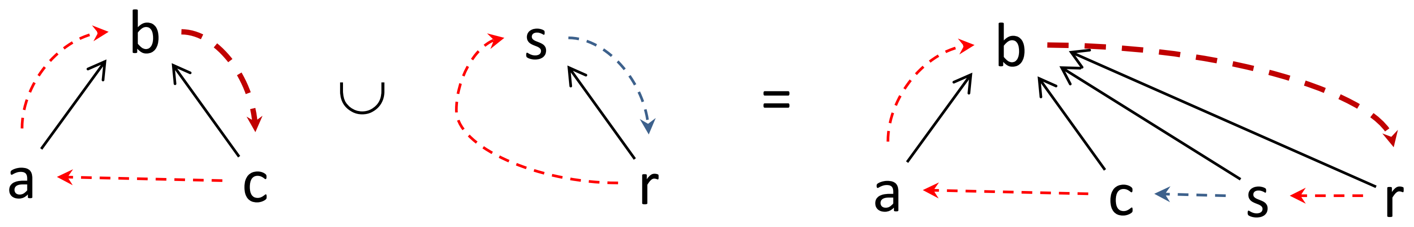


Fields: root, next, size

Microsoft Research

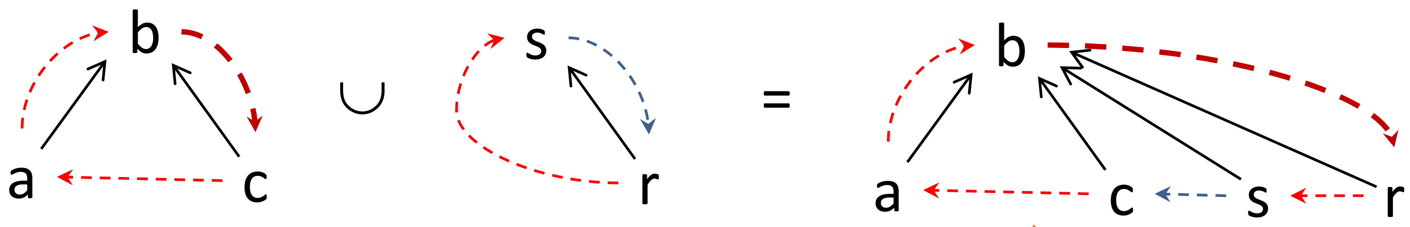
Deciding Equality + (uninterpreted) Functions

New union-find:



Deciding Equality + (uninterpreted) Functions

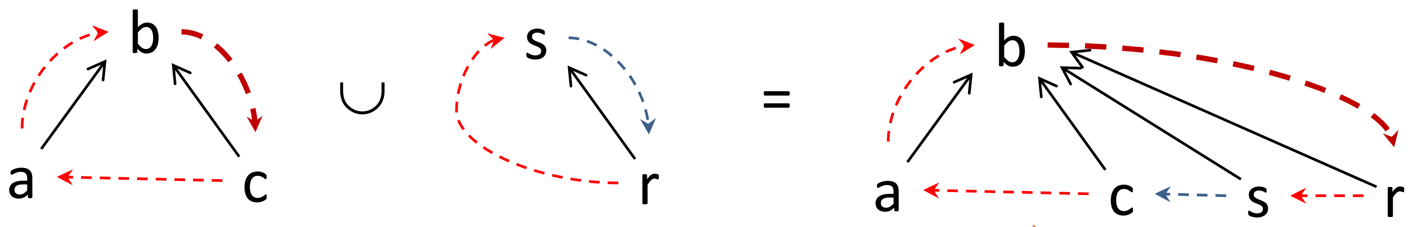
New union-find:



What was updated?
root[s], root[r],
next[b], next[s],
size[b]

Deciding Equality + (uninterpreted) Functions

New union-find:



We only need to store s in the undo stack!

What was updated? root[c], root[r], next[b], next[s], size[b]

Deciding Equality + (uninterpreted) Functions

What about the congruence table?

hash table used to implement the congruence rule.

Let us use an additional field cg.

It is only relevant for subterms: $v_3 \equiv f(a, v_1)$

Invariant: a constant (e.g., v_3) is in the table iff $cg[v_3] = v_3$

Otherwise, $cg[v_3]$ contains the subterm congruent to v_3

Example:

$v_3 \equiv f(a, v_1)$, $v_4 \equiv f(b, v_2)$

Assume v_3 and v_4 are congruent (i.e., a = b and v1 = v2)

Moreover, v_3 is in the congruence table.

Then: $cg[v_4] = v_3$ and $cg[v_3] = v_3$

Deciding Equality + (uninterpreted) Functions

```
procedure Merge(a, b)
    $a_r$ := root[a]; $b_r$ := root[b]
    if $a_r = b_r$ then return
    if not CheckDiseqs($a_r$, $b_r$) then return
    if size[a] < size[b] then swap  a, b; swap  $a_r$, $b_r$
    AddToTrailStack(MERGE, $b_r$)
    RemoveParentsFromHashTable($b_r$)
    c := $b_r$
    do
        root[c] := $a_r$
        c := next[c]
    while $c \neq b_r$
    ReinsertParentsToHashTable($b_r$)
    swap next[$a_r$], next[$b_r$]
    size[$a_r$] := size[$a_r$] + size[$b_r$]
```

Microsoft Research

Deciding Equality + (uninterpreted) Functions

```
procedure UndoMerge($b_r$)
    $a_r$ := root[$b_r$]
    size[$a_r$] := size[$a_r$] – size[$b_r$]
    swap next[$a_r$], next[$b_r$]
    RemoveParentsFromHashTable($b_r$)
    c := $b_r$
    do
        root[c] := $b_r$
        c := next[c]
    while c ≠ $b_r$
    for each parent p of $b_r$
        if p = cg[p] or not congruent(p, cg[p])
            add p to hash table
            cg[p] := p
```

Microsoft Research

Deciding Equality + (uninterpreted) Functions

```
procedure UndoMerge($b_r$)
    $a_r$ := root[$b_r$]
    size[$a_r$] := size[$a_r$] – size[$b_r$]
    swap next[$a_r$], next[$b_r$]
    RemoveParentsFromHashTa
    while c
    for each p    ent p of $b_r$
        if p = cg[p] or not congruent(p, cg[p])
            add p to hash table
            cg[p] := p
```

p was in the hash table before and after the merge

p was in the hash table before but not after the merge.

Microsoft Research

Deciding Equality + (uninterpreted) Functions

Propagating equalities (and disequalities)

Store the atom occurrences of a constant.

$p_1 \equiv a = b$, $p_2 \equiv b = c$,
$p_3 \equiv d = e$, $p_4 \equiv a = c$

atom_occs[a] = $\{ p_1, p_4 \}$
atom_occs[b] = $\{ p_1, p_2 \}$
atom_occs[c] = $\{ p_2, p_4 \}$
atom_occs[d] = $\{ p_3 \}$
atom_occs[e] = $\{ p_4 \}$

When merging or adding new disequalities traverse these sets.

Deciding Equality + (uninterpreted) Functions

Propagating disequalities (hard case)

$v_1 \equiv f(a, b)$, $v_2 \equiv f(c, d)$

Assume we know that

$v_1 \neq v_2$

$a = c$

Then, $b \neq d$

More about that later.

Deciding Equality + (uninterpreted) Functions

Efficient Lemma Generation (computing a small S')

In EUF (equality + UF) a minimal unsatisfiable set is composed on:

- n equalities
- 1 disequality

It is easy to find the disequality $a \neq b$.

So, our problem consists in finding the minimal set of equalities that implies $a = b$.

Deciding Equality + (uninterpreted) Functions

Efficient Lemma Generation (computing a small S')

First idea:

If $a = b$ is implied by a set of equalities, then a and b are in the same equivalence class.

Store all equalities used to "create" the equivalence class.

$p_1 \equiv (a = c)$, $p_2 \equiv (b = c)$,
$p_3 \equiv (s = r)$, $p_4 \equiv (c = r)$
$p_1, p_2, p_3, p_4, \ldots \mid \ldots$

$$\begin{tikzcd}
 & & s \arrow[drr, dashed, bend left] & \\
a \arrow[urr] \arrow[urr, dashed, bend left] & b \arrow[l, dashed] \arrow[ur] & c \arrow[l, dashed] \arrow[u] & r \arrow[l, dashed] \arrow[ul]
\end{tikzcd}$$

Too imprecise for justifying $a = b$.
We need only p_1, p_2.

The equivalence class was "created" using p_1, p_2, p_3, p_4

Deciding Equality + (uninterpreted) Functions

Efficient Lemma Generation (computing a small S')

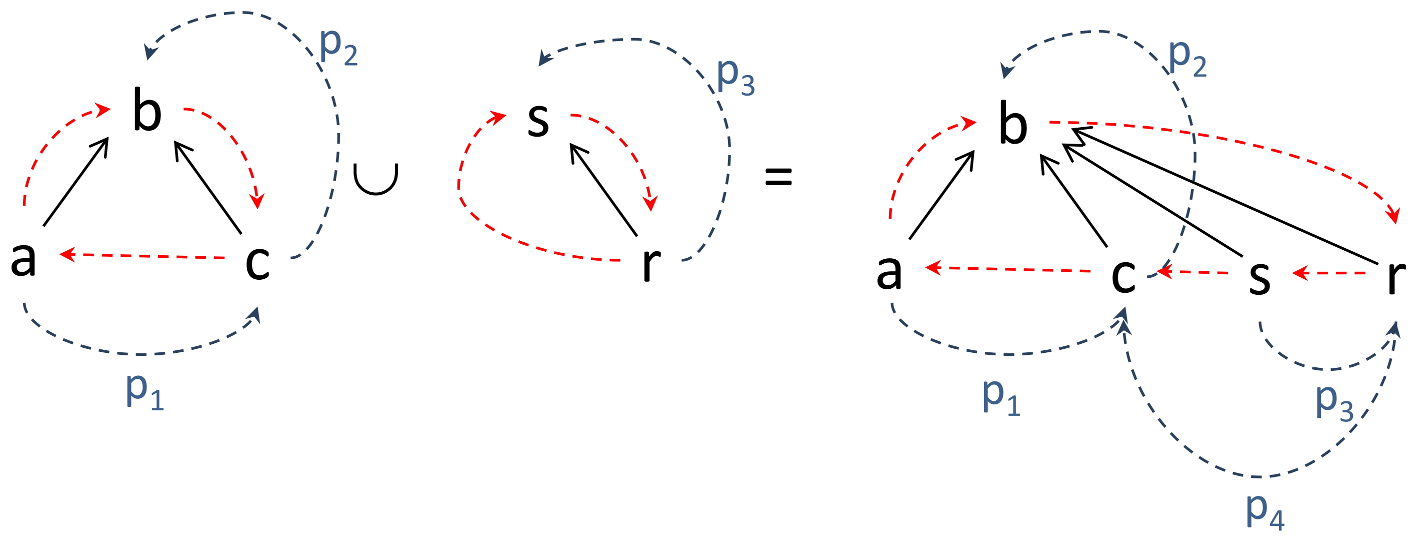
Second idea: Store a "proof tree".

Each constant c has a non-redundant "proof" for c = root[c].

The proof is a path from c to root[c]

$p_1 \equiv (a = c)$, $p_2 \equiv (b = c)$,

$p_3 \equiv (s = r)$, $p_4 \equiv (c = r)$

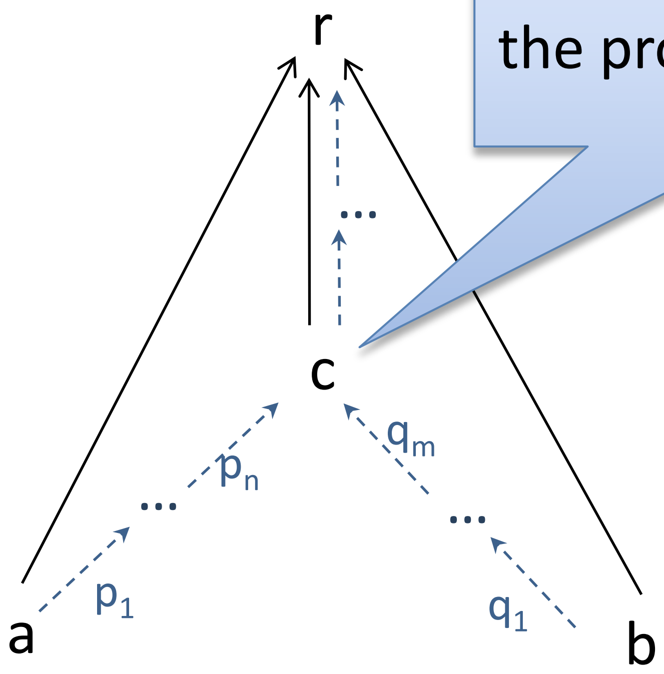


Microsoft Research

Deciding Equality + (uninterpreted) Functions

```
procedure Merge(a, b, $p_i$)
    $a_r$ := root[a]; $b_r$ := root[b]
    if $a_r = b_r$ then return
    if not CheckDiseqs($a_r$, $b_r$) then return
    if size[a] < size[b] then swap  a, b; swap  $a_r$, $b_r$
    InvertPathFrom(b, $b_r$); AddProofEdge(b, a, $p_i$)
    AddToTrailStack(MERGE, $b_r$ , b)
    …
```

Deciding Equality + (uninterpreted) Functions



Common ancestor in the proof tree.

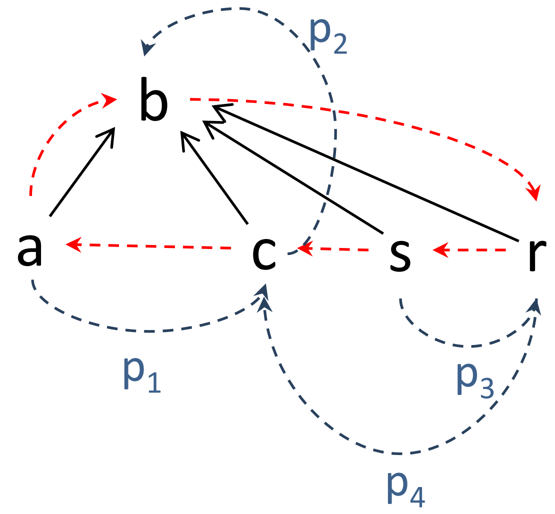
Non redundant proof for **a = b**

$p_1, \ldots, p_n, q_1, \ldots, q_m$

Microsoft Research

Deciding Equality + (uninterpreted) Functions

Extract a non redundant proof for a = r, a = b and a = s.



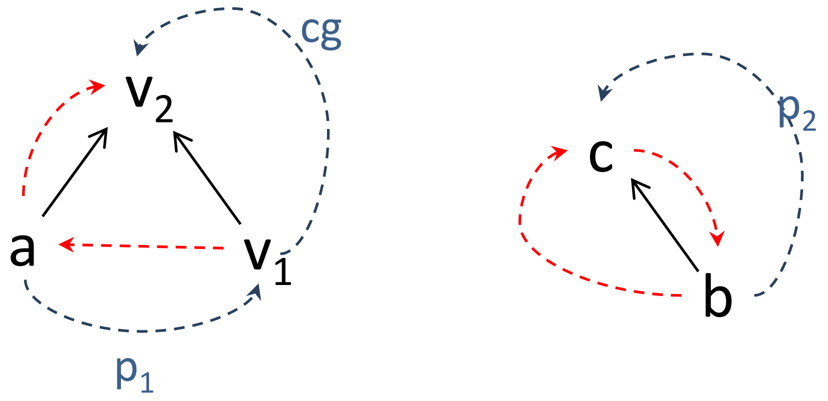
$$\begin{tikzcd}
 & b \arrow[rrr, dashed, bend left, red] & & & \\
a \arrow[ur] \arrow[ur, dashed, bend left, red] \arrow[rr, dashed, bend right, "p_1"'] & & c \arrow[ul] \arrow[ll, dashed, red] \arrow[ul, dashed, bend right, "p_2"'] \arrow[rr, dashed, bend right=60, "p_4"'] & s \arrow[ull] \arrow[l, dashed, red] \arrow[r, dashed, bend right, "p_3"'] & r \arrow[ulll] \arrow[l, dashed, red]
\end{tikzcd}$$

Deciding Equality + (uninterpreted) Functions

What about congruence?

New form of justification for an edge in the “proof tree”.

$v_1 \equiv f(b),\ v_2 \equiv f(c)$

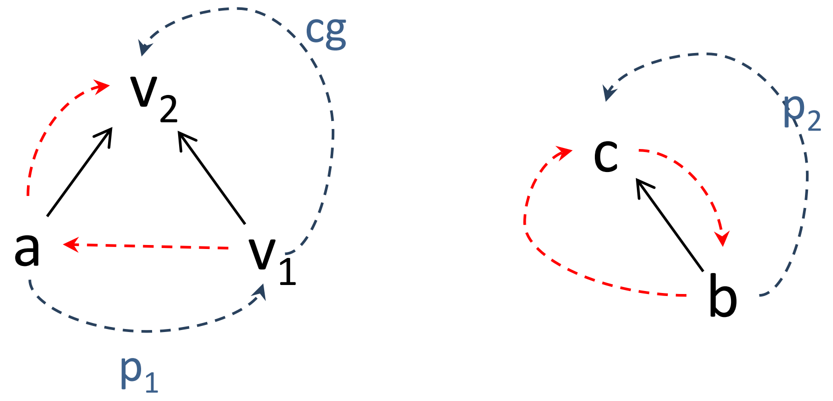


Deciding Equality + (uninterpreted) Functions

What about congruence?

New form of justification for an edge in the "proof tree".

$v_1 \equiv f(b)$, $v_2 \equiv f(c)$



When computing the "proof" for $a = v_2$

Recursive call for computing the proof for $v_1 = v_2$

Result: $\{p_1, p_2\}$

Deciding Equality + (uninterpreted) Functions

The new algorithm may compute redundant proofs for EUF.

Using notation $a \overset{p}{=} b$ for $p \equiv a = b$, and p assigned by SAT solver

$f_1(a_1) \overset{p_1}{=} a_1 \overset{q_1}{=} a_2 \overset{s_1}{=} f_1(a_5)$
$f_2(a_1) \overset{p_2}{=} a_2 \overset{q_2}{=} a_3 \overset{s_2}{=} f_2(a_5)$
$f_3(a_1) \overset{p_3}{=} a_3 \overset{q_3}{=} a_4 \overset{s_3}{=} f_3(a_5)$
$f_4(a_1) \overset{p_4}{=} a_4 \overset{q_4}{=} a_5 \overset{s_4}{=} f_4(a_5)$

Deciding Equality + (uninterpreted) Functions

The new algorithm may compute redundant proofs for EUF.

Using notation $a \overset{p}{=} b$ for $p \equiv a = b$, and p assigned by SAT solver

$f_1(a_1) \overset{p_1}{=} a_1 \overset{q_1}{=} a_2 \overset{s_1}{=} f_1(a_5)$ Two non redundant proofs $f_2(a_1) = f_2(a_5)$:

$f_2(a_1) \overset{p_2}{=} a_2 \overset{q_2}{=} a_3 \overset{s_2}{=} f_2(a_5)$ $\{p_2, q_2, s_2\}$ using transitivity

$f_3(a_1) \overset{p_3}{=} a_3 \overset{q_3}{=} a_4 \overset{s_3}{=} f_3(a_5)$ $\{q_1, q_2, q_3, q_4\}$ using congruence $a_1 = a_5$

$f_4(a_1) \overset{p_4}{=} a_4 \overset{q_4}{=} a_5 \overset{s_4}{=} f_4(a_5)$ Similar for f_1, f_3, f_4.

Deciding Equality + (uninterpreted) Functions

The new algorithm may compute redundant proofs for EUF.

Using notation $a \overset{p}{=} b$ for $p \equiv a = b$, and p assigned by SAT solver

$f_1(a_1) \overset{p_1}{=} a_1 \overset{q_1}{=} a_2 \overset{s_1}{=} f_1(a_5)$

$f_2(a_1) \overset{p_2}{=} a_2 \overset{q_2}{=} a_3 \overset{s_2}{=} f_2(a_5)$

$f_3(a_1) \overset{p_3}{=} a_3 \overset{q_3}{=} a_4 \overset{s_3}{=} f_3(a_5)$

$f_4(a_1) \overset{p_4}{=} a_4 \overset{q_4}{=} a_5 \overset{s_4}{=} f_4(a_5)$

Two non redundant proofs $f_2(a_1) = f_2(a_5)$:

$\{p_2, q_2, s_2\}$ using transitivity

$\{q_1, q_2, q_3, q_4\}$ using congruence $a_1 = a_5$

Similar for f_1, f_3, f_4.

So there are 16 proofs for

$g(f_1(a_1), f_2(a_1), f_3(a_1), f_4(a_1)) = g(f_1(a_5), f_2(a_5), f_3(a_5), f_4(a_5))$

The only non redundant is $\{q_1, q_2, q_3, q_4\}$

Microsoft Research

Deciding Equality + (uninterpreted) Functions

Some benchmarks are very hard for our procedure.

$p_1 \vee a_1 = c_0, \neg p_1 \vee a_1 = c_1, \quad p_1 \vee b_1 = c_0, \neg p_1 \vee b_1 = c_1,$

$p_2 \vee a_2 = c_0, \neg p_2 \vee a_2 = c_1, \quad p_2 \vee b_2 = c_0, \neg p_2 \vee b_2 = c_1,$

$\ldots,$

$p_n \vee a_n = c_0, \neg p_n \vee a_n = c_1, \quad p_n \vee b_n = c_0, \neg p_n \vee b_n = c_1,$

$f(a_n, \ldots, f(a_2, a_1)\ldots) \neq f(b_n, \ldots, f(b_2, b_1)\ldots)$

Deciding Equality + (uninterpreted) Functions

Some benchmarks are very hard for our procedure.

$p_1 \vee \mathbf{a_1 = c_0}$, $\neg p_1 \vee a_1 = c_1$, $p_1 \vee \mathbf{b_1 = c_0}$, $\neg p_1 \vee b_1 = c_1$,
$p_2 \vee a_2 = c_0$, $\neg p_2 \vee \mathbf{a_2 = c_1}$, $p_2 \vee b_2 = c_0$, $\neg p_2 \vee \mathbf{b_2 = c_1}$,
…,
$p_n \vee \mathbf{a_n = c_0}$, $\neg p_n \vee a_n = c_1$, $p_n \vee \mathbf{b_n = c_0}$, $\neg p_n \vee b_n = c_1$,
$\mathbf{f(a_n, \ldots, f(a_2, a_1)\ldots) \neq f(b_n, \ldots, f(b_2, b_1)\ldots)}$

Lemmas learned during the search are not useful.

They only use atoms that are already in the problem!

Deciding Equality + (uninterpreted) Functions

Some benchmarks are very hard for our procedure.

$p_1 \vee a_1 = c_0$, $\neg p_1 \vee a_1 = c_1$, $p_1 \vee b_1 = c_0$, $\neg p_1 \vee b_1 = c_1$,
$p_2 \vee a_2 = c_0$, $\neg p_2 \vee a_2 = c_1$, $p_2 \vee b_2 = c_0$, $\neg p_2 \vee b_2 = c_1$,
…,
$p_n \vee a_n = c_0$, $\neg p_n \vee a_n = c_1$, $p_n \vee b_n = c_0$, $\neg p_n \vee b_n = c_1$,
$f(a_n, \ldots, f(a_2, a_1)\ldots) \neq f(b_n, \ldots, f(b_2, b_1)\ldots)$

Lemmas learned during the search are not useful.

They only use atoms that are already in the problem!

Solution: congruence rule suggests which new atoms must be created.

Deciding Equality + (uninterpreted) Functions

Some benchmarks are very hard for our procedure.

$p_1 \vee a_1 = c_0$, $\neg p_1 \vee a_1 = c_1$, $p_1 \vee b_1 = c_0$, $\neg p_1 \vee b_1 = c_1$,

$p_2 \vee a_2 = c_0$, $\neg p_2 \vee a_2 = c_1$, $p_2 \vee b_2 = c_0$, $\neg p_2 \vee b_2 = c_1$,

…,

$p_n \vee a_n = c_0$, $\neg p_n \vee a_n = c_1$, $p_n \vee b_n = c_0$, $\neg p_n \vee b_n = c_1$,

$f(a_n, \ldots, f(a_2, a_1)\ldots) \neq f(b_n, \ldots, f(b_2, b_1)\ldots)$

Solution: congruence rule suggests which new atoms must be created.

Whenever, the congruence rules

$a_i = b_i$, $a_j = b_j$ implies $f(a_i, a_j) = f(b_i, b_j)$

is used to (immediately) deduce a conflict. Add the clause:

$a_i \neq b_i \vee a_j \neq b_j \vee f(a_i, a_j) = f(b_i, b_j)$

Deciding Equality + (uninterpreted) Functions

Solution: congruence rule suggests which new atoms must be created.

Whenever, the congruence rules

$a_i = b_i$, $a_j = b_j$ implies $f(a_i, a_j) = f(b_i, b_j)$

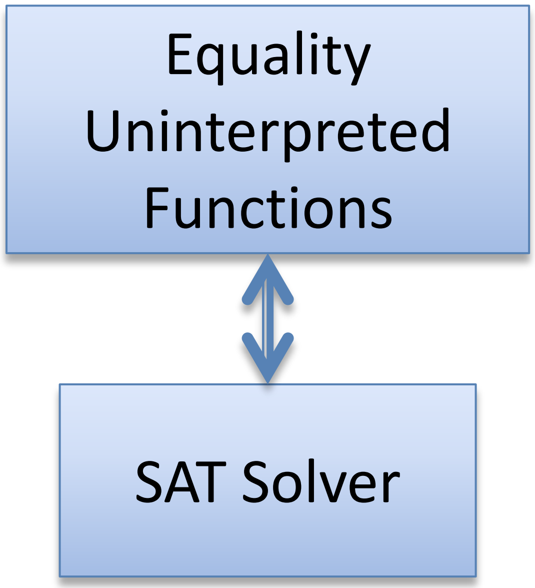
is used to (immediately) deduce a conflict. Add the clause:

$a_i \neq b_i \vee a_j \neq b_j \vee f(a_i, a_j) = f(b_i, b_j)$

"Dynamic Ackermannization"

It allows the solver to perform the missing disequality propagation.

Summary



We can solve the QF_UF SMT-Lib benchmarks!

Linear Arithmetic

- Many approaches
  - Graph-based for difference logic: $a - b \leq 3$
  - Fourier-Motzkin elimination:

$$t_1 \leq ax, \;\; bx \leq t_2 \;\; \Rightarrow \;\; bt_1 \leq at_2$$

  - Standard Simplex
  - General Form Simplex

Difference Logic: $a - b \leq 5$

Very useful in practice!

Most arithmetical constraints in software verification/analysis are in this fragment.

$$x := x + 1$$

$$\Downarrow$$

$$x_1 = x_0 + 1$$

$$\Downarrow$$

$$x_1 - x_0 \leq 1,\ x_0 - x_1 \leq -1$$

Microsoft Research

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

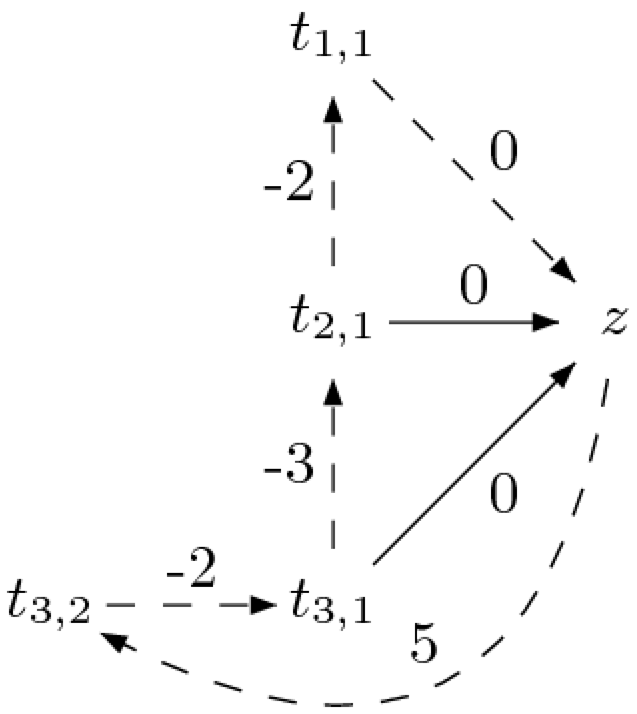
$$\begin{array}{l}
(t_{1,1} \geq 0) \wedge (t_{1,2} \geq t_{1,1} + 2) \wedge (t_{1,2} + 1 \leq 8) \wedge \\
(t_{2,1} \geq 0) \wedge (t_{2,2} \geq t_{2,1} + 3) \wedge (t_{2,2} + 1 \leq 8) \wedge \\
(t_{3,1} \geq 0) \wedge (t_{3,2} \geq t_{3,1} + 2) \wedge (t_{3,2} + 3 \leq 8) \wedge \\
((t_{1,1} \geq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

Difference Logic

Chasing negative cycles!

Algorithms based on Bellman-Ford (O(mn)).

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



$$\begin{tikzcd}
 & t_{1,1} \arrow[dr, dashed, "0"] & \\
 & t_{2,1} \arrow[u, dashed, "-2"] \arrow[r, "0"] & z \arrow[dll, dashed, bend left, "5"] \\
t_{3,2} \arrow[r, dashed, "-2"] & t_{3,1} \arrow[u, dashed, "-3"] \arrow[ur, "0"'] &
\end{tikzcd}$$

Standard Simplex

Many solvers (e.g., ICS, Simplify) are based on the Standard Simplex.

$$
\begin{aligned}
a - d + 2e &= 3 \\
b - d &= 1 \\
c + d - e &= -1 \\
a, b, c, d, e &\geq 0
\end{aligned}
$$

Microsoft Research

Standard Simplex

Many solvers (e.g., ICS, Simplify) are based on the Standard Simplex.

a - d + 2e = 3

b - d = 1

c + d - e = -1

a, b, c, d, e ≥ 0

$$\begin{pmatrix} 1 & 0 & 0 & -1 & 2 \\ 0 & 1 & 0 & -1 & 0 \\ 0 & 0 & 1 & 1 & -1 \end{pmatrix} \begin{pmatrix} a \\ b \\ c \\ d \\ e \end{pmatrix} = \begin{pmatrix} 3 \\ 1 \\ -1 \end{pmatrix}$$

$Ax = b$ and $x \geq 0.$

Standard Simplex

Many solvers (e.g., ICS, Simplify) are based on the Standard Simplex.

$$\begin{aligned} a - d + 2e &= 3 \\ b - d &= 1 \\ c + d - e &= -1 \\ a, b, c, d, e &\geq 0 \end{aligned}$$

We say a,b,c are the basic (or dependent) variables

$$\begin{pmatrix} 1 & 0 & 0 & -1 & 2 \\ 0 & 1 & 0 & -1 & 0 \\ 0 & 0 & 1 & 1 & -1 \end{pmatrix} \begin{pmatrix} a \\ b \\ c \\ d \\ e \end{pmatrix} = \begin{pmatrix} 3 \\ 1 \\ -1 \end{pmatrix}$$

$Ax = b$ and $x \geq 0$.

Microsoft Research

Standard Simplex

Many solvers (e.g., ICS, Simplify) are based on the Standard Simplex.

$$
\begin{aligned}
a - d + 2e &= 3\\
b - d &= 1\\
c + d - e &= -1\\
a, b, c, d, e &\geq 0
\end{aligned}
$$

We say a,b,c are the basic (or dependent) variables

We say d,e are the non-basic (or non-dependent) variables.

$$
\begin{pmatrix} 1 & 0 & 0 & -1 & 2\\ 0 & 1 & 0 & -1 & 0\\ 0 & 0 & 1 & 1 & -1 \end{pmatrix}
\begin{pmatrix} a\\ b\\ c\\ d\\ e \end{pmatrix}
=
\begin{pmatrix} 3\\ 1\\ -1 \end{pmatrix}
$$

$Ax = b$ and $x \geq 0.$

Microsoft Research

Standard Simplex

- Incrementality: add/remove equations
- Slow backtracking
- No theory propagation

Fast Linear Arithmetic

- Simplex General Form
- Algorithm based on the dual simplex
- Non redundant proofs
- Efficient backtracking
- Efficient theory propagation
- Support for string inequalities: $t > 0$
- Preprocessing step
- Integer problems:

    Gomory cuts, Branch & Bound, GCD test

General Form

General Form: $Ax = 0$ and $l_j \leq x_j \leq u_j$

Example:

$$x \geq 0, (x + y \leq 2 \vee x + 2y \geq 6), (x + y = 2 \vee x + 2y > 4)$$

$$\leadsto$$

$$s_1 \equiv x + y, s_2 \equiv x + 2y,$$

$$x \geq 0, (s_1 \leq 2 \vee s_2 \geq 6), (s_1 = 2 \vee s_2 > 4)$$

Only bounds (e.g., $s_1 \leq 2$) are asserted during the search.

Unconstrained variables can be eliminated before the beginning of the search.

From Definitions to a Tableau

$$s_1 \equiv x + y, \quad s_2 \equiv x + 2y$$

From Definitions to a Tableau

$$s_1 \equiv x + y, \quad s_2 \equiv x + 2y$$

$$\Downarrow$$

$$s_1 = x + y,$$
$$s_2 = x + 2y$$

From Definitions to a Tableau

$$s_1 \equiv x + y, \quad s_2 \equiv x + 2y$$

$$\Downarrow$$

$$s_1 = x + y,$$
$$s_2 = x + 2y$$

$$\Downarrow$$

$$s_1 - x - y = 0$$
$$s_2 - x - 2y = 0$$

From Definitions to a Tableau

$s_1 \equiv x + y, \quad s_2 \equiv x + 2y$

$\Downarrow$

$s_1 = x + y,$

$s_2 = x + 2y$

$\Downarrow$

$s_1 - x - y \;\;= 0$ $\qquad$ s_1, s_2 are basic (dependent)

$s_2 - x - 2y = 0$ $\qquad$ x,y are non-basic

Microsoft Research

Pivoting

A way to swap a basic with a non-basic variable!

It is just equational reasoning.

Key invariant: a basic variable occurs in only one equation.

Example: swap s_1 and y

$$s_1 - x - y \quad = 0$$
$$s_2 - x - 2y = 0$$

Microsoft Research

Pivoting

A way to swap a basic with a non-basic variable!

It is just equational reasoning.

Key invariant: a basic variable occurs in only one equation.

Example: swap s_1 and y

$$s_1 - x - y \quad = 0$$
$$s_2 - x - 2y = 0$$

$$\Downarrow$$

$$-s_1 + x + y \quad = 0$$
$$s_2 - x - 2y = 0$$

Microsoft Research

Pivoting

A way to swap a basic with a non-basic variable!

It is just equational reasoning.

Key invariant: a basic variable occurs in only one equation.

Example: swap s_1 and y

$$s_1 - x - y = 0$$
$$s_2 - x - 2y = 0$$

$$\Downarrow$$

$$-s_1 + x + y = 0$$
$$s_2 - x - 2y = 0$$

$$\Downarrow$$

$$-s_1 + x + y = 0$$
$$s_2 - 2s_1 + x = 0$$

Pivoting

A way to swap a basic with a non-basic variable!

It is just equational reasoning.

Key invariant: a basic variable occurs in only one equation.

Example: swap s_1 and y

$$s_1 - x - y = 0$$
$$s_2 - x - 2y = 0$$

It is just substituting equals by equals.

$$\Downarrow$$

$$-s_1 + x + y = 0$$
$$s_2 - x - 2y = 0$$

$$\Downarrow$$

$$-s_1 + x + y = 0$$
$$s_2 - 2s_1 + x = 0$$

Microsoft Research

Pivoting

Definition:
An assignment (model) is a mapping from variables to values

A way to swap a basic with a non-basic variable!

It is just equational reasoning.

Key invariant: a basic variable occurs in only one equation.

Example: swap s_1 and y

$$s_1 - x - y = 0$$
$$s_2 - x - 2y = 0$$
$$\Downarrow$$
$$-s_1 + x + y = 0$$
$$s_2 - x - 2y = 0$$
$$\Downarrow$$
$$-s_1 + x + y = 0$$
$$s_2 - 2s_1 + x = 0$$

It is just substituting equals by equals.

Key Property:
If an assignment satisfies the equations before a pivoting step, then it will also satisfy them after!

Pivoting

Definition:
An assignment (model) is a mapping from variables to values

A way to swap a basic with a non-basic variable!

It is just equational reasoning.

Key invariant: a basic variable occurs in only one equation.

Example: swap s_2 and y

$$s_1 - x - y = 0$$
$$s_2 - x - 2y = 0$$

$$\Downarrow$$

$$-s_1 + x + y = 0$$
$$s_2 - x - 2y = 0$$

$$\Downarrow$$

$$-s_1 + x + y = 0$$
$$s_2 - 2s_1 + x = 0$$

It is just substituting equals by equals.

Example:
$M(x) = 1$
$M(y) = 1$
$M(s_1) = 2$
$M(s_2) = 3$

Key Property:
If an assignment satisfies the equations before a pivoting step, then it will also satisfy them after!

Equations + Bounds + Assignment

An assignment (model) is a mapping from variables to values.

We maintain an assignment that satisfies all equations and bounds.

The assignment of non dependent variables implies the assignment of dependent variables.

Equations + Bounds can be used to derive new bounds.

Example: $x = y - z,\ y \leq 2,\ z \geq 3 \rightsquigarrow x \leq -1.$

The new bound may be inconsistent with the already known bounds.

Example: $x \leq -1,\ x \geq 0.$

“Repairing Models”

If the assignment of a non-basic variable does not satisfy a bound, then fix it and propagate the change to all dependent variables.

$a = c - d$	$a = c - d$
$b = c + d$	$b = c + d$
$M(a) = 0$	$M(a) = 1$
$M(b) = 0$	$M(b) = 1$
$M(c) = 0$	$M(c) = 1$
$M(d) = 0$	$M(d) = 0$
$1 \leq c$	$1 \leq c$

“Repairing Models”

If the assignment of a non-basic variable does not satisfy a bound, then fix it and propagate the change to all dependent variables. Of course, we may introduce new “problems”.

$a = c - d$
$b = c + d$
$M(a) = 0$
$M(b) = 0$
$M(c) = 0$
$M(d) = 0$
$1 \leq c$
$a \leq 0$

$\Rightarrow$

$a = c - d$
$b = c + d$
$M(a) = 1$
$M(b) = 1$
$M(c) = 1$
$M(d) = 0$
$1 \leq c$
$\mathbf{a \leq 0}$

Microsoft Research

"Repairing Models"

If the assignment of a basic variable does not satisfy a bound, then pivot it, fix it, and propagate the change to its new dependent variables.

$$
\begin{array}{lclcl}
a = c - d & & c = a + d & & c = a + d \\
b = c + d & & b = a + 2d & & b = a + 2d \\
M(a) = 0 & & M(a) = 0 & & M(a) = 1 \\
M(b) = 0 & \Rightarrow & M(b) = 0 & \Rightarrow & M(b) = 1 \\
M(c) = 0 & & M(c) = 0 & & M(c) = 1 \\
M(d) = 0 & & M(d) = 0 & & M(d) = 0 \\
1 \leq a & & 1 \leq a & & 1 \leq a
\end{array}
$$

"Repairing Models"

Sometimes, a model cannot be repaired. It is pointless to pivot.

$a = b - c$

$a \leq 0,\ 1 \leq b,\ c \leq 0$

$M(a) = 1$

$M(b) = 1$

$M(c) = 0$

The value of M(a) is too big. We can reduce it by:

- reducing M(b)
  not possible b is at lower bound
- increasing M(c)
  not possible c is at upper bound

"Repairing Models"

Extracting proof from failed repair attempts is easy.

$s_1 \equiv a + d,\ s_2 \equiv c + d$

$a = s_1 - s_2 + c$

$a \leq 0,\ 1 \leq s_1,\ s_2 \leq 0,\ 0 \leq c$

$M(a) = 1$

$M(s_1) = 1$

$M(s_2) = 0$

$M(c) = 0$

"Repairing Models"

Extracting proof from failed repair attempts is easy.

$s_1 \equiv a + d,\ s_2 \equiv c + d$

$a = s_1 - s_2 + c$

$a \leq 0,\ 1 \leq s_1,\ s_2 \leq 0,\ 0 \leq c$

$M(a) = 1$

$M(s_1) = 1$

$M(s_2) = 0$

$M(c) = 0$

$\{ a \leq 0,\ 1 \leq s_1,\ s_2 \leq 0,\ 0 \leq c \}$ is inconsistent

“Repairing Models”

Extracting proof from failed repair attempts is easy.

$s_1 \equiv a + d,\ s_2 \equiv c + d$

$a = s_1 - s_2 + c$

$a \leq 0,\ 1 \leq s_1,\ s_2 \leq 0,\ 0 \leq c$

$M(a) = 1$

$M(s_1) = 1$

$M(s_2) = 0$

$M(c) = 0$

$\{ a \leq 0,\ 1 \leq s_1,\ s_2 \leq 0,\ 0 \leq c \}$ is inconsistent

$\{ a \leq 0,\ 1 \leq a + d,\ c + d \leq 0,\ 0 \leq c \}$ is inconsistent

Strict Inequalities

The method described only handles non-strict inequalities (e.g., $x \leq 2$).

For integer problems, strict inequalities can be converted into non-strict inequalities. $x < 1 \leadsto x \leq 0$.

For rational/real problems, strict inequalities can be converted into non-strict inequalities using a small δ. $x < 1 \leadsto x \leq 1 - \delta$.

We do not compute a δ, we treat it symbolically.

δ is an infinitesimal parameter: $(c, k) = c + k\delta$

Example

- Initial state

$$s \geq 1, x \geq 0$$

$$(y \leq 1 \vee v \geq 2), (v \leq -2 \vee v \geq 0), (v \leq -2 \vee u \leq -1)$$

Model	Equations	Bounds
$M(x) = 0$	$s = x + y$	
$M(y) = 0$	$u = x + 2y$	
$M(s) = 0$	$v = x - y$	
$M(u) = 0$		
$M(v) = 0$		

Example

- Asserting $s \geq 1$

$$s \geq 1, x \geq 0$$

$$(y \leq 1 \vee v \geq 2), (v \leq -2 \vee v \geq 0), (v \leq -2 \vee u \leq -1)$$

Model	Equations	Bounds
$M(x) = 0$	$s = x + y$	
$M(y) = 0$	$u = x + 2y$	
$M(s) = 0$	$v = x - y$	
$M(u) = 0$		
$M(v) = 0$		

Example

- Asserting $s \geq 1$ assignment does not satisfy new bound.

$$s \geq 1, x \geq 0$$

$$(y \leq 1 \vee v \geq 2), (v \leq -2 \vee v \geq 0), (v \leq -2 \vee u \leq -1)$$

Model	Equations	Bounds
$M(x) = 0$	$s = x + y$	$s \geq 1$
$M(y) = 0$	$u = x + 2y$	
$M(s) = 0$	$v = x - y$	
$M(u) = 0$		
$M(v) = 0$		

Example

- Asserting $s \geq 1$ pivot s and x (s is a dependent variable).

$$s \geq 1, x \geq 0$$

$$(y \leq 1 \vee v \geq 2), (v \leq -2 \vee v \geq 0), (v \leq -2 \vee u \leq -1)$$

Model	Equations	Bounds
$M(x) = 0$	$s = x + y$	$s \geq 1$
$M(y) = 0$	$u = x + 2y$	
$M(s) = 0$	$v = x - y$	
$M(u) = 0$		
$M(v) = 0$		

Example

- Asserting $s \geq 1$ pivot s and x (s is a dependent variable).

$$s \geq 1, x \geq 0$$

$$(y \leq 1 \vee v \geq 2), (v \leq -2 \vee v \geq 0), (v \leq -2 \vee u \leq -1)$$

Model	Equations	Bounds
$M(x) = 0$	$x = s - y$	$s \geq 1$
$M(y) = 0$	$u = x + 2y$	
$M(s) = 0$	$v = x - y$	
$M(u) = 0$		
$M(v) = 0$		

Example

- Asserting $s \geq 1$ pivot s and x (s is a dependent variable).

$$s \geq 1, x \geq 0$$

$$(y \leq 1 \vee v \geq 2), (v \leq -2 \vee v \geq 0), (v \leq -2 \vee u \leq -1)$$

Model	Equations	Bounds
$M(x) = 0$	$x = s - y$	$s \geq 1$
$M(y) = 0$	$u = s + y$	
$M(s) = 0$	$v = s - 2y$	
$M(u) = 0$		
$M(v) = 0$		

Example

- Asserting $s \geq 1$ update assignment.

$$s \geq 1, x \geq 0$$

$$(y \leq 1 \vee v \geq 2), (v \leq -2 \vee v \geq 0), (v \leq -2 \vee u \leq -1)$$

Model	Equations	Bounds
$M(x) = 0$	$x = s - y$	$s \geq 1$
$M(y) = 0$	$u = s + y$	
$M(s) = 1$	$v = s - 2y$	
$M(u) = 0$		
$M(v) = 0$		

Example

- Asserting $s \geq 1$ update dependent variables assignment.

$$s \geq 1, x \geq 0$$

$$(y \leq 1 \vee v \geq 2), (v \leq -2 \vee v \geq 0), (v \leq -2 \vee u \leq -1)$$

Model	Equations	Bounds
$M(x) = 1$	$x = s - y$	$s \geq 1$
$M(y) = 0$	$u = s + y$	
$M(s) = 1$	$v = s - 2y$	
$M(u) = 1$		
$M(v) = 1$		

Example

- Asserting $x \geq 0$

$$s \geq 1, x \geq 0$$

$$(y \leq 1 \vee v \geq 2), (v \leq -2 \vee v \geq 0), (v \leq -2 \vee u \leq -1)$$

Model	Equations	Bounds
$M(x) = 1$	$x = s - y$	$s \geq 1$
$M(y) = 0$	$u = s + y$	
$M(s) = 1$	$v = s - 2y$	
$M(u) = 1$		
$M(v) = 1$		

Example

- Asserting $x \geq 0$ assignment satisfies new bound.

$$s \geq 1, x \geq 0$$

$$(y \leq 1 \vee v \geq 2), (v \leq -2 \vee v \geq 0), (v \leq -2 \vee u \leq -1)$$

Model	Equations	Bounds
$M(x) = 1$	$x = s - y$	$s \geq 1$
$M(y) = 0$	$u = s + y$	$x \geq 0$
$M(s) = 1$	$v = s - 2y$	
$M(u) = 1$		
$M(v) = 1$		

Example

- Case split $\neg y \leq 1$

$$s \geq 1, x \geq 0$$

$$(y \leq 1 \vee v \geq 2), (v \leq -2 \vee v \geq 0), (v \leq -2 \vee u \leq -1)$$

Model	Equations	Bounds
$M(x) = 1$	$x = s - y$	$s \geq 1$
$M(y) = 0$	$u = s + y$	$x \geq 0$
$M(s) = 1$	$v = s - 2y$	
$M(u) = 1$		
$M(v) = 1$		

Example

- Case split $\neg y \leq 1$ assignment does not satisfies new bound.

$$s \geq 1, x \geq 0$$

$$(y \leq 1 \vee v \geq 2), (v \leq -2 \vee v \geq 0), (v \leq -2 \vee u \leq -1)$$

Model	Equations	Bounds
$M(x) = 1$	$x = s - y$	$s \geq 1$
$M(y) = 0$	$u = s + y$	$x \geq 0$
$M(s) = 1$	$v = s - 2y$	$y > 1$
$M(u) = 1$		
$M(v) = 1$		

Example

- Case split $\neg y \leq 1$ update assignment.

$$s \geq 1, x \geq 0$$

$$(y \leq 1 \vee v \geq 2), (v \leq -2 \vee v \geq 0), (v \leq -2 \vee u \leq -1)$$

Model	Equations	Bounds
$M(x) = 1$	$x = s - y$	$s \geq 1$
$M(y) = 1 + \delta$	$u = s + y$	$x \geq 0$
$M(s) = 1$	$v = s - 2y$	$y > 1$
$M(u) = 1$		
$M(v) = 1$		

Example

- Case split $\neg y \le 1$ update dependent variables assignment.

$$s \ge 1, x \ge 0$$

$$(y \le 1 \lor v \ge 2), (v \le -2 \lor v \ge 0), (v \le -2 \lor u \le -1)$$

Model	Equations	Bounds
$M(x) = -\delta$	$x = s - y$	$s \ge 1$
$M(y) = 1 + \delta$	$u = s + y$	$x \ge 0$
$M(s) = 1$	$v = s - 2y$	$y > 1$
$M(u) = 2 + \delta$		
$M(v) = -1 - 2\delta$		

Example

- Bound violation

$$s \geq 1, x \geq 0$$

$$(y \leq 1 \vee v \geq 2), (v \leq -2 \vee v \geq 0), (v \leq -2 \vee u \leq -1)$$

Model

$$\begin{aligned} M(x) &= -\delta \\ M(y) &= 1+\delta \\ M(s) &= 1 \\ M(u) &= 2+\delta \\ M(v) &= -1-2\delta \end{aligned}$$

Equations

$$\begin{aligned} x &= s-y \\ u &= s+y \\ v &= s-2y \end{aligned}$$

Bounds

$$\begin{array}{ccc} s & \geq & 1 \\ x & \geq & 0 \\ \hline y & > & 1 \end{array}$$

Example

- Bound violation pivot x and s (x is a dependent variables).

$$s \geq 1, x \geq 0$$

$$(y \leq 1 \vee v \geq 2), (v \leq -2 \vee v \geq 0), (v \leq -2 \vee u \leq -1)$$

Model	Equations	Bounds
$M(x) = -\delta$	$x = s - y$	$s \geq 1$
$M(y) = 1 + \delta$	$u = s + y$	$x \geq 0$
$M(s) = 1$	$v = s - 2y$	$\overline{y > 1}$
$M(u) = 2 + \delta$		
$M(v) = -1 - 2\delta$		

Example

- Bound violation pivot x and s (x is a dependent variables).

$$s \geq 1, x \geq 0$$

$$(y \leq 1 \vee v \geq 2), (v \leq -2 \vee v \geq 0), (v \leq -2 \vee u \leq -1)$$

Model

$$\begin{aligned} M(x) &= -\delta \\ M(y) &= 1+\delta \\ M(s) &= 1 \\ M(u) &= 2+\delta \\ M(v) &= -1-2\delta \end{aligned}$$

Equations

$$\begin{aligned} s &= x + y \\ u &= s + y \\ v &= s - 2y \end{aligned}$$

Bounds

$$\begin{array}{ccc} s & \geq & 1 \\ x & \geq & 0 \\ \hline y & > & 1 \end{array}$$

Example

- Bound violation pivot x and s (x is a dependent variables).

$$s \geq 1, x \geq 0$$

$$(y \leq 1 \vee v \geq 2), (v \leq -2 \vee v \geq 0), (v \leq -2 \vee u \leq -1)$$

Model	Equations	Bounds
$M(x) = -\delta$	$s = x + y$	$s \geq 1$
$M(y) = 1 + \delta$	$u = x + 2y$	$x \geq 0$
$M(s) = 1$	$v = x - y$	$y > 1$
$M(u) = 2 + \delta$		
$M(v) = -1 - 2\delta$		

Example

- Bound violation   update assignment.

$$s \geq 1, x \geq 0$$

$$(y \leq 1 \vee v \geq 2), (v \leq -2 \vee v \geq 0), (v \leq -2 \vee u \leq -1)$$

Model	Equations	Bounds
$M(x) = 0$	$s = x + y$	$s \geq 1$
$M(y) = 1 + \delta$	$u = x + 2y$	$x \geq 0$
$M(s) = 1$	$v = x - y$	$y > 1$
$M(u) = 2 + \delta$		
$M(v) = -1 - 2\delta$		

Example

- Bound violation update dependent variables assignment.

$$s \geq 1, x \geq 0$$

$$(y \leq 1 \vee v \geq 2), (v \leq -2 \vee v \geq 0), (v \leq -2 \vee u \leq -1)$$

Model

$$\begin{aligned} M(x) &= 0 \\ M(y) &= 1 + \delta \\ M(s) &= 1 + \delta \\ M(u) &= 2 + 2\delta \\ M(v) &= -1 - \delta \end{aligned}$$

Equations

$$\begin{aligned} s &= x + y \\ u &= x + 2y \\ v &= x - y \end{aligned}$$

Bounds

$$\begin{array}{ccc} s & \geq & 1 \\ x & \geq & 0 \\ \hline y & > & 1 \end{array}$$

Example

- Theory propagation $x \geq 0, y > 1 \rightsquigarrow u > 2$

$$s \geq 1, x \geq 0$$

$$(y \leq 1 \vee v \geq 2), (v \leq -2 \vee v \geq 0), (v \leq -2 \vee u \leq -1)$$

Model	Equations	Bounds
$M(x) = 0$	$s = x + y$	$s \geq 1$
$M(y) = 1 + \delta$	$u = x + 2y$	$x \geq 0$
$M(s) = 1 + \delta$	$v = x - y$	$y > 1$
$M(u) = 2 + 2\delta$		
$M(v) = -1 - \delta$		

Example

- Theory propagation $u > 2 \leadsto \neg u \leq -1$

$$s \geq 1, x \geq 0$$

$$(y \leq 1 \vee v \geq 2), (v \leq -2 \vee v \geq 0), (v \leq -2 \vee u \leq -1)$$

Model	Equations	Bounds
$M(x) = 0$	$s = x + y$	$s \geq 1$
$M(y) = 1 + \delta$	$u = x + 2y$	$x \geq 0$
$M(s) = 1 + \delta$	$v = x - y$	$y > 1$
$M(u) = 2 + 2\delta$		$u > 2$
$M(v) = -1 - \delta$		

Example

- Boolean propagation $\neg y \leq 1 \rightsquigarrow v \geq 2$

$$s \geq 1, x \geq 0$$

$$(y \leq 1 \vee v \geq 2), (v \leq -2 \vee v \geq 0), (v \leq -2 \vee u \leq -1)$$

Model	Equations	Bounds
$M(x) = 0$	$s = x + y$	$s \geq 1$
$M(y) = 1 + \delta$	$u = x + 2y$	$x \geq 0$
$M(s) = 1 + \delta$	$v = x - y$	$y > 1$
$M(u) = 2 + 2\delta$		$u > 2$
$M(v) = -1 - \delta$		

Example

- Theory propagation $v \geq 2 \rightsquigarrow \neg v \leq -2$

$$s \geq 1, x \geq 0$$

$$(y \leq 1 \vee v \geq 2), (v \leq -2 \vee v \geq 0), (v \leq -2 \vee u \leq -1)$$

Model

$$\begin{aligned} M(x) &= 0 \\ M(y) &= 1 + \delta \\ M(s) &= 1 + \delta \\ M(u) &= 2 + 2\delta \\ M(v) &= -1 - \delta \end{aligned}$$

Equations

$$\begin{aligned} s &= x + y \\ u &= x + 2y \\ v &= x - y \end{aligned}$$

Bounds

$$\begin{array}{ccc} s & \geq & 1 \\ x & \geq & 0 \\ \hline y & > & 1 \\ u & > & 2 \end{array}$$

Example

- Conflict empty clause

$$s \geq 1, x \geq 0$$

$$(y \leq 1 \vee v \geq 2), (v \leq -2 \vee v \geq 0), (v \leq -2 \vee u \leq -1)$$

Model

$$\begin{aligned} M(x) &= 0 \\ M(y) &= 1+\delta \\ M(s) &= 1+\delta \\ M(u) &= 2+2\delta \\ M(v) &= -1-\delta \end{aligned}$$

Equations

$$\begin{aligned} s &= x+y \\ u &= x+2y \\ v &= x-y \end{aligned}$$

Bounds

$$\begin{array}{ccc} s & \geq & 1 \\ x & \geq & 0 \\ \hline y & > & 1 \\ u & > & 2 \end{array}$$

Example

- Backtracking

$$s \geq 1, x \geq 0$$

$$(y \leq 1 \vee v \geq 2), (v \leq -2 \vee v \geq 0), (v \leq -2 \vee u \leq -1)$$

Model	Equations	Bounds
$M(x) = 0$	$s = x + y$	$s \geq 1$
$M(y) = 1 + \delta$	$u = x + 2y$	$x \geq 0$
$M(s) = 1 + \delta$	$v = x - y$	
$M(u) = 2 + 2\delta$		
$M(v) = -1 - \delta$		

Example

- Asserting $y \leq 1$

$$s \geq 1, x \geq 0$$

$$(y \leq 1 \vee v \geq 2), (v \leq -2 \vee v \geq 0), (v \leq -2 \vee u \leq -1)$$

Model

$$\begin{aligned} M(x) &= 0 \\ M(y) &= 1+\delta \\ M(s) &= 1+\delta \\ M(u) &= 2+2\delta \\ M(v) &= -1-\delta \end{aligned}$$

Equations

$$\begin{aligned} s &= x+y \\ u &= x+2y \\ v &= x-y \end{aligned}$$

Bounds

$$\begin{aligned} s &\geq 1 \\ x &\geq 0 \end{aligned}$$

Example

- Asserting $y \leq 1$ assignment does not satisfy new bound.

$$s \geq 1, x \geq 0$$

$$(y \leq 1 \vee v \geq 2), (v \leq -2 \vee v \geq 0), (v \leq -2 \vee u \leq -1)$$

Model

$$\begin{aligned} M(x) &= 0 \\ M(y) &= 1+\delta \\ M(s) &= 1+\delta \\ M(u) &= 2+2\delta \\ M(v) &= -1-\delta \end{aligned}$$

Equations

$$\begin{aligned} s &= x+y \\ u &= x+2y \\ v &= x-y \end{aligned}$$

Bounds

$$\begin{array}{ccc} s & \geq & 1 \\ x & \geq & 0 \\ \hline y & \leq & 1 \end{array}$$

Example

- Asserting $y \leq 1$ update assignment.

$$s \geq 1, x \geq 0$$

$$(y \leq 1 \vee v \geq 2), (v \leq -2 \vee v \geq 0), (v \leq -2 \vee u \leq -1)$$

Model	Equations	Bounds
$M(x) = 0$	$s = x + y$	$s \geq 1$
$M(y) = 1$	$u = x + 2y$	$x \geq 0$
$M(s) = 1 + \delta$	$v = x - y$	$y \leq 1$
$M(u) = 2 + 2\delta$		
$M(v) = -1 - \delta$		

Example

- Asserting $y \leq 1$ update dependent variables assignment.

$$s \geq 1, x \geq 0$$

$$(y \leq 1 \vee v \geq 2), (v \leq -2 \vee v \geq 0), (v \leq -2 \vee u \leq -1)$$

Model	Equations	Bounds
$M(x) = 0$	$s = x + y$	$s \geq 1$
$M(y) = 1$	$u = x + 2y$	$x \geq 0$
$M(s) = 1$	$v = x - y$	$y \leq 1$
$M(u) = 2$		
$M(v) = -1$		

Example

- Theory propagation $s \geq 1, y \leq 1 \leadsto v \geq -1$

$$s \geq 1, x \geq 0$$

$$(y \leq 1 \vee v \geq 2), (v \leq -2 \vee v \geq 0), (v \leq -2 \vee u \leq -1)$$

Model	Equations	Bounds
$M(x) = 0$	$x = s - y$	$s \geq 1$
$M(y) = 1$	$u = s + y$	$x \geq 0$
$M(s) = 1$	$v = s - 2y$	$y \leq 1$
$M(u) = 2$		
$M(v) = -1$		

Example

- Theory propagation $v \geq -1 \rightsquigarrow \neg v \leq -2$

$$s \geq 1, x \geq 0$$

$$(y \leq 1 \vee v \geq 2), (v \leq -2 \vee v \geq 0), (v \leq -2 \vee u \leq -1)$$

Model	Equations	Bounds
$M(x) = 0$	$x = s - y$	$s \geq 1$
$M(y) = 1$	$u = s + y$	$x \geq 0$
$M(s) = 1$	$v = s - 2y$	$y \leq 1$
$M(u) = 2$		$v \geq -1$
$M(v) = -1$		

Example

- Boolean propagation $\neg v \le -2 \leadsto v \ge 0$

$$s \ge 1, x \ge 0$$

$$(y \le 1 \vee v \ge 2), (v \le -2 \vee v \ge 0), (v \le -2 \vee u \le -1)$$

Model	Equations	Bounds
$M(x) = 0$	$x = s - y$	$s \ge 1$
$M(y) = 1$	$u = s + y$	$x \ge 0$
$M(s) = 1$	$v = s - 2y$	$y \le 1$
$M(u) = 2$		$v \ge -1$
$M(v) = -1$		

Example

- Bound violation   assignment does not satisfy new bound.

$$s \geq 1, x \geq 0$$

$$(y \leq 1 \vee v \geq 2), (v \leq -2 \vee v \geq 0), (v \leq -2 \vee u \leq -1)$$

Model	Equations	Bounds
$M(x) = 0$	$x = s - y$	$s \geq 1$
$M(y) = 1$	$u = s + y$	$x \geq 0$
$M(s) = 1$	$v = s - 2y$	$y \leq 1$
$M(u) = 2$		$v \geq 0$
$M(v) = -1$		

Example

- Bound violation pivot u and s (u is a dependent variable).

$$s \geq 1, x \geq 0$$

$$(y \leq 1 \vee v \geq 2), (v \leq -2 \vee v \geq 0), (v \leq -2 \vee u \leq -1)$$

Model	Equations	Bounds
$M(x) = 0$	$x = s - y$	$s \geq 1$
$M(y) = 1$	$u = s + y$	$x \geq 0$
$M(s) = 1$	$v = s - 2y$	$y \leq 1$
$M(u) = 2$		$v \geq 0$
$M(v) = -1$		

Example

- Bound violation pivot u and s (u is a dependent variable).

$$s \geq 1, x \geq 0$$

$$(y \leq 1 \vee v \geq 2), (v \leq -2 \vee v \geq 0), (v \leq -2 \vee u \leq -1)$$

Model	Equations	Bounds
$M(x) = 0$	$x = s - y$	$s \geq 1$
$M(y) = 1$	$u = s + y$	$x \geq 0$
$M(s) = 1$	$s = v + 2y$	$y \leq 1$
$M(u) = 2$		$v \geq 0$
$M(v) = -1$		

Example

- Bound violation pivot u and s (u is a dependent variable).

$$s \geq 1, x \geq 0$$

$$(y \leq 1 \vee v \geq 2), (v \leq -2 \vee v \geq 0), (v \leq -2 \vee u \leq -1)$$

Model	Equations	Bounds
$M(x) = 0$	$x = v + y$	$s \geq 1$
$M(y) = 1$	$u = v + 3y$	$x \geq 0$
$M(s) = 1$	$s = v + 2y$	$y \leq 1$
$M(u) = 2$		$v \geq 0$
$M(v) = -1$		

Example

- Bound violation update assignment.

$$s \geq 1, x \geq 0$$

$$(y \leq 1 \vee v \geq 2), (v \leq -2 \vee v \geq 0), (v \leq -2 \vee u \leq -1)$$

Model	Equations	Bounds
$M(x) = 0$	$x = v + y$	$s \geq 1$
$M(y) = 1$	$u = v + 3y$	$x \geq 0$
$M(s) = 1$	$s = v + 2y$	$y \leq 1$
$M(u) = 2$		$v \geq 0$
$M(v) = 0$		

Example

- Bound violation update dependent variables assignment.

$$s \geq 1, x \geq 0$$

$$(y \leq 1 \vee v \geq 2), (v \leq -2 \vee v \geq 0), (v \leq -2 \vee u \leq -1)$$

Model	Equations	Bounds
$M(x) = 1$	$x = v + y$	$s \geq 1$
$M(y) = 1$	$u = v + 3y$	$x \geq 0$
$M(s) = 2$	$s = v + 2y$	$y \leq 1$
$M(u) = 3$		$v \geq 0$
$M(v) = 0$		

Example

- Boolean propagation $\neg v \leq -2 \rightsquigarrow u \leq -1$

$$s \geq 1, x \geq 0$$

$$(y \leq 1 \vee v \geq 2), (v \leq -2 \vee v \geq 0), (v \leq -2 \vee u \leq -1)$$

Model	Equations	Bounds
$M(x) = 1$	$x = v + y$	$s \geq 1$
$M(y) = 1$	$u = v + 3y$	$x \geq 0$
$M(s) = 2$	$s = v + 2y$	$y \leq 1$
$M(u) = 3$		$v \geq 0$
$M(v) = 0$		

Example

- Bound violation   assignment does not satisfy new bound.

$$s \geq 1, x \geq 0$$

$$(y \leq 1 \vee v \geq 2), (v \leq -2 \vee v \geq 0), (v \leq -2 \vee u \leq -1)$$

Model	Equations	Bounds
$M(x) = 1$	$x = v + y$	$s \geq 1$
$M(y) = 1$	$u = v + 3y$	$x \geq 0$
$M(s) = 2$	$s = v + 2y$	$y \leq 1$
$M(u) = 3$		$v \geq 0$
$M(v) = 0$		$u \leq -1$

Example

- Bound violation pivot u and y (u is a dependent variable).

$$s \geq 1, x \geq 0$$

$$(y \leq 1 \vee v \geq 2), (v \leq -2 \vee v \geq 0), (v \leq -2 \vee u \leq -1)$$

Model	Equations	Bounds
$M(x) = 1$	$x = v + y$	$s \geq 1$
$M(y) = 1$	$u = v + 3y$	$x \geq 0$
$M(s) = 2$	$s = v + 2y$	$y \leq 1$
$M(u) = 3$		$v \geq 0$
$M(v) = 0$		$u \leq -1$

Example

- Bound violation pivot u and y (u is a dependent variable).

$$s \geq 1, x \geq 0$$

$$(y \leq 1 \vee v \geq 2), (v \leq -2 \vee v \geq 0), (v \leq -2 \vee u \leq -1)$$

Model	Equations	Bounds
$M(x) = 1$	$x = v + y$	$s \geq 1$
$M(y) = 1$	$y = \frac{1}{3}u - \frac{1}{3}v$	$x \geq 0$
$M(s) = 2$	$s = v + 2y$	$y \leq 1$
$M(u) = 3$		$v \geq 0$
$M(v) = 0$		$u \leq -1$

Example

- Bound violation pivot u and y (u is a dependent variable).

$$s \geq 1, x \geq 0$$

$$(y \leq 1 \vee v \geq 2), (v \leq -2 \vee v \geq 0), (v \leq -2 \vee u \leq -1)$$

Model	Equations	Bounds
$M(x) = 1$	$x = \frac{1}{3}u + \frac{2}{3}v$	$s \geq 1$
$M(y) = 1$	$y = \frac{1}{3}u - \frac{1}{3}v$	$x \geq 0$
$M(s) = 2$	$s = \frac{2}{3}u + \frac{1}{3}v$	$y \leq 1$
$M(u) = 3$		$v \geq 0$
$M(v) = 0$		$u \leq -1$

Example

- Bound violation   update assignment.

$$s \geq 1, x \geq 0$$

$$(y \leq 1 \vee v \geq 2), (v \leq -2 \vee v \geq 0), (v \leq -2 \vee u \leq -1)$$

Model

$$\begin{array}{lcr} M(x) & = & 1 \\ M(y) & = & 1 \\ M(s) & = & 2 \\ M(u) & = & -1 \\ M(v) & = & 0 \end{array}$$

Equations

$$\begin{array}{lcl} x & = & \frac{1}{3}u + \frac{2}{3}v \\ y & = & \frac{1}{3}u - \frac{1}{3}v \\ s & = & \frac{2}{3}u + \frac{1}{3}v \end{array}$$

Bounds

$$\begin{array}{lcr} s & \geq & 1 \\ x & \geq & 0 \\ \hline y & \leq & 1 \\ v & \geq & 0 \\ u & \leq & -1 \end{array}$$

Example

- Bound violation   update dependent variables assignment.

$$s \geq 1, x \geq 0$$

$$(y \leq 1 \vee v \geq 2), (v \leq -2 \vee v \geq 0), (v \leq -2 \vee u \leq -1)$$

Model

$$\begin{aligned} M(x) &= -\tfrac{1}{3} \\ M(y) &= -\tfrac{1}{3} \\ M(s) &= -\tfrac{2}{3} \\ M(u) &= -1 \\ M(v) &= 0 \end{aligned}$$

Equations

$$\begin{aligned} x &= \tfrac{1}{3}u + \tfrac{2}{3}v \\ y &= \tfrac{1}{3}u - \tfrac{1}{3}v \\ s &= \tfrac{2}{3}u + \tfrac{1}{3}v \end{aligned}$$

Bounds

$$\begin{array}{ccc} s & \geq & 1 \\ x & \geq & 0 \\ \hline y & \leq & 1 \\ v & \geq & 0 \\ u & \leq & -1 \end{array}$$

Example

- Bound violations

$$s \geq 1, x \geq 0$$

$$(y \leq 1 \vee v \geq 2), (v \leq -2 \vee v \geq 0), (v \leq -2 \vee u \leq -1)$$

Model

$$\begin{aligned} M(x) &= -\tfrac{1}{3} \\ M(y) &= -\tfrac{1}{3} \\ M(s) &= -\tfrac{2}{3} \\ M(u) &= -1 \\ M(v) &= 0 \end{aligned}$$

Equations

$$\begin{aligned} x &= \tfrac{1}{3}u + \tfrac{2}{3}v \\ y &= \tfrac{1}{3}u - \tfrac{1}{3}v \\ s &= \tfrac{2}{3}u + \tfrac{1}{3}v \end{aligned}$$

Bounds

$$\begin{array}{ccc} s & \geq & 1 \\ x & \geq & 0 \\ \hline y & \leq & 1 \\ v & \geq & 0 \\ u & \leq & -1 \end{array}$$

Example

- Bound violations pivot s and v (s is a dependent variable).

$$s \geq 1, x \geq 0$$

$$(y \leq 1 \vee v \geq 2), (v \leq -2 \vee v \geq 0), (v \leq -2 \vee u \leq -1)$$

Model	Equations	Bounds
$M(x) = -\frac{1}{3}$	$x = \frac{1}{3}u + \frac{2}{3}v$	$s \geq 1$
$M(y) = -\frac{1}{3}$	$y = \frac{1}{3}u - \frac{1}{3}v$	$x \geq 0$
$M(s) = -\frac{2}{3}$	$s = \frac{2}{3}u + \frac{1}{3}v$	$y \leq 1$
$M(u) = -1$		$v \geq 0$
$M(v) = 0$		$u \leq -1$

Example

- Bound violations pivot s and v (s is a dependent variable).

$$s \geq 1, x \geq 0$$

$$(y \leq 1 \vee v \geq 2), (v \leq -2 \vee v \geq 0), (v \leq -2 \vee u \leq -1)$$

Model

$$\begin{aligned} M(x) &= -\tfrac{1}{3} \\ M(y) &= -\tfrac{1}{3} \\ M(s) &= -\tfrac{2}{3} \\ M(u) &= -1 \\ M(v) &= 0 \end{aligned}$$

Equations

$$\begin{aligned} x &= \tfrac{1}{3}u + \tfrac{2}{3}v \\ y &= \tfrac{1}{3}u - \tfrac{1}{3}v \\ v &= 3s - 2u \end{aligned}$$

Bounds

$$\begin{array}{ccc} s & \geq & 1 \\ x & \geq & 0 \\ \hline y & \leq & 1 \\ v & \geq & 0 \\ u & \leq & -1 \end{array}$$

Example

- Bound violations pivot s and v (s is a dependent variable).

$$s \geq 1, x \geq 0$$

$$(y \leq 1 \vee v \geq 2), (v \leq -2 \vee v \geq 0), (v \leq -2 \vee u \leq -1)$$

Model	Equations	Bounds
$M(x) = -\frac{1}{3}$	$x = 2s - u$	$s \geq 1$
$M(y) = -\frac{1}{3}$	$y = -s + u$	$x \geq 0$
$M(s) = -\frac{2}{3}$	$v = 3s - 2u$	$y \leq 1$
$M(u) = -1$		$v \geq 0$
$M(v) = 0$		$u \leq -1$

Example

- Bound violations update assignment.

$$s \geq 1, x \geq 0$$

$$(y \leq 1 \vee v \geq 2), (v \leq -2 \vee v \geq 0), (v \leq -2 \vee u \leq -1)$$

Model	Equations	Bounds
$M(x) = -\frac{1}{3}$	$x = 2s - u$	$s \geq 1$
$M(y) = -\frac{1}{3}$	$y = -s + u$	$x \geq 0$
$M(s) = 1$	$v = 3s - 2u$	$y \leq 1$
$M(u) = -1$		$v \geq 0$
$M(v) = 0$		$u \leq -1$

Example

- Bound violations update dependent variables assignment.

$$s \geq 1, x \geq 0$$

$$(y \leq 1 \vee v \geq 2), (v \leq -2 \vee v \geq 0), (v \leq -2 \vee u \leq -1)$$

Model	Equations	Bounds
$M(x) = 3$	$x = 2s - u$	$s \geq 1$
$M(y) = -2$	$y = -s + u$	$x \geq 0$
$M(s) = 1$	$v = 3s - 2u$	$y \leq 1$
$M(u) = -1$		$v \geq 0$
$M(v) = 5$		$u \leq -1$

Example

- Found satisfying assignment

$$s \geq 1, x \geq 0$$

$$(y \leq 1 \lor v \geq 2), (v \leq -2 \lor v \geq 0), (v \leq -2 \lor u \leq -1)$$

Model	Equations	Bounds
$M(x) = 3$	$x = 2s - u$	$s \geq 1$
$M(y) = -2$	$y = -s + u$	$x \geq 0$
$M(s) = 1$	$v = 3s - 2u$	$y \leq 1$
$M(u) = -1$		$v \geq 0$
$M(v) = 5$		$u \leq -1$

Correctness

Completeness: trivial

Soundness: also trivial

Termination: non trivial.

We cannot choose arbitrary variable to pivot.

Assume the variables are ordered.

Bland's rule: select the smallest basic variable c that does not satisfy its bounds, then select the smallest non-basic in the row of c that can be used for pivoting.

Too technical.

Uses the fact that a tableau has a finite number of configurations. Then, any infinite trace will have cycles.

Combining Theories

In practice, we need a combination of theories.

b + 2 = c and f(read(write(a,b,3), c-2)) ≠ f(c-b+1)

A theory is a set (potentially infinite) of first-order sentences.

Main questions:

Is the union of two theories T1 ∪ T2 consistent?

Given a solvers for T1 and T2, how can we build a solver for T1 ∪ T2?

Disjoint Theories

Two theories are disjoint if they do not share function/constant and predicate symbols.

= is the only exception.

Example:

The theories of arithmetic and arrays are disjoint.

Arithmetic symbols: {0, -1, 1, -2, 2, ..., +, -, *, >, <, ≥, ≤}

Array symbols: { read, write }

Purification

It is a different name for our "naming" subterms procedure.

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

$\Downarrow$

$b + 2 = c, v_6 \neq v_7$

$v_1 \equiv 3, v_2 \equiv write(a, b, v_1), v_3 \equiv c-2, v_4 \equiv read(v_2, v_3),$

$v_5 \equiv c-b+1, v_6 \equiv f(v_4), v_7 \equiv f(v_5)$

Microsoft Research

Purification

It is a different name for our "naming" subterms procedure.

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

$b + 2 = c,\ v_6 \neq v_7$

$v_1 \equiv 3,\ v_2 \equiv write(a,\ b,\ v_1),\ v_3 \equiv c-2,\ v_4 \equiv read(v_2,\ v_3),$

$v_5 \equiv c-b+1,\ v_6 \equiv f(v_4),\ v_7 \equiv f(v_5)$

$b + 2 = c,\ v_1 \equiv 3,\ v_3 \equiv c-2,\ v_5 \equiv c-b+1,$

$v_2 \equiv write(a,\ b,\ v_1),\ v_4 \equiv read(v_2,\ v_3),$

$v_6 \equiv f(v_4),\ v_7 \equiv f(v_5),\ v_6 \neq v_7$

Stably Infinite Theories

A theory is stably infinite if every satisfiable QFF is satisfiable in an infinite model.

EUF and arithmetic are stably infinite.

Bit-vectors are not.

Important Result

The union of two consistent, disjoint, stably infinite theories is consistent.

Convexity

A theory T is convex iff

for all finite sets S of literals and

for all $a_1 = b_1 \vee \ldots \vee a_n = b_n$

S implies $a_1 = b_1 \vee \ldots \vee a_n = b_n$

iff

S implies $a_i = b_i$ for some $1 \leq i \leq n$

Microsoft Research

Convexity: Results

Every convex theory with non trivial models is stably infinite.

All Horn equational theories are convex.

formulas of the form $s_1 \neq r_1 \vee \ldots \vee s_n \neq r_n \vee t = t'$

Linear rational arithmetic is convex.

Convexity: Negative Results

Linear integer arithmetic is not convex

$1 \leq a \leq 2, b = 1, c = 2$ implies $a = b \vee a = c$

Nonlinear arithmetic

$a^2 = 1, b = 1, c = -1$ implies $a = b \vee a = c$

Theory of bit-vectors

Theory of arrays

$c_1 = read(write(a, i, c_2), j), c_3 = read(a, j)$

implies $c_1 = c_2 \vee c_1 = c_3$

Microsoft Research

Combination of non-convex theories

EUF is convex (O(n log n))

IDL is non-convex (O(nm))

EUF $\cup$ IDL is NP-Complete

Reduce 3CNF to EUF $\cup$ IDL

For each boolean variable p_i add $0 \leq a_i \leq 1$

For each clause $p_1 \vee \neg p_2 \vee p_3$ add

$f(a_1, a_2, a_3) \neq f(0, 1, 0)$

Combination of non-convex theories

EUF is convex (O(n log n))

IDL is non-convex (O(nm))

EUF $\cup$ IDL is NP-Complete

Reduce 3CNF to EUF $\cup$ IDL

For each boolean variable p_i add $0 \leq a_i \leq 1$

For each clause $p_1 \vee \neg p_2 \vee p_3$ add

$$f(a_1, a_2, a_3) \neq f(0, 1, 0)$$

$\Downarrow$ implies

$$a_1 \neq 0 \vee a_2 \neq 1 \vee a_3 \neq 0$$

Nelson-Oppen Combination

Let $\mathcal{T}_1$ and $\mathcal{T}_2$ be consistent, stably infinite theories over disjoint (countable) signatures. Assume satisfiability of conjunction of literals can decided in $O(T_1(n))$ and $O(T_2(n))$ time respectively.

Then,

1. The combined theory $\mathcal{T}$ is consistent and stably infinite.
2. Satisfiability of quantifier free conjunction of literals in $\mathcal{T}$ can be decided in $O(2^{n^2} \times (T_1(n) + T_2(n))$.
3. If $\mathcal{T}_1$ and $\mathcal{T}_2$ are convex, then so is $\mathcal{T}$ and satisfiability in $\mathcal{T}$ is in $O(n^3 \times (T_1(n) + T_2(n)))$.

Nelson-Oppen Combination

The combination procedure:

Initial State: ϕ is a conjunction of literals over $\Sigma_1 \cup \Sigma_2$.

Purification: Preserving satisfiability transform ϕ into $\phi_1 \wedge \phi_2$, such that, $\phi_i \in \Sigma_i$.

Interaction: Guess a partition of $\mathcal{V}(\phi_1) \cap \mathcal{V}(\phi_2)$ into disjoint subsets. Express it as conjunction of literals ψ.
Example. The partition $\{x_1\}, \{x_2, x_3\}, \{x_4\}$ is represented as $x_1 \neq x_2, x_1 \neq x_4, x_2 \neq x_4, x_2 = x_3$.

Component Procedures : Use individual procedures to decide whether $\phi_i \wedge \psi$ is satisfiable.

Return: If both return yes, return yes. No, otherwise.

Microsoft Research

Soundness

Each step is satisfiability preserving.

Say ϕ is satisfiable (in the combination).

- Purification: $\phi_1 \land \phi_2$ is satisfiable.
- Iteration: for some partition ψ, $\phi_1 \land \phi_2 \land \psi$ is satisfiable.
- Component procedures: $\phi_1 \land \psi$ and $\phi_2 \land \psi$ are both satisfiable in component theories.
- Therefore, if the procedure return unsatisfiable, then ϕ is unsatisfiable.

Completeness

Suppose the procedure returns satisfiable.

- Let ψ be the partition and A and B be models of $\mathcal{T}_1 \wedge \phi_1 \wedge \psi$ and $\mathcal{T}_2 \wedge \phi_2 \wedge \psi$.
- The component theories are stably infinite. So, assume the models are infinite (of same cardinality).
- Let h be a bijection between $|A|$ and $|B|$ such that $h(A(x)) = B(x)$ for each shared variable.
- Extend B to $\bar{B}$ by interpretations of symbols in Σ_1: $\bar{B}(f)(b_1, \ldots, b_n) = h(A(f)(h^{-1}(b_1), \ldots, h^{-1}(b_n)))$
- $\bar{B}$ is a model of: $\mathcal{T}_1 \wedge \phi_1 \wedge \mathcal{T}_2 \wedge \phi_2 \wedge \psi$

oft
Research

NO deterministic procedure (for convex theories)

Instead of guessing, we can deduce the equalities to be shared.

Purification: no changes.

Interaction: Deduce an equality $x = y$:

$$\mathcal{T}_1 \vdash (\phi_1 \Rightarrow x = y)$$

Update $\phi_2 := \phi_2 \wedge x = y$. And vice-versa. Repeat until no further changes.

Component Procedures : Use individual procedures to decide whether ϕ_i is satisfiable.

Remark: $\mathcal{T}_i \vdash (\phi_i \Rightarrow x = y)$ iff $\phi_i \wedge x \neq y$ is not satisfiable in $\mathcal{T}_i$.

NO deterministic procedure Completeness

Assume the theories are convex.

- Suppose ϕ_i is satisfiable.
- Let E be the set of equalities $x_j = x_k$ $(j \neq k)$ such that, $\mathcal{T}_i \not\vdash \phi_i \Rightarrow x_j = x_k$.
- By convexity, $\mathcal{T}_i \not\vdash \phi_i \Rightarrow \bigvee_E x_j = x_k$.
- $\phi_i \wedge \bigwedge_E x_j \neq x_k$ is satisfiable.
- The proof now is identical to the nondeterministic case.
- Sharing equalities is sufficient, because a theory $\mathcal{T}_1$ can assume that $x^B \neq y^B$ whenever $x = y$ is not implied by $\mathcal{T}_2$ and vice versa.

Microsoft Research

NO procedure: Example

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

Arithmetic	Arrays	EUF
$b + 2 = c,$	$v_2 \equiv write(a, b, v_1),$	$v_6 \equiv f(v_4),$
$v_1 \equiv 3,$	$v_4 \equiv read(v_2, v_3)$	$v_7 \equiv f(v_5),$
$v_3 \equiv c-2,$		$v_6 \neq v_7$
$v_5 \equiv c-b+1$		

NO procedure: Example

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

Arithmetic

$b + 2 = \mathbf{c},$

$v_1 \equiv 3,$

$v_3 \equiv c-2,$

$v_5 \equiv c-b+1$

Arrays

$v_2 \equiv write(a, b, v_1),$

$v_4 \equiv read(v_2, v_3)$

EUF

$v_6 \equiv f(v_4),$

$v_7 \equiv f(v_5),$

$v_6 \neq v_7$

Substituting c

NO procedure: Example

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

Arithmetic	Arrays	EUF
$b + 2 = c,$	$v_2 \equiv write(a, b, v_1),$	$v_6 \equiv f(v_4),$
$v_1 \equiv 3,$	$v_4 \equiv read(v_2, v_3),$	$v_7 \equiv f(v_5),$
$\mathbf{v_3 \equiv b},$		$v_6 \neq v_7$
$v_5 \equiv 3$		

Propagating $v_3 = b$

NO procedure: Example

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

Arithmetic	Arrays	EUF
$b + 2 = c,$	$\mathbf{v_2} \equiv write(a, \mathbf{b}, v_1),$	$v_6 \equiv f(v_4),$
$v_1 \equiv 3,$	$v_4 \equiv read(\mathbf{v_2}, \mathbf{v_3}),$	$v_7 \equiv f(v_5),$
$v_3 \equiv b,$	$v_3 = b$	$v_6 \neq v_7,$
$v_5 \equiv 3$		$v_3 = b$

Deducing $v_4 = v_1$

NO procedure: Example

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

Arithmetic	Arrays	EUF
$b + 2 = c,$	$v_2 \equiv write(a, b, v_1),$	$v_6 \equiv f(v_4),$
$v_1 \equiv 3,$	$v_4 \equiv read(v_2, v_3),$	$v_7 \equiv f(v_5),$
$v_3 \equiv b,$	$v_3 = b,$	$v_6 \neq v_7,$
$v_5 \equiv 3$	$\mathbf{v_4 = v_1}$	$v_3 = b$

Propagating $v_4 = v_1$

NO procedure: Example

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

Arithmetic	Arrays	EUF
$b + 2 = c,$	$v_2 \equiv \mathrm{write}(a, b, v_1),$	$v_6 \equiv f(v_4),$
$\mathbf{v_1 \equiv 3},$	$v_4 \equiv \mathrm{read}(v_2, v_3),$	$v_7 \equiv f(v_5),$
$v_3 \equiv b,$	$v_3 = b,$	$v_6 \neq v_7,$
$\mathbf{v_5 \equiv 3},$	$v_4 = v_1$	$v_3 = b,$
$v_4 = v_1$		$v_4 = v_1$

Propagating $v_5 = v_1$

NO procedure: Example

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

Arithmetic

$b + 2 = c,$

$v_1 \equiv 3,$

$v_3 \equiv b,$

$v_5 \equiv 3,$

$v_4 = v_1$

Arrays

$v_2 \equiv write(a, b, v_1),$

$v_4 \equiv read(v_2, v_3),$

$v_3 = b,$

$v_4 = v_1$

EUF

$v_6 \equiv f(\mathbf{v_4}),$

$v_7 \equiv f(\mathbf{v_5}),$

$v_6 \neq v_7,$

$v_3 = b,$

$\mathbf{v_4 = v_1},$

$\mathbf{v_5 = v_1}$

Congruence: $v_6 = v_7$

NO procedure: Example

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

Arithmetic

$b + 2 = c,$

$v_1 \equiv 3,$

$v_3 \equiv b,$

$v_5 \equiv 3,$

$v_4 = v_1$

Unsatisfiable

Arrays

$v_2 \equiv write(a, b, v_1),$

$v_4 \equiv read(v_2, v_3),$

$v_3 = b,$

$v_4 = v_1$

EUF

$v_6 \equiv f(v_4),$

$v_7 \equiv f(v_5),$

$\mathbf{v_6 \neq v_7},$

$v_3 = b,$

$v_4 = v_1,$

$v_5 = v_1\ ,$

$\mathbf{v_6 = v_7}$

NO deterministic procedure

Deterministic procedure may fail for non-convex theories.

$0 \leq a \leq 1, 0 \leq b \leq 1, 0 \leq c \leq 1,$

$f(a) \neq f(b),$

$f(a) \neq f(c),$

$f(b) \neq f(c)$

Combining Procedures in Practice

Propagate all implied equalities.

- Deterministic Nelson-Oppen.
- Complete only for convex theories.
- It may be expensive for some theories.

Delayed Theory Combination.

- Nondeterministic Nelson-Oppen.
- Create set of interface equalities $(x = y)$ between shared variables.
- Use SAT solver to guess the partition.
- Disadvantage: the number of additional equality literals is quadratic in the number of shared variables.

Combining Procedures in Practice

Common to these methods is that they are pessimistic about which equalities are propagated.

Model-based Theory Combination

- Optimistic approach.
- Use a candidate model M_i for one of the theories $\mathcal{T}_i$ and propagate all equalities implied by the candidate model, hedging that other theories will agree.

  if $M_i \models \mathcal{T}_i \cup \Gamma_i \cup \{u = v\}$ **then** propagate $u = v$.
- If not, use backtracking to fix the model.
- It is cheaper to enumerate equalities that are implied in a particular model than of all models.

Example

$$x = f(y - 1), f(x) \neq f(y), 0 \leq x \leq 1, 0 \leq y \leq 1$$

Purifying

Example

$$x = f(z), f(x) \neq f(y), 0 \leq x \leq 1, 0 \leq y \leq 1, z = y - 1$$

Example

$\mathcal{T}_E$			$\mathcal{T}_A$	
Literals	*Eq. Classes*	*Model*	*Literals*	*Model*
$x = f(z)$	$\{x, f(z)\}$	$E(x) = *_1$	$0 \leq x \leq 1$	$A(x) = 0$
$f(x) \neq f(y)$	$\{y\}$	$E(y) = *_2$	$0 \leq y \leq 1$	$A(y) = 0$
	$\{z\}$	$E(z) = *_3$	$z = y - 1$	$A(z) = -1$
	$\{f(x)\}$	$E(f) = \{*_1 \mapsto *_4,$		
	$\{f(y)\}$	$*_2 \mapsto *_5,$		
		$*_3 \mapsto *_1,$		
		else $\mapsto *_6\}$		

Assume x = y

Example

$\mathcal{T}_E$			$\mathcal{T}_A$	
Literals	*Eq. Classes*	*Model*	*Literals*	*Model*
$x = f(z)$	$\{x, y, f(z)\}$	$E(x) = *_1$	$0 \leq x \leq 1$	$A(x) = 0$
$f(x) \neq f(y)$	$\{z\}$	$E(y) = *_1$	$0 \leq y \leq 1$	$A(y) = 0$
$x = y$	$\{f(x), f(y)\}$	$E(z) = *_2$	$z = y - 1$	$A(z) = -1$
		$E(f) = \{*_1 \mapsto *_3,$	$x = y$	
		$*_2 \mapsto *_1,$		
		else $\mapsto *_4\}$		

Unsatisfiable

Example

$\mathcal{T}_E$			$\mathcal{T}_A$	
Literals	*Eq. Classes*	*Model*	*Literals*	*Model*
$x = f(z)$	$\{x, f(z)\}$	$E(x) = *_1$	$0 \leq x \leq 1$	$A(x) = 0$
$f(x) \neq f(y)$	$\{y\}$	$E(y) = *_2$	$0 \leq y \leq 1$	$A(y) = 0$
$x \neq y$	$\{z\}$	$E(z) = *_3$	$z = y - 1$	$A(z) = -1$
	$\{f(x)\}$	$E(f) = \{*_1 \mapsto *_4,$	$x \neq y$	
	$\{f(y)\}$	$*_2 \mapsto *_5,$		
		$*_3 \mapsto *_1,$		
		else $\mapsto *_6\}$		

Backtrack, and assert $x \neq y$.

$\mathcal{T}_A$ model need to be fixed.

Example

$\mathcal{T}_E$			$\mathcal{T}_A$	
Literals	*Eq. Classes*	*Model*	*Literals*	*Model*
$x = f(z)$	$\{x, f(z)\}$	$E(x) = *_1$	$0 \leq x \leq 1$	$A(x) = 0$
$f(x) \neq f(y)$	$\{y\}$	$E(y) = *_2$	$0 \leq y \leq 1$	$A(y) = 1$
$x \neq y$	$\{z\}$	$E(z) = *_3$	$z = y - 1$	$A(z) = 0$
	$\{f(x)\}$	$E(f) = \{*_1 \mapsto *_4,$	$x \neq y$	
	$\{f(y)\}$	$*_2 \mapsto *_5,$		
		$*_3 \mapsto *_1,$		
		else $\mapsto *_6\}$		

Assume x = z

Example

$\mathcal{T}_E$			$\mathcal{T}_A$	
Literals	*Eq. Classes*	*Model*	*Literals*	*Model*
$x = f(z)$	$\{x, z,$	$E(x) = *_1$	$0 \leq x \leq 1$	$A(x) = 0$
$f(x) \neq f(y)$	$f(x), f(z)\}$	$E(y) = *_2$	$0 \leq y \leq 1$	$A(y) = 1$
$x \neq y$	$\{y\}$	$E(z) = *_1$	$z = y - 1$	$A(z) = 0$
$x = z$	$\{f(y)\}$	$E(f) = \{*_1 \mapsto *_1,$	$x \neq y$	
		$*_2 \mapsto *_3,$	$x = z$	
		else $\mapsto *_4\}$		

Satisfiable

Example

$\mathcal{T}_E$			$\mathcal{T}_A$	
Literals	*Eq. Classes*	*Model*	*Literals*	*Model*
$x = f(z)$	$\{x, z,$	$E(x) = *_1$	$0 \leq x \leq 1$	$A(x) = 0$
$f(x) \neq f(y)$	$f(x), f(z)\}$	$E(y) = *_2$	$0 \leq y \leq 1$	$A(y) = 1$
$x \neq y$	$\{y\}$	$E(z) = *_1$	$z = y - 1$	$A(z) = 0$
$x = z$	$\{f(y)\}$	$E(f) = \{*_1 \mapsto *_1,$	$x \neq y$	
		$*_2 \mapsto *_3,$	$x = z$	
		else $\mapsto *_4\}$		

Let h be the bijection between $|E|$ and $|A|$.

$$h = \{*_1 \mapsto 0, *_2 \mapsto 1, *_3 \mapsto -1, *_4 \mapsto 2, \ldots\}$$

Example

$\mathcal{T}_E$		$\mathcal{T}_A$	
Literals	*Model*	*Literals*	*Model*
$x = f(z)$	$E(x) = *_1$	$0 \leq x \leq 1$	$A(x) = 0$
$f(x) \neq f(y)$	$E(y) = *_2$	$0 \leq y \leq 1$	$A(y) = 1$
$x \neq y$	$E(z) = *_1$	$z = y - 1$	$A(z) = 0$
$x = z$	$E(f) = \{*_1 \mapsto *_1,$	$x \neq y$	$A(f) = \{0 \mapsto 0$
	$*_2 \mapsto *_3,$	$x = z$	$1 \mapsto -1$
	else $\mapsto *_4\}$		*else* $\mapsto 2\}$

Extending A using h.

$$h = \{*_1 \mapsto 0, *_2 \mapsto 1, *_3 \mapsto -1, *_4 \mapsto 2, \ldots\}$$

Non-stably infinite theories in practice

Bit-vector theory is not stably-infinite.

How can we support it?

Solution: add a predicate *is-bv*(x) to the bit-vector theory (intuition: *is-bv*(x) is true iff x is a bitvector).

The result of the bit-vector operation *op*(x, y) is not specified if $\neg$*is-bv*(x) or $\neg$*is-bv*(y).

The new bit-vector theory is stably-infinite.

Reduction Functions

A reduction function reduces the satifiability problem for a complex theory into the satisfiability problem of a simpler theory.

Ackermannization is a reduction function.

Reduction Functions

Theory of commutative functions.

- $\forall x, y. f(x, y) = f(y, x)$
- Reduction to EUF
- For every $f(a, b)$ in ϕ, do $\phi := \phi \wedge f(a, b) = f(b, a)$.

Applications

- Test case generation
- Verifying Compilers
- Predicate Abstraction
- Invariant Generation
- Type Checking
- Model Based Testing

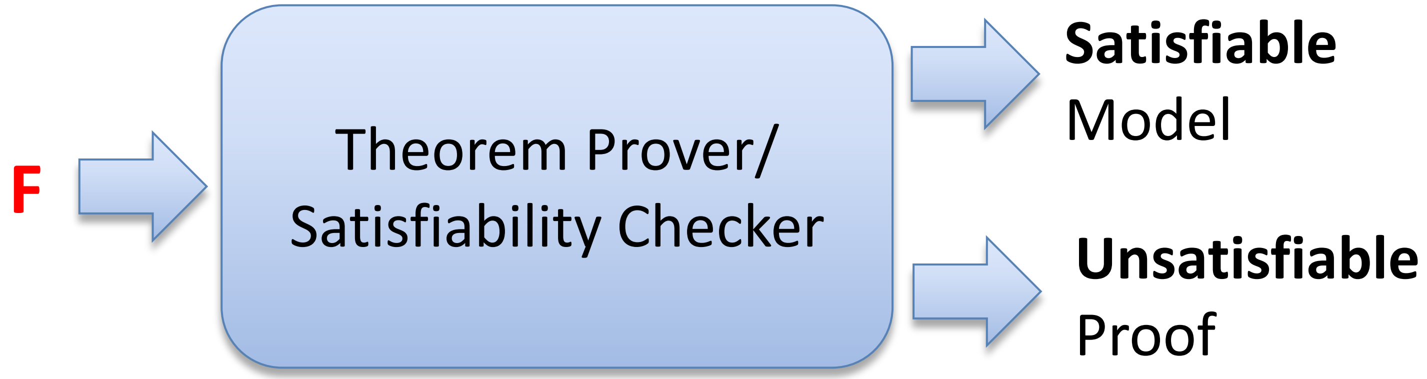
Microsoft Research

Theorem Provers/Satisfiability Checkers

A formula F is valid

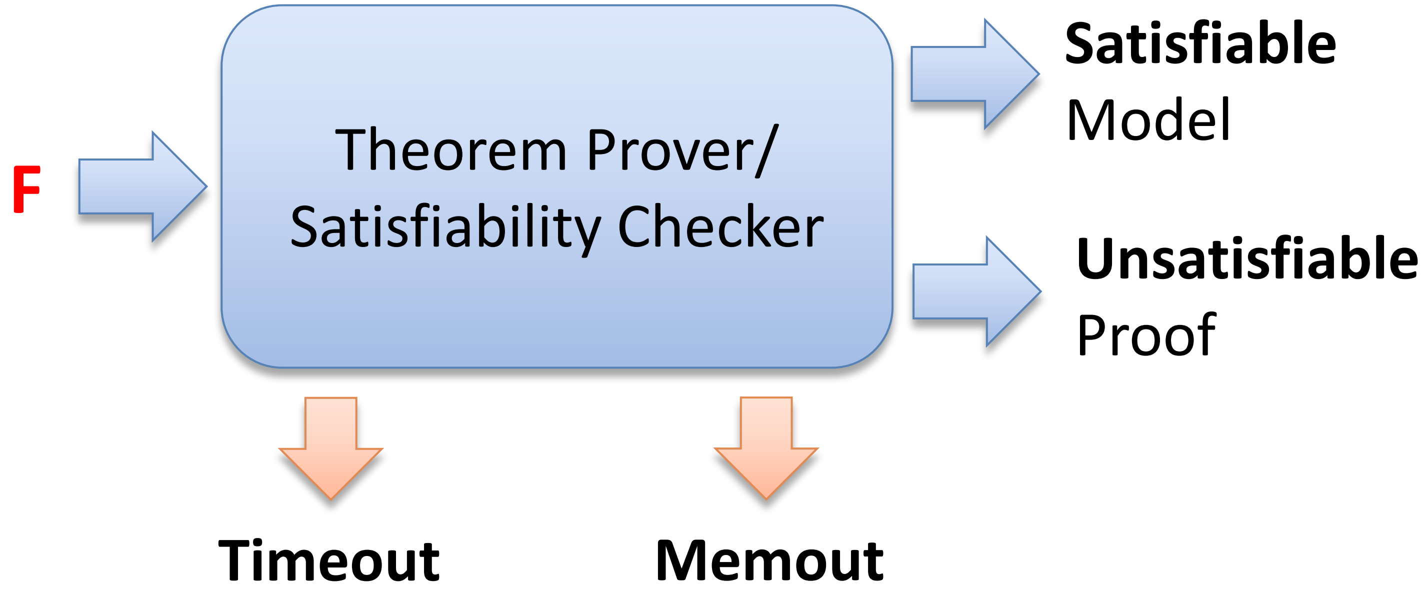
Iff

$\neg$F is unsatisfiable



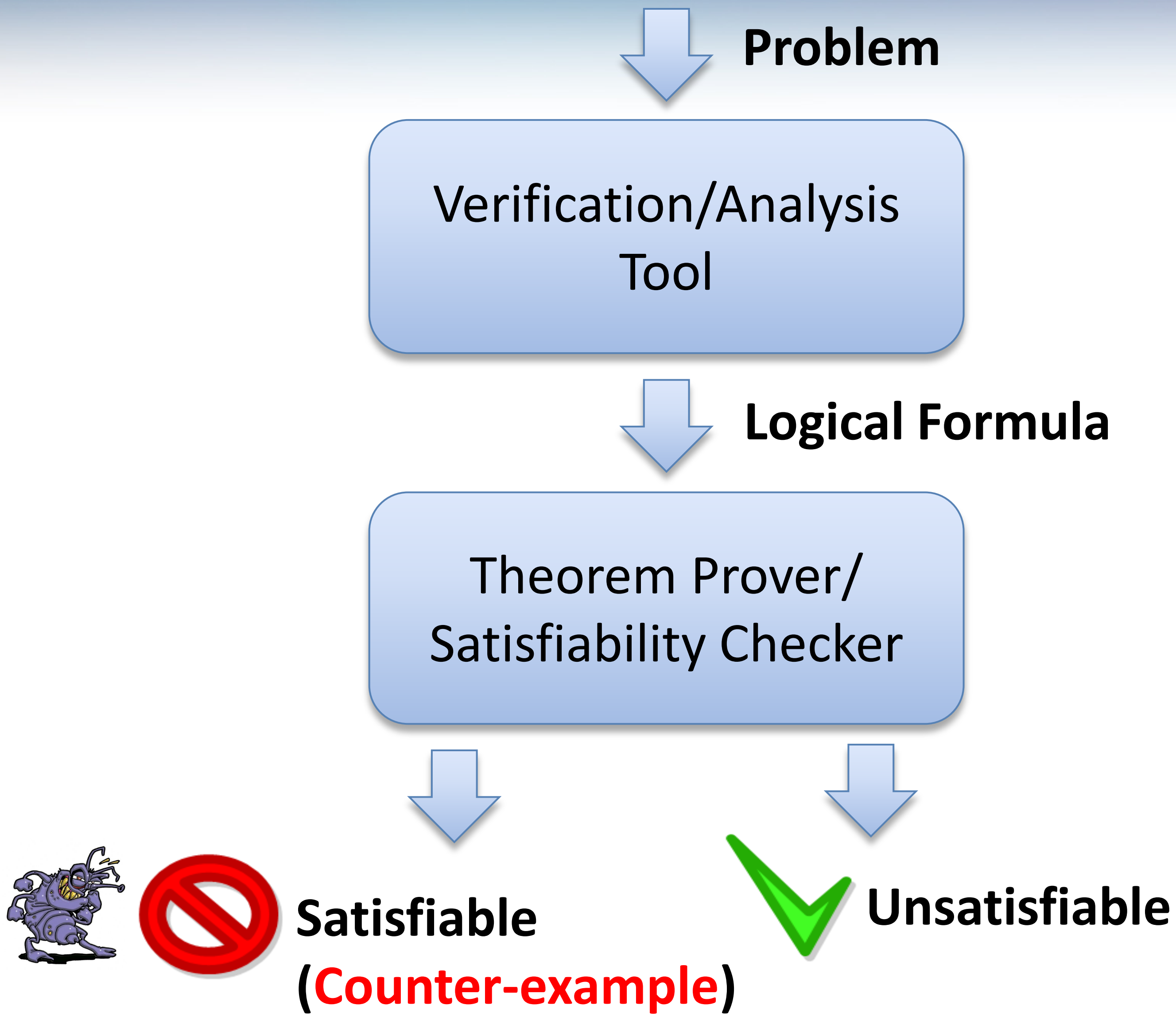
Theorem Provers/Satisfiability Checkers

A formula F is valid
Iff
$\neg$F is unsatisfiable



Microsoft Research

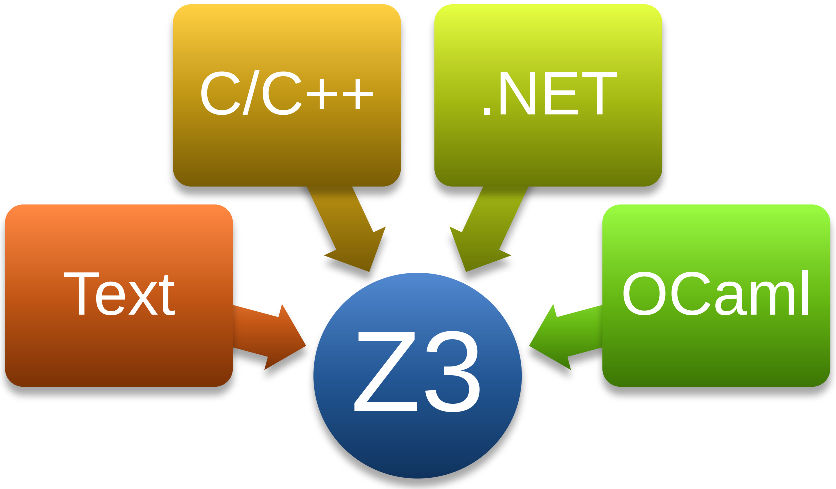
Verification/Analysis Tool: “Template”


Problem
Verification/Analysis Tool
Logical Formula
Theorem Prover/ Satisfiability Checker
Satisfiable (Counter-example)
Unsatisfiable

Microsoft Research

SMT@Microsoft: Solver

- Z3 is a new solver developed at Microsoft Research.
- Development/Research driven by internal customers.
- Free for academic research.
- Interfaces:



- http://research.microsoft.com/projects/z3

Microsoft Research

Test case generation

Test case generation

- Test (correctness + usability) is 95% of the deal:
  - Dev/Test is 1-1 in products.
  - Developers are responsible for unit tests.
- Tools:
  - Annotations and static analysis (SAL + ESP)
  - File Fuzzing
  - Unit test case generation

Microsoft® Research

Security is critical

- Security bugs can be very expensive:
  - Cost of each MS Security Bulletin: $600k to $Millions.
  - Cost due to worms: $Billions.
  - The real victim is the customer.
- Most security exploits are initiated via files or packets.
  - Ex: Internet Explorer parses dozens of file formats.
- Security testing: hunting for million dollar bugs
  - Write A/V
  - Read A/V
  - Null pointer dereference
  - Division by zero

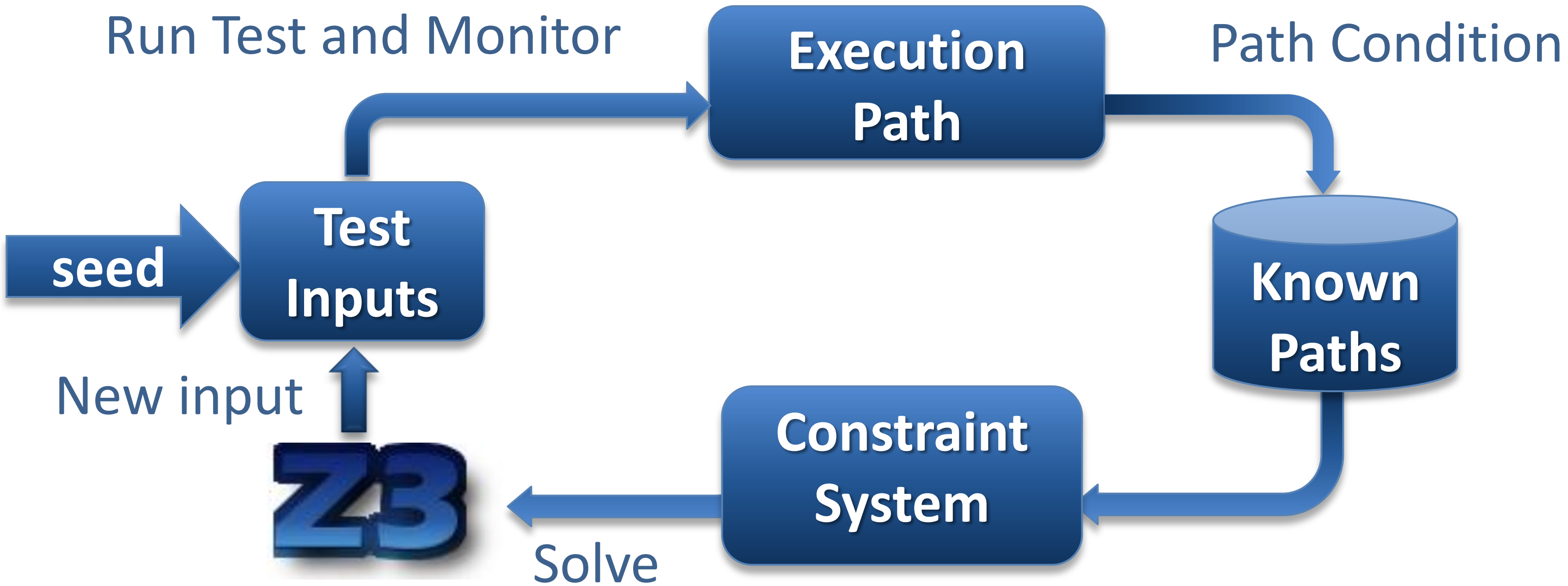
Microsoft Research

Hunting for Security Bugs.

- Two main techniques used by *"black hats"*:
  - Code inspection (of binaries).
  - *Black box fuzz testing.*
- **Black box** fuzz testing:
  - A form of black box random testing.
  - Randomly *fuzz* (=modify) a well formed input.
  - Grammar-based fuzzing: rules to encode how to fuzz.
- **Heavily** used in security testing
  - At MS: several internal tools.
  - Conceptually simple yet effective in practice

Microsoft Research

Directed Automated Random Testing (DART)



DARTish projects at Microsoft

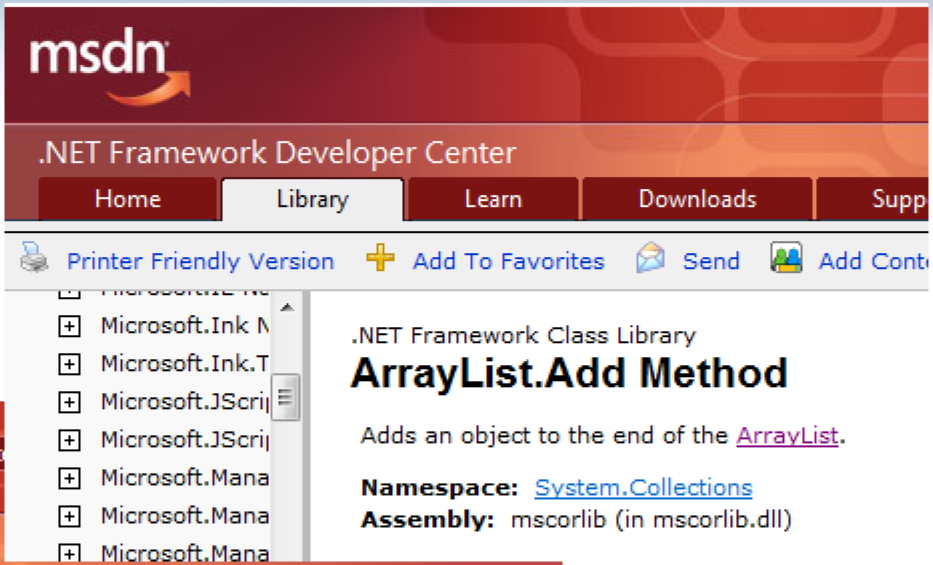
PEX	Implements DART for .NET.
SAGE	Implements DART for x86 binaries.
YOGI	Implements DART to check the feasibility of program paths generated statically.
Vigilante	Partially implements DART to dynamically generate worm filters.

Microsoft Research

What is Pex?

- Test input generator
  - Pex starts from parameterized unit tests
  - Generated tests are emitted as traditional unit tests

ArrayList: The Spec

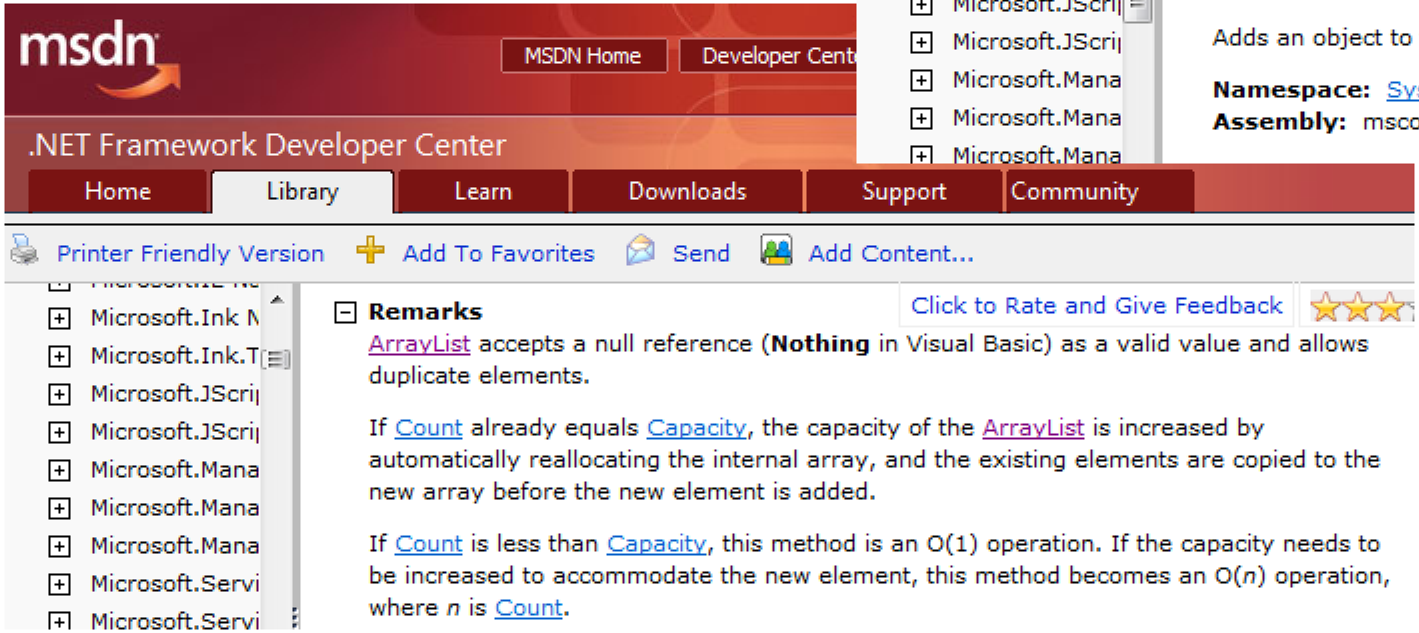


.NET Framework Class Library

ArrayList.Add Method

Adds an object to the end of the ArrayList.

Namespace: System.Collections
Assembly: mscorlib (in mscorlib.dll)



Remarks

ArrayList accepts a null reference (**Nothing** in Visual Basic) as a valid value and allows duplicate elements.

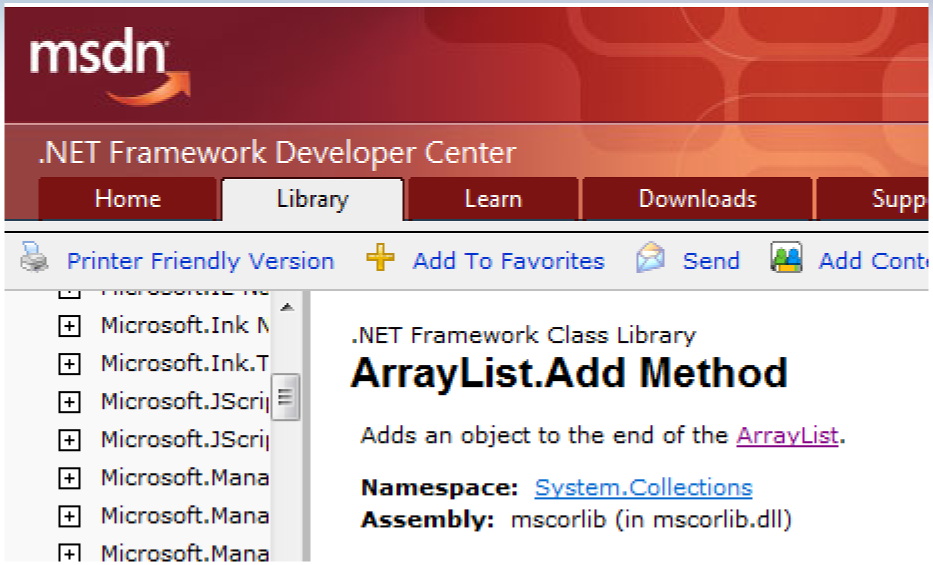
If Count already equals Capacity, the capacity of the ArrayList is increased by automatically reallocating the internal array, and the existing elements are copied to the new array before the new element is added.

If Count is less than Capacity, this method is an O(1) operation. If the capacity needs to be increased to accommodate the new element, this method becomes an O(*n*) operation, where *n* is Count.

ArrayList: AddItem Test

```
class ArrayListTest {
  [PexMethod]
  void AddItem(int c, object item) {
      var list = new ArrayList(c);
      list.Add(item);
      Assert(list[0] == item); }
}
```

```
class ArrayList {
  object[] items;
  int count;

  ArrayList(int capacity) {
    if (capacity < 0) throw ...;
    items = new object[capacity];
  }

  void Add(object item) {
    if (count == items.Length)
      ResizeArray();

    items[this.count++] = item; }
...
```



.NET Framework Class Library

ArrayList.Add Method

Adds an object to the end of the ArrayList.

Namespace: System.Collections
Assembly: mscorlib (in mscorlib.dll)

Microsoft Research

ArrayList: Starting Pex…

```
class ArrayListTest {
  [PexMethod]
  void AddItem(int c, object item) {
      var list = new ArrayList(c);
      list.Add(item);
      Assert(list[0] == item); }
}
```

```
class ArrayList {
  object[] items;
  int count;

  ArrayList(int capacity) {
    if (capacity < 0) throw ...;
    items = new object[capacity];
  }

  void Add(object item) {
    if (count == items.Length)
      ResizeArray();

    items[this.count++] = item; }
...
```

Inputs

Microsoft Research

ArrayList: Run 1, (0,null)

```
class ArrayListTest {
  [PexMethod]
  void AddItem(int c, object item) {
      var list = new ArrayList(c);
      list.Add(item);
      Assert(list[0] == item); }
}
```

```
class ArrayList {
  object[] items;
  int count;

  ArrayList(int capacity) {
    if (capacity < 0) throw ...;
    items = new object[capacity];
  }

  void Add(object item) {
    if (count == items.Length)
      ResizeArray();

    items[this.count++] = item; }
...
```

Inputs
(0,null)

Microsoft Research

ArrayList: Run 1, (0,null)

```
class ArrayListTest {
  [PexMethod]
  void AddItem(int c, object item) {
      var list = new ArrayList(c);
      list.Add(item);
      Assert(list[0] == item); }
}
```

	Inputs	Observed Constraints
	(0,null)	!(c<0)

```
class ArrayList {
  object[] items;
  int count;

  ArrayList(int capacity) {
    if (capacity < 0) throw ...;
    items = new object[capacity];
  }

  void Add(object item) {
    if (count == items.Length)
      ResizeArray();

    items[this.count++] = item; }
...
```

c < 0 → false

Microsoft Research

ArrayList: Run 1, (0,null)

```
class ArrayListTest {
  [PexMethod]
  void AddItem(int c, object item) {
      var list = new ArrayList(c);
      list.Add(item);
      Assert(list[0] == item); }
}
```

	Inputs	Observed Constraints
	(0,null)	!(c<0) && 0==c

```
class ArrayList {
  object[] items;
  int count;

  ArrayList(int capacity) {
    if (capacity < 0) throw ...;
    items = new object[capacity];
  }

  void Add(object item) {
    if (count == items.Length)
      ResizeArray();

    items[this.count++] = item; }
...
```

0 == c → true

Microsoft Research

ArrayList: Run 1, (0,null)

```
class ArrayListTest {
  [PexMethod]
  void AddItem(int c, object item) {
      var list = new ArrayList(c);
      list.Add(item);
      Assert(list[0] == item); }
}
```

	Inputs	Observed Constraints
	(0,null)	!(c<0) && 0==c

```
item == item  →  true
```

```
class ArrayList {
  object[] items;
  int count;

  ArrayList(int capacity) {
    if (capacity < 0) throw ...;
    items = new object[capacity];
  }

  void Add(object item) {
    if (count == items.Length)
      ResizeArray();

    items[this.count++] = item; }
...
```

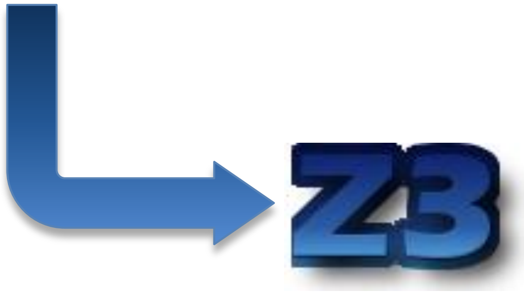
Microsoft Research

ArrayList: Picking the next branch to cover

```
class ArrayListTest {
  [PexMethod]
  void AddItem(int c, object item) {
      var list = new ArrayList(c);
      list.Add(item);
      Assert(list[0] == item); }
}
```

```
class ArrayList {
  object[] items;
  int count;

  ArrayList(int capacity) {
    if (capacity < 0) throw ...;
    items = new object[capacity];
  }

  void Add(object item) {
    if (count == items.Length)
      ResizeArray();

    items[this.count++] = item; }
...
```

Constraints to solve	Inputs	Observed Constraints
	`(0,null)`	`!(c<0) && 0==c`
`!(c<0) && 0!=c`		



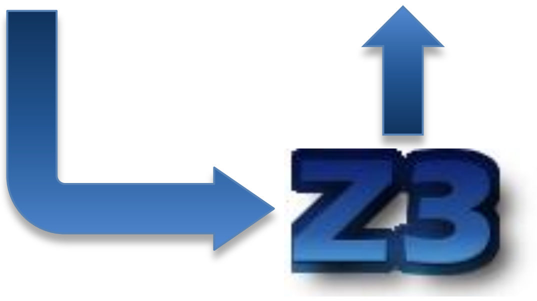
Microsoft Research

ArrayList: Solve constraints using SMT solver

```
class ArrayListTest {
  [PexMethod]
  void AddItem(int c, object item) {
      var list = new ArrayList(c);
      list.Add(item);
      Assert(list[0] == item); }
}
```

```
class ArrayList {
  object[] items;
  int count;

  ArrayList(int capacity) {
    if (capacity < 0) throw ...;
    items = new object[capacity];
  }

  void Add(object item) {
    if (count == items.Length)
      ResizeArray();

    items[this.count++] = item; }
...
```

Constraints to solve	Inputs	Observed Constraints
	(0,null)	!(c<0) && 0==c
!(c<0) && 0!=c	**(1,null)**	



Microsoft Research

ArrayList: Run 2, (1, null)

```
class ArrayListTest {
  [PexMethod]
  void AddItem(int c, object item) {
      var list = new ArrayList(c);
      list.Add(item);
      Assert(list[0] == item); }
}
```

Constraints to solve	Inputs	Observed Constraints
	(0,null)	!(c<0) && 0==c
!(c<0) && 0!=c	(1,null)	!(c<0) && 0!=c

```
class ArrayList {
  object[] items;
  int count;

  ArrayList(int capacity) {
    if (capacity < 0) throw ...;
    items = new object[capacity];
  }

  void Add(object item) {
    if (count == items.Length)
      ResizeArray();

    items[this.count++] = item; }
...
```

0 == c → false

Microsoft Research

ArrayList: Pick new branch

```
class ArrayListTest {
  [PexMethod]
  void AddItem(int c, object item) {
      var list = new ArrayList(c);
      list.Add(item);
      Assert(list[0] == item); }
}
```

```
class ArrayList {
  object[] items;
  int count;

  ArrayList(int capacity) {
    if (capacity < 0) throw ...;
    items = new object[capacity];
  }

  void Add(object item) {
    if (count == items.Length)
      ResizeArray();

    items[this.count++] = item; }
...
```

Constraints to solve	Inputs	Observed Constraints
	(0,null)	!(c<0) && 0==c
!(c<0) && 0!=c	(1,null)	!(c<0) && 0!=c
c<0		

Microsoft® Research

ArrayList: Run 3, (-1, null)

```
class ArrayListTest {
  [PexMethod]
  void AddItem(int c, object item) {
      var list = new ArrayList(c);
      list.Add(item);
      Assert(list[0] == item); }
}
```

```
class ArrayList {
  object[] items;
  int count;

  ArrayList(int capacity) {
    if (capacity < 0) throw ...;
    items = new object[capacity];
  }

  void Add(object item) {
    if (count == items.Length)
      ResizeArray();

    items[this.count++] = item; }
...
```

Constraints to solve	Inputs	Observed Constraints
	(0,null)	!(c<0) && 0==c
!(c<0) && 0!=c	(1,null)	!(c<0) && 0!=c
c<0	**(-1,null)**	

Microsoft Research

ArrayList: Run 3, (-1, null)

```
class ArrayListTest {
  [PexMethod]
  void AddItem(int c, object item) {
      var list = new ArrayList(c);
      list.Add(item);
      Assert(list[0] == item); }
}
```

Constraints to solve	Inputs	Observed Constraints
	(0,null)	!(c<0) && 0==c
!(c<0) && 0!=c	(1,null)	!(c<0) && 0!=c
c<0	**(-1,null)**	c<0

```
class ArrayList {
  object[] items;
  int count;

  ArrayList(int capacity) {
    if (capacity < 0) throw ...;
    items = new object[capacity];
  }

  void Add(object item) {
    if (count == items.Length)
      ResizeArray();

    items[this.count++] = item; }
...
```

c < 0 → true

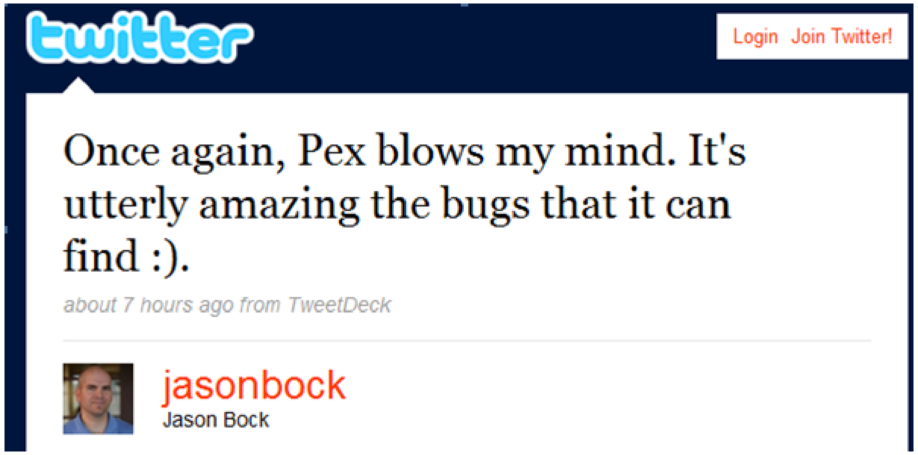
Microsoft Research

ArrayList: Run 3, (-1, null)

```
class ArrayListTest {
  [PexMethod]
  void AddItem(int c, object item) {
      var list = new ArrayList(c);
      list.Add(item);
      Assert(list[0] == item); }
}
```

Constraints to solve	Inputs	Observed Constraints
	(0,null)	!(c<0) && 0==c
!(c<0) && 0!=c	(1,null)	!(c<0) && 0!=c
c<0	(-1,null)	c<0

```
class ArrayList {
  object[] items;
  int count;

  ArrayList(int capacity) {
    if (capacity < 0) throw ...;
    items = new object[capacity];
  }

  void Add(object item) {
    if (count == items.Length)
      ResizeArray();

    items[this.count++] = item; }
...
```


Once again, Pex blows my mind. It's utterly amazing the bugs that it can find :).

about 7 hours ago from TweetDeck

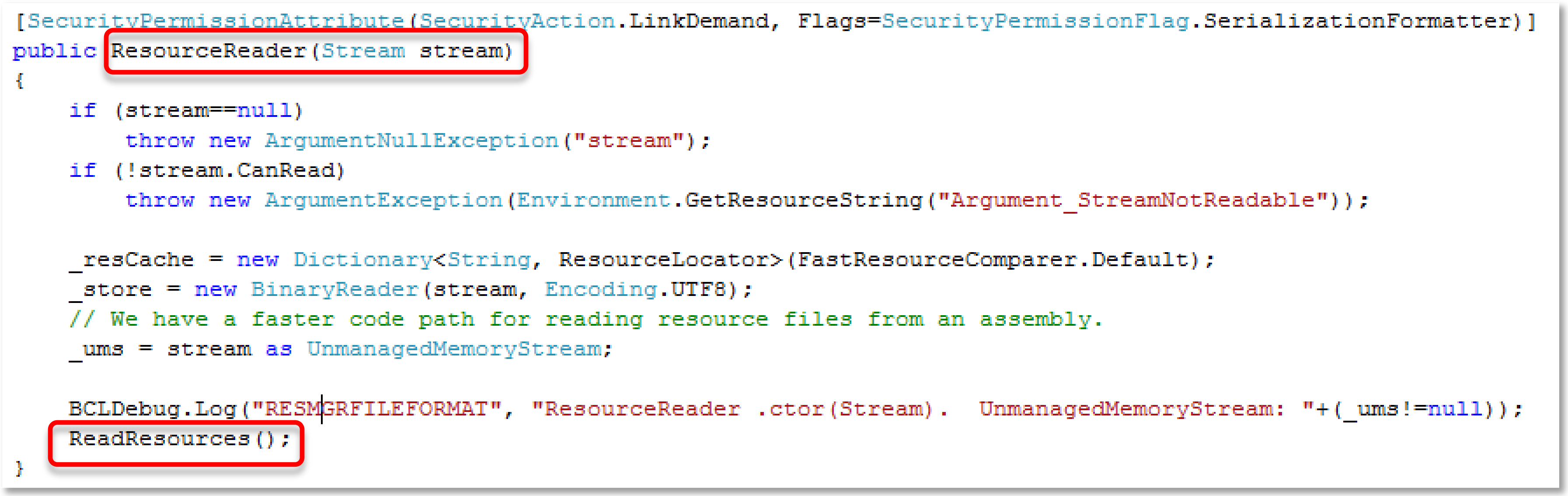
jasonbock
Jason Bock

Microsoft Research

White box testing in practice

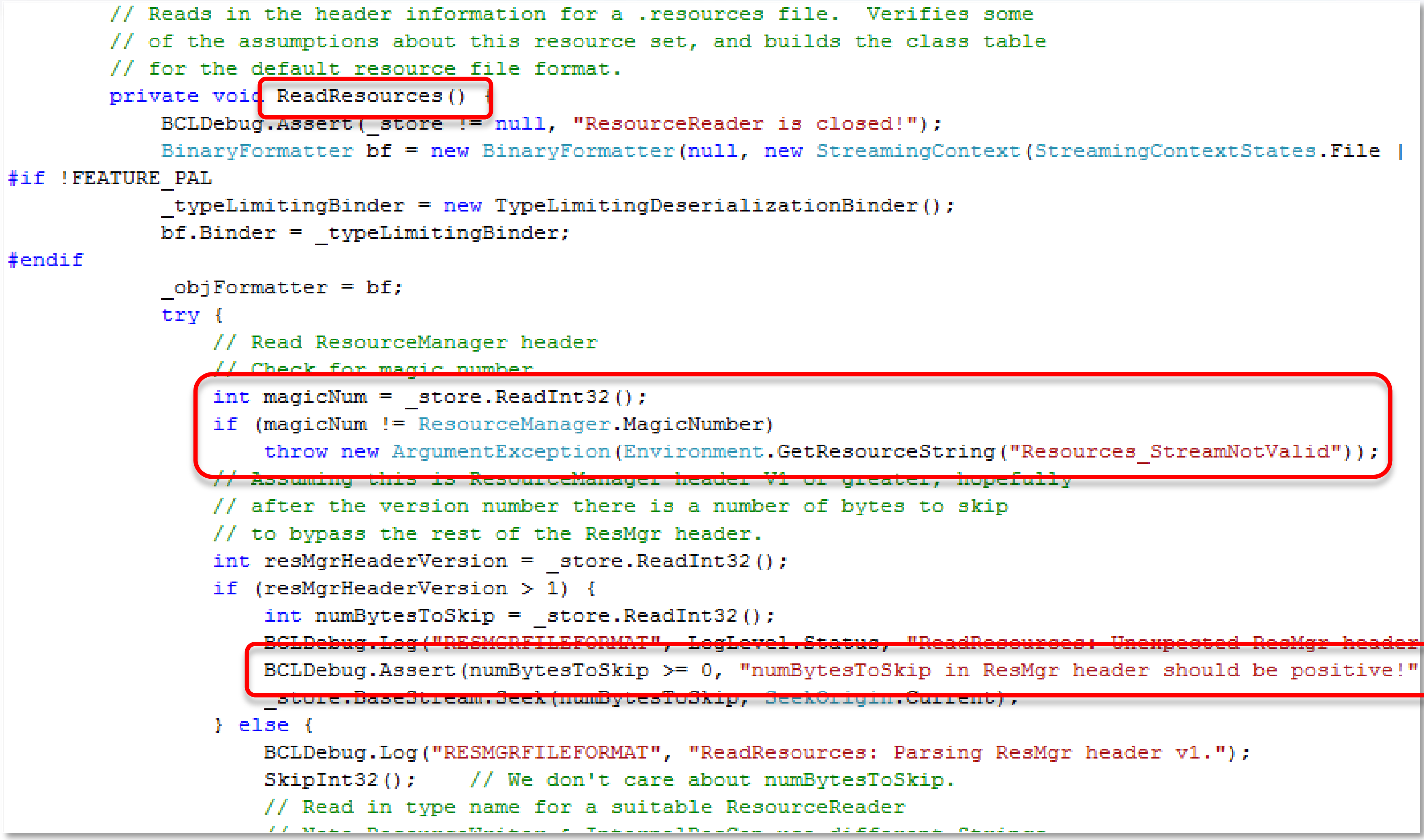
How to test this code?

(Real code from .NET base class libraries.)



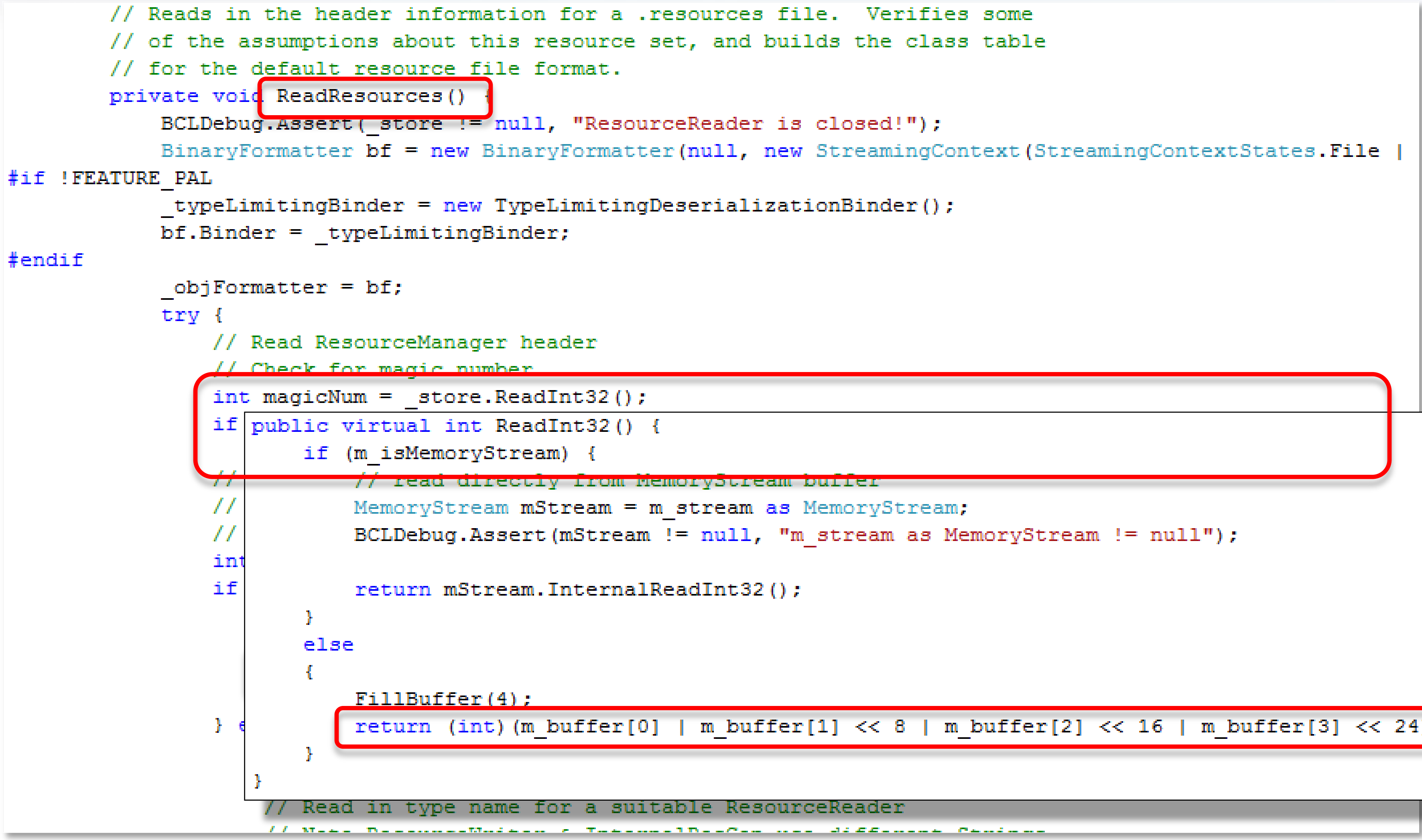
```
[SecurityPermissionAttribute(SecurityAction.LinkDemand, Flags=SecurityPermissionFlag.SerializationFormatter)]
public ResourceReader(Stream stream)
{
    if (stream==null)
        throw new ArgumentNullException("stream");
    if (!stream.CanRead)
        throw new ArgumentException(Environment.GetResourceString("Argument_StreamNotReadable"));

    _resCache = new Dictionary<String, ResourceLocator>(FastResourceComparer.Default);
    _store = new BinaryReader(stream, Encoding.UTF8);
    // We have a faster code path for reading resource files from an assembly.
    _ums = stream as UnmanagedMemoryStream;

    BCLDebug.Log("RESMGRFILEFORMAT", "ResourceReader .ctor(Stream).  UnmanagedMemoryStream: "+(_ums!=null));
    ReadResources();
}
```

White box testing in practice

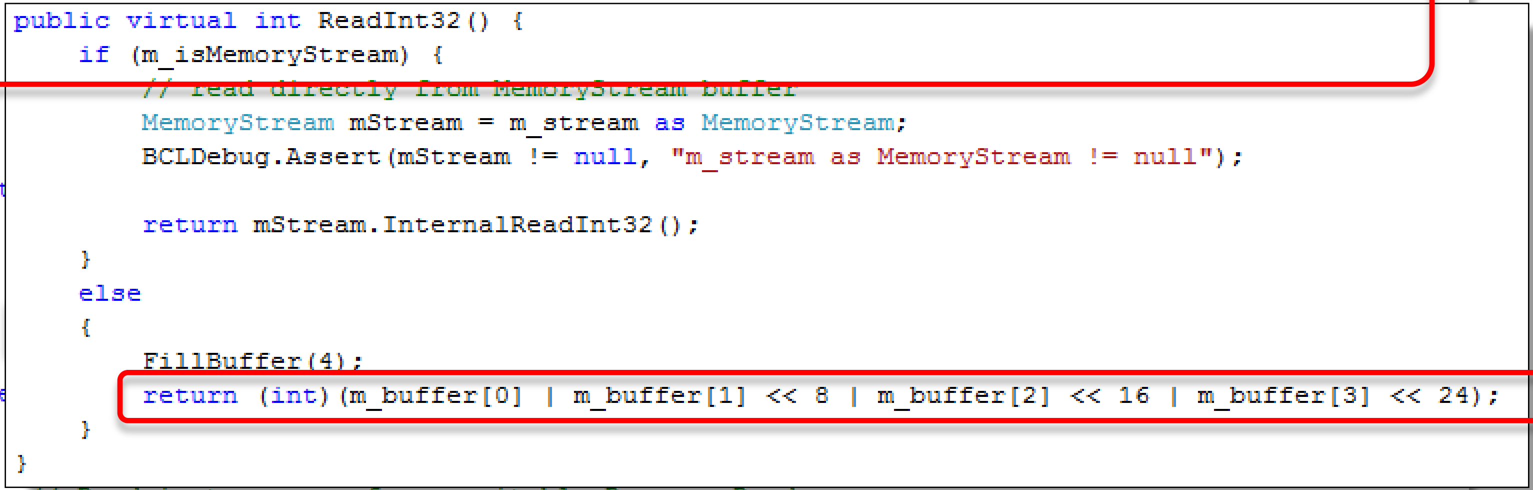


```
        // Reads in the header information for a .resources file.  Verifies some
        // of the assumptions about this resource set, and builds the class table
        // for the default resource file format.
        private void ReadResources() {
            BCLDebug.Assert(_store != null, "ResourceReader is closed!");
            BinaryFormatter bf = new BinaryFormatter(null, new StreamingContext(StreamingContextStates.File |
#if !FEATURE_PAL
            _typeLimitingBinder = new TypeLimitingDeserializationBinder();
            bf.Binder = _typeLimitingBinder;
#endif
            _objFormatter = bf;
            try {
                // Read ResourceManager header
                // Check for magic number
                int magicNum = _store.ReadInt32();
                if (magicNum != ResourceManager.MagicNumber)
                    throw new ArgumentException(Environment.GetResourceString("Resources_StreamNotValid"));
                // Assuming this is ResourceManager header V1 or greater, hopefully
                // after the version number there is a number of bytes to skip
                // to bypass the rest of the ResMgr header.
                int resMgrHeaderVersion = _store.ReadInt32();
                if (resMgrHeaderVersion > 1) {
                    int numBytesToSkip = _store.ReadInt32();
                    BCLDebug.Log("RESMGRFILEFORMAT", LogLevel.Status, "ReadResources: Unexpected ResMgr header
                    BCLDebug.Assert(numBytesToSkip >= 0, "numBytesToSkip in ResMgr header should be positive!"
                    _store.BaseStream.Seek(numBytesToSkip, SeekOrigin.Current);
                } else {
                    BCLDebug.Log("RESMGRFILEFORMAT", "ReadResources: Parsing ResMgr header v1.");
                    SkipInt32();    // We don't care about numBytesToSkip.
                    // Read in type name for a suitable ResourceReader
                    // Note ResourceWriter & InternalResGen use different Strings
```

White box testing in practice

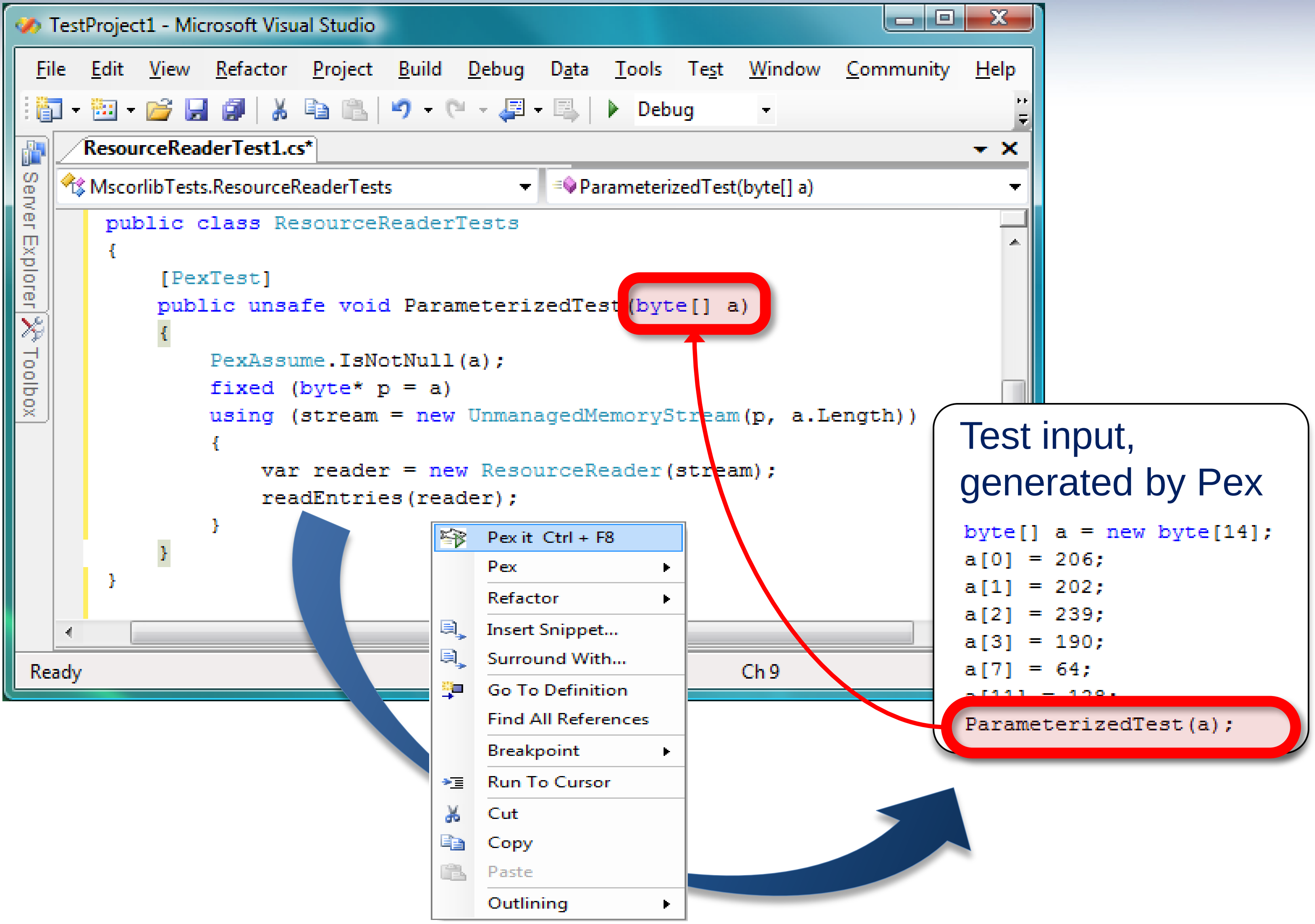


```
        // Reads in the header information for a .resources file.  Verifies some
        // of the assumptions about this resource set, and builds the class table
        // for the default resource file format.
        private void ReadResources() {
            BCLDebug.Assert(_store != null, "ResourceReader is closed!");
            BinaryFormatter bf = new BinaryFormatter(null, new StreamingContext(StreamingContextStates.File |
#if !FEATURE_PAL
            _typeLimitingBinder = new TypeLimitingDeserializationBinder();
            bf.Binder = _typeLimitingBinder;
#endif
            _objFormatter = bf;
            try {
                // Read ResourceManager header
                // Check for magic number
                int magicNum = _store.ReadInt32();
                if
                //
                //
                //
                int
                if



                } e
                    // Read in type name for a suitable ResourceReader
```



```
public virtual int ReadInt32() {
    if (m_isMemoryStream) {
        // read directly from MemoryStream buffer
        MemoryStream mStream = m_stream as MemoryStream;
        BCLDebug.Assert(mStream != null, "m_stream as MemoryStream != null");

        return mStream.InternalReadInt32();
    }
    else
    {
        FillBuffer(4);
        return (int)(m_buffer[0] | m_buffer[1] << 8 | m_buffer[2] << 16 | m_buffer[3] << 24);
    }
}
```

Pex – Test Input Generation

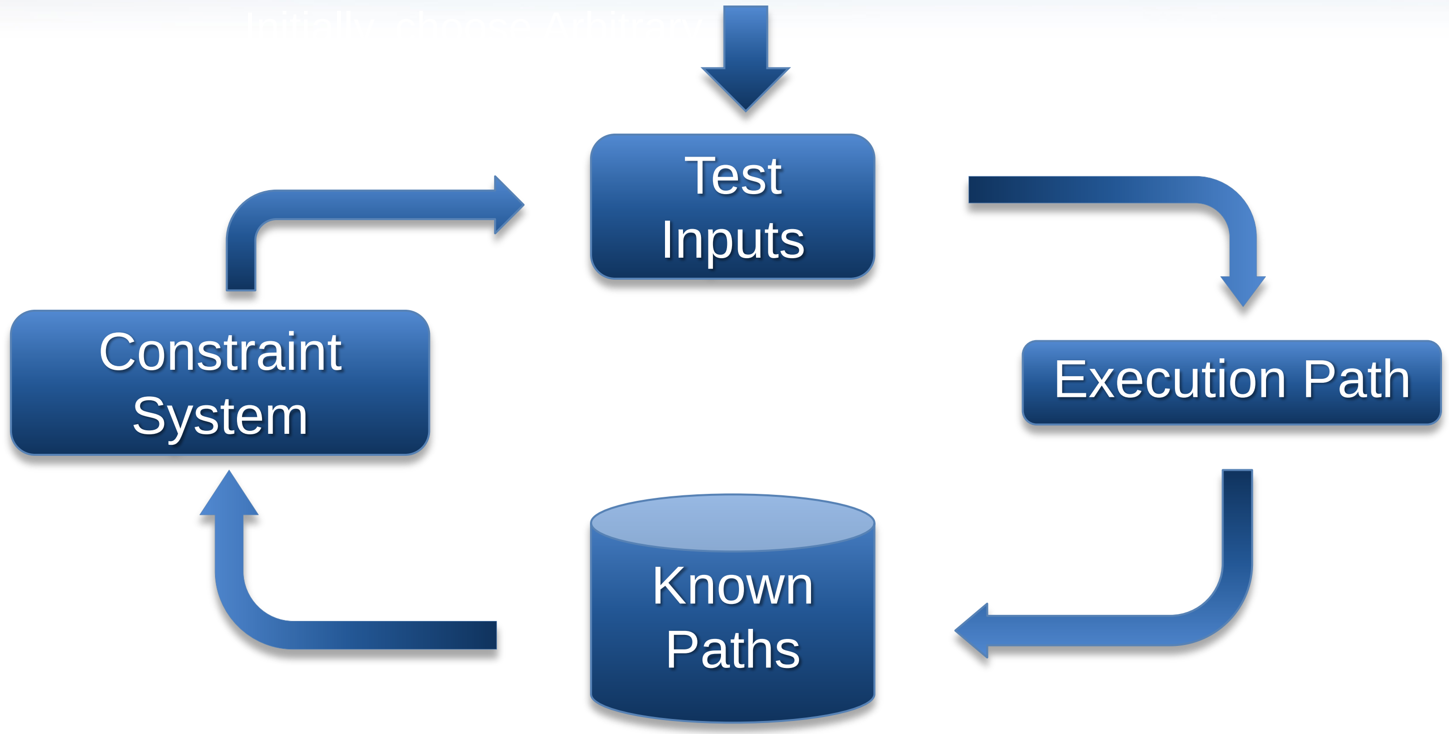


```
public class ResourceReaderTests
{
    [PexTest]
    public unsafe void ParameterizedTest(byte[] a)
    {
        PexAssume.IsNotNull(a);
        fixed (byte* p = a)
        using (stream = new UnmanagedMemoryStream(p, a.Length))
        {
            var reader = new ResourceReader(stream);
            readEntries(reader);
        }
    }
}
```

Test input, generated by Pex

```
byte[] a = new byte[14];
a[0] = 206;
a[1] = 202;
a[2] = 239;
a[3] = 190;
a[7] = 64;
a[11] = 128;
ParameterizedTest(a);
```

Test Input Generation by Dynamic Symbolic Execution



Result: small test suite,
high code coverage

Finds only real bugs
No false warnings

PEX ↔ Z3

Rich Combination — Linear arithmetic — Bitvector — Arrays — Free Functions

Models — Model used as test inputs

∀-Quantifier — Used to model custom theories (e.g., .NET type system)

API — Huge number of small problems. Textual interface is too inefficient.

PEX $\leftrightarrow$ Z3

Rich Combination | Linear arithmetic | Bitvector | Arrays | Free Functions

∀-Quantifier | Used to model custom theories (e.g., .NET type system)

Undecidable (in general)

Microsoft Research

PEX ↔ Z3

Rich Combination | Linear arithmetic | Bitvector | Arrays | Free Functions

∀-Quantifier | Used to model custom theories (e.g., .NET type system)

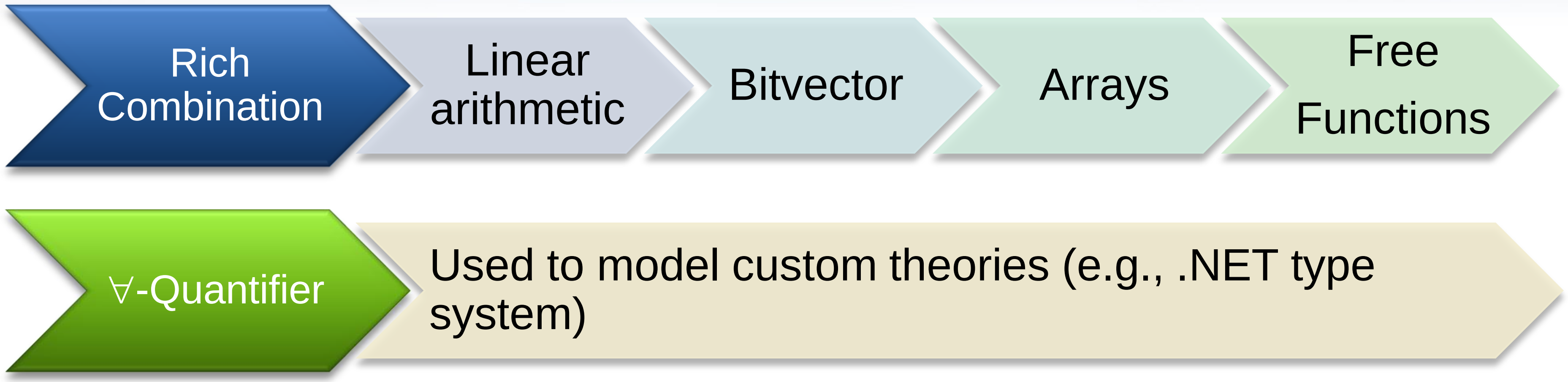
Undecidable (in general)

Solution:

Return "Candidate" Model

Check if trace is valid by executing it

PEX ↔ Z3



Undecidable (in general)

Refined solution:

Support for decidable fragments.

Microsoft Research

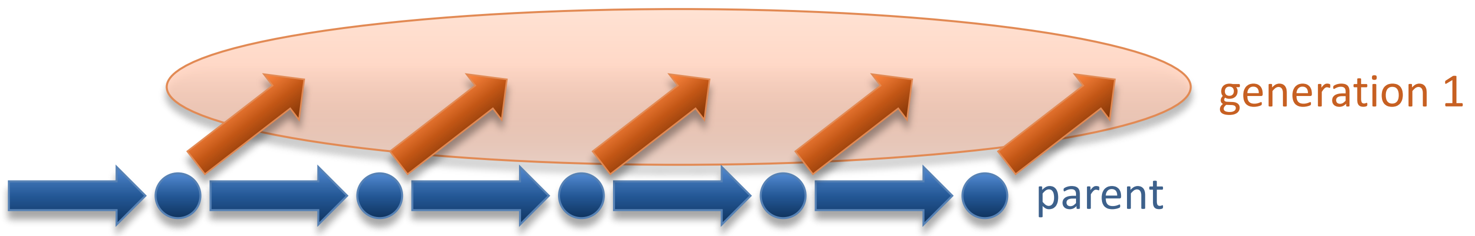
SAGE

- Apply DART to large applications (not units).
- Start with well-formed input (not random).
- Combine with generational search (not DFS).
  - Negate 1-by-1 each constraint in a path constraint.
  - Generate many children for each parent run.

Microsoft Research

SAGE

- Apply DART to large applications (not units).
- Start with well-formed input (not random).
- Combine with generational search (not DFS).
  - Negate 1-by-1 each constraint in a path constraint.
  - Generate many children for each parent run.



Microsoft Research

Zero to Crash in 10 Generations

- Starting with 100 zero bytes …
- SAGE generates a crashing test for Media1 parser

```
00000000h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
00000010h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
00000020h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
00000030h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
00000040h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
00000050h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
00000060h: 00 00 00 00                                     ; ....
```

Generation 0 – seed file

Microsoft Research

Zero to Crash in 10 Generations

- Starting with 100 zero bytes …
- SAGE generates a crashing test for Media1 parser

```
00000000h: 52 49 46 46 3D 00 00 00 ** ** ** 20 00 00 00 00 ; RIFF=...*** ....
00000010h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
00000020h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
00000030h: 00 00 00 00 73 74 72 68 00 00 00 00 76 69 64 73 ; ....strh....vids
00000040h: 00 00 00 00 73 74 72 66 B2 75 76 3A 28 00 00 00 ; ....strf²uv:(...
00000050h: 00 00 00 00 00 00 00 00 00 00 00 00 01 00 00 00 ; ................
00000060h: 00 00 00 00                                     ; ....
```

Generation 10 – CRASH

Microsoft Research

SAGE (cont.)

- SAGE is very effective at finding bugs.
- Works on large applications.
- Fully automated
- Easy to deploy (x86 analysis – any language)
- Used in various groups inside Microsoft
- Powered by Z3.

SAGE $\leftrightarrow$ Z3

- Formulas are usually big conjunctions.
- SAGE uses only the bitvector and array theories.
- Pre-processing step has a huge performance impact.
  - Eliminate variables.
  - Simplify formulas.
- Early unsat detection.

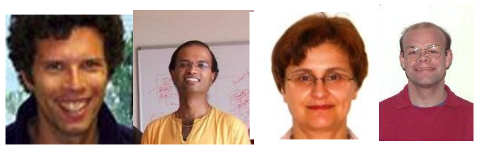
Microsoft Research

Static Driver Verifier

SLAM

Static Driver Verifier

- Z3 is part of SDV 2.0 (Windows 7)
- It is used for:
  - Predicate abstraction (c2bp)
  - Counter-example refinement (newton)



Ella Bounimova, Vlad Levin, Jakob Lichtenberg,
Tom Ball, Sriram Rajamani, Byron Cook

Overview

- *http://research.microsoft.com/slam/*
- *SLAM/SDV* is a software model checker.
- Application domain: *device drivers*.
- Architecture:

  c2bp C program → boolean program (*predicate abstraction).*

  bebop Model checker for boolean programs.

  newton Model refinement (check for path feasibility)
- SMT solvers are used to perform predicate abstraction and to check path feasibility.
- c2bp makes several calls to the SMT solver. The formulas are relatively small.

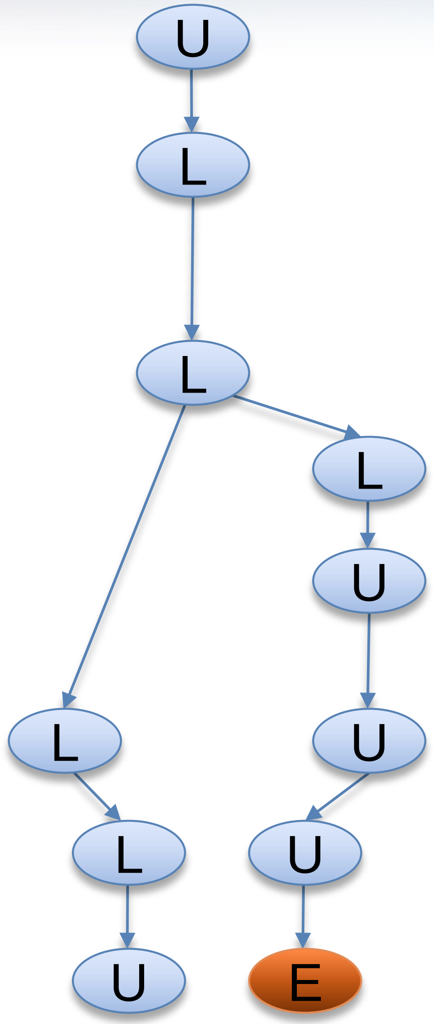
Example

Do this code obey the looking rule?

```
do {
   KeAcquireSpinLock();

   nPacketsOld = nPackets;

   if(request){
        request = request->Next;
        KeReleaseSpinLock();
        nPackets++;
   }
} while (nPackets != nPacketsOld);

KeReleaseSpinLock();
```

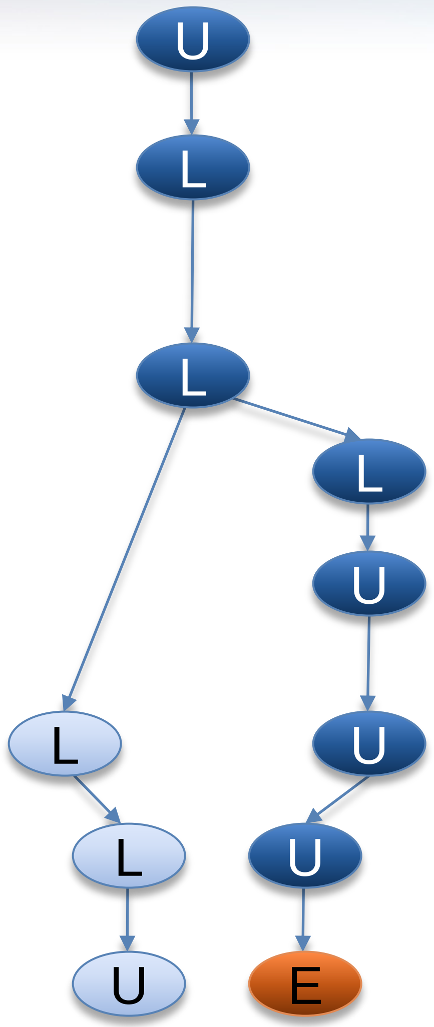
Example

Model checking
Boolean program



```
do {
   KeAcquireSpinLock();



   if(*){

      KeReleaseSpinLock();

   }
} while (*);

KeReleaseSpinLock();
```

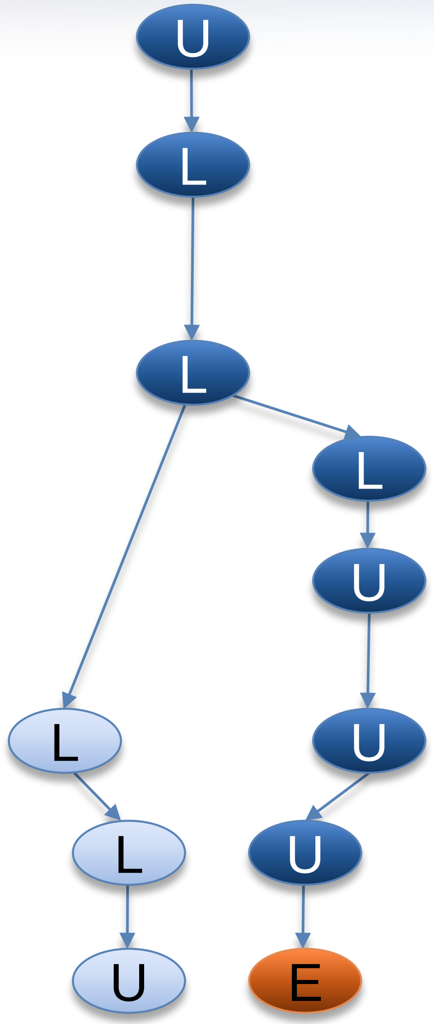
Example



```
do {
   KeAcquireSpinLock();

   nPacketsOld = nPackets;

   if(request){
        request = request->Next;
        KeReleaseSpinLock();
        nPackets++;
   }
} while (nPackets != nPacketsOld);

KeReleaseSpinLock();
```

Example

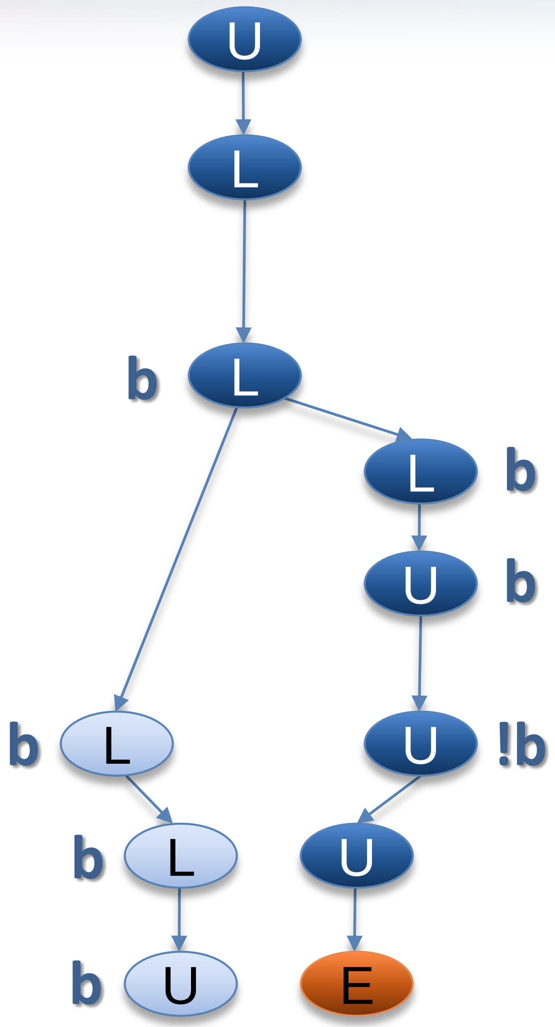
Add new predicate to
Boolean program
b: (nPacketsOld == nPackets)



```
do {
   KeAcquireSpinLock();

   nPacketsOld = nPackets;
     b = true;
   if(request){
        request = request->Next;
        KeReleaseSpinLock();
        nPackets++;
   }        b = b ? false : *;
} while (nPackets != nPacketsOld);
              !b
KeReleaseSpinLock();
```

Example

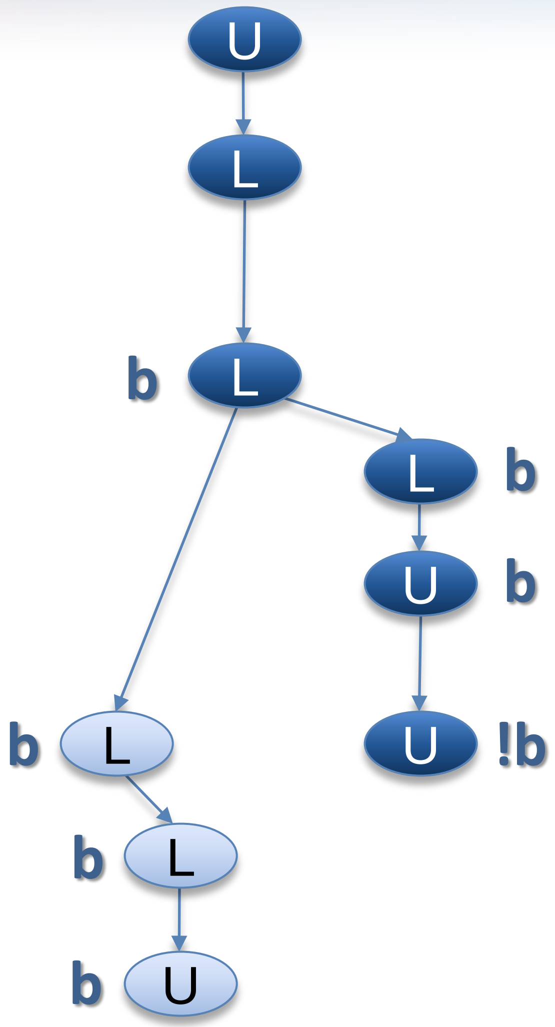
Model Checking
Refined Program
b: (nPacketsOld == nPackets)



```
do {
   KeAcquireSpinLock();

   b = true;

   if(*){

      KeReleaseSpinLock();
      b = b ? false : *;
   }
} while (!b);

KeReleaseSpinLock();
```

Example

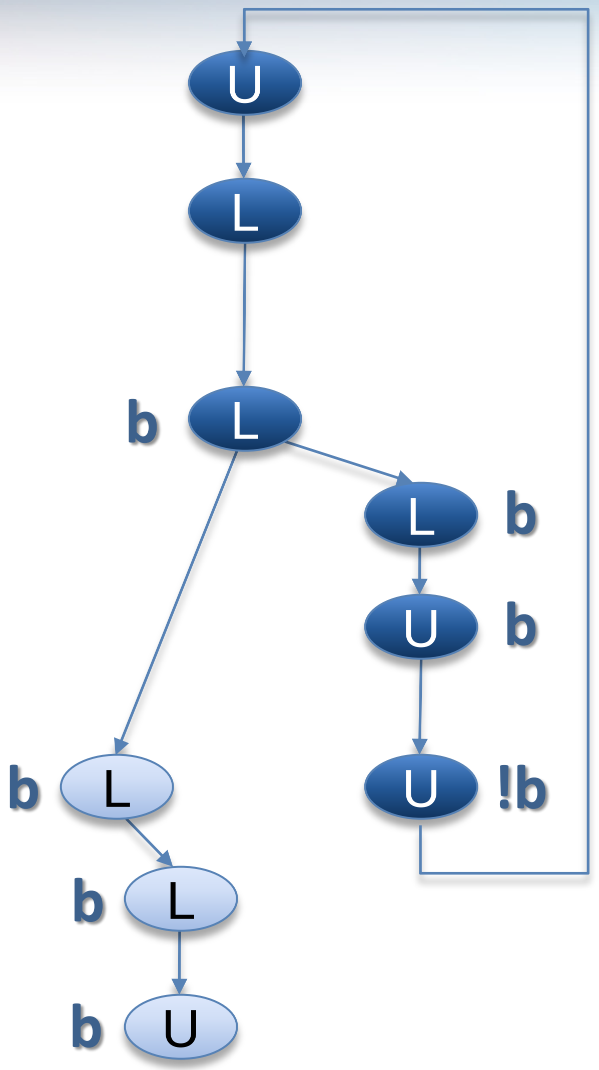
Model Checking
Refined Program
b: (nPacketsOld == nPackets)



```
do {
   KeAcquireSpinLock();

   b = true;

   if(*){

      KeReleaseSpinLock();
      b = b ? false : *;
   }
} while (!b);

KeReleaseSpinLock();
```

Example

Model Checking
Refined Program
b: (nPacketsOld == nPackets)



```
do {
   KeAcquireSpinLock();

   b = true;

   if(*){

      KeReleaseSpinLock();
      b = b ? false : *;
   }
} while (!b);

KeReleaseSpinLock();
```

Observations about SLAM

- Automatic discovery of invariants
  - driven by property and a finite set of (false) execution paths
  - predicates are ***not*** invariants, but *observations*
  - abstraction + model checking computes inductive invariants (Boolean combinations of observations)
- A hybrid dynamic/static analysis
  - newton executes path through C code symbolically
  - c2bp+bebop explore all paths through abstraction
- A new form of program slicing
  - program code and data not relevant to property are dropped
  - non-determinism allows slices to have more behaviors

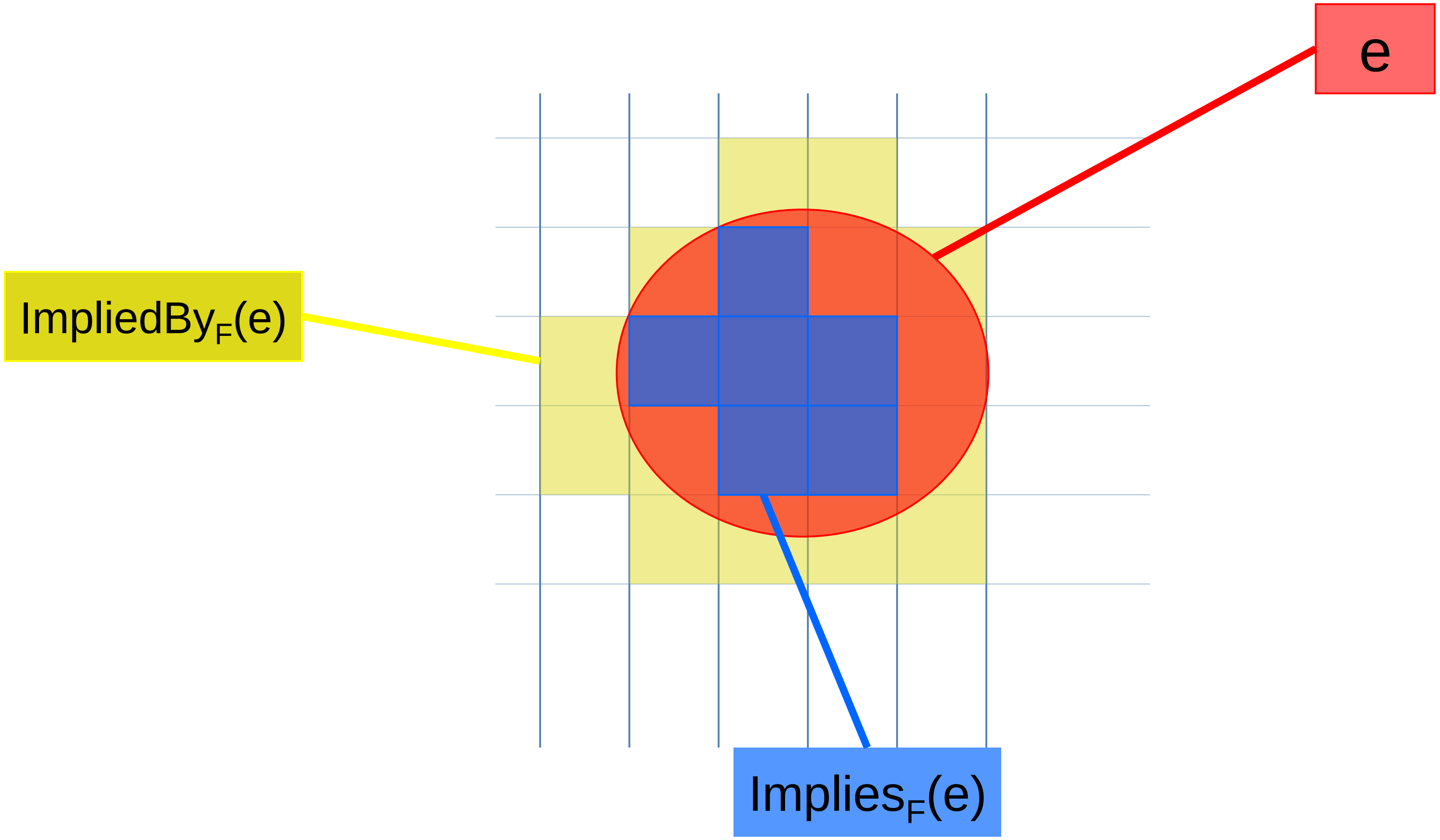
Predicate Abstraction: *c2bp*

- **Given** a C program P and $F = \{p_1, \ldots , p_n\}$.
- **Produce** a Boolean program $B(P, F)$
  - Same control flow structure as P.
  - Boolean variables $\{b_1, \ldots , b_n\}$ to match $\{p_1, \ldots , p_n\}$.
  - Properties true in $B(P, F)$ are true in P.
- Each p_i is a pure Boolean expression.
- Each p_i represents set of states for which p_i is true.
- Performs modular abstraction.

Abstracting Expressions via *F*

- *$Implies_F$ (e)*
  - Best Boolean function over *F* that implies *e.*
- *$ImpliedBy_F$ (e)*
  - Best Boolean function over *F* that is implied by *e.*
  - *$ImpliedBy_F$ (e) = not $Implies_F$ (not e)*

$\text{Implies}_F(e)$ and $\text{ImpliedBy}_F(e)$



Computing *Implies*$_F$*(e)*

- minterm m = l_1 and ... and l_n, where $l_i = p_i$, or $l_i = not\ p_i$.
- *Implies*$_F$*(e)*: disjunction of all minterms that imply *e.*
- Naive approach
  - Generate all 2^n possible minterms.
  - For each minterm m, use SMT solver to check validity of *m* implies *e*.
- Many possible optimizations

Computing *Implies$_F$(e)*

- F = { x < y, x = 2}
- *e* : y > 1
- Minterms over F
  - !x<y, !x=2 implies y>1
  - x<y, !x=2 implies y>1
  - !x<y, x=2 implies y>1
  - x<y, x=2 implies y>1

Computing *Implies$_F$(e)*

- F = { x < y, x = 2}
- *e* : y > 1
- Minterms over F
  - !x<y, !x=2 implies y>1 ⊘
  - x<y, !x=2 implies y>1 ⊘
  - !x<y, x=2 implies y>1 ⊘
  - x<y, x=2 implies y>1 ✓

Computing *Implies$_F$(e)*

- F = { x < y, x = 2}
- *e* : y > 1
- Minterms over F
  - !x<y, !x=2 implies y>1 🚫
  - x<y, !x=2 implies y>1 🚫
  - !x<y, x=2 implies y>1 🚫
  - x<y, x=2 implies y>1 ✓

Implies$_F$(y>1) = x<y $\wedge$ x=2

Computing *Implies$_F$(e)*

- F = { x < y, x = 2}
- *e* : y > 1
- Minterms over F
  - !x<y, !x=2 implies y>1 🚫
  - x<y, !x=2 implies y>1 🚫
  - !x<y, x=2 implies y>1 🚫
  - x<y, x=2 implies y>1 ✓

Implies$_F$(y>1) = $b_1 \wedge b_2$

Newton

- Given an error path *p* in the Boolean program *B*.
- Is *p* a feasible path of the corresponding C program?
  - Yes: found a bug.
  - No: find predicates that explain the infeasibility.
- Execute path symbolically.
- Check conditions for inconsistency using SMT solver.

Z3 & Static Driver Verifier

- All-SAT
  - Better (more precise) Predicate Abstraction
- Unsatisfiable cores
  - Why the abstract path is not feasible?
  - Fast Predicate Abstraction

Bit-precise Scalable Static Analysis

PREfix [Moy, Bjorner, Sielaff 2009]

What is wrong here?

```
int binary_search(int[] arr, int low,
                           int high, int key)
while (low <= high)
   {
       // Find middle value
       int mid = (low + high) / 2;
       int val = arr[mid];
       if (val == key) return mid;
       if (val < key) low = mid+1;
       else high = mid-1;
   }
   return -1;
```

Package: java.util.Arrays
Function: binary_search

```
void itoa(int n, char* s) {
     if (n < 0) {
        *s++ = '-';
        n = -n;
     }
     // Add digits to s
     ….
```

Book: Kernighan and Ritchie
Function: itoa (integer to ascii)

What is wrong here?

3(INT_MAX+1)/4 +
(INT_MAX+1)/4
= INT_MIN

```
int binary_se

while (low <= hig
   {
     // Find middle value
     int mid = (low + high) / 2;
     int val = arr[mid];
     if (val == key) return mid;
     if (val < key) low = mid+1;
     else high = mid-1;
   }
   return -1;
```

Package: java.util.Arrays
Function: binary_search

```
id itoa(int n, char* s) {
  if (n < 0) {
     *s++ = '-';
     n = -n;
  }
  // Add digits to s
  ….
```

Book: Kernighan and Ritchie
Function: itoa (integer to ascii)

What is wrong here?

-INT_MIN=
INT_MIN

3(INT_MAX+1)/4 +
(INT_MAX+1)/4
= INT_MIN

```
int binary_se

while (low <= hig
    {
        // Find middle value
        int mid = (low + high) / 2;
        int val = arr[mid];
        if (val == key) return mid;
        if (val < key) low = mid+1;
        else high = mid-1;
     }
     return -1;
```

```
id itoa(int n,    ar* s) {
  if (n < 0) {
      *s++ = '-';
      n = -n;
  }
  // Add digits to s
  ….
```

Package: java.util.Arrays
Function: binary_search

Book: Kernighan and Ritchie
Function: itoa (integer to ascii)

The PREfix Static Analysis Engine

```
int init_name(char **outname, uint n)
{
   if (n == 0) return 0;
   else if (n > UINT16_MAX) exit(1);
   else if ((*outname = malloc(n)) == NULL) {
      return 0xC0000095; // NT_STATUS_NO_MEM;
   }
   return 0;
}

int get_name(char* dst, uint size)
{
   char* name;
   int status = 0;
   status = init_name(&name, size);
   if (status != 0) {
      goto error;
   }
   strcpy(dst, name);
error:
   return status;
}
```

C/C++ functions

The PREfix Static Analysis Engine

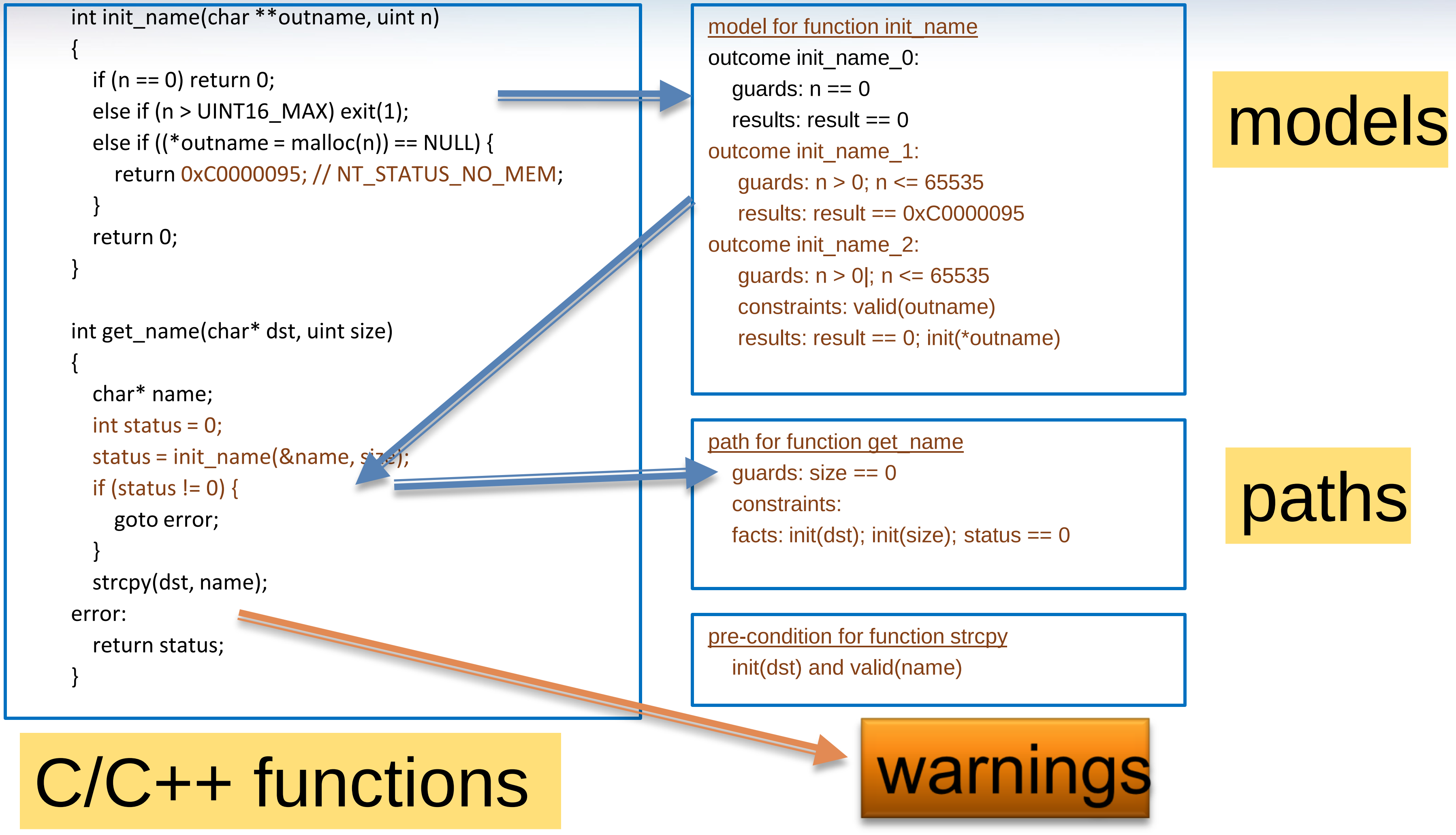
```
int init_name(char **outname, uint n)
{
   if (n == 0) return 0;
   else if (n > UINT16_MAX) exit(1);
   else if ((*outname = malloc(n)) == NULL) {
      return 0xC0000095; // NT_STATUS_NO_MEM;
   }
   return 0;
}

int get_name(char* dst, uint size)
{
   char* name;
   int status = 0;
   status = init_name(&name, size);
   if (status != 0) {
      goto error;
   }
   strcpy(dst, name);
error:
   return status;
}
```

```
model for function init_name
outcome init_name_0:
   guards: n == 0
   results: result == 0
outcome init_name_1:
    guards: n > 0; n <= 65535
    results: result == 0xC0000095
outcome init_name_2:
    guards: n > 0|; n <= 65535
    constraints: valid(outname)
    results: result == 0; init(*outname)
```

models

C/C++ functions

The PREfix Static Analysis Engine



```
int init_name(char **outname, uint n)
{
   if (n == 0) return 0;
   else if (n > UINT16_MAX) exit(1);
   else if ((*outname = malloc(n)) == NULL) {
      return 0xC0000095; // NT_STATUS_NO_MEM;
   }
   return 0;
}

int get_name(char* dst, uint size)
{
   char* name;
   int status = 0;
   status = init_name(&name, size);
   if (status != 0) {
      goto error;
   }
   strcpy(dst, name);
error:
   return status;
}
```

model for function init_name
outcome init_name_0:
   guards: n == 0
   results: result == 0
outcome init_name_1:
   guards: n > 0; n <= 65535
   results: result == 0xC0000095
outcome init_name_2:
   guards: n > 0|; n <= 65535
   constraints: valid(outname)
   results: result == 0; init(*outname)

path for function get_name
   guards: size == 0
   constraints:
   facts: init(dst); init(size); status == 0

pre-condition for function strcpy
   init(dst) and valid(name)

models

paths

C/C++ functions

warnings

Overflow on unsigned addition

m_nSize == m_nMaxSize == UINT_MAX

```
iElement = m_nSize;
if( iElement >= m_nMaxSize )
{
    bool bSuccess = GrowBuffer( iElement+1 );
    …
}
::new( m_pData+iElement ) E( element );
m_nSize++;
```

iElement + 1 == 0

Write in unallocated memory

Code was written for address space < 4GB

Using an overflown value as allocation size

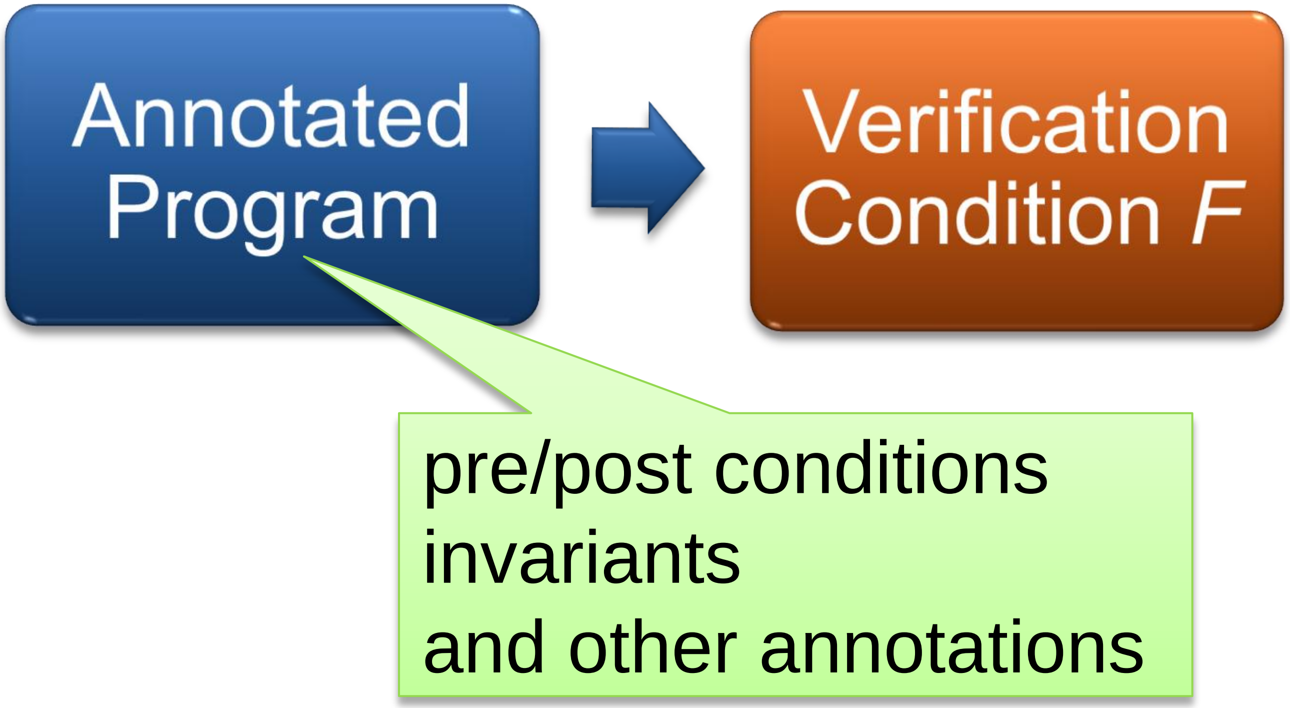
```
ULONG AllocationSize;
while (CurrentBuffer != NULL) {
        if (NumberOfBuffers > MAX_ULONG / sizeof(MYBUFFER)) {
                return NULL;
         }
         NumberOfBuffers++;
         CurrentBuffer = CurrentBuffer->NextBuffer;
}
AllocationSize = sizeof(MYBUFFER)*NumberOfBuffers;
UserBuffersHead = malloc(AllocationSize);
```

Overflow check

Increment and exit from loop

Possible overflow

Verifying Compilers



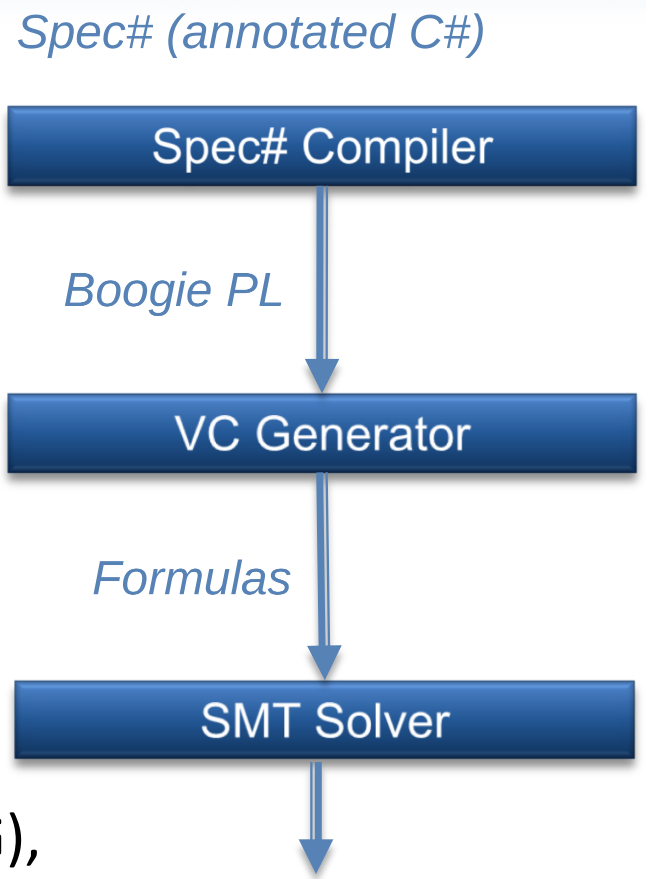
pre/post conditions
invariants
and other annotations

Annotations: Example

```
class C {
    private int a, z;
    invariant z > 0

    public void M()
        requires a != 0
        {
            z = 100/a;
        }
}
```

Spec# Approach for a Verifying Compiler

- *Source Language*
  - C# + goodies = Spec#
- *Specifications*
  - method contracts,
  - invariants,
  - field and type annotations.
- *Program Logic:*
  - *Dijkstra's weakest preconditions.*
- *Automatic Verification*
  - type checking,
  - verification condition generation (VCG),
  - **SMT**



Microsoft Research

Command language

- x := E
  - x := x + 1
  - x := 10
- havoc x
- S ; T
- assert P
- assume P
- S □ T

Reasoning about execution traces

- Hoare triple { P } S { Q } says that

  every terminating execution trace of S that starts in a state satisfying P

  - does not go wrong, and
  - terminates in a state satisfying Q

Reasoning about execution traces

- Hoare triple { P } S { Q } says that every terminating execution trace of S that starts in a state satisfying P
  - does not go wrong, and
  - terminates in a state satisfying Q
- Given S and Q, what is the weakest P’ satisfying {P’} S {Q} ?
  - P' is called the *weakest precondition* of S with respect to Q, written wp(S, Q)
  - to check {P} S {Q}, check P $\Rightarrow$ P’

Weakest preconditions

$\mathrm{wp}(\ x := E,\ \ Q\ ) = \quad Q[\ E\ /\ x\ ]$

$\mathrm{wp}(\ \textbf{havoc}\ x,\ \ Q\ ) = \quad (\forall x \bullet\ \ Q\ )$

$\mathrm{wp}(\ \textbf{assert}\ P,\ \ Q\ ) = \quad P \land Q$

$\mathrm{wp}(\ \textbf{assume}\ P,\ \ Q\ ) = \quad P \Rightarrow Q$

$\mathrm{wp}(\ S\ ;\ T,\ \ Q\ ) = \quad \mathrm{wp}(\ S,\ \ \mathrm{wp}(\ T,\ Q\ ))$

$\mathrm{wp}(\ S \ \square\ T,\ \ Q\ ) = \quad \mathrm{wp}(\ S,\ Q\ ) \land \mathrm{wp}(\ T,\ Q\ )$

Structured if statement

```
if E then S else T end  =

        assume E;  S
        □
        assume ¬E;  T
```

While loop with loop invariant

```
while E
    invariant J
do
    S
end
```

where x denotes the assignment targets of S

check that the loop invariant holds initially

```
=   assert J;
    havoc x;  assume J;
    (   assume E;  S;  assert J;  assume false
    □   assume ¬E
    )
```

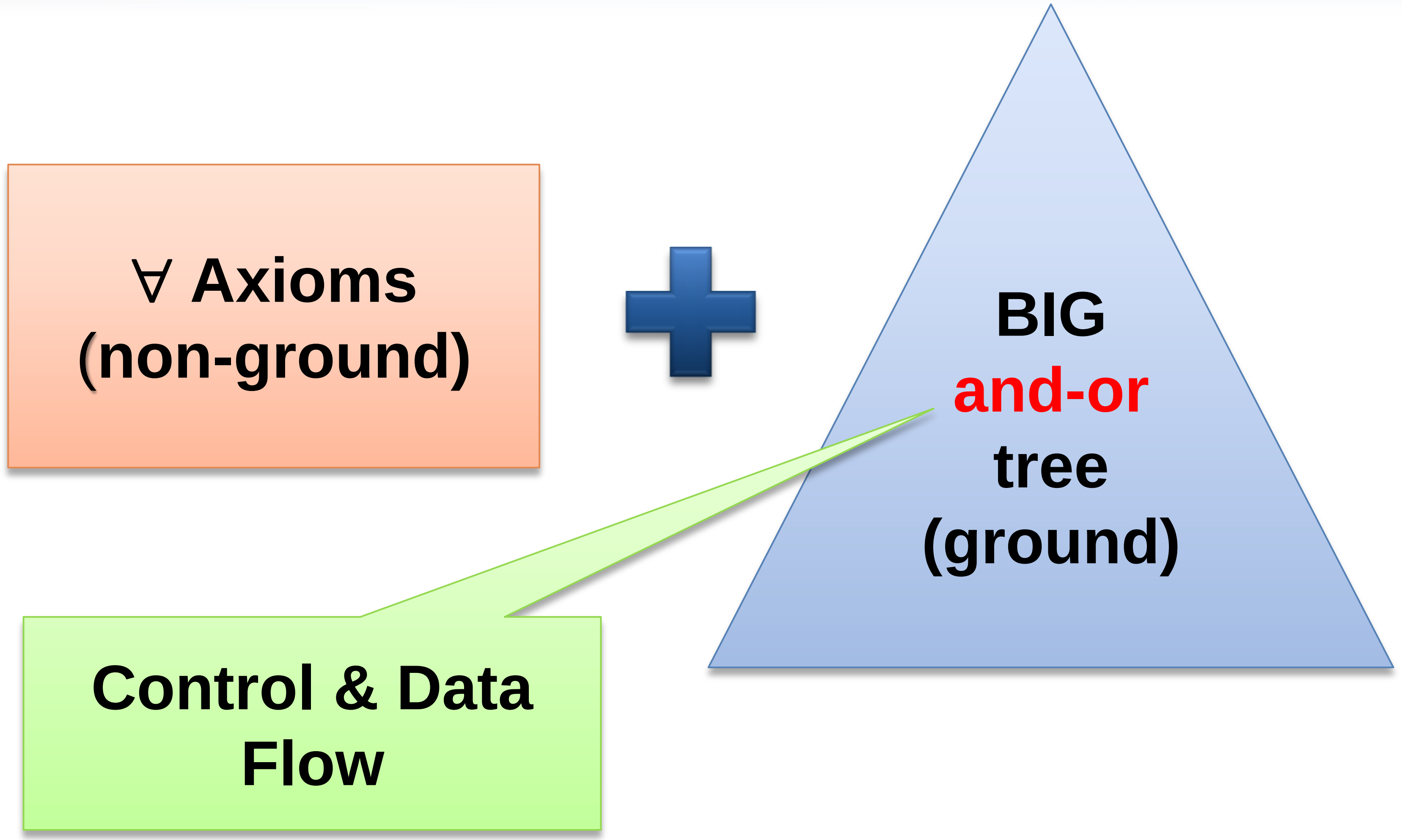
"fast forward" to an arbitrary iteration of the loop

check that the loop invariant is maintained by the loop body

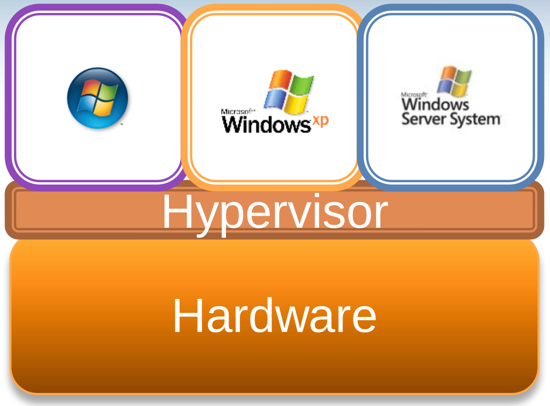
Spec# Chunker.NextChunk translation

```
procedure Chunker.NextChunk(this: ref where $IsNotNull(this, Chunker)) returns ($result: ref where $IsNotNull($result, System.String));
  // in-parameter:  target object
  free requires $Heap[this, $allocated];
  requires ($Heap[this, $ownerFrame] == $PeerGroupPlaceholder || !($Heap[$Heap[this, $ownerRef], $inv] <: $Heap[this, $ownerFrame]) ||
    $Heap[$Heap[this, $ownerRef], $localinv] == $BaseClass($Heap[this, $ownerFrame])) && (forall $pc: ref :: $pc != null && $Heap[$pc, $allocated]
    && $Heap[$pc, $ownerRef] == $Heap[this, $ownerRef] && $Heap[$pc, $ownerFrame] == $Heap[this, $ownerFrame] ==> $Heap[$pc, $inv] ==
    $typeof($pc) && $Heap[$pc, $localinv] == $typeof($pc));
  // out-parameter:  return value
  free ensures $Heap[$result, $allocated];
  ensures ($Heap[$result, $ownerFrame] == $PeerGroupPlaceholder || !($Heap[$Heap[$result, $ownerRef], $inv] <: $Heap[$result, $ownerFrame]) ||
    $Heap[$Heap[$result, $ownerRef], $localinv] == $BaseClass($Heap[$result, $ownerFrame])) && (forall $pc: ref :: $pc != null && $Heap[$pc,
    $allocated] && $Heap[$pc, $ownerRef] == $Heap[$result, $ownerRef] && $Heap[$pc, $ownerFrame] == $Heap[$result, $ownerFrame] ==>
    $Heap[$pc, $inv] == $typeof($pc) && $Heap[$pc, $localinv] == $typeof($pc));
  // user-declared postconditions
  ensures $StringLength($result) <= $Heap[this, Chunker.ChunkSize];
  // frame condition
  modifies $Heap;
  free ensures (forall $o: ref, $f: name :: { $Heap[$o, $f] } $f != $inv && $f != $localinv && $f != $FirstConsistentOwner && (!IsStaticField($f) ||
    !IsDirectlyModifiableField($f)) && $o != null && old($Heap)[$o, $allocated] && (old($Heap)[$o, $ownerFrame] == $PeerGroupPlaceholder ||
    !(old($Heap)[old($Heap)[$o, $ownerRef], $inv] <: old($Heap)[$o, $ownerFrame]) || old($Heap)[old($Heap)[$o, $ownerRef], $localinv] ==
    $BaseClass(old($Heap)[$o, $ownerFrame])) && old($o != this || !(Chunker <: DeclType($f)) || !$IncludedInModifiesStar($f)) && old($o != this || $f
    != $exposeVersion) ==> old($Heap)[$o, $f] == $Heap[$o, $f]);
  // boilerplate
  free requires $BeingConstructed == null;
  free ensures (forall $o: ref :: { $Heap[$o, $localinv] } { $Heap[$o, $inv] } $o != null && !old($Heap)[$o, $allocated] && $Heap[$o, $allocated] ==>
    $Heap[$o, $inv] == $typeof($o) && $Heap[$o, $localinv] == $typeof($o));
  free ensures (forall $o: ref :: { $Heap[$o, $FirstConsistentOwner] } old($Heap)[old($Heap)[$o, $FirstConsistentOwner], $exposeVersion] ==
    $Heap[old($Heap)[$o, $FirstConsistentOwner], $exposeVersion] ==> old($Heap)[$o, $FirstConsistentOwner] == $Heap[$o,
    $FirstConsistentOwner]);
  free ensures (forall $o: ref :: { $Heap[$o, $localinv] } { $Heap[$o, $inv] } old($Heap)[$o, $allocated] ==> old($Heap)[$o, $inv] == $Heap[$o, $inv] &&
    old($Heap)[$o, $localinv] == $Heap[$o, $localinv]);
  free ensures (forall $o: ref :: { $Heap[$o, $allocated] } old($Heap)[$o, $allocated] ==> $Heap[$o, $allocated]) && (forall $ot: ref :: { $Heap[$ot,
    $ownerFrame] } { $Heap[$ot, $ownerRef] } old($Heap)[$ot, $allocated] && old($Heap)[$ot, $ownerFrame] != $PeerGroupPlaceholder ==>
    old($Heap)[$ot, $ownerRef] == $Heap[$ot, $ownerRef] && old($Heap)[$ot, $ownerFrame] == $Heap[$ot, $ownerFrame]) &&
    old($Heap)[$BeingConstructed, $NonNullFieldsAreInitialized] == $Heap[$BeingConstructed, $NonNullFieldsAreInitialized];
```

Verification conditions: Structure



Hypervisor: A Manhattan Project

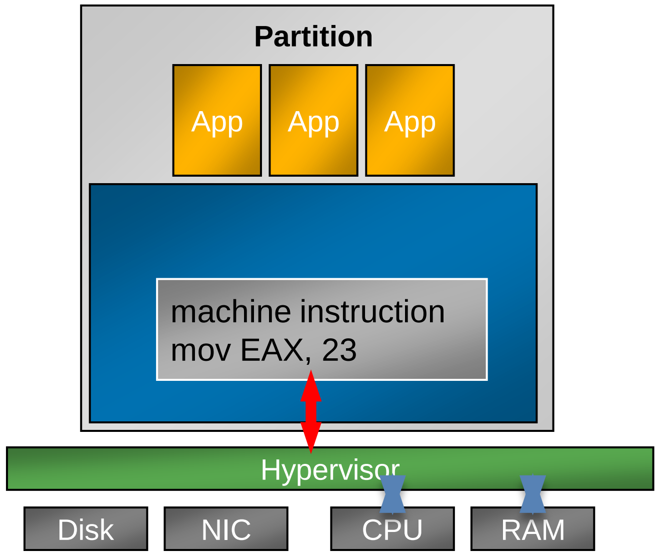


- **Meta OS**: small layer of software between hardware and OS
- **Mini**: 100K lines of non-trivial concurrent systems C code
- **Critical:** must provide functional resource abstraction
- **Trusted**: a verification grand challenge

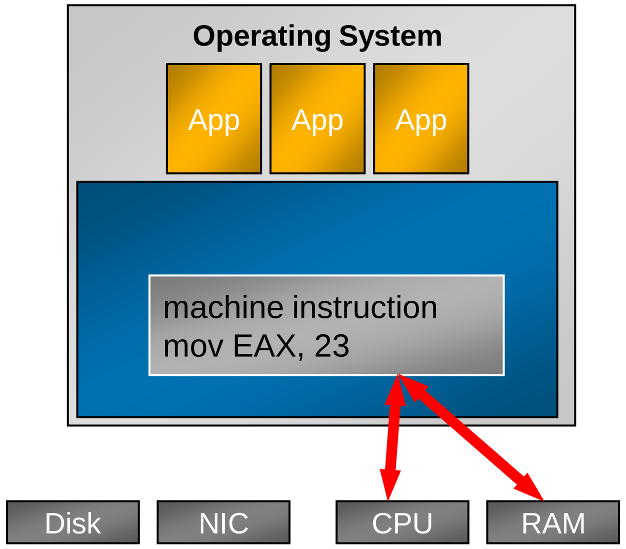
HV Correctness: Simulation

A partition cannot distinguish (with some exceptions) whether a machine instruction is executed

a) through the HV OR b) directly on a processor



≈



Hypervisor Implementation

- real code, as shipped with Windows Server 2008
- ca. 100 000 lines of C, 5 000 lines of x64 assembly
- concurrency
  - spin locks, r/w locks, rundowns, turnstiles
  - lock-free accesses to volatile data and hardware covered by implicit protocols
- scheduler, memory allocator, etc.
- access to hardware registers (memory management, virtualization support)

Hypervisor Verification (2007 – 2010)

Partners:

- European Microsoft Innovation Center
- Microsoft Research
- Microsoft’s Windows Div.
- Universität des Saarlandes

co-funded by the German Ministry of Education and Research

http://www.verisoftxt.de

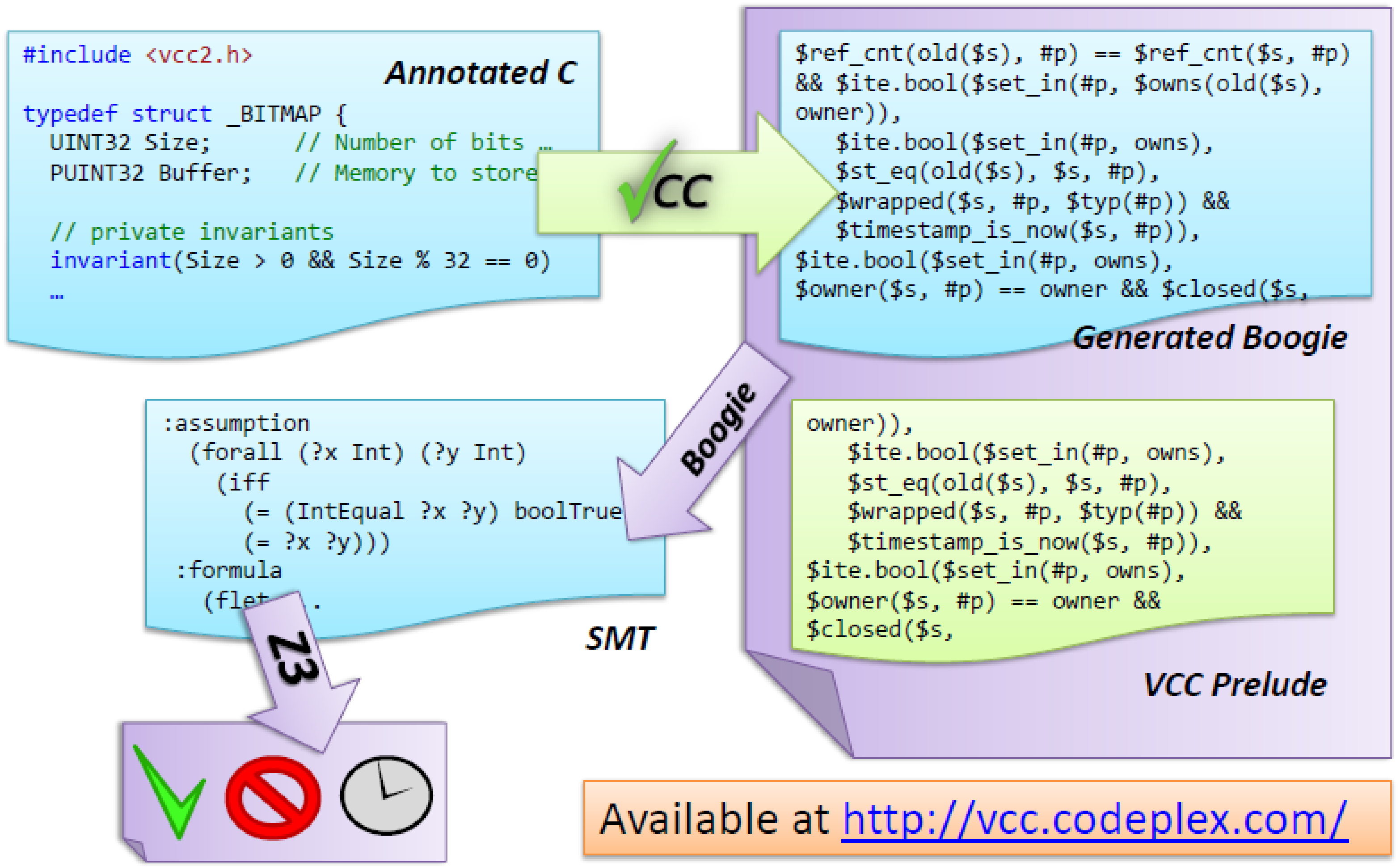
Challenges for Verification of Concurrent C

1. **Memory model** that is adequate and efficient to reason about
2. **Modular reasoning** about concurrent code
3. **Invariants** for (large and complex) C data structures
4. Huge verification conditions to be proven **automatically**
5. “Live” specifications that **evolve with the code**

The Microsoft Verifying C Compiler (VCC)

- Source Language
  - ANSI C +
  - Design-by-Contract Annotations +
  - Ghost state +
  - Theories +
  - Metadata Annotations
- Program Logic
  - Dijkstra's weakest preconditions
- Automatic Verification
  - verification condition generation (VCG)
  - automatic theorem proving (SMT)

VCC Architecture



```
#include <vcc2.h>

typedef struct _BITMAP {
  UINT32 Size;       // Number of bits …
  PUINT32 Buffer;    // Memory to store

  // private invariants
  invariant(Size > 0 && Size % 32 == 0)
  …
```

```
$ref_cnt(old($s), #p) == $ref_cnt($s, #p)
&& $ite.bool($set_in(#p, $owns(old($s),
owner)),
   $ite.bool($set_in(#p, owns),
   $st_eq(old($s), $s, #p),
   $wrapped($s, #p, $typ(#p)) &&
   $timestamp_is_now($s, #p)),
$ite.bool($set_in(#p, owns),
$owner($s, #p) == owner && $closed($s,
```

```
:assumption
  (forall (?x Int) (?y Int)
    (iff
      (= (IntEqual ?x ?y) boolTrue
      (= ?x ?y)))
 :formula
   (flet   .
```

```
owner)),
   $ite.bool($set_in(#p, owns),
   $st_eq(old($s), $s, #p),
   $wrapped($s, #p, $typ(#p)) &&
   $timestamp_is_now($s, #p)),
$ite.bool($set_in(#p, owns),
$owner($s, #p) == owner &&
$closed($s,
```

Available at http://vcc.codeplex.com/

Contracts / Modular Verification

```
int foo(int x)
  requires(x > 5)          // precond
  ensures(result > x)      // postcond
{
…
}
```

```
void bar(int y; int *z)
  writes(z)                  // framing
  requires(y > 7)
  maintains(*z > 7)          // invariant
{
   *z = foo(y);
   assert(*z > 7);
}
```

- function contracts: pre-/postconditions, framing
- modularity: **`bar`** only knows contract (but not code) of **`foo`**

advantages:

- modular verification: one function at a time
- no unfolding of code: scales to large applications

Hypervisor: Some Statistics

- VCs have several Mb
- Thousands of non ground clauses
- Developers are willing to wait at most 5 min per VC

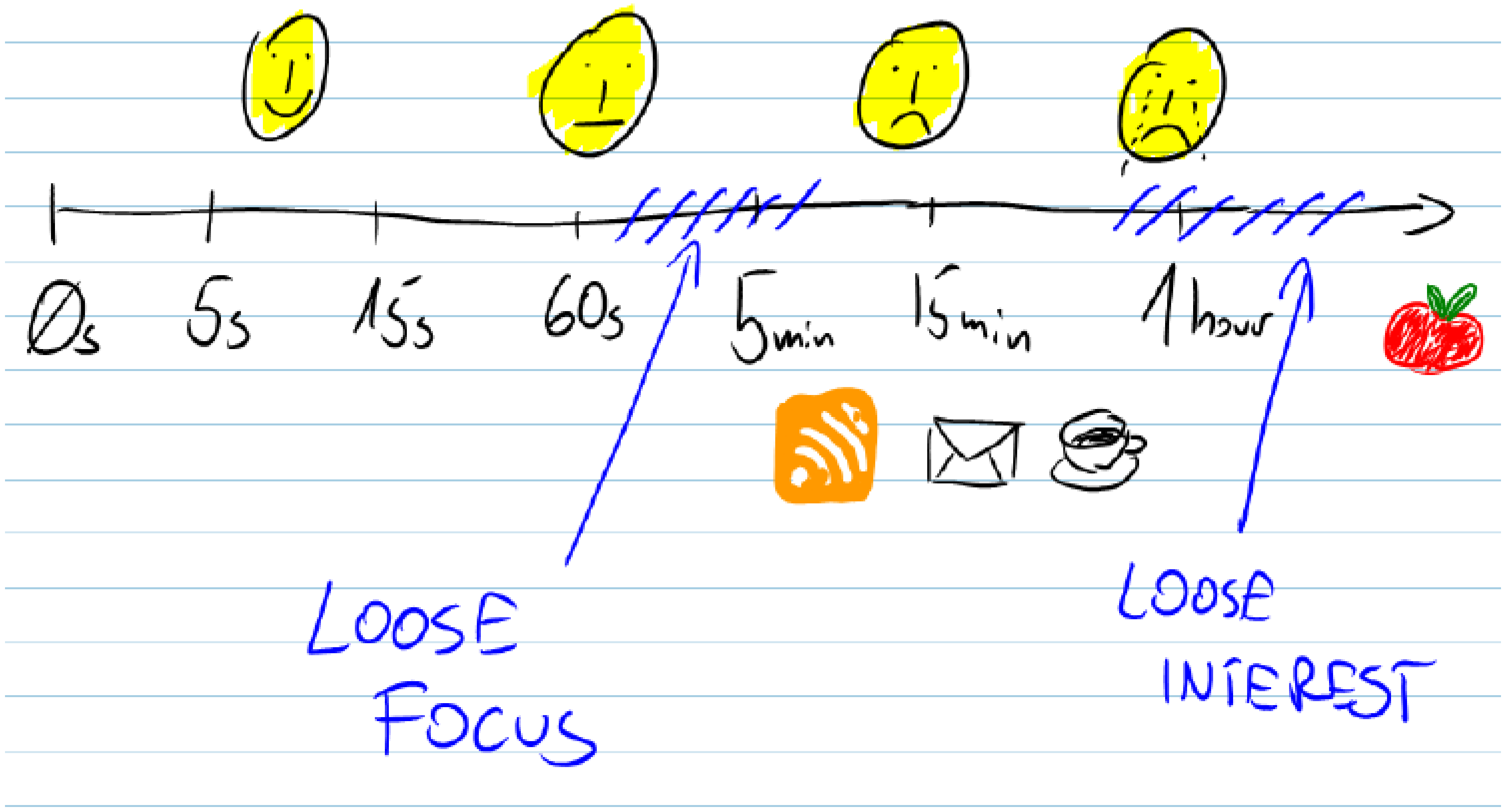
Hypervisor: Some Statistics

- VCs have several Mb
- Thousands of non ground clauses
- Developers are willing to wait at most 5 min per VC

Are you willing to wait more than 5 min for your compiler?

Microsoft Research

Verification Attempt Time vs. Satisfaction and Productivity



By Michal Moskal (VCC Designer and Software Verification Expert)

Why did my proof attempt fail?

1. My annotations are not strong enough!
weak loop invariants and/or contracts

2. My theorem prover is not strong (or fast) enough.
Send "angry" email to Nikolaj and Leo.

Challenge

- Quantifiers, quantifiers, quantifiers, …
- Modeling the runtime

$$
\begin{aligned}
&\forall\ h,o,f: \\
&\qquad \mathrm{IsHeap}(h) \wedge o \neq \mathrm{null} \wedge \mathrm{read}(h, o, \mathrm{alloc}) = t \\
&\qquad \Rightarrow \\
&\qquad \mathrm{read}(h,o, f) = \mathrm{null} \vee \mathrm{read}(h, \mathrm{read}(h,o,f),\mathrm{alloc}) = t
\end{aligned}
$$

Microsoft Research

Challenge

- Quantifiers, quantifiers, quantifiers, …
- Modeling the runtime
- Frame axioms

$$\forall\ o, f:$$
$$\quad o \neq \text{null} \wedge \text{read}(h_0, o, \text{alloc}) = t \Rightarrow$$
$$\quad\quad \text{read}(h_1,o,f) = \text{read}(h_0,o,f) \vee (o,f) \in M$$

Challenge

- Quantifiers, quantifiers, quantifiers, …
- Modeling the runtime
- Frame axioms
- User provided assertions

$\forall$ i,j: i $\leq$ j $\Rightarrow$ read(a,i) $\leq$ read(b,j)

Challenge

- Quantifiers, quantifiers, quantifiers, …
- Modeling the runtime
- Frame axioms
- User provided assertions
- Theories

  $\forall$ x: p(x,x)

  $\forall$ x,y,z: p(x,y), p(y,z) $\Rightarrow$ p(x,z)

  $\forall$ x,y: p(x,y), p(y,x) $\Rightarrow$ x = y

Challenge

- Quantifiers, quantifiers, quantifiers, …
- Modeling the runtime
- Frame axioms
- User provided assertions
- Theories
- Solver must be fast in satisfiable instances.

We want to find bugs!

Microsoft Research

Bad news

There is no sound and refutationally complete procedure for linear integer arithmetic + free function symbols

Many Approaches

- Heuristic quantifier instantiation
- Combining SMT with Saturation provers
- Complete quantifier instantiation
- Decidable fragments
- Model based quantifier instantiation

Challenge: Modeling Runtime

- Is the axiomatization of the runtime consistent?
- False implies everything
- Partial solution: SMT + Saturation Provers
- Found many bugs using this approach

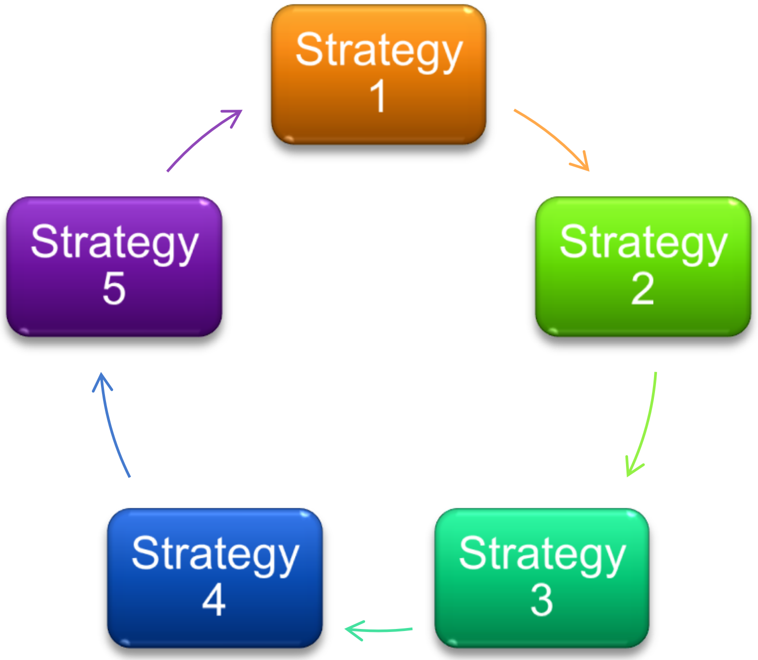
Challenge: Robustness

- Standard complain

  “I made a small modification in my Spec, and Z3 is timingout”
- This also happens with SAT solvers (NP-complete)
- In our case, the problems are undecidable
- Partial solution: parallelization

Parallel Z3

- Joint work with Y. Hamadi (MSRC) and C. Wintersteiger
- Multi-core & Multi-node (HPC)
- Different strategies in parallel
- Collaborate exchanging lemmas



Microsoft Research

Hey, I don’t trust these proofs

Z3 may be buggy.

Solution: proof/certificate generation.

Engineering problem: these certificates are too big.

Hey, I don’t trust these proofs

Z3 may be buggy.

Solution: proof/certificate generation.

Engineering problem: these certificates are too big.

The Axiomatization of the runtime may be buggy or inconsistent.

Yes, this is true. We are working on new techniques for proving satisfiability (building a model for these axioms)

Hey, I don't trust these proofs

Z3 may be buggy.

- Solution: proof/certificate generation.
- Engineering problem: these certificates are too big.

The Axiomatization of the runtime may be buggy or inconsistent.

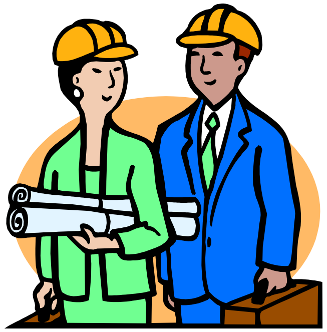
- Yes, this is true. We are working on new techniques for proving satisfiability (building a model for these axioms)

The VCG generator is buggy (i.e., it makes the wrong assumptions)

- Do you trust your compiler?

Engineer Perspective

- These are bug-finding tools!
  - When they return "Proved", it just means they cannot find more bugs.
- I add Loop invariants to speedup the process.
  - I don't want to waste time analyzing paths with 1,2,...,k,... iterations.
- They are successful if they expose bugs not exposed by regular testing.



Microsoft Research

Conclusion

Powerful, mature, and versatile tools like SMT solvers can now be exploited in very useful ways.

The construction and application of satisfiability procedures is an active research area with exciting challenges.

SMT is hot at Microsoft.

Z3 is a new SMT solver.

Main applications:

- Test-case generation.
- Verifying compiler.
- Model Checking & Predicate Abstraction.

Books

- Bradley & Manna: The Calculus of Computation
- Kroening & Strichman: Decision Procedures, An Algorithmic Point of View
- Chapter in the Handbook of Satisfiability

Web Links

Z3:

`http://research.microsoft.com/projects/z3`

`http://research.microsoft.com/~leonardo`

- Slides & Papers

`http://www.smtlib.org`

`http://www.smtcomp.org`

References

[Ack54] W. Ackermann. Solvable cases of the decision problem. *Studies in Logic and the Foundation of Mathematics*, 1954

[ABC$^+$02] G. Audemard, P. Bertoli, A. Cimatti, A. Kornilowicz, and R. Sebastiani. A SAT based approach for solving formulas over boolean and linear mathematical propositions. In *Proc. of CADE'02*, 2002

[BDS00] C. Barrett, D. Dill, and A. Stump. A framework for cooperating decision procedures. In *17th International Conference on Computer-Aided Deduction*, volume 1831 of *Lecture Notes in Artificial Intelligence*, pages 79–97. Springer-Verlag, 2000

[BdMS05] C. Barrett, L. de Moura, and A. Stump. SMT-COMP: Satisfiability Modulo Theories Competition. In *Int. Conference on Computer Aided Verification (CAV'05)*, pages 20–23. Springer, 2005

[BDS02] C. Barrett, D. Dill, and A. Stump. Checking satisfiability of first-order formulas by incremental translation to SAT. In Ed Brinksma and Kim Guldstrand Larsen, editors, *Proceedings of the* 14^{th} *International Conference on Computer Aided Verification (CAV '02)*, volume 2404 of *Lecture Notes in Computer Science*, pages 236–249. Springer-Verlag, July 2002. Copenhagen, Denmark

[BBC$^+$05] M. Bozzano, R. Bruttomesso, A. Cimatti, T. Junttila, P. van Rossum, S. Ranise, and R. Sebastiani. Efficient satisfiability modulo theories via delayed theory combination. In *Int. Conf. on Computer-Aided Verification (CAV)*, volume 3576 of *LNCS*. Springer, 2005

[Chv83] V. Chvatal. *Linear Programming*. W. H. Freeman, 1983

References

[CG96] B. Cherkassky and A. Goldberg. Negative-cycle detection algorithms. In *European Symposium on Algorithms*, pages 349–363, 1996

[DLL62] M. Davis, G. Logemann, and D. Loveland. A machine program for theorem proving. *Communications of the ACM*, 5(7):394–397, July 1962

[DNS03] D. Detlefs, G. Nelson, and J. B. Saxe. Simplify: A theorem prover for program checking. Technical Report HPL-2003-148, HP Labs, 2003

[DST80] P. J. Downey, R. Sethi, and R. E. Tarjan. Variations on the Common Subexpression Problem. *Journal of the Association for Computing Machinery*, 27(4):758–771, 1980

[dMR02] L. de Moura and H. Rueß. Lemmas on demand for satisfiability solvers. In *Proceedings of the Fifth International Symposium on the Theory and Applications of Satisfiability Testing (SAT 2002)*. Cincinnati, Ohio, 2002

[DdM06] B. Dutertre and L. de Moura. Integrating simplex with DPLL(T). Technical report, CSL, SRI International, 2006

[dMB07b] L. de Moura and N. Bjørner. Efficient E-Matching for SMT solvers. In *CADE-21*, pages 183–198, 2007

References

[dMB07c] L. de Moura and N. Bjørner. Model Based Theory Combination. In *SMT'07*, 2007

[dMB07a] L. de Moura and N. Bjørner. Relevancy Propagation . Technical Report MSR-TR-2007-140, Microsoft Research, 2007

[dMB08a] L. de Moura and N. Bjørner. Z3: An Efficient SMT Solver. In *TACAS 08*, 2008

[dMB08c] L. de Moura and N. Bjørner. Engineering DPLL(T) + Saturation. In *IJCAR'08*, 2008

[dMB08b] L. de Moura and N. Bjørner. Deciding Effectively Propositional Logic using DPLL and substitution sets. In *IJCAR'08*, 2008

[GHN$^+$04] H. Ganzinger, G. Hagen, R. Nieuwenhuis, A. Oliveras, and C. Tinelli. DPLL(T): Fast decision procedures. In R. Alur and D. Peled, editors, *Int. Conference on Computer Aided Verification (CAV 04)*, volume 3114 of *LNCS*, pages 175–188. Springer, 2004

[MSS96] J. Marques-Silva and K. A. Sakallah. GRASP - A New Search Algorithm for Satisfiability. In *Proc. of ICCAD'96*, 1996

[NO79] G. Nelson and D. C. Oppen. Simplification by cooperating decision procedures. *ACM Transactions on Programming Languages and Systems*, 1(2):245–257, 1979

[NO05] R. Nieuwenhuis and A. Oliveras. DPLL(T) with exhaustive theory propagation and its application to difference logic. In *Int. Conference on Computer Aided Verification (CAV'05)*, pages 321–334. Springer, 2005

References

[Opp80] D. Oppen. Reasoning about recursively defined data structures. *J. ACM*, 27(3):403–411, 1980

[PRSS99] A. Pnueli, Y. Rodeh, O. Shtrichman, and M. Siegel. Deciding equality formulas by small domains instantiations. *Lecture Notes in Computer Science*, 1633:455–469, 1999

[Pug92] William Pugh. The Omega test: a fast and practical integer programming algorithm for dependence analysis. In *Communications of the ACM*, volume 8, pages 102–114, August 1992

[RT03] S. Ranise and C. Tinelli. The smt-lib format: An initial proposal. In *Proceedings of the 1st International Workshop on Pragmatics of Decision Procedures in Automated Reasoning (PDPAR'03), Miami, Florida*, pages 94–111, 2003

[RS01] H. Ruess and N. Shankar. Deconstructing shostak. In *16th Annual IEEE Symposium on Logic in Computer Science*, pages 19–28, June 2001

[SLB03] S. Seshia, S. Lahiri, and R. Bryant. A hybrid SAT-based decision procedure for separation logic with uninterpreted functions. In *Proc. 40th Design Automation Conference*, pages 425–430. ACM Press, 2003

[Sho81] R. Shostak. Deciding linear inequalities by computing loop residues. *Journal of the ACM*, 28(4):769–779, October 1981

References

[dMB09] L. de Moura and N. Bjørner. Generalized and Efficient Array Decision Procedures. FMCAD, 2009.
[GdM09] Y. Ge and L. de Moura. Complete Quantifier Instantiation for quantified SMT formulas, CAV, 2009.