On Designing and Implementing Satisfiability Modulo Theory (SMT) Solvers

Summer School 2009 , Nancy
Verification Technology, Systems and Applications

Leonardo de Moura
Microsoft Research

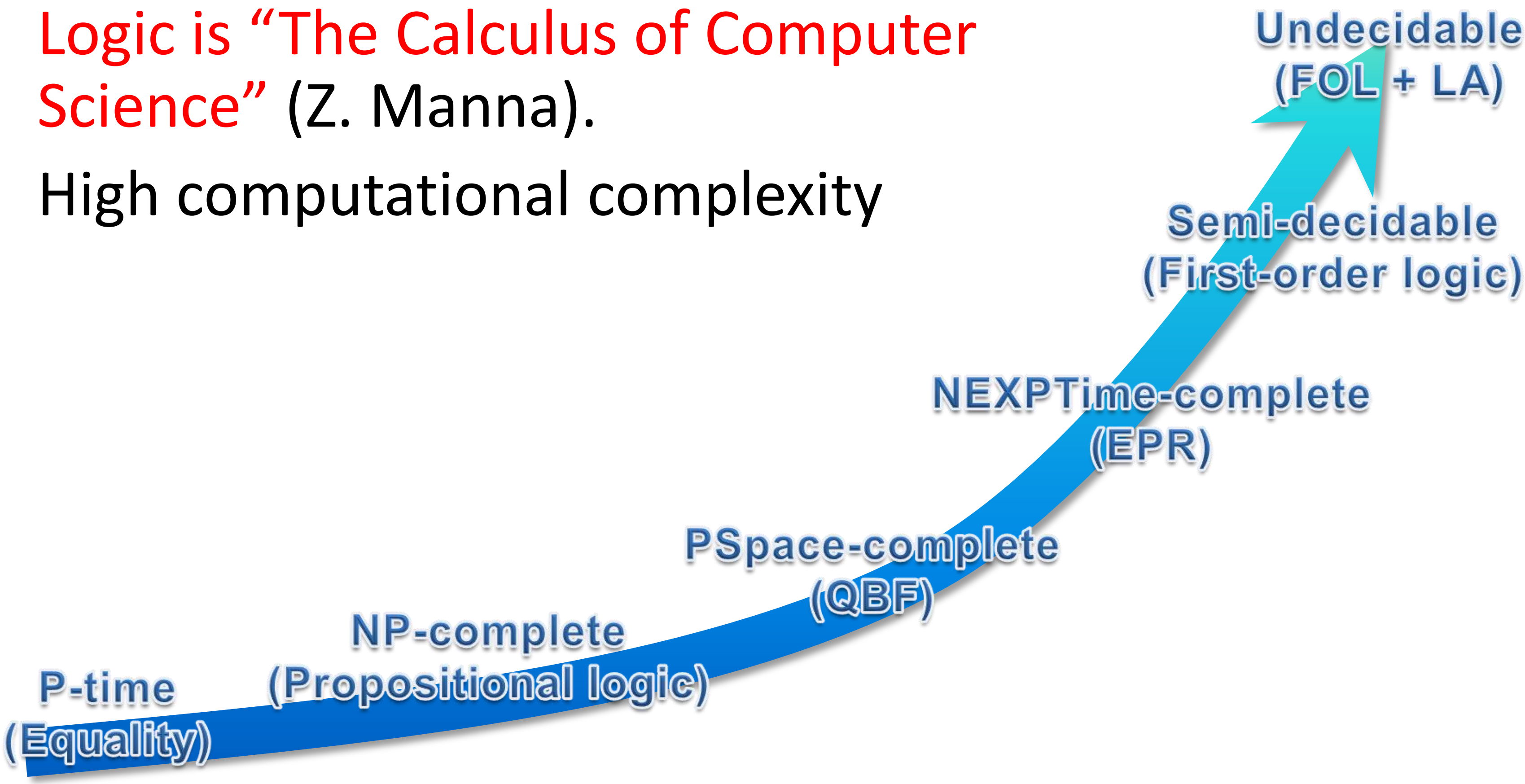
Symbolic Reasoning

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

Microsoft Research

Symbolic Reasoning

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



Microsoft Research

Applications

- Test case generation
- Verifying Compilers
- Predicate Abstraction
- Invariant Generation
- Type Checking
- Model Based Testing

Microsoft Research

Some Applications @ Microsoft

The Spec# Programming System

HAVOC

ForμLa

Hyper-V

Microsoft Virtualization

Terminator T-2

VCC

SLAM

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.

Type checking

Signature:

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

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

Microsoft® Research

Satisfiability Modulo Theories (SMT)

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

Array Theory

Microsoft Research

Satisfiability Modulo Theories (SMT)

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

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

Satisfiability Modulo Theories (SMT)

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

Simplifying

Microsoft Research

Satisfiability Modulo Theories (SMT)

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

Microsoft Research

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

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

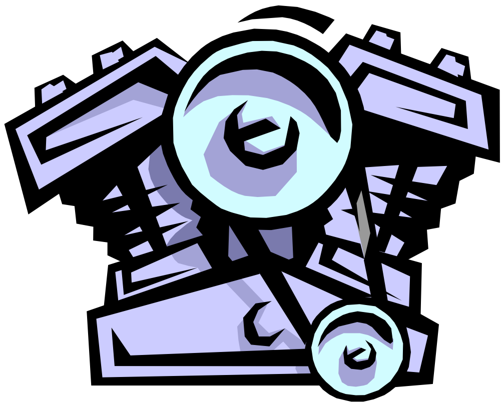
Bounded Model Checking

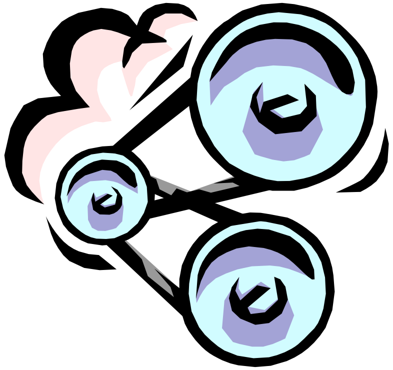
Test-Case Generation

Little Engines of Proof

An SMT Solver is a collection of

Little Engines of Proof



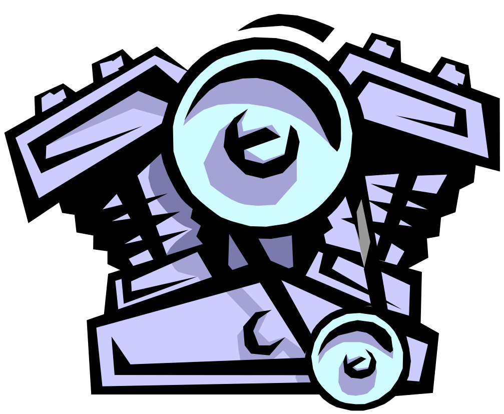


Microsoft Research

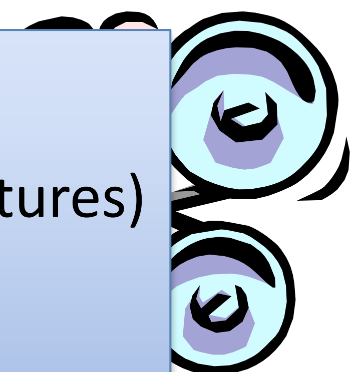
Little Engines of Proof

An SMT Solver is a collection of

Little Engines of Proof



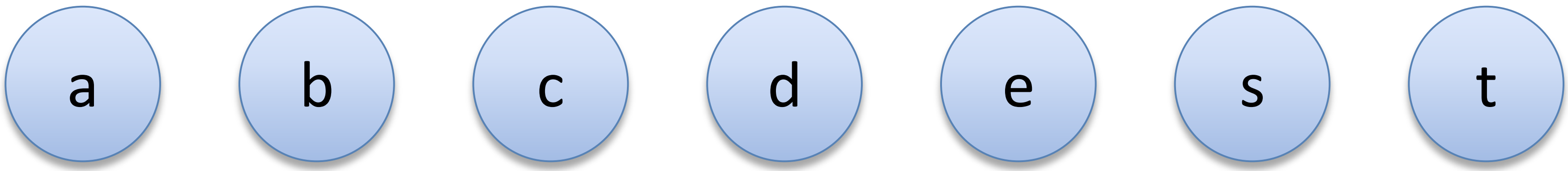
Examples:
SAT Solver (Daniel's lectures)
Equality solver



Microsoft Research

Deciding Equality

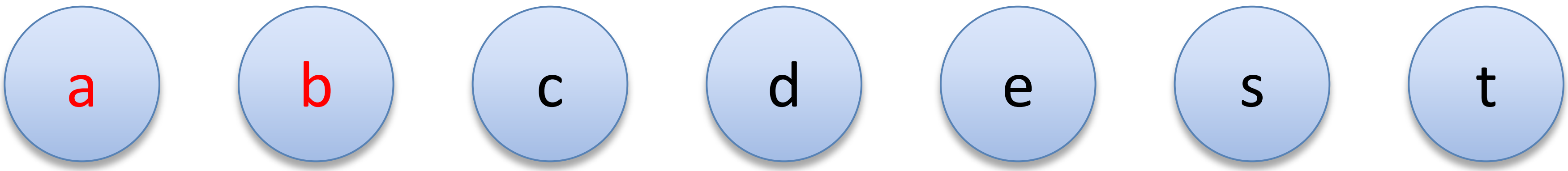
$a = b, b = c, d = e, b = s, d = t, a \neq e, a \neq s$



Microsoft Research

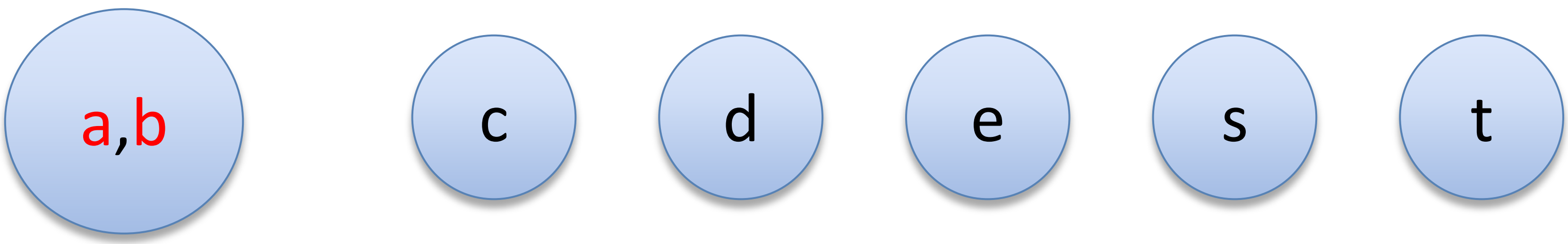
Deciding Equality

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



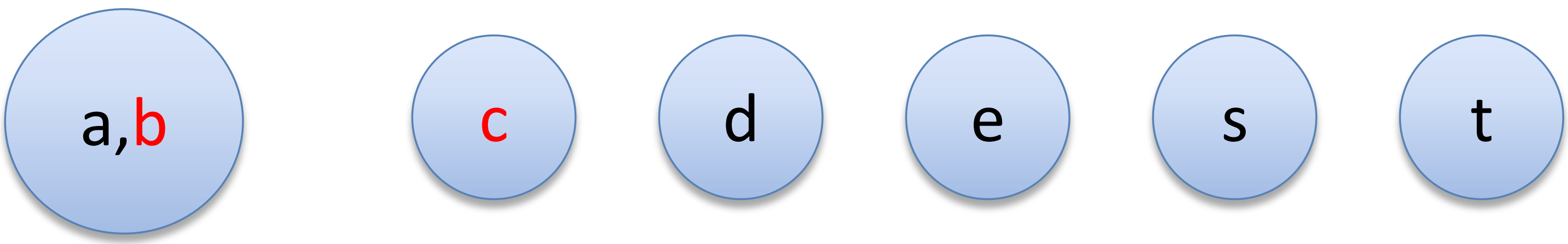
Deciding Equality

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



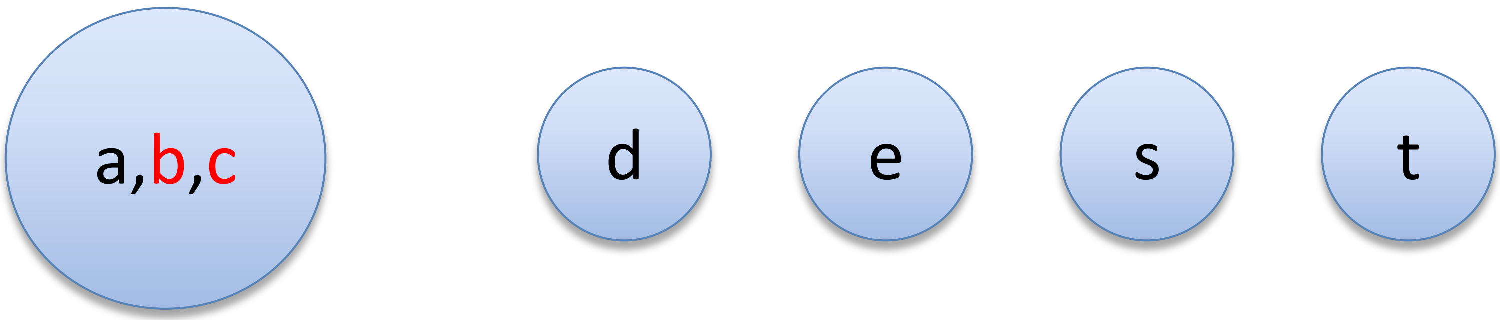
Deciding Equality

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



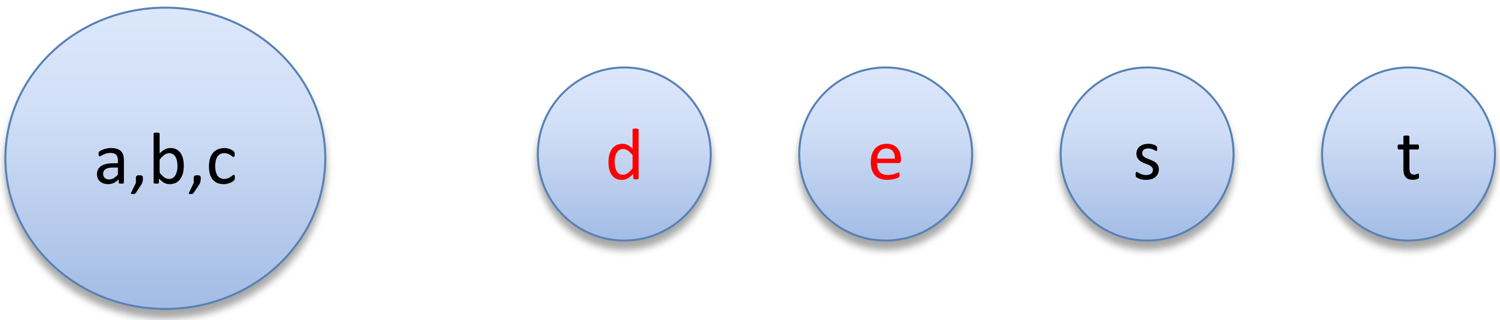
Deciding Equality

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



Deciding Equality

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

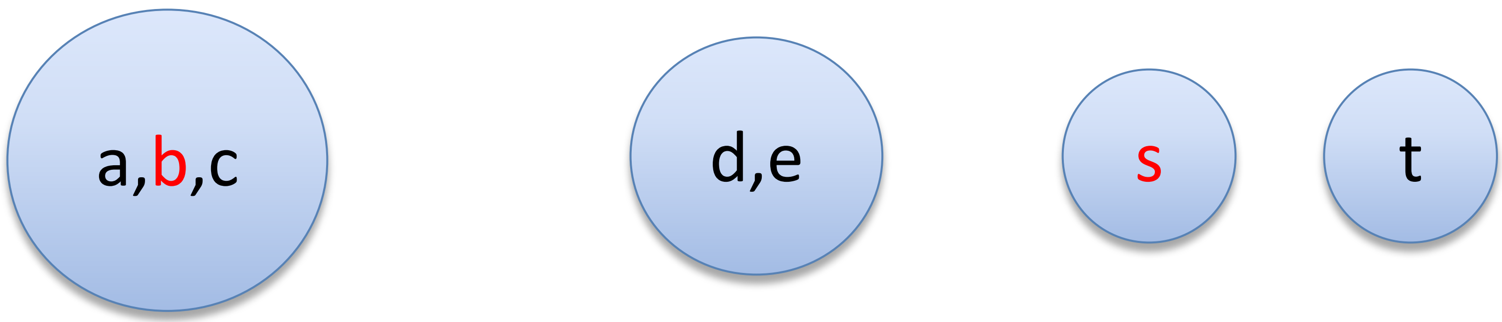


Deciding Equality

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

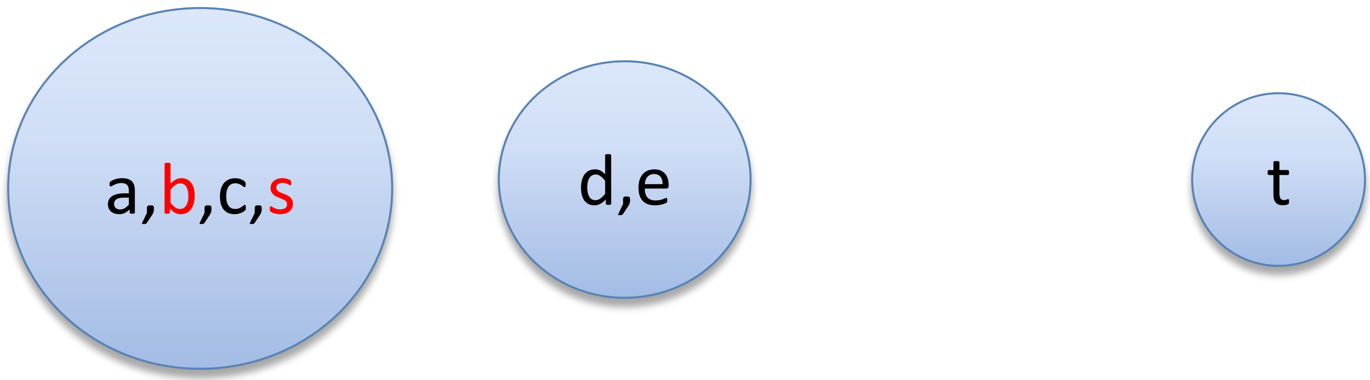
Deciding Equality

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



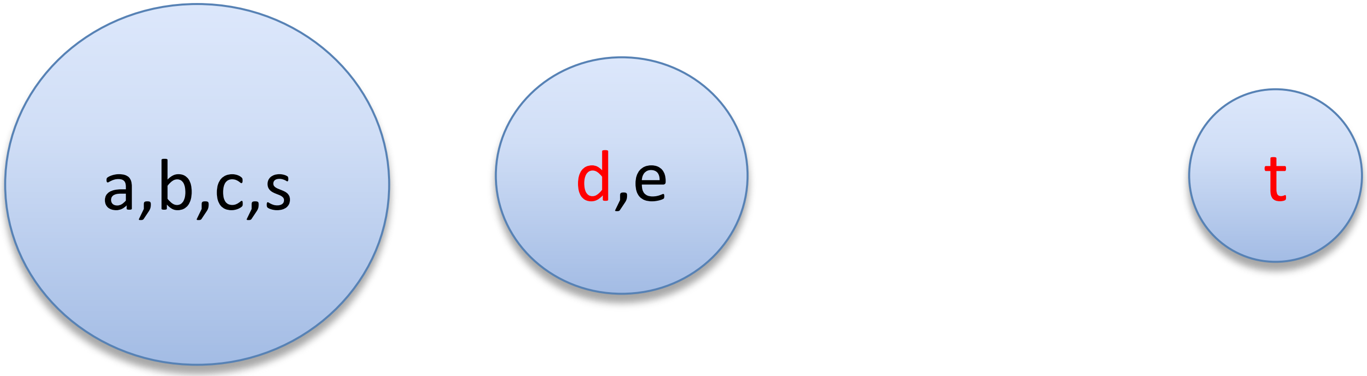
Deciding Equality

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



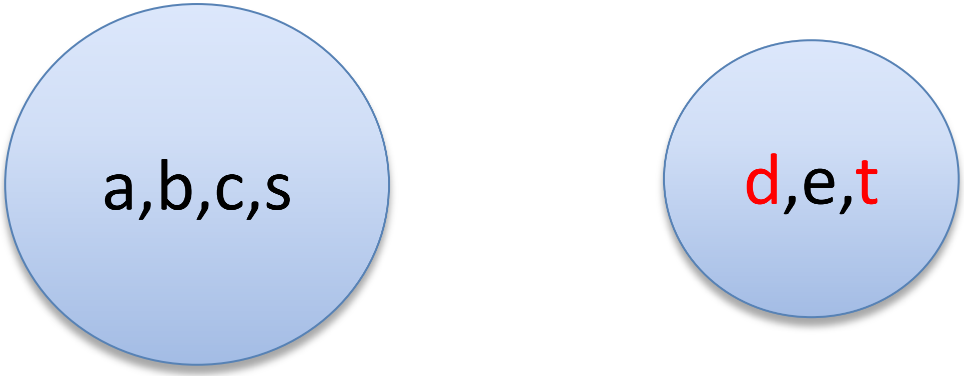
Deciding Equality

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



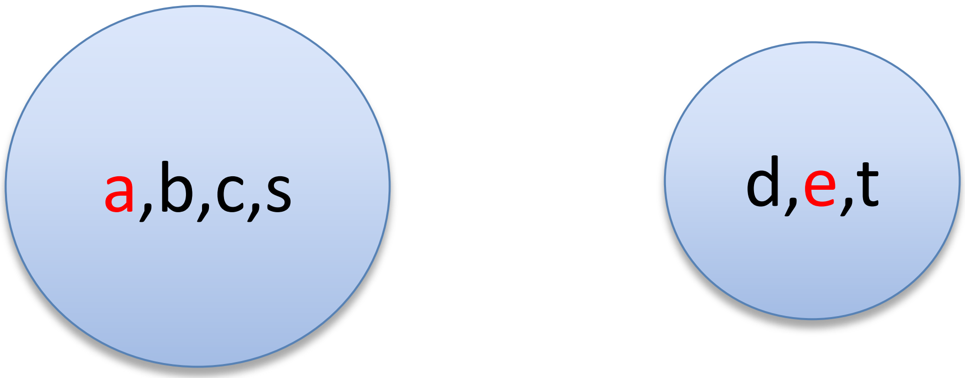
Deciding Equality

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



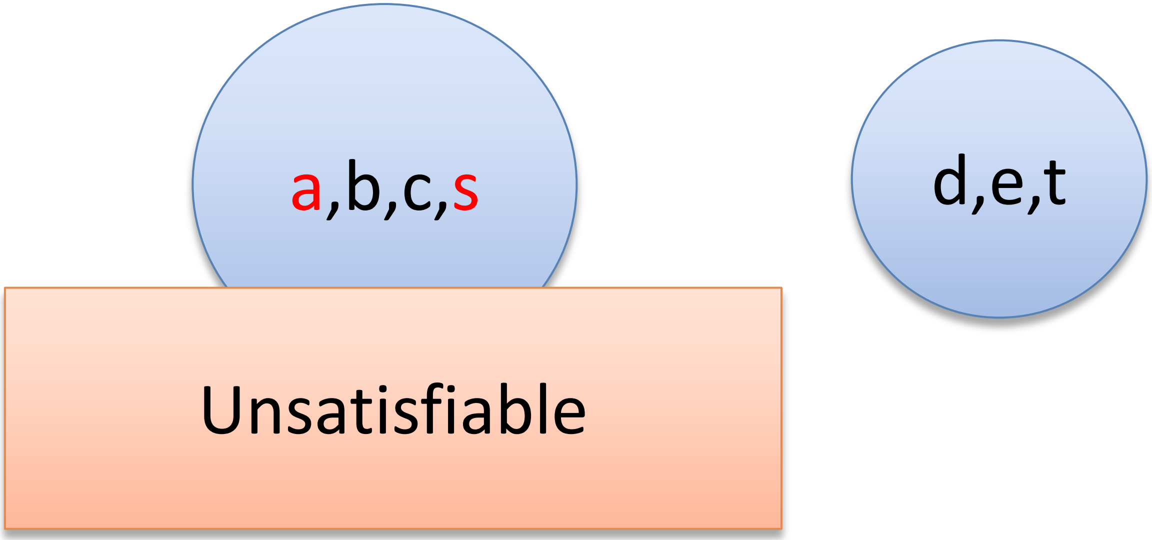
Deciding Equality

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



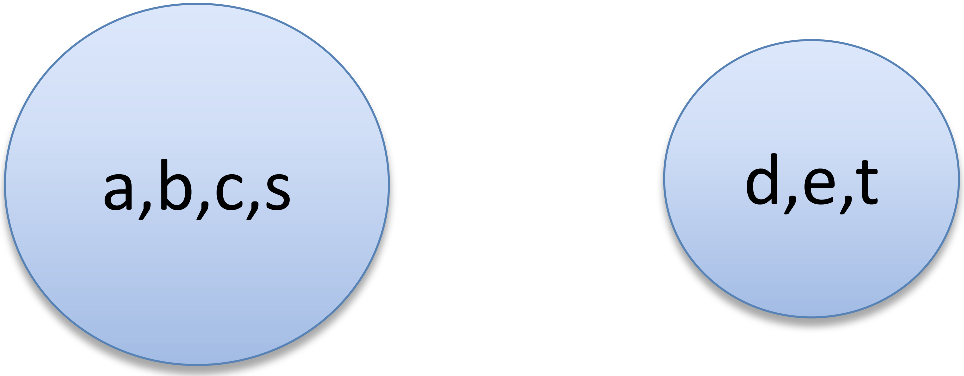
Deciding Equality

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



Deciding Equality

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

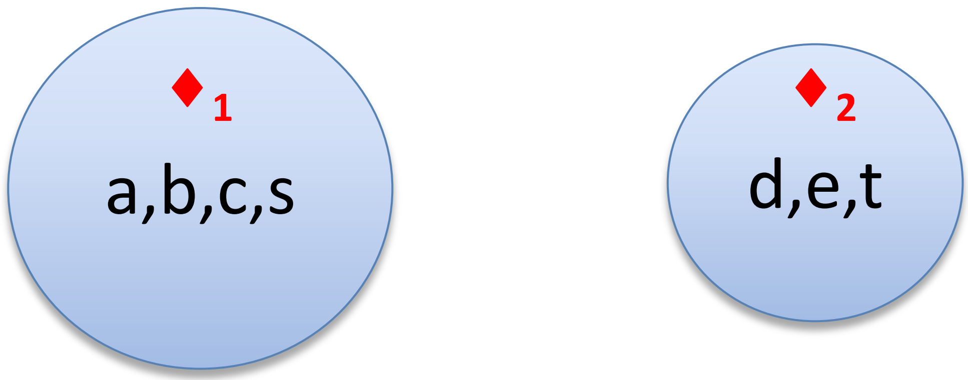


Model construction

Microsoft Research

Deciding Equality

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

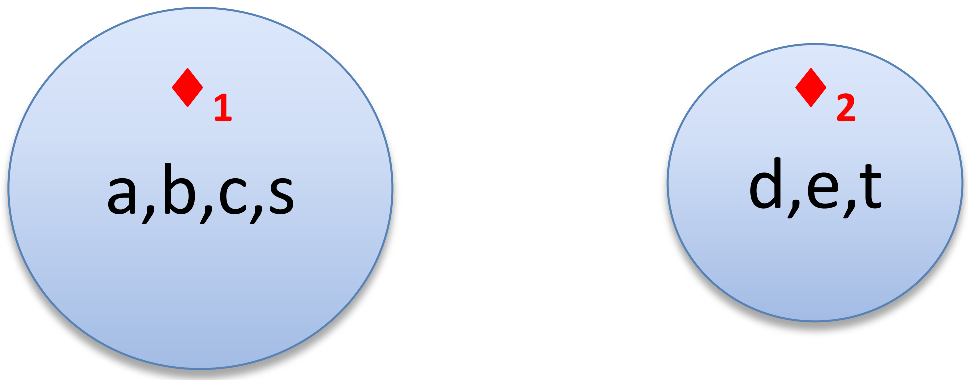


Model construction

$|M| = \{\blacklozenge_1, \blacklozenge_2\}$ (universe, aka domain)

Deciding Equality

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



Model construction

$|M| = \{\blacklozenge_1, \blacklozenge_2\}$ (universe, aka domain)

$M(a) = \blacklozenge_1$ (assignment)

Microsoft Research

Deciding Equality

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

$\blacklozenge_1$

a,b,c,s

$\blacklozenge_2$

d,e,t

Alternative notation:
$a^M = \blacklozenge_1$

Model construction

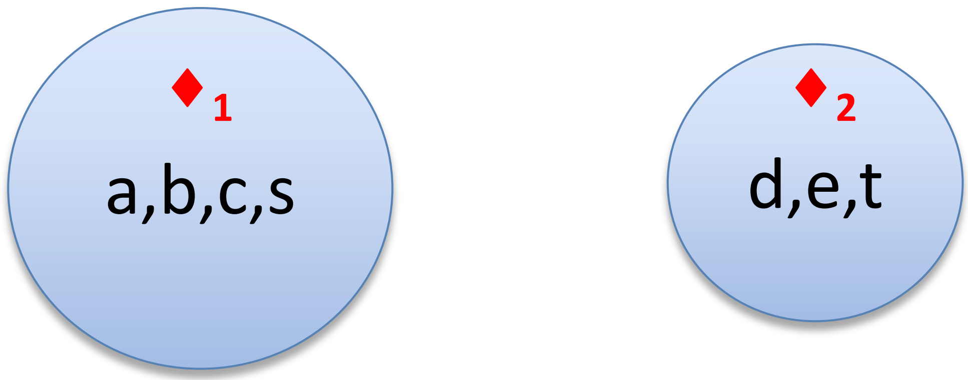
$|M| = \{\blacklozenge_1, \blacklozenge_2\}$ (universe, aka domain)

$M(a) = \blacklozenge_1$ (assignment)

Microsoft Research

Deciding Equality

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



Model construction

$|M| = \{\blacklozenge_1, \blacklozenge_2\}$ (universe, aka domain)

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

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

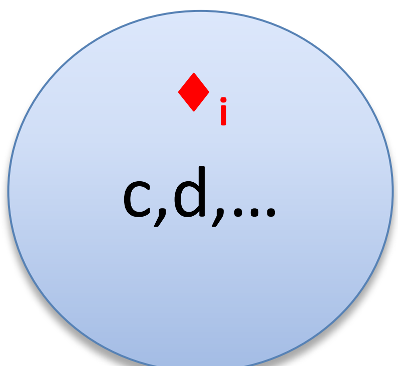
Deciding Equality: Termination, Soundness, Completeness

- 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".

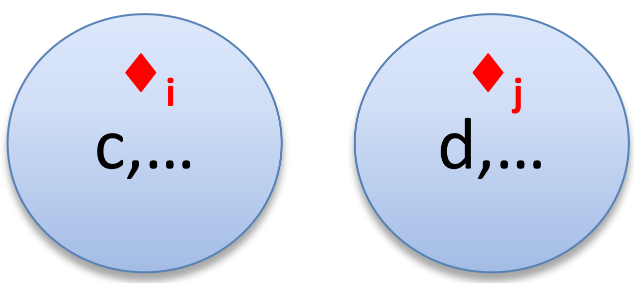


$$M(c) = M(d) = \blacklozenge_i$$

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 $c \neq d$ 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"



$M(c) = \blacklozenge_i$

$M(d) = \blacklozenge_j$

Microsoft Research

Deciding Equality + (uninterpreted) Functions

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

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

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

First Step: "Naming" subterms

Congruence Rule:

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

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

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

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

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

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

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

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

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

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:

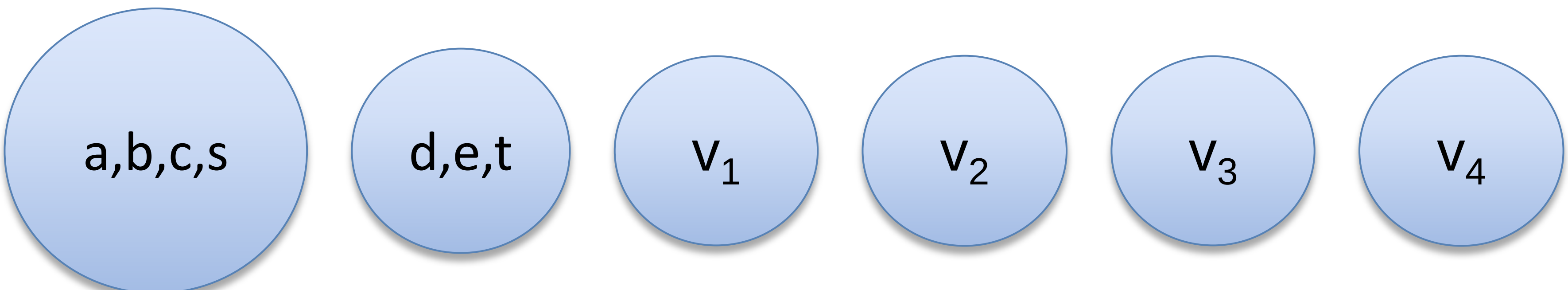
$x_1 = y_1, \ldots, x_n = y_n$ implies $f(x_1, \ldots, x_n) = f(y_1, \ldots, y_n)$

Microsoft Research

Deciding Equality + (uninterpreted) Functions

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

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



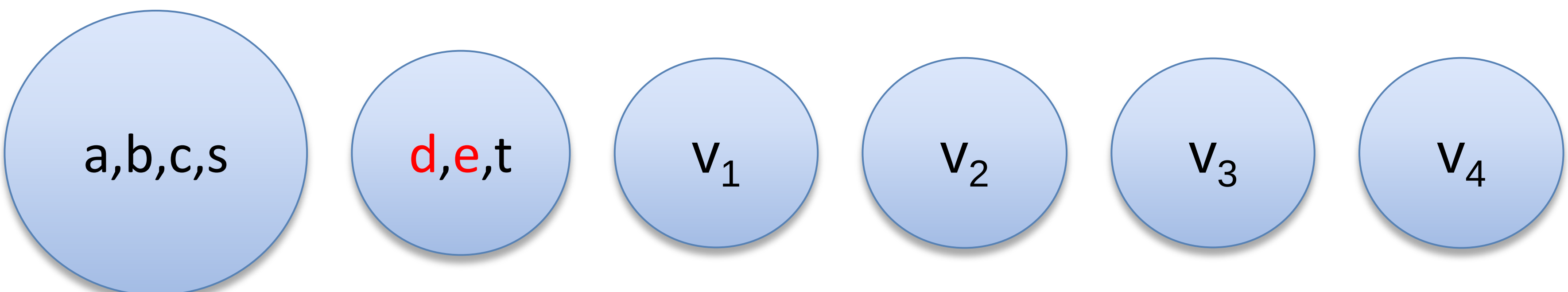
Congruence Rule:

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

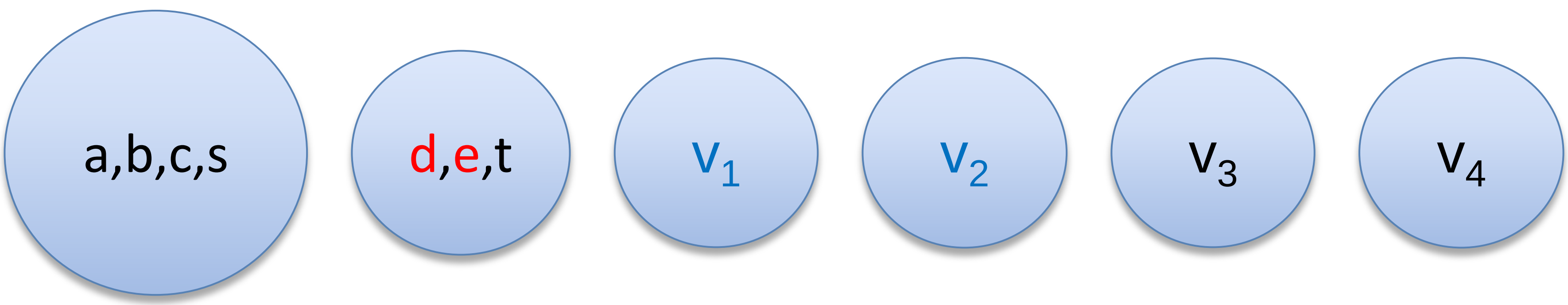
$d = e$ implies $g(d) = g(e)$

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:

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

$$d = e \text{ implies } v_1 = v_2$$

Deciding Equality + (uninterpreted) Functions

We say:
v_1 and v_2 are congruent.

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

$$v_1 \equiv g(d),\ v_2 \equiv g(e),\ v_3 \equiv f(a, v_1)\ \ldots_4 \equiv f(b, v_2)$$

a,b,c,s

d,e,t

v_1, v_2

v_3

v_4

Congruence Rule:

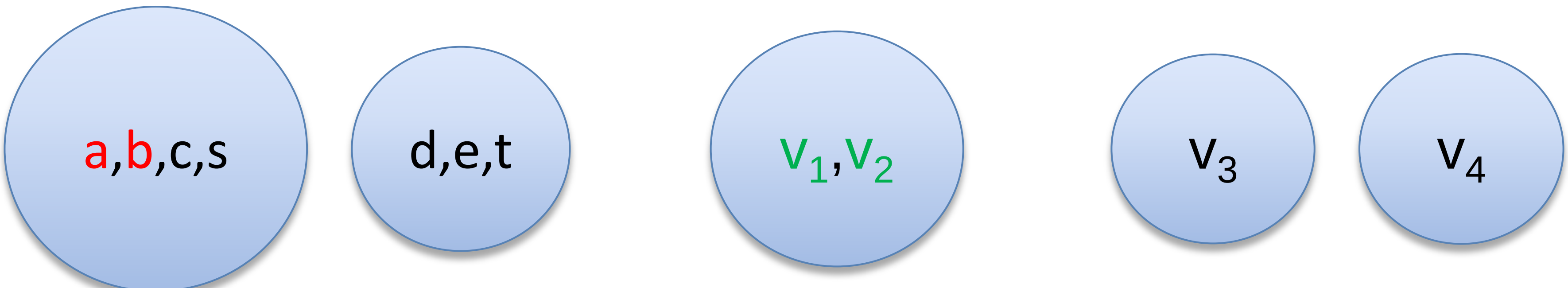
$$x_1 = y_1, \ldots, x_n = y_n \text{ implies } f(x_1, \ldots, x_n) = f(y_1, \ldots, y_n)$$

$$d = e \text{ implies } v_1 = v_2$$

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:

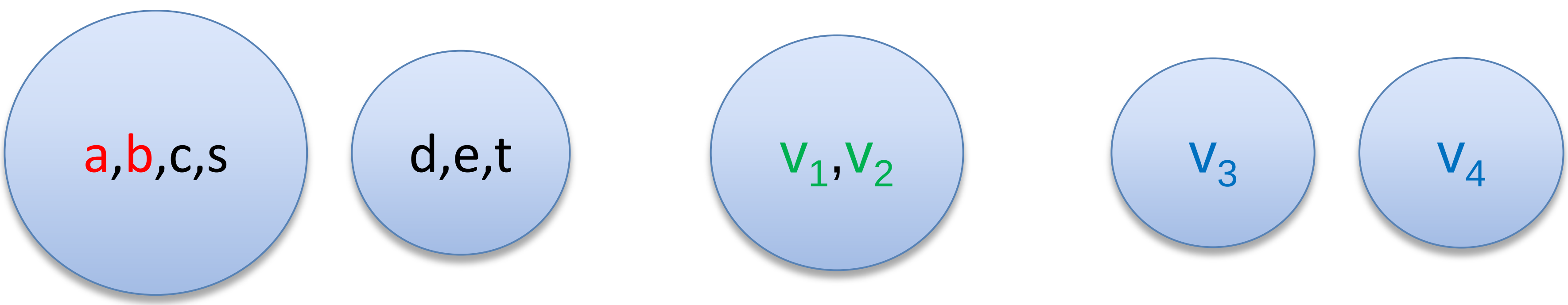
$$x_1 = y_1, \ldots, x_n = y_n \text{ implies } f(x_1, \ldots, x_n) = f(y_1, \ldots, y_n)$$

$$a = b,\ v_1 = v_2 \text{ implies } f(a, v_1) = f(b, v_2)$$

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

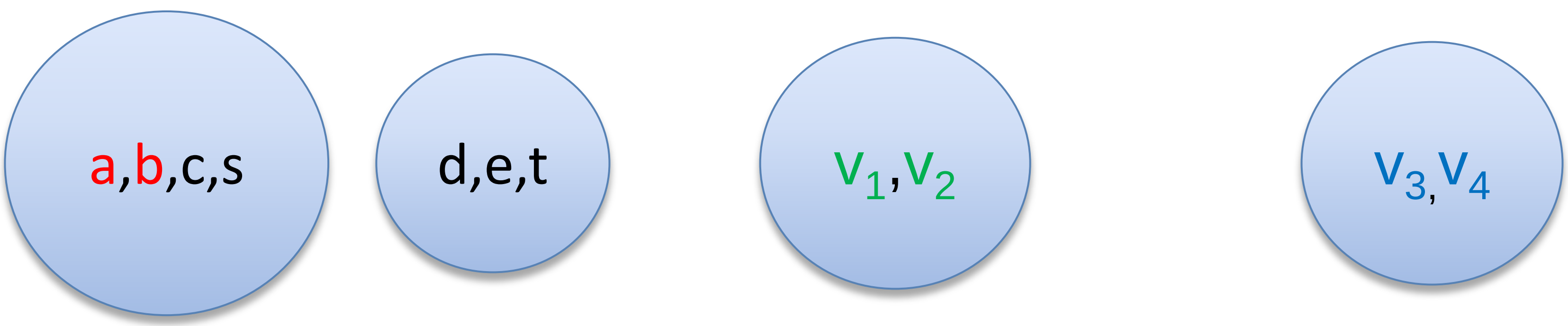
$a = b,\ v_1 = v_2$ implies $v_3 = v_4$

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:

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

$$a = b,\ v_1 = v_2 \text{ implies } v_3 = v_4$$

Microsoft Research

Deciding Equality + (uninterpreted) Functions

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

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

a,b,c,s

d,e,t

v_1, v_2

v_3, v_4

Unsatisfiable

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

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

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

Changing the problem

a,b,c,s

d,e,t

v_1, v_2

v_3, v_4

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

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

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

a,b,c,s | d,e,t | v_1,v_2 | v_3, v_4

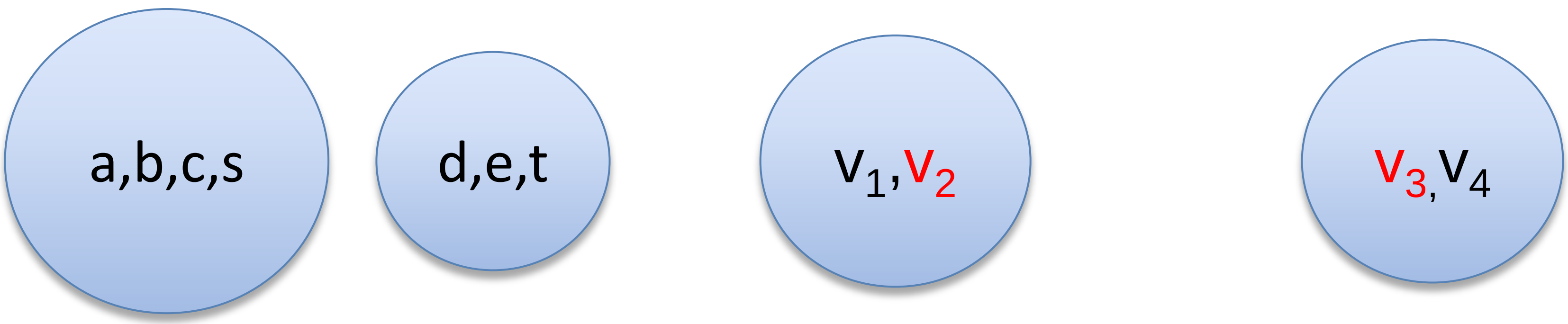
Congruence Rule:

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

Deciding Equality + (uninterpreted) Functions

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

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



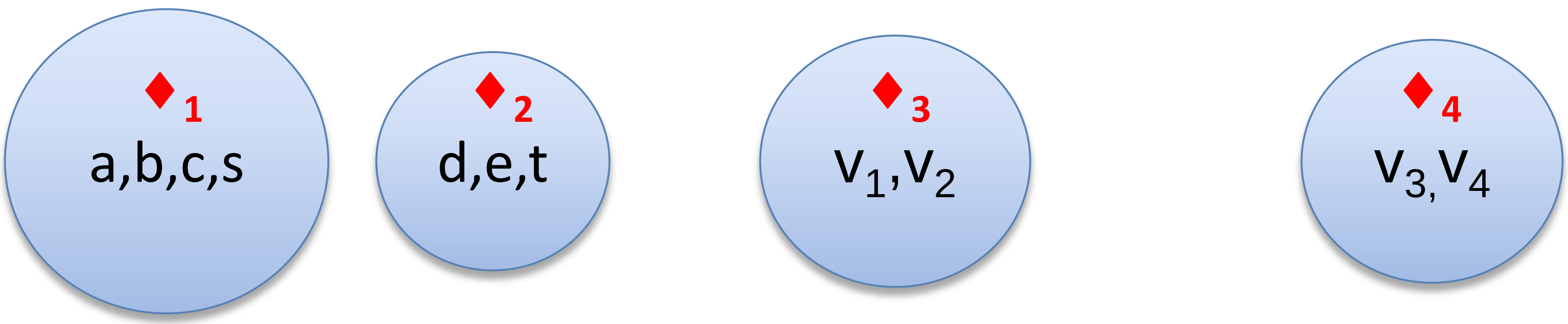
Congruence Rule:

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

Deciding Equality + (uninterpreted) Functions

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

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



Model construction:

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

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

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

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

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

Microsoft Research

Deciding Equality + (uninterpreted) Functions

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

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

$\blacklozenge_1$ 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.

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?

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 $v \equiv g(a)$ 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 $v \equiv g$

    $M(a) = \blacklozenge_i$

    $M(v) = \blacklozenge_j$

    $= \blacklozenge_k$, otherwise ($\blacklozenge_k$ is a ... ary 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 $M(g)(\blacklozenge_1) = \blacklozenge_3$?

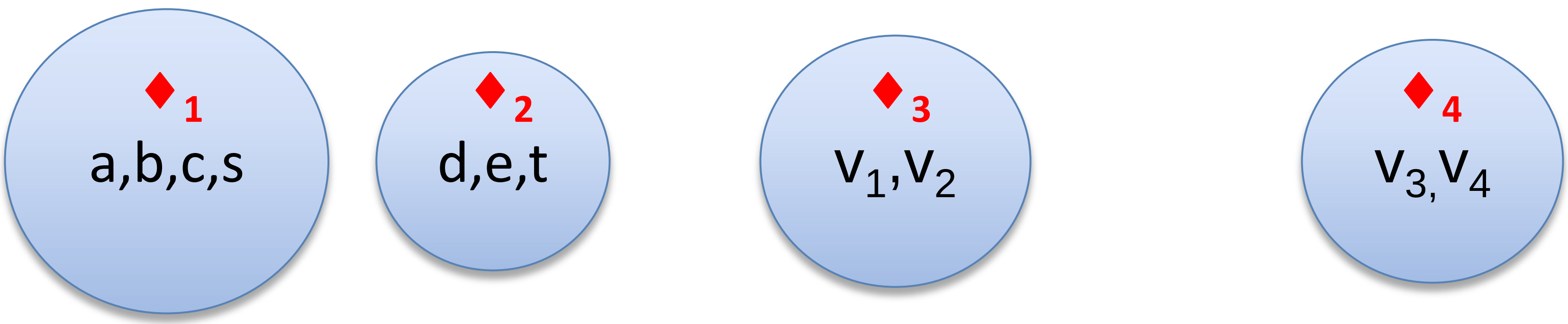
This is impossible because of the congruence rule!

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

Deciding Equality + (uninterpreted) Functions

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

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



Model construction:

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

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

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

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

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

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:

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

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

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

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

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

$M(g)(\blacklozenge_i) = \blacklozenge_j$ 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:

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

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

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

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

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

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

$M(g)(\blacklozenge_i) = \blacklozenge_j$ 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:

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

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

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

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

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

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

$M(g)(\blacklozenge_i) = \blacklozenge_j$ 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

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

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

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?

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

Microsoft Research

Deciding Equality + (uninterpreted) Functions

What about predicates?

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

$$\Downarrow$$

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

Ackermannization

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

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

$$\Downarrow$$

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

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

Deciding Equality + (uninterpreted) Functions

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

Deciding Equality + (uninterpreted) Functions

d,e,t   Sets (equivalence classes)

d,e $\cup$ t = d,e,t   Union

a,b,c,s $a \neq s$ Membership

Microsoft Research

Deciding Equality + (uninterpreted) Functions

d,e,t

Sets (equivale

Key observation:

The sets are disjoint!

d,e ∪ t = d,e,t

Union

a,b,c,s

$a \neq s$

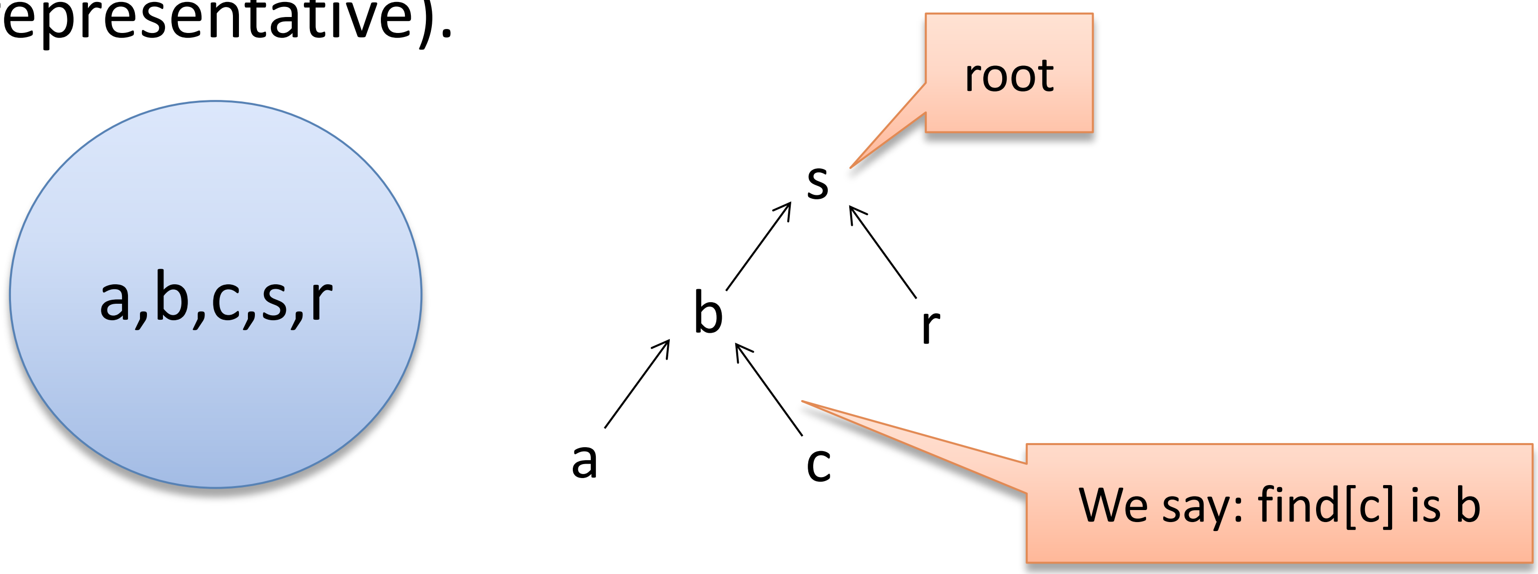
Membership

Microsoft Research

Deciding Equality + (uninterpreted) Functions

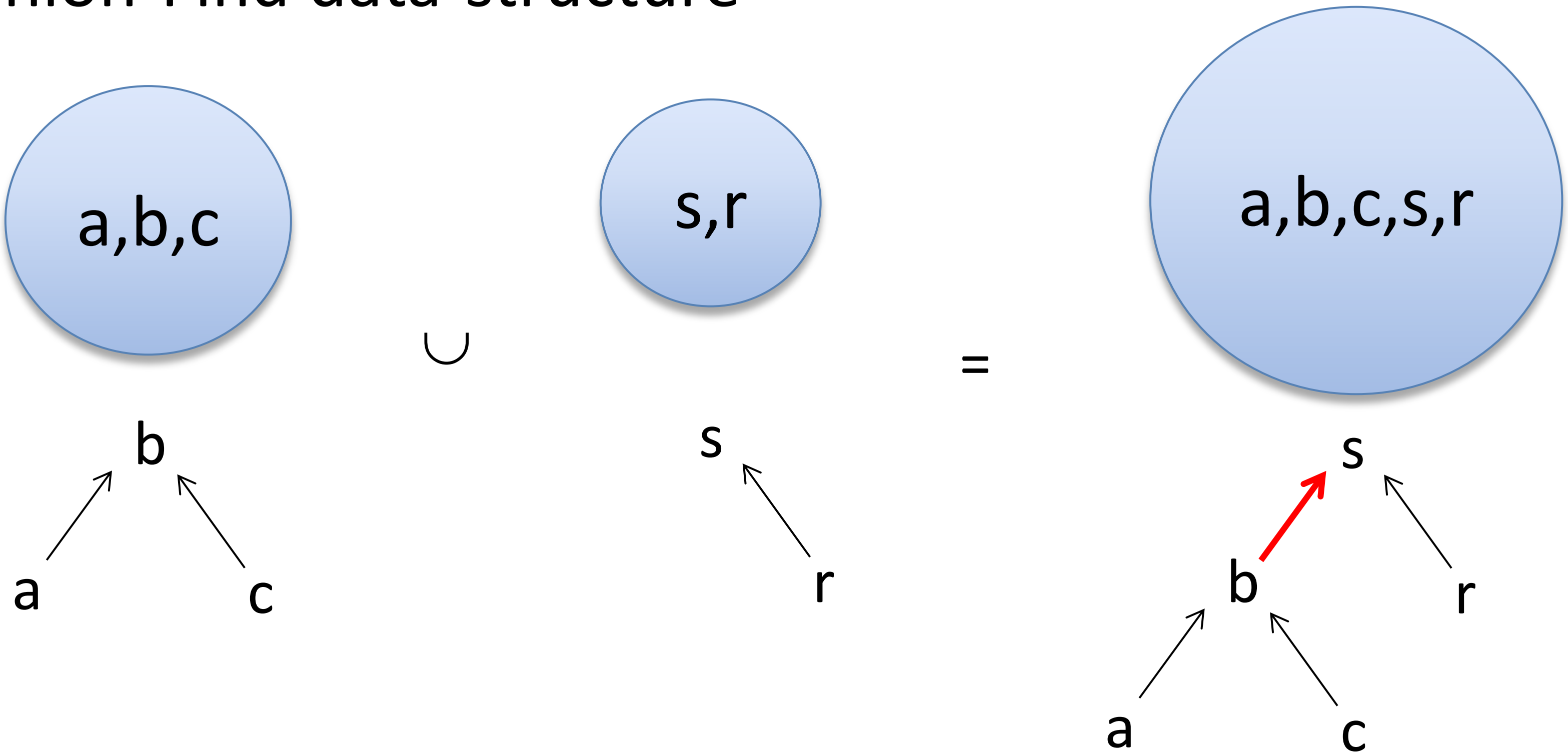
Union-Find data-structure

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



Deciding Equality + (uninterpreted) Functions

Union-Find data-structure



Microsoft Research

Deciding Equality + (uninterpreted) Functions

Tracking the equivalence classes size is important!

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

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

…

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

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

Microsoft Research

Deciding Equality + (uninterpreted) Functions

Tracking the equivalence classes size is important!

$$a_1 \rightarrow a_2 \quad \cup \quad a_3 \quad = \quad a_1 \rightarrow a_2 \leftarrow a_3$$

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

...

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

Deciding Equality + (uninterpreted) Functions

Tracking the equivalence classes size is important!

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

$$a_1 \rightarrow a_2 \quad \cup \quad a_3 \quad = \quad a_1 \rightarrow a_2 \leftarrow a_3$$

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

…

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

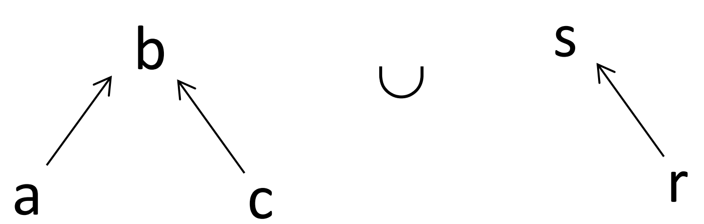
Each constant has two fields: find and size.

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.



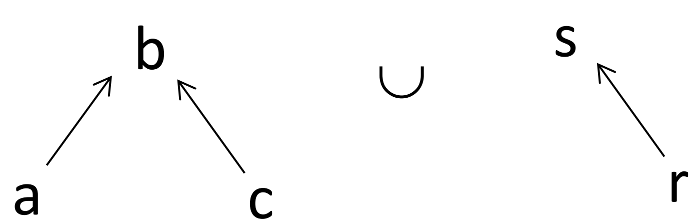
occurrences[b] = { $v_1 \equiv g(b)$, $v_2 \equiv f(a)$ }
occurrences[s] = { $v_3 \equiv f(r)$ }

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.



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

occurrences(s) = { $v_3 \equiv f(r)$ }

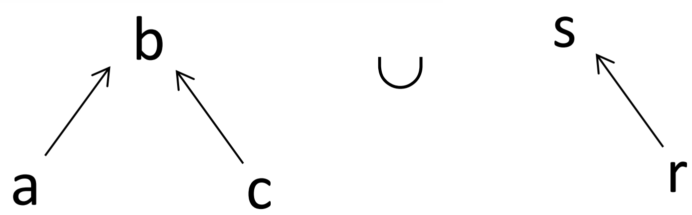
Inefficient version:

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.

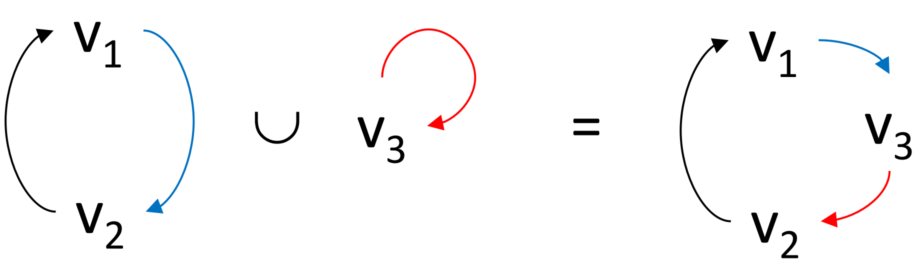
Research

Deciding Equality + (uninterpreted) Functions



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)



Microsoft Research

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.

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

Microsoft Research

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!

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

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

Microsoft Research

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.

Microsoft Research

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

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

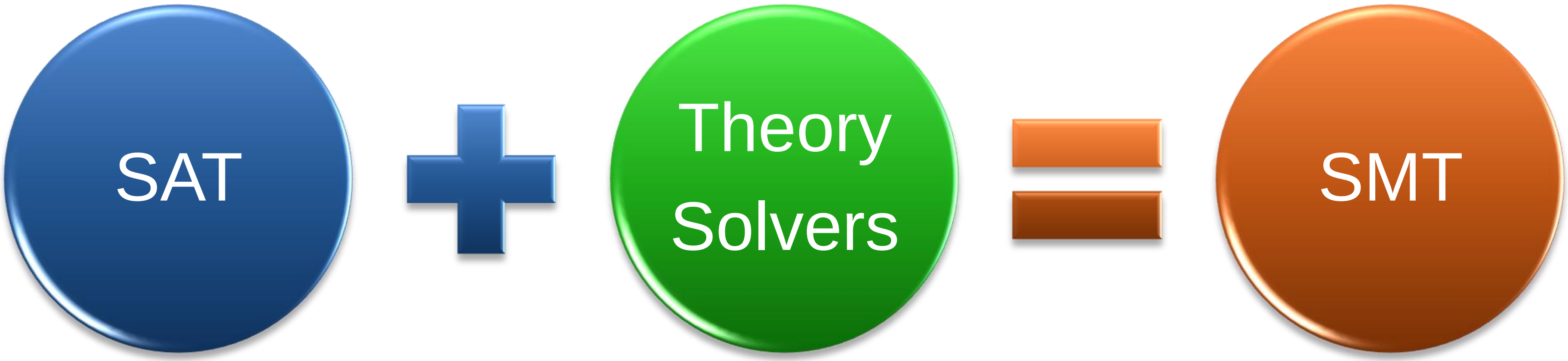
Naïve Solution: Convert to DNF

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

Too Inefficient!
(exponential blowup)

Microsoft Research

SMT : Basic Architecture



Case Analysis

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

SAT (propositional checkers): Case Analysis

$$
\begin{aligned}
p &\vee q,\\
p &\vee \neg q,\\
\neg p &\vee q,\\
\neg p &\vee \neg q
\end{aligned}
$$

Microsoft Research

SAT (propositional checkers): Case Analysis

$$
\begin{aligned}
p &\lor q, \\
p &\lor \neg q, \\
\neg p &\lor q, \\
\neg p &\lor \neg q
\end{aligned}
$$

Assignment:
p = false,
q = false

Microsoft Research

SAT (propositional checkers): Case Analysis

$$\begin{array}{r} p \lor q, \\ p \lor \neg q, \\ \neg p \lor q, \\ \neg p \lor \neg q \end{array}$$

Assignment:
p = false,
q = true

Microsoft Research

SAT (propositional checkers): Case Analysis

$$p \vee q,$$
$$p \vee \neg q,$$
$$\neg p \vee q,$$
$$\neg p \vee \neg q$$

Assignment:
p = true,
q = false

Microsoft Research

SAT (propositional checkers): Case Analysis

$$
\begin{aligned}
p &\vee q, \\
p &\vee \neg q, \\
\neg p &\vee q, \\
\neg p &\vee \neg q
\end{aligned}
$$

Assignment:
p = true,
q = true

Microsoft Research

DPLL

M | F

Partial model

Set of clauses

Microsoft Research

DPLL

Guessing

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

$$\Downarrow$$

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

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

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

$$\Downarrow$$

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

Microsoft Research

Modern DPLL

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

SAT + Theory solvers

Basic Idea

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

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 + 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

SAT + Theory solvers

Basic Idea

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

Abstract (aka “naming” atoms)

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

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

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

SAT Solver

Assignment

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

SAT + Theory solvers

Basic Idea

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

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

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

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

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

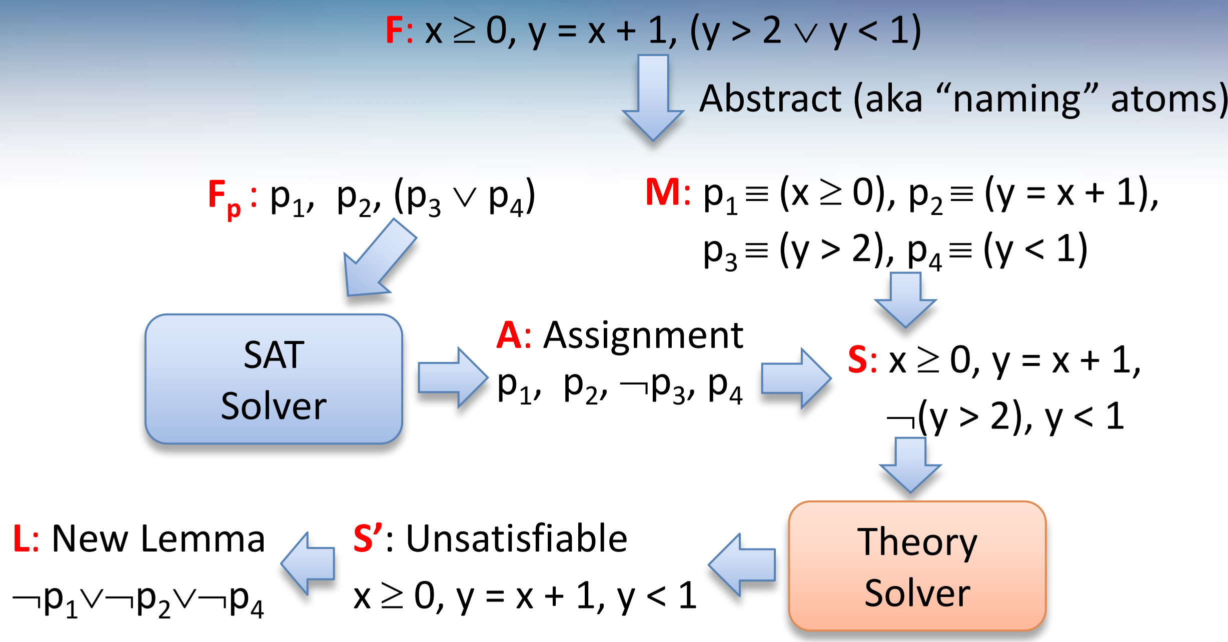
S: $x \geq 0,\ y = x + 1,$
$\neg(y > 2),\ y < 1$

Theory Solver

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

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

SAT + Theory solvers



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

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

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

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

Partial assignment is already Theory inconsistent.

SAT + Theory solvers

Efficient Backtracking

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

Deciding Equality + (uninterpreted) Functions

Efficient Lemma Generation (computing a small S')

(R, S') := Theory_solver(S)

When R = UNSAT (i.e., S is unsatisfiable),
S' $\subseteq$ S is also unsatisfiable

We say S' is redundant
iff
Exists S'' $\subset$ S' which is also unsatisfiable.

SAT + Theory solvers

Efficient Lemma Generation (computing a small S')

Avoid lemmas containing redundant literals.

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

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

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

$\neg p_1 \lor \neg p_2 \lor \neg p_3 \lor \neg p_4$

Imprecise Lemma

SAT + Theory solvers

Theory Propagation

It is the SMT equivalent of unit propagation.

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

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

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

$\Downarrow$ 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.

Deciding Equality + (uninterpreted) Functions

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

- Incrementality
- Efficient Backtracking
- Theory Propagation
- Lemma Learning

Microsoft Research

Deciding Equality + (uninterpreted) Functions

Incrementality (main problem):

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

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

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

$\Downarrow$

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

Deciding Equality + (uninterpreted) Functions

Incrementality (main problem):

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

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

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

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

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 ≠ c }

diseqs[c] = { a ≠ c }

Deciding Equality + (uninterpreted) Functions

Incrementality

Store the disequalities of a constant.

Very similar to the structure occurrences.

$a = b, a \neq c$

$$a \rightarrow b \qquad c$$

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

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

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

Deciding Equality + (uninterpreted) Functions

Incrementality

Store the disequalities of a constant.

Very similar to the structure occurrences.

$a = b, a \neq c$

$$\begin{array}{ccc} & b & \quad c \\ \nearrow & & \\ a & & \end{array}$$

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

diseqs(b) = $\{ a \neq c \}$

diseqs(c) = $\{ a \neq c \}$

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.

Microsoft Research

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

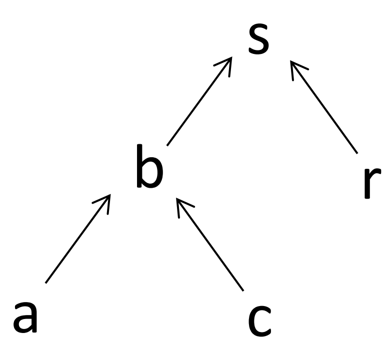
We can do better: coarse grain backtracking.

Minimize the size of the undo stack.

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

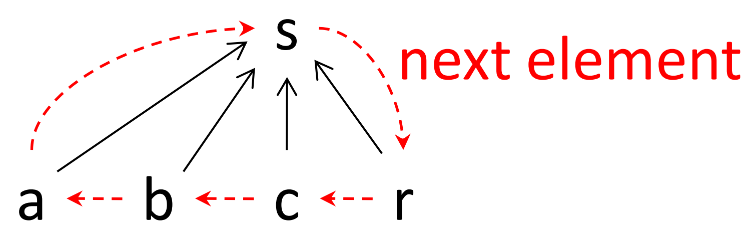
Let us change the union-find data-structure a little bit.

Before:



Fields: find, size

After:



Fields: root, next, size

Microsoft Research

Deciding Equality + (uninterpreted) Functions

Backtracking

We can do b

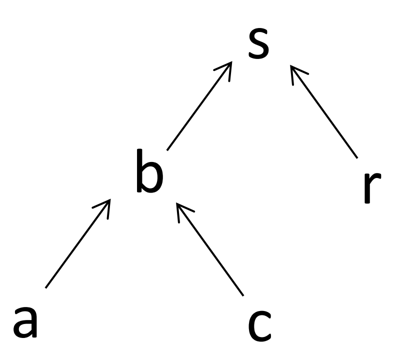
Minimiz

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.

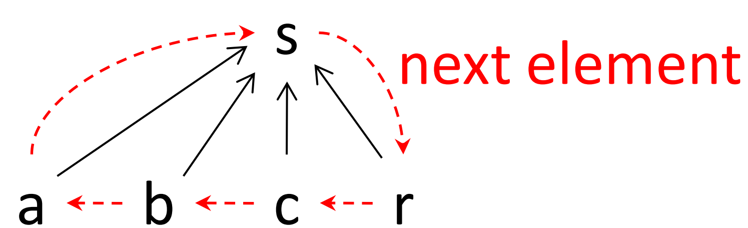
Let us change the union-find d ructure a little bit.

Before:



Fields: find, size

After:

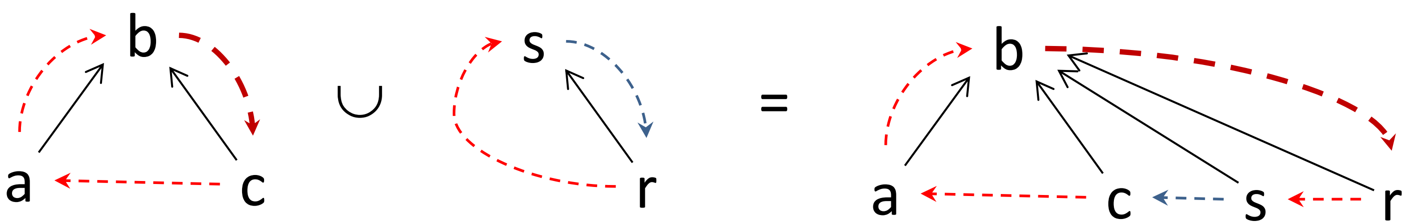


Fields: root, next, size

Microsoft Research

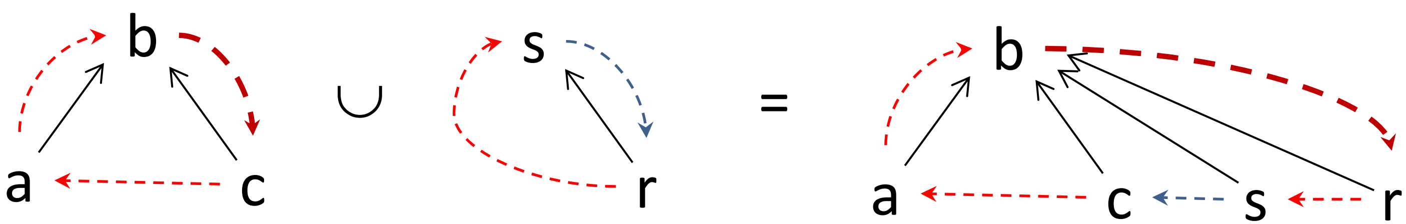
Deciding Equality + (uninterpreted) Functions

New union-find:



Deciding Equality + (uninterpreted) Functions

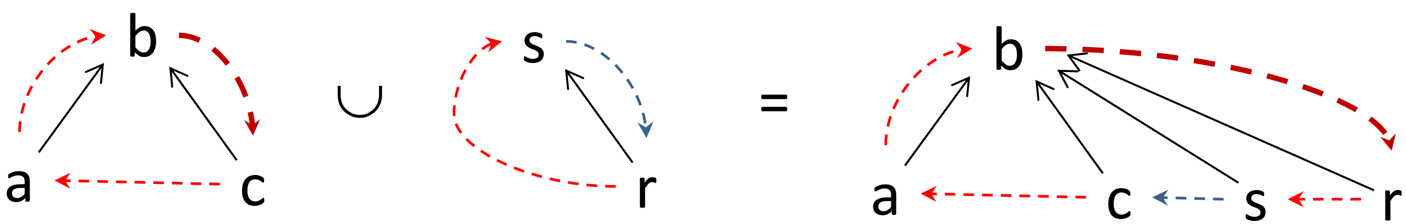
New union-find:



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]

Microsoft Research

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]

p was in the hash table before and after the merge

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

while ...

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

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.

Microsoft Research

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

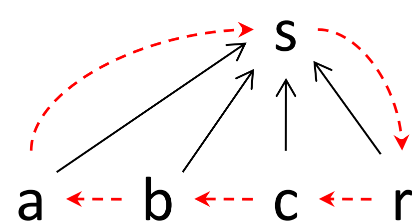
Efficient Lemma Generation (computing a small S')

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.

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



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

Microsoft Research

Deciding Equality + (uninterpreted) Functions

Efficient Lemma Generation (computing a small S')

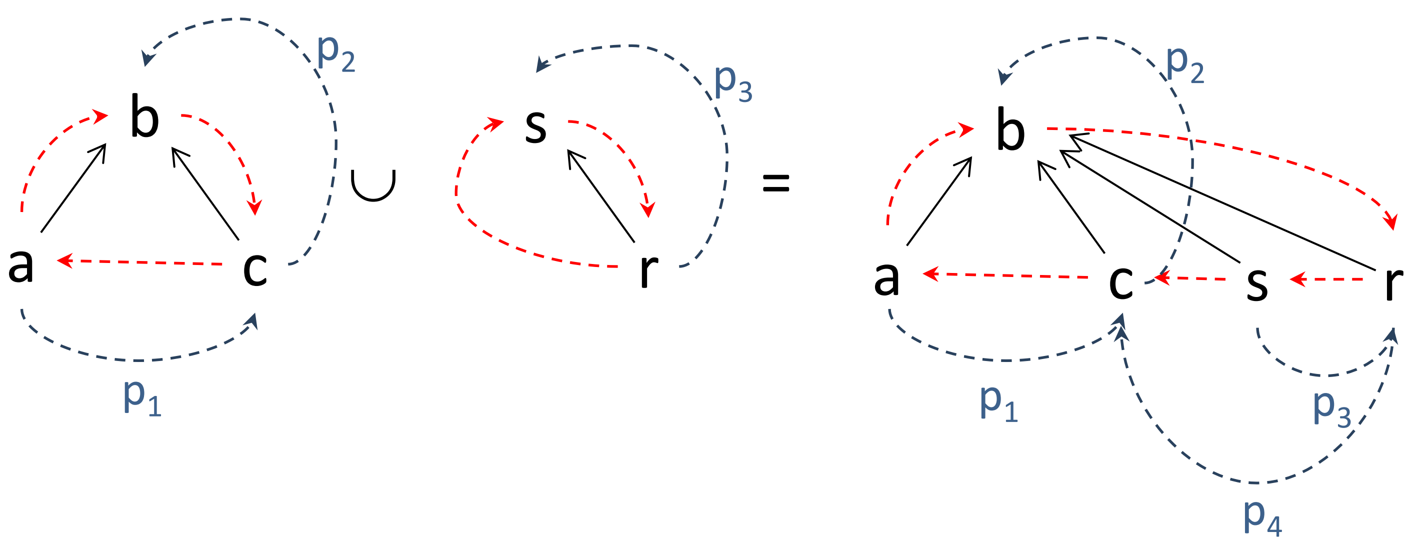
Second idea: Store a "proof tree".

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

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

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

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

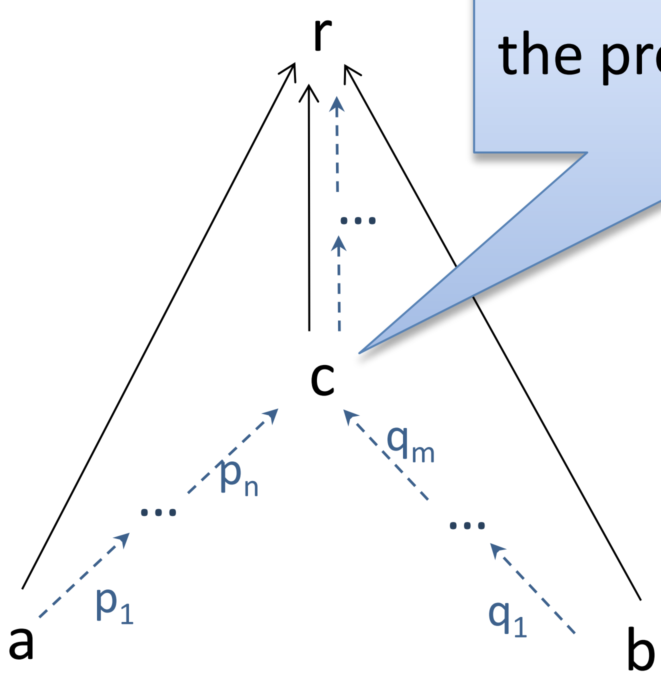


Microsoft Research

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



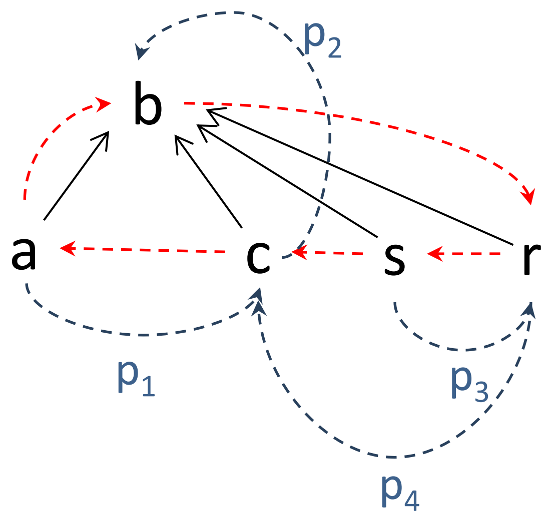
Common ancestor in the proof tree.

Non redundant proof for **a = b**

$p_1, \dots, p_n, q_1, \dots, q_m$

Deciding Equality + (uninterpreted) Functions

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

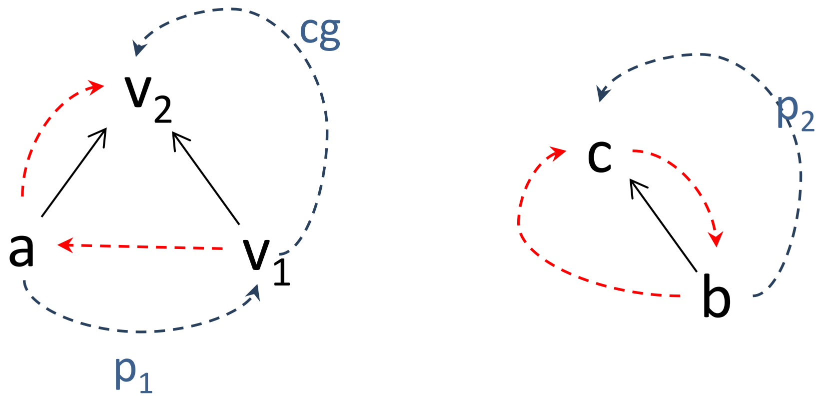


Deciding Equality + (uninterpreted) Functions

What about congruence?

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

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

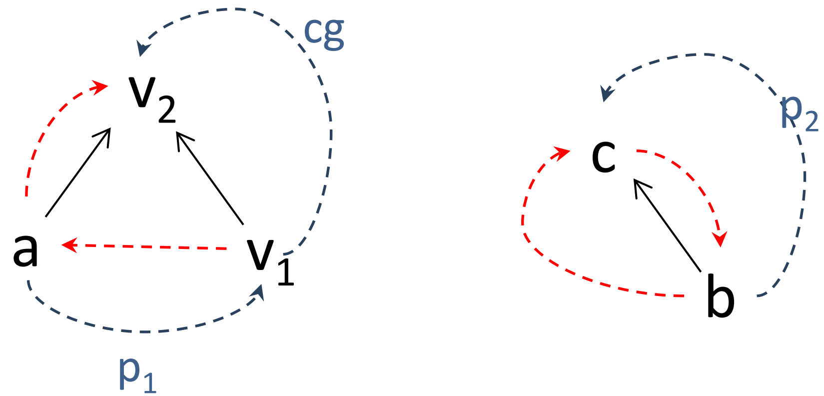


Deciding Equality + (uninterpreted) Functions

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: $\{p_1, p_2\}$

Microsoft Research

Deciding Equality + (uninterpreted) Functions

The new algorithm may compute redundant proofs for EUF.

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

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

Microsoft Research

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.

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 $\{q_1, q_2, q_3, q_4\}$

Deciding Equality + (uninterpreted) Functions

Some benchmarks are very hard for our procedure.

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

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

$\ldots,$

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

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

Microsoft Research

Deciding Equality + (uninterpreted) Functions

Some benchmarks are very hard for our procedure.

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

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

$\ldots,$

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

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

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.

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

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

$\ldots,$

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

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

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:

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

Microsoft Research

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:

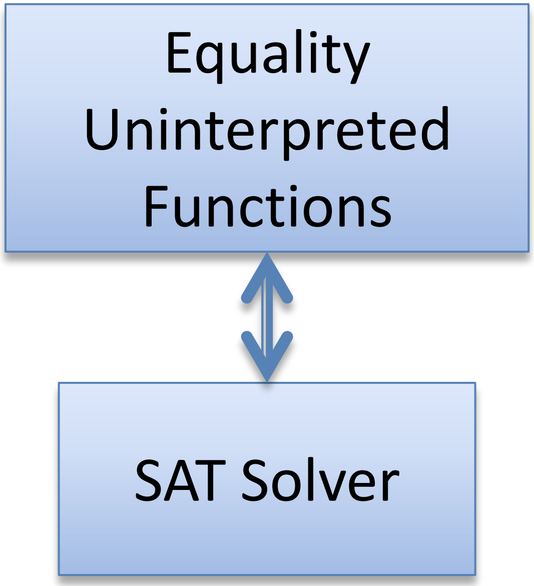
$a_i \neq b_i \lor a_j \neq b_j \lor f(a_i, a_j) = f(b_i, b_j)$

"Dynamic Ackermannization"

It allows the solver to perform the missing disequality propagation.

Microsoft Research

Summary



We can solve the QF_UF SMT-Lib benchmarks!