Decision Procedures in Verification

Decision Procedures (2)

17.12.2012

Viorica Sofronie-Stokkermans

e-mail: sofronie@uni-koblenz.de

Until now:

Logical Theories; Decision procedures

Generalities

The theory of uninterpreted function symbols (UIF): Motivation

3.3. Theory of Uninterpreted Function Symbols

Application: Compiler Verification

Example: prove equivalence of source and target program

```
1:  y := 1
2:  if z = x*x*x
3:    then y := x*x + y
4:  endif
```

```
1: y := 1
2: R1 := x*x
3: R2 := R1*x
4: jmpNE(z,R2,6)
5: y := R1+1
```

To prove: (indexes refer to values at line numbers)

$$y_1 \approx 1 \wedge [(z_0 \approx x_0 * x_0 * x_0 \wedge y_3 \approx x_0 * x_0 + y_1) \vee (z_0 \not\approx x_0 * x_0 * x_0 \wedge y_3 \approx y_1)] \wedge$$

$$y_1' \approx 1 \wedge R1_2 \approx x_0' * x_0' \wedge R2_3 \approx R1_2 * x_0' \wedge$$

$$\wedge [(z_0' \approx R2_3 \wedge y_5' \approx R1_2 + 1) \vee (z_0' \neq R2_3 \wedge y_5' \approx y_1')] \wedge$$

$$x_0 \approx x_0' \wedge y_0 \approx y_0' \wedge z_0 \approx z_0' \implies x_0 \approx x_0' \wedge y_3 \approx y_5' \wedge z_0 \approx z_0'$$

Uninterpreted function symbols

Let $\Sigma = (\Omega, \Pi)$ be arbitrary

Let $\mathcal{M} = \Sigma$-alg be the class of all Σ-structures

The theory of uninterpreted function symbols is Th(Σ-alg) the family of all first-order formulae which are true in all Σ-algebras.

in general undecidable

Decidable fragment:

e.g. the class $\text{Th}_\forall(\Sigma\text{-alg})$ of all universal formulae which are true in all Σ-algebras.

Theorem 3.3.1

The following are equivalent:

(1) testing validity of universal formulae w.r.t. UIF is decidable

(2) testing validity of (universally quantified) clauses w.r.t. UIF is decidable

Solution 1

Task:

Check if $UIF \models \forall \overline{x}(s_1(\overline{x}) \approx t_1(\overline{x}) \wedge \cdots \wedge s_k(\overline{x}) \approx t_k(\overline{x}) \rightarrow \bigvee_{j=1}^m s'_j(\overline{x}) \approx t'_j t(\overline{x}))$

Solution 1:

The following are equivalent:

(1) $(\bigwedge_i s_i \approx t_i) \rightarrow \bigvee_j s'_j \approx t'_j$ is valid

(2) $Eq(\sim) \wedge \text{Con}(f) \wedge (\bigwedge_i s_i \sim t_i) \wedge (\bigwedge_j s'_j \not\sim t'_j)$ is unsatisfiable.

where $Eq(\sim)$: $\text{Refl}(\sim) \wedge \text{Sim}(\sim) \wedge \text{Trans}(\sim)$

$\text{Con}(f)$: $\forall x_1, \dots, x_n, y_1, \dots, y_n (\bigwedge x_i \sim y_i \rightarrow f(x_1, \dots, x_n) \sim f(y_1, \dots, y_n))$

Resolution: inferences between transitivity axioms – nontermination

Solution 2

Task:
Check if $UIF \models \forall \overline{x}(s_1(\overline{x}) \approx t_1(\overline{x}) \wedge \dots \wedge s_k(\overline{x}) \approx t_k(\overline{x}) \rightarrow \bigvee_{j=1}^{m} s'_j(\overline{x}) \approx t'_j(\overline{x}))$

Solution 2: Ackermann's reduction.

Flatten the formula (replace, bottom-up, $f(c)$ with a new constant c_f
$\phi \mapsto FLAT(\phi)$

Theorem 3.3.2: The following are equivalent:

(1) $(\bigwedge_i s_i(\overline{c}) \approx t_i(\overline{c})) \wedge \bigwedge_j s'_j(\overline{c}) \not\approx t'_j(\overline{c})$ is satisfiable
(2) $FC \wedge FLAT[(\bigwedge_i s_i(\overline{c}) \approx t_i(\overline{c})) \wedge \bigwedge_j s'_j(\overline{c}) \not\approx t'_j(\overline{c})]$ is satisfiable

where $FC = \{c_1{=}d_1, \dots c_n{=}d_n \rightarrow c_f{=}d_f \mid$ whenever $f(c_1, \dots, c_n)$ was renamed to c_f
$f(d_1, \dots, d_n)$ was renamed to $d_f\}$

Note: The problem is decidable in PTIME (see next pages)
Problem: Naive handling of transitivity/congruence axiom $\mapsto O(n^3)$
Goal: Give a faster algorithm

Example

The following are equivalent:

(1) $C := f(a, b) \approx a \wedge f(f(a, b), b) \not\approx a$

(2) $FC \wedge FLAT[C]$, where:

$FLAT[f(a, b) \approx a \wedge f(f(a, b), b) \not\approx a]$ is computed by introducing new constants renaming terms starting with f and then replacing in C the terms with the constants:

- $FLAT[\underbrace{f(a, b)}_{a_1} \approx a \wedge \underbrace{f(\underbrace{f(a, b)}_{a_1}, b)}_{a_2} \not\approx a] := a_1 \approx a \wedge a_2 \not\approx a$

$$f(a, b) = a_1$$
$$f(a_1, b) = a_2$$

- $FC := (a \approx a_1 \rightarrow a_1 \approx a_2)$

Thus, the following are equivalent:

(1) $C := f(a, b) \approx a \wedge f(f(a, b), b) \not\approx a$

(2) $\underbrace{(a \approx a_1 \rightarrow a_1 \approx a_2)}_{FC} \wedge \underbrace{a_1 \approx a \wedge a_2 \not\approx a}_{FLAT[C]}$

Solution 3

Task:

Check if $UIF \models \forall \overline{x}(s_1(\overline{x}) \approx t_1(\overline{x}) \wedge \cdots \wedge s_k(\overline{x}) \approx t_k(\overline{x}) \rightarrow \bigvee_{j=1}^{m} s'_j(\overline{x}) \approx t'_j(\overline{x}))$

i.e. if $(s_1(\overline{c}) \approx t_1(\overline{c}) \wedge \cdots \wedge s_k(\overline{c}) \approx t_k(\overline{c}) \wedge \bigwedge_j s'_j(\overline{c}) \not\approx t'_j(\overline{c}))$ unsatisfiable.

Solution 3

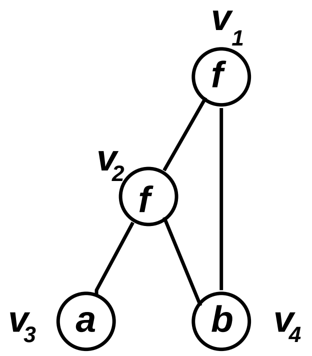
Task:

Check if $(s_1(\overline{c}) \approx t_1(\overline{c}) \wedge \cdots \wedge s_k(\overline{c}) \approx t_k(\overline{c}) \wedge \bigwedge_k s'_k(\overline{c}) \not\approx t'_k(\overline{c}))$ unsatisfiable.

Solution 3 [Downey-Sethi, Tarjan'76; Nelson-Oppen'80]

represent the terms occurring in the problem as DAG's

Example: Check whether $f(f(a, b), b) \approx a$ is a consequence of $f(a, b) \approx a$.



$$
\begin{aligned}
v_1 &: \quad f(f(a, b), b) \\
v_2 &: \quad f(a, b) \\
v_3 &: \quad a \\
v_4 &: \quad b
\end{aligned}
$$

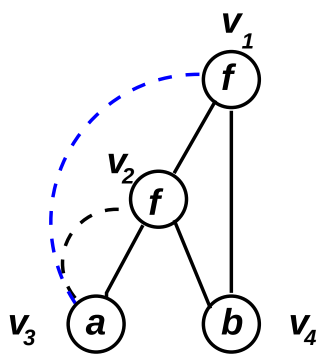
Solution 3

Task: Check if $(s_1(\overline{c})\approx t_1(\overline{c}) \wedge \cdots \wedge s_k(\overline{c})\approx t_k(\overline{c}) \wedge s(\overline{c})\not\approx t(\overline{c}))$ unsatisfiable.

Solution 3 [Downey-Sethi, Tarjan'76; Nelson-Oppen'80]

- represent the terms occurring in the problem as DAG's
- represent premise equalities by a relation on the vertices of the DAG

Example: Check whether $f(f(a, b), b) \approx a$ is a consequence of $f(a, b) \approx a$.



$$
\begin{array}{ll}
v_1: & f(f(a, b), b) \\
v_2: & f(a, b) \\
v_3: & a \\
v_4: & b \\
R: & \{(v_2, v_3)\}
\end{array}
$$

- compute the "congruence closure" R^c of R
- check whether $(v_1, v_3) \in R^c$

Computing the congruence closure of a DAG

Example

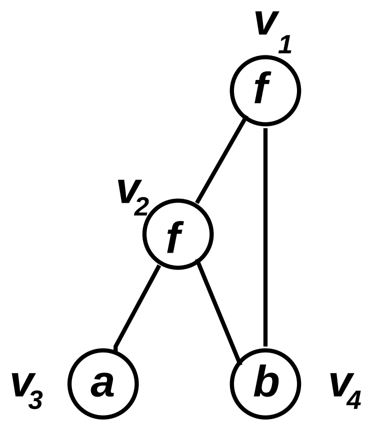
- **DAG structures:**

  - $G = (V, E)$ directed graph

  - Labelling on vertices

    $\lambda(v)$: label of vertex v
    $\delta(v)$: outdegree of vertex v

  - Edges leaving the vertex v are ordered
    ($v[i]$: denotes i-th successor of v)



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

$\delta(v_1) = \delta(v_2) = 2$
$\delta(v_3) = \delta(v_4) = 0$

$v_1[1] = v_2$, $v_2[2] = v_4$

...

Congruence closure of a DAG/Relation

Given: $G = (V, E)$ DAG + labelling

$R \subseteq V \times V$

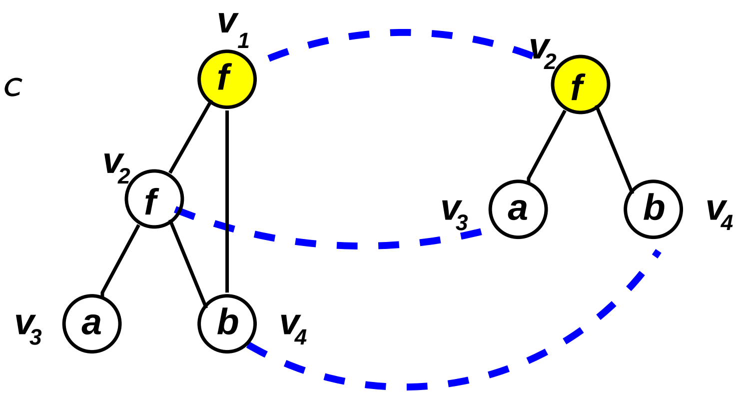
The congruence closure of R is the smallest relation R^c on V which is:

- reflexive
- symmetric
- transitive
- congruence:

  If $\lambda(u) = \lambda(v)$ and $\delta(u) = \delta(v)$

  and for all $1 \leq i \leq \delta(u)$: $(u[i], v[i]) \in R^c$

  then $(u, v) \in R^c$.



Congruence closure of a relation

Recursive definition

$$\frac{(u, v) \in R}{(u, v) \in R^c}$$

$$\frac{}{(v, v) \in R^c} \qquad \frac{(u, v) \in R^c}{(v, u) \in R^c} \qquad \frac{(u, v) \in R^c \quad (v, w) \in R^c}{(u, w) \in R^c}$$

$$\frac{\lambda(u) = \lambda(v) \quad u, v \text{ have } n \text{ successors} \quad \text{and } (u[i], v[i]) \in R^c \text{ for all } 1 \leq i \leq n}{(u, v) \in R^c}$$

- The congruence closure of R is the smallest set closed under these rules

Congruence closure and UIF

Assume that we have an algorithm $\mathbb{A}$ for computing the congruence closure of a graph G and a set R of pairs of vertices

- Use $\mathbb{A}$ for checking whether $\bigwedge_{i=1}^{n} s_i \approx t_i \wedge \bigwedge_{j=1}^{m} s'_j \not\approx t'_j$ is satisfiable.

  (1) Construct graph corresponding to the terms occurring in s_i, t_i, s'_j, t'_j

  Let v_t be the vertex corresponding to term t

  (2) Let $R = \{(v_{s_i}, v_{t_i}) \mid i \in \{1, \dots, n\}\}$

  (3) Compute R^c.

  (4) Output "Sat" if $(v_{s'_j}, v_{t'_j}) \notin R^c$ for all $1 \leq j \leq m$, otherwise "Unsat"

Theorem 3.3.3 (Correctness)

$\bigwedge_{i=1}^{n} s_i \approx t_i \wedge \bigwedge_{j=1}^{m} s'_j \not\approx t'_j$ is satisfiable iff $[v_{s'_j}]_{R^c} \neq [v_{t'_j}]_{R^c}$ for all $1 \leq j \leq m$.

Congruence closure and UIF

Theorem 3.3.3 (Correctness)

$\bigwedge_{i=1}^{n} s_i \approx t_i \wedge \bigwedge_{j=1}^{m} s'_j \not\approx t'_j$ is satisfiable iff $[v_{s'_j}]_{R^c} \neq [v_{t'_j}]_{R^c}$ for all $1 \leq j \leq m$.

Proof ($\Rightarrow$)

Assume $\mathcal{A}$ is a Σ-structure such that $\mathcal{A} \models \bigwedge_{i=1}^{n} s_i \approx t_i \wedge \bigwedge_{j=1}^{m} s'_j \not\approx t'_j$.

We can show that $[v_s]_{R^c} = [v_t]_{R^c}$ implies that $\mathcal{A} \models s = t$ (Exercise).

(We use the fact that if $[v_s]_{R^c} = [v_t]_{R^c}$ then there is a derivation for $(v_s, v_t) \in R^c$ in the calculus defined before; use induction on length of derivation to show that $\mathcal{A} \models s = t$.)

As $\mathcal{A} \models s'_j \not\approx t'_j$, it follows that $[v_{s'_j}]_{R^c} \neq [v_{t'_j}]_{R^c}$ for all $1 \leq j \leq m$.

Congruence closure and UIF

Theorem 3.3.3 (Correctness)

$\bigwedge_{i=1}^{n} s_i \approx t_i \wedge \bigwedge_{j=1}^{m} s_j' \not\approx t_j'$ is satisfiable iff $[v_{s_j'}]_{R^c} \neq [v_{t_j'}]_{R^c}$ for all $1 \leq j \leq m$.

Proof($\Leftarrow$) Assume that $[v_{s_j'}]_{R^c} \neq [v_{t_j'}]_{R^c}$ for all $1 \leq j \leq m$. We construct a structure that satisfies $\bigwedge_{i=1}^{n} s_i \approx t_i \wedge \bigwedge_{j=1}^{m} s_j' \not\approx t_j'$

- Universe is quotient of V w.r.t. R^c plus new element 0.
- c constant $\mapsto c_{\mathcal{A}} = [v_c]_{R^c}$.
- $f/n \mapsto f_{\mathcal{A}}([v_1]_{R^c}, \dots, [v_n]_{R^c}) = \begin{cases} [v_{f(t_1,\dots,t_n)}]_{R^c} & \text{if } v_{f(t_1,\dots,t_n)} \in V, \\ & [v_{t_i}]_{R^c} = [v_i]_{R^c} \text{ for } 1 \leq i \leq n \\ 0 & \text{otherwise} \end{cases}$

well-defined because R^c is a congruence.

- It holds that $\mathcal{A} \models s_j' \not\approx t_j'$ and $\mathcal{A} \models s_i \approx t_i$

Computing the congruence closure of a DAG

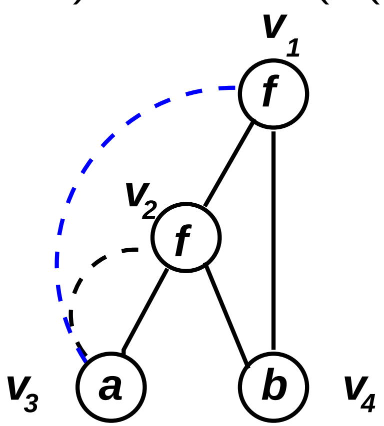
Given: $G = (V, E)$ DAG + labelling

$R \subseteq V \times V$

Task: Compute R^c (the congruence closure of R)

Example:

$f(a, b) \approx a \rightarrow f(f(a, b), b) \approx a$



$R = \{(v_2, v_3)\}$

Idea:

- Start with the identity relation $R^c = Id$
- Successively add new pairs of nodes to R^c;
  close relation under congruence.

Task: Compute R^c

Computing the congruence closure of a DAG

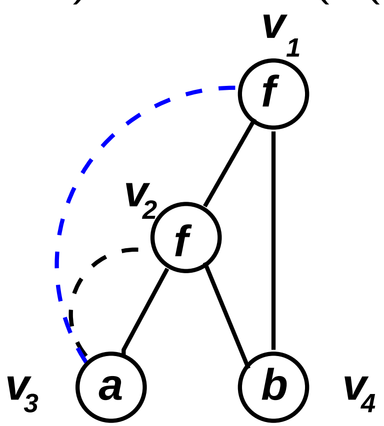
Given: $G = (V, E)$ DAG + labelling

$R \subseteq V \times V$; $(v, v') \in V^2$

Task: Check whether $(v, v') \in R^c$

Example:

$f(a, b) \approx a \rightarrow f(f(a, b), b) \approx a$



$R = \{(v_2, v_3)\}$

Idea:

- Start with the identity relation $R^c = Id$
- Successively add new pairs of nodes to R^c; close relation under congruence.

Task: Decide whether $(v_1, v_3) \in R^c$

Computing the congruence closure of a DAG

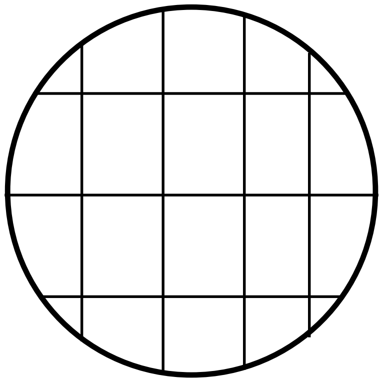
Given: $G = (V, E)$ DAG + labelling

$R \subseteq V \times V$

Task: Compute R^c (the congruence closure of R)

Idea: Recursively construct relations closed under congruence R_i (approximating R^c) by identifying congruent vertices u, v and computing $R_{i+1} :=$ congruence closure of $R_i \cup \{(u, v)\}$.

Representation:



- Congruence relation $\mapsto$ corresponding partition

Computing the congruence closure of a DAG

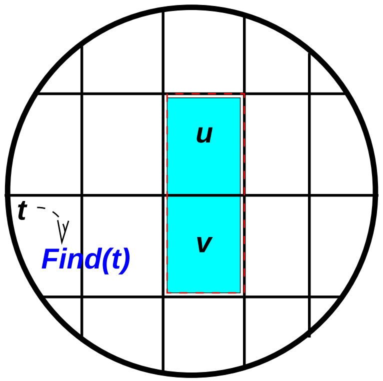
Given: $G = (V, E)$ DAG + labelling

$R \subseteq V \times V$

Task: Compute R^c (the congruence closure of R)

Idea: Recursively construct relations closed under congruence R_i (approximating R^c) by identifying congruent vertices u, v and computing $R_{i+1} :=$ congruence closure of $R_i \cup \{(u, v)\}$.

Representation:



- Congruence relation $\mapsto$ corresponding partition
- Use procedures which operate on the partition:

  FIND(u): unique name of equivalence class of u

  UNION(u, v) combines equivalence classes of u, v

  finds repr. t_u, t_v of equiv.cl. of u, v; sets FIND(u) to t_v

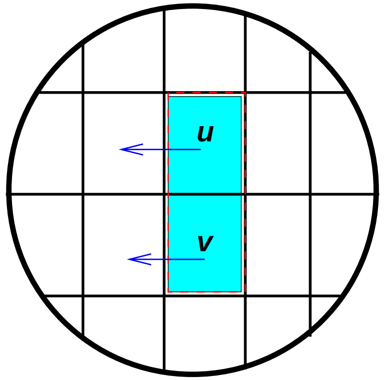
Computing the congruence closure of a DAG

MERGE(u, v)

g

Input:	$G = (V, E)$ DAG + labelling
	R relation on V closed under congruence
	$u, v \in V$
Output:	the congruence closure of $R \cup \{(u, v)\}$

If FIND(u) = FIND(v) [same canonical representative] **then** Return
If FIND$(u) \neq$ FIND(v) **then** [merge u, v; recursively-predecessors]
P_u := set of all predecessors of vertices w with FIND(w) = FIND(u)
P_v := set of all predecessors of vertices w with FIND(w) = FIND(v)
Call UNION(u, v) [merge congruence classes]
For all $(x, y) \in P_u \times P_v$ **do**: [merge congruent predecessors]
if FIND$(x) \neq$ FIND(y) **and** CONGRUENT(x, y) **then** MERGE(x, y)



CONGRUENT(x, y)

if $\lambda(x) \neq \lambda(y)$ **then** Return FALSE
For $1 \leq i \leq \delta(x)$ **if** FIND$(x[i]) \neq$ FIND$(y[i])$ **then** Return FALSE

Return TRUE.

Correctness

Proof:

(1) Returned equivalence relation is not too coarse

If x, y merged then $(x, y) \in (R \cup \{(u, v)\})^c$
(UNION only on initial pair and on congruent pairs)

(2) Returned equivalence relation is not too fine

If x, y vertices s.t. $(x, y) \in (R \cup \{(u, v)\})^c$ then they are merged by the algorithm.
Induction of length of derivation of (x, y) from $(R \cup \{(u, v)\})^c$

(1) $(x, y) \in R$ OK (they are merged)
(2) $(x, y) \not\in R$. The only non-trivial case is the following:
$\lambda(x) = \lambda(y), x, y$ have n successors x_i, y_i where
$(x_i, y_i) \in (R \cup \{(u, v)\})^c$ for all $1 \leq i \leq b$.
Induction hypothesis: (x_i, y_i) are merged at some point
(become equal during some call of UNION(a, b), made in some MERGE(a, b))
Successor of x equivalent to a (or b) before this call of UNION; same for y.

$\Rightarrow$ MERGE must merge x and y

Computing the Congruence Closure

Let $G = (V, E)$ graph and $R \subseteq V \times V$

$CC(G, R)$ computes the R^c:

(1) $R_0 := \emptyset$; $i := 1$

(2) while R contains "fresh" elements do:

pick "fresh" element $(u, v) \in R$

$R_i :=$ MERGE(u, v) for G and R_{i-1}; $i := i + 1$.

Complexity: $O(n^2)$
Downey-Sethi-Tarjan congruence closure algorithm:
more sophisticated version of MERGE (complexity $O(n \cdot logn)$)

Reference: G. Nelson and D.C. Oppen. Fast decision procedures based on congruence closure. Journal of the ACM, 27(2):356-364, 1980.

Decision procedure for the QF theory of equality

Signature: Σ (function symbols)

Problem: Test satisfiability of the formula

$$F \quad = \quad s_1 \approx t_1 \wedge \cdots \wedge s_n \approx t_n \quad \wedge \quad s'_1 \not\approx t'_1 \wedge \cdots \wedge s'_m \not\approx t'_m$$

Solution: Let S_F be the set of all subterms occurring in F

1. Construct the DAG for S_F; $R_0 = Id$
2. [Build R_n the congruence closure of $\{(v(s_1), v(t_1)), \dots, (v(s_n), v(t_n))\}$]

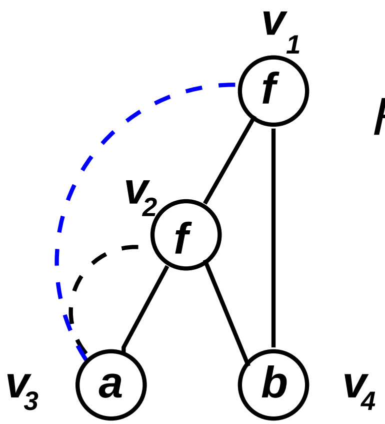
   For $i \in \{1, \dots, n\}$ **do** $R_i :=$ MERGE(v_{s_i}, v_{t_i}) w.r.t. R_{i-1}
3. **If** FIND$(v_{s'_j})$ = FIND$(v_{t'_j})$ for some $j \in \{1, \dots, m\}$ **then** return unsatisfiable
4. **else** [if FIND$(v_{s'_j}) \neq$ FIND$(v_{t'_j})$ for all $j \in \{1, \dots, m\}$] **then** return satisfiable

Example

$f(a, b) \approx a \rightarrow f(f(a, b), b) \approx a$

Test: unsatisfiability of

$f(a, b) \approx a \wedge f(f(a, b), b) \not\approx a$



$R = \{(v_2, v_3)\}$

Task:

- Compute R^c
- Decide whether $(v_1, v_3) \in R^c$

Solution:

1. Construct DAG in the figure; $R_0 = Id$.
2. Compute $R_1 :=$ MERGE$((v_2, v_3)$

   [Test representatives]

   FIND$(v_2) = v_2 \neq v_3 =$ FIND(v_3)

   $P_{v_2} := \{v_1\}$; $P_{v_3} := \{v_2\}$

   [Merge congruence classes]

   UNION(v_2, v_3): sets FIND(v_2) to v_3.

   [Compute and recursively merge predecessors]

   Test: FIND$(v_1) = v_1 \neq v_3 =$ FIND(v_2)

   CONGR(v_1, v_2)

   MERGE(v_1, v_2): (different representatives)

   calls UNION(v_1, v_2) which

   sets FIND(v_1) to v_3.

3. Test whether FIND$(v_1) =$ FIND(v_3). Yes.

   Return unsatisfiable.

3.4. Decision procedures for numeric domains

- Peano arithmetic
- Theory of real numbers

- Linear arithmetic
  - over $\mathbb{N}/\mathbb{Z}$
  - over $\mathbb{R}/\mathbb{Q}$

Decision procedures

- Light-weight fragments of linear arithmetic: Difference logic
- Full fragment ($LI(\mathbb{R})$ or $LI(\mathbb{Q})$)

Peano arithmetic

Peano axioms:

$$\forall x\, \neg(x + 1 \approx 0) \qquad \text{(zero)}$$

$$\forall x \forall y\, (x + 1 \approx y + 1 \rightarrow x \approx y \qquad \text{(successor)}$$

$$F[0] \wedge (\forall x\, (F[x] \rightarrow F[x + 1]) \rightarrow \forall x F[x]) \qquad \text{(induction)}$$

$$\forall x\, (x + 0 \approx x) \qquad \text{(plus zero)}$$

$$\forall x, y\, (x + (y + 1) \approx (x + y) + 1) \qquad \text{(plus successor)}$$

$$\forall x, y\, (x * 0 \approx 0) \qquad \text{(times 0)}$$

$$\forall x, y\, (x * (y + 1) \approx x * y + x) \qquad \text{(times successor)}$$

$3 * y + 5 > 2 * y$ expressed as $\exists z (z \not\approx 0 \wedge 3 * y + 5 \approx 2 * y + z)$

Intended interpretation: $(\mathbb{N}, \{0, 1, +, *\}, \{<\})$ (also with $\approx$)

(does not capture true arithmetic by Goedel's incompleteness theorem)

Undecidable

Theory of integers

- $\mathsf{Th}((\mathbb{Z}, \{0, 1, +, *\}, \{<\}))$

Undecidable

Theory of real numbers

Theory of real closed fields (real closed fields: fields with same properties as real numbers)

Axioms:

- the ordered field axioms;
- axiom asserting that every positive number has a square root; and
- an axiom scheme asserting that all polynomials of odd order have at least one real root.

Tarski (1951) proved that the theory of real closed fields, including the binary predicate symbols "$=$", "$\neq$", and "$<$", and the operations of addition and multiplication, admits elimination of quantifiers, which implies that it is a complete and decidable theory.

Linear arithmetic

Syntax

- Signature $\Sigma = (\{0/0, s/1, +/2\}, \{< /2\})$
- Terms, atomic formulae – as usual

Example: $3 * x_1 + 2 * x_2 \leq 5 * x_3$ abbreviation for

$$(x_1 + x_1 + x_1) + (x_2 + x_2) \leq (x_3 + x_3 + x_3 + x_3 + x_3)$$

Linear arithmetic

There are several ways to define linear arithmetic.

We need at least the following signature: $\Sigma = (\{0/0, 1/0, +/2\}, \{< /2\})$ and the predefined binary predicate $\approx$.

Linear arithmetic

There are several ways to define linear arithmetic.

We need at least the following signature: $\Sigma = (\{0/0, 1/0, +/2\}, \{< /2\})$ and the predefined binary predicate $\approx$.

Linear arithmetic over $\mathbb{N}/\mathbb{Z}$

Th($\mathbb{Z}_+$) $\quad \mathbb{Z}_+ = (\mathbb{Z}, 0, s, +, <)$ the standard interpretation of integers.

Axiomatization

Linear arithmetic over $\mathbb{Q}/\mathbb{R}$

Th($\mathbb{R}$) $\quad \mathbb{R} = (\mathbb{R}, \{0, 1, +\}, \{<\})$ the standard interpretation of reals;

Th($\mathbb{Q}$) $\quad \mathbb{Q} = (\mathbb{Q}, \{0, 1, +\}, \{<\})$ the standard interpretation of rationals.

Axiomatization

Outline

We first present an efficient method for checking the satisfiability of formulae in a very simple fragment of linear arithmetic.

We will then give more details about possibilities of checking the satisfiability of arbitrary formulae in linear arithmetic.

Simple fragments of linear arithmetic

- Difference logic

  check satisfiability of conjunctions of constraints of the form

$$x - y \leq c$$

- UTVPI (unit, two variables per identity)

  check satisfiability of conjunctions of constraints of the form

$$ax + by \leq c, \text{ where } a, b \in \{-1, 0, 1\}$$

Application: Program Verification

```
i := 1,  n < m
while i < n
do
   i := i + 1
    [** part of a program in which  i  is used as an index in an array
        which was declared to be of size s > m (and i is not changed)
     **]
   ....
od
```

Task: $i \leq s$ always during the execution of this program.

Application: Program Verification

```
i := 1,  n < m
while i < n
do
   i := i + 1
    [** part of a program in which  i  is used as an index in an array
        which was declared to be of size s > m (and i is not changed)
     **]
   ....
od
```

Task: $i \leq s$ always during the execution of this program.

Solution: Show that this is true at the beginning and remains true after every update of i.

Application: Program Verification

```
i := 1,  n < m
while i < n
do
   i := i + 1
    [** part of a program in which  i  is used as an index in an array
        which was declared to be of size s > m (and i is not changed)
     **]
   ....
od
```

Task: $i \leq s$ always during the execution of this program.

Solution: Show that $i \leq s$ is an invariant of the program:

1) It holds at the first line: $i = 1 \rightarrow i \leq s$

2) It is preserved under the updates in the while loop:
$\forall n, m, s, i, i' \ \ (n < m \wedge 1 < m < s \wedge i \leq n \wedge i \leq s \wedge i' \approx i + 1 \rightarrow i' \leq s)$

Positive difference logic

Syntax

The syntax of formulae in positive difference logic is defined as follows:

- Atomic formulae (also called difference constraints) are of the form:

  $$x - y \leq c$$

  where x, y are variables and c is a numerical constant.

- The set of formulae is:

  $$\begin{array}{lcll} F, G, H & ::= & A & \text{(atomic formula)} \\ & \mid & (F \wedge G) & \text{(conjunction)} \end{array}$$

Semantics:

Versions of difference logic exist, where the standard interpretation is $\mathbb{Q}$ or resp. $\mathbb{Z}$.

Positive difference logic

A decision procedure for positive difference logic ($\leq$ only)

Let S be a set (i.e. conjunction) of atoms in (positive) difference logic. $G(S) = (V, E, w)$, the inequality graph of S, is a weighted graph with:

- $V = X(S)$, the set of variables occurring in S
- $e = (x, y) \in E$ with $w(e) = c$ iff $x - y \leq c \in S$

Theorem 3.4.1.
Let S be a conjunction of difference constraints, and $G(S)$ the inequality graph of S. Then S is satisfiable iff there is no negative cycle in $G(S)$.

Searching for negative cycles in a graph can be done with the Bellman-Ford algorithm for finding the single-source shortest paths in a directed weighted graph in time $O(|V| \cdot |E|)$. (Side-effect of the algorithm exploited - if there exists a negative cycle in the graph then the algorithm finds it and aborts.)

Positive difference logic

Theorem 3.4.1.
Let S be a conjunction of difference constraints, and $G(S)$ the inequality graph of S. Then S is satisfiable iff there is no negative cycle in $G(S)$.

Proof: ($\Rightarrow$) Assume S satisfiable. Let $\beta : X \rightarrow \mathbb{Z}$ satisfying assignment.
Let $v_1 \stackrel{c_{12}}{\rightarrow} v_2 \stackrel{c_{23}}{\rightarrow} \dots \stackrel{c_{n-1,n}}{\rightarrow} v_n \stackrel{c_{n1}}{\rightarrow} v_1$ be a cycle in $G(S)$.

$$
\begin{array}{rrcl}
\text{Then:} & \beta(v_1) - \beta(v_2) & \leq & c_{12} \\
 & \beta(v_2) - \beta(v_3) & \leq & c_{23} \\
 & \dots & & \\
g & \beta(v_n) - \beta(v_1) & \leq & c_{n1} \\
\hline
0 = & \beta(v_1) - \beta(v_1) & \leq & \sum_{i=1}^{n-1} c_{i,i+1} + c_{n1}
\end{array}
$$

Thus, for satisfiability it is necessary that all cycles are positive.

Positive difference logic

Theorem 3.4.1.
Let S be a conjunction of difference constraints, and $G(S)$ the inequality graph of S. Then S is satisfiable iff there is no negative cycle in $G(S)$.

Proof: ($\Leftarrow$) Assume that there is no negative cycle.

Add a new vertex s and an 0-weighted edge from every vertex in V to s. (This does not introduce negative cycles.)

Let δ_{uv} denote the minimal weight of the paths from u to v.

- $\delta_{uv} = \infty$ if there is no path from u to v.
- well-defined since there are no negative cycles

Define $\beta : V \longrightarrow \mathbb{Z}$ by $\beta(v) = \delta_{vs}$. Claim: β satisfying assignment for S.

Let $x - y \leq c \in S$. Consider the shortest paths from x to s and from y to s. By the triangle inequality, $\delta_{xs} \leq c + \delta_{ys}$, i.e. $\beta(x) - \beta(y) \leq c$.