Decision Procedures in Verification

Decision Procedures (3)

22.12.2022

Viorica Sofronie-Stokkermans

e-mail: sofronie@uni-koblenz.de

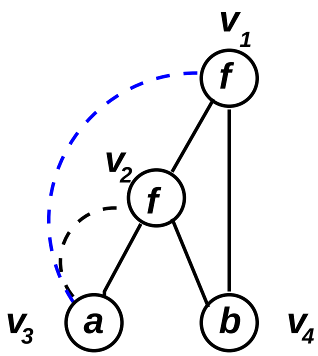
Solution 3

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



v_1 : $f(f(a, b), b)$
v_2 : $f(a, b)$
v_3 : a
v_4 : b

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

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

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 $1 \leq j \leq m$.

Computing the congruence closure of a DAG

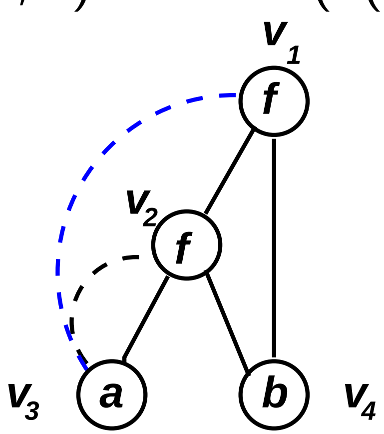
Given: $G = (V, E)$ DAG + labelling

$R \subseteq V \times V$

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

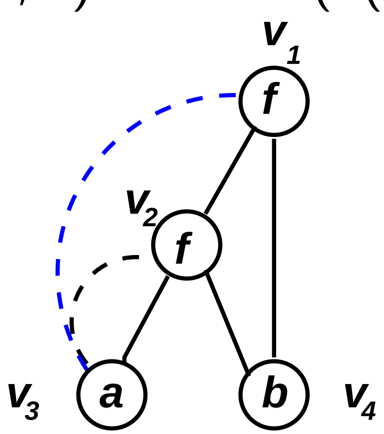
Given: $G = (V, E)$ DAG + labelling

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

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

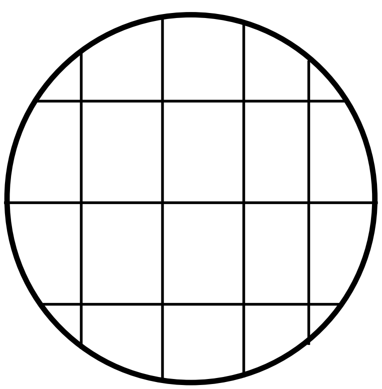
Given: $G = (V, E)$ DAG + labelling

$R \subseteq V \times V$

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

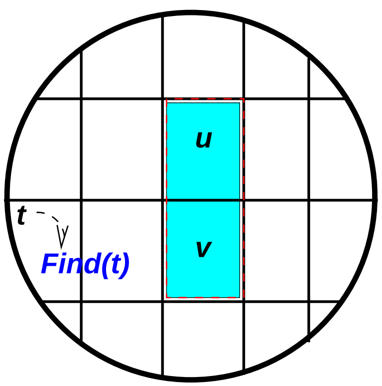
Given: $G = (V, E)$ DAG + labelling

$R \subseteq V \times V$

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

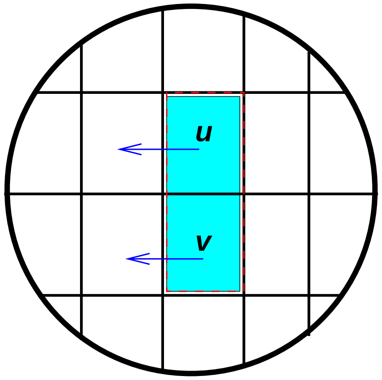
Computing the congruence closure of a DAG

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) $(x, y) \notin R$. 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 $1 \leq i \leq n$.
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 $\{(v(s_1), v(t_1)), \dots, (v(s_n), v(t_n))\}$]

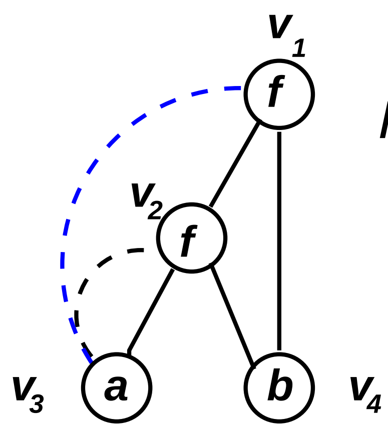
   For $i \in \{1, \dots, n\}$ **do** $R_i :=$ MERGE(v_{s_i}, v_{t_i}) w.r.t. R_{i-1}
3. **If** FIND$(v_{s'_j})$ $=$ FIND$(v_{t'_j})$ for some $j \in \{1, \dots, m\}$ **then** return unsatisfiable
4. **else** [if FIND$(v_{s'_j}) \neq$ FIND$(v_{t'_j})$ for all $j \in \{1, \dots, m\}$] **then** return satisfiable

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 $R_1 := \text{MERGE}((v_2, v_3)$

   [Test representatives]

   $\text{FIND}(v_2) = v_2 \neq v_3 = \text{FIND}(v_3)$

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

   [Merge congruence classes]

   $\text{UNION}(v_2, v_3)$: sets $\text{FIND}(v_2)$ to v_3.

   [Compute and recursively merge predecessors]

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

   $\text{CONGR}(v_1, v_2)$

   $\text{MERGE}(v_1, v_2)$: (different representatives)

   calls $\text{UNION}(v_1, v_2)$ which

   sets $\text{FIND}(v_1)$ to v_3.
3. Test whether $\text{FIND}(v_1) = \text{FIND}(v_3)$. 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 $\exists z(z \neq 0 \wedge 3 * y + 5 \approx 2 * y + z)$

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 inequality)

  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;            [**  where 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: Check whether $i \leq s$ always during the execution of this program.

Application: Program Verification

```
i := 1;               [**  where 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: Check whether $i \leq s$ 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;              [** where 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: Check whether $i \leq s$ 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:
$\forall n, m, s, i, i' \ \ (1 \leq n < m < s \wedge i < n \wedge i \leq s \wedge i' \approx i + 1 \rightarrow i' \leq s)$

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

Example - blackboard

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: next time.