Decision Procedures in Verification

Decision Procedures (3)

10.01.2022

Viorica Sofronie-Stokkermans

e-mail: sofronie@uni-koblenz.de
Until now:

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

i.e. if $(s_1(c) \approx t_1(c) \land \cdots \land s_k(c) \approx t_k(c) \land \bigwedge_j s_j'(c) \not\approx t_j'(c))$ unsatisfiable.
Solution 3

Task:
Check if \((s_1(c) \approx t_1(c) \land \cdots \land s_k(c) \approx t_k(c) \land \bigwedge_k s'_k(c) \not\approx t'_k(c))\) 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 \(f(a, b) \approx a\).
Computing the congruence closure of a DAG

- **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 $v$)

Example

$$\lambda(v_1) = \lambda(v_2) = f$$
$$\lambda(v_3) = a, \lambda(v_4) = b$$
$$\delta(v_1) = \delta(v_2) = 2$$
$$\delta(v_3) = \delta(v_4) = 0$$
$$v_1[1] = v_2, v_2[2] = v_4$$
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 contains \( R \) and 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 \( (u, v) \in R^c \).
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 $\bigwedge_{i=1}^{n} s_i \approx t_i \land \bigwedge_{j=1}^{m} s'_j \not\approx t'_j$ 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 $R = \{(v_{s_i}, v_{t_i}) \mid i \in \{1, \ldots, n\}\}$

3. Compute $R^c$.

4. Output “Sat” if $(v_{s'_j}, v_{t'_j}) \not\in R^c$ for all $1 \leq j \leq m$, otherwise “Unsat”

**Theorem 3.3.3 (Correctness)**

$\bigwedge_{i=1}^{n} s_i \approx t_i \land \bigwedge_{j=1}^{m} s'_j \not\approx t'_j$ 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:
\( f(a, b) \approx a \rightarrow f(f(a, b), b) \approx a \)

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 \( R_{i+1} := \text{congruence closure of } R_i \cup \{(u, v)\} \).

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:
  - \( \text{FIND}(u) \): unique name of equivalence class of \( u \)
  - \( \text{UNION}(u, v) \) combines equivalence classes of \( u, v \)
    finds repr. \( t_u, t_v \) of equiv.cl. of \( u, v \); sets \( \text{FIND}(u) \) to \( t_v \)
## Computing the congruence closure of a DAG

**MERGE**(\(u, v\))

<table>
<thead>
<tr>
<th>Input:</th>
<th>(G = (V, E)) DAG + labelling</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(R) relation on (V) closed under congruence</td>
</tr>
<tr>
<td></td>
<td>(u, v \in V)</td>
</tr>
</tbody>
</table>

| Output: | the congruence closure of \(R \cup \{(u, v)\}\) |

If \(\text{FIND}(u) = \text{FIND}(v)\) [same canonical representative] then Return

If \(\text{FIND}(u) \neq \text{FIND}(v)\) then [merge \(u, v\); recursively-predecessors]

\(P_u :=\) set of all predecessors of vertices \(w\) with \(\text{FIND}(w) = \text{FIND}(u)\)

\(P_v :=\) set of all predecessors of vertices \(w\) with \(\text{FIND}(w) = \text{FIND}(v)\)

Call \(\text{UNION}(u, v)\) [merge congruence classes]

For all \((x, y) \in P_u \times P_v\) do: [merge congruent predecessors]

if \(\text{FIND}(x) \neq \text{FIND}(y)\) and \(\text{CONGRUENT}(x, y)\) then \(\text{MERGE}(x, y)\)

**CONGRUENT**(\(x, y\))

if \(\lambda(x) \neq \lambda(y)\) then Return FALSE

For 1 \(\leq i \leq \delta(x)\) if \(\text{FIND}(x[i]) \neq \text{FIND}(y[i])\) then Return FALSE

Return TRUE.
Correctness

Proof:

(1) Returned equivalence relation is not too coarse

If \( x, y \) merged then \((x, y) \in (R \cup \{(u, v)\})^c\)

\((\text{UNION} \text{ only on initial pair and on congruent pairs})\)

(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 \( 1 \leq i \leq n \).

Induction hypothesis: \((x_i, y_i)\) are merged at some point

(become equal during some call of \(\text{UNION}(a, b)\), made in some \(\text{MERGE}(a, b)\))

Successor of \(x\) equivalent to \(a\) (or \(b\)) before this call of \(\text{UNION}\); same for \(y\).

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

Complexity: $O(n^2)$

Downey-Sethi-Tarjan congruence closure algorithm:
   more sophisticated version of $\text{MERGE}$ (complexity $O(n \cdot \log n)$)

Decision procedure for the QF theory of equality

Signature: $\Sigma$ (function symbols)

Problem: Test satisfiability of the formula

$$F = s_1 \approx t_1 \wedge \cdots \wedge s_n \approx t_n \wedge s_1' \not\approx t_1' \wedge \cdots \wedge s_m' \not\approx t_m'$$

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 $\{(v(s_1), v(t_1)), \ldots, (v(s_n), v(t_n))\}$]
   
   For $i \in \{1, \ldots, n\}$ do $R_i := \text{MERGE}(v_{s_i}, v_{t_i})$ w.r.t. $R_{i-1}$
3. If $\text{FIND}(v_{s'_j}) = \text{FIND}(v_{t'_j})$ for some $j \in \{1, \ldots, m\}$ then return unsatisfiable
4. else [if $\text{FIND}(v_{s'_j}) \neq \text{FIND}(v_{t'_j})$ for all $j \in \{1, \ldots, m\}$] then return satisfiable
**Example**

\[ f(a, b) \approx a \rightarrow f(f(a, b), b) \approx a \]

**Test:** unsatisfiability of

\[ f(a, b) \approx a \land f(f(a, b), b) \not\approx a \]

**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]**
   
   \[
   \text{UNION}(v_2, v_3) : \text{sets FIND}(v_2) \text{ to } v_3.
   \]

   **[Compute and recursively merge predecessors]**
   
   Test: \( \text{FIND}(v_1) = v_1 \neq v_3 = \text{FIND}(v_2) \)

   \[
   \text{CONGR}(v_1, v_2) \\
   \text{MERGE}(v_1, v_2) : \text{(different representatives)}
   \]

   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:  
\[ \forall x \neg(x + 1 \approx 0) \]  
(zero)  
\[ \forall x \forall y (x + 1 \approx y + 1 \rightarrow x \approx y) \]  
(successor)  
\[ F[0] \land (\forall x (F[x] \rightarrow F[x + 1]) \rightarrow \forall x F[x]) \]  
(induction)  
\[ \forall x (x + 0 \approx x) \]  
(plus zero)  
\[ \forall x, y (x + (y + 1) \approx (x + y) + 1) \]  
(plus successor)  
\[ \forall x, y (x \ast 0 \approx 0) \]  
(times 0)  
\[ \forall x, y (x \ast (y + 1) \approx x \ast y + x) \]  
(times successor)

\[ 3 \ast y + 5 > 2 \ast y \text{ expressed as } \exists z(z \neq 0 \land 3 \ast y + 5 \approx 2 \ast y + z) \]

**Intended interpretation:**  
\((\mathbb{N}, \{0, 1, +, \ast\}, \{<\})\) (also with \(\approx\))  
(does not capture true arithmetic by Goedel’s incompleteness theorem)

Undecidable
Theory of integers

- $\text{Th}((\mathbb{Z}, \{0, 1, +, \ast\}, \{<\}))$

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 "=" , "≠" , 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:** $3 \times x_1 + 2 \times x_2 \leq 5 \times x_3$ 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 \( \mathbb{N}/\mathbb{Z} \)

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

Axiomatization

Linear arithmetic over \( \mathbb{Q}/\mathbb{R} \)

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

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

Axiomatization
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 <= s always during the execution of this program.
Application: Program Verification

\[
i := 1; \quad [** \text{where } 1 \leq n < m \quad **]\n\]
while \(i < n\)
do
\[
i := i + 1; \\
[** \text{part of a program in which } i \text{ is used as an index in an array} \\
\quad \text{which was declared to be of size } s > m \text{ (and } i \text{ is not changed)} \quad **]
\]
\[
\ldots
\]
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:
   $\forall n, m, s, i, i'(1 \leq n < m < s \land i < n \land i \leq s \land i' \approx i + 1 \rightarrow i' \leq s)$
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:

  \[
  F, G, H ::= A \quad \text{(atomic formula)} \hspace{1cm} (F \land G) \quad \text{(conjunction)}
  \]

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 $v_1 \xrightarrow{c_{12}} v_2 \xrightarrow{c_{23}} \cdots \xrightarrow{c_{n-1,n}} v_n \xrightarrow{c_{1}} v_1$ be a cycle in $G(S)$.

Then:
\[
\beta(v_1) - \beta(v_2) \leq c_{12} \\
\beta(v_2) - \beta(v_3) \leq c_{23} \\
\vdots \\
\beta(v_n) - \beta(v_1) \leq c_{n1}
\]
\[
g \leq 0 = \beta(v_1) - \beta(v_1) \leq \sum_{i=1}^{n-1} c_{i,i+1} + c_{n1}
\]

Thus, for satisfiability it is necessary that all cycles are positive.
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 $\delta_{uv}$ 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 $\beta : V \rightarrow \mathbb{Z}$ by $\beta(v) = \delta_{vs}$. Claim: $\beta$ 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. $\beta(x) - \beta(y) \leq c$. 