Decision Procedures for Verification

Decision Procedures (4)

16.01.2017

Viorica Sofronie-Stokkermans

sofronie@uni-koblenz.de

Until now:

Decision Procedures

- Uninterpreted functions

  congruence closure

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)
$\forall x \forall y\, (x + 1 \approx y + 1 \to x \approx y$	(successor)
$F[0] \wedge (\forall x\, (F[x] \to F[x+1]) \to \forall x F[x])$	(induction)
$\forall x\, (x + 0 \approx x)$	(plus zero)
$\forall x, y\, (x + (y + 1) \approx (x + y) + 1)$	(plus successor)
$\forall x, y\, (x * 0 \approx 0)$	(times 0)
$\forall x, y\, (x * (y + 1) \approx x * y + x)$	(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:

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

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

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

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 $\beta : V \to \mathbb{Z}$ 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. $\beta(x) - \beta(y) \leq c$.

Difference logic

Syntax

- Atomic formulae (difference constraints): $x - y \leq c$

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

- Formulae:

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

Note: $\neg(x - y \leq c)$ is equivalent to $y - x < -c$.

Difference logic

Syntax

- Atomic formulae (difference constraints): $x - y \leq c$

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

- Formulae: $F, G, H \quad ::= \quad A$ (atomic formula)

  $\quad\quad\quad\quad\quad\quad\quad\quad | \quad \neg A$

  $\quad\quad\quad\quad\quad\quad\quad\quad | \quad (F \wedge G)$ (conjunction)

Note: $\neg(x - y \leq c)$ is equivalent to $y - x < c$.

Satisfiability over $\mathbb{Z}$

$y - x < c$ iff $y - x \leq c - 1$

Natural reduction to positive difference logic.

Difference logic

Syntax

- Atomic formulae (difference constraints): $x - y \leq c$

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

- Formulae:

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

Note: $\neg(x - y \leq c)$ is equivalent to $y - x < c$.

Theorem 3.4.2.
Let S be a conjunction of strict and non-strict difference constraints, and $G(S)$ the inequality graph of S. Then S is satisfiable iff there is no negative cycle in $G(S)$.

Difference logic

Theorem 3.4.2.
Let S be a conjunction of strict and non-strict difference constraints, and $G(S)$ the inequality graph of S. Then S is satisfiable iff there is no negative cycle in $G(S)$.

Proof:
Need to extend the graph construction and the unsatisfiability condition:

$x_1 - x_2 \prec_1 c_1, \ldots, x_n - x_1 \prec_n c_n$ unsatisfiable iff

- $\sum_{i=1}^{n} c_i < 0$, or
- $\sum_{i=1}^{n} c_i = 0$ and one $\prec_i$ is strict.

Consider pairs $(\prec, c)$ instead of numbers c

- $(\prec, c) <_B (\prec', c')$ iff $c < c'$ or ($c = c'$, $\prec_1 = <$ and $\prec_2 = \leq$)
- $(\prec, c) + (\prec', c') = (\prec'', c + c')$ where $\prec'' = <$ iff $\prec$ or $\prec'$ is $<$.