Decision Procedures for Verification

Decision Procedures (2)

7.01.2019

Viorica Sofronie-Stokkermans

sofronie@uni-koblenz.de

Until now:

Theory of Uninterpreted Function Symbols

in general undecidable

Decidable fragment:

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

$\mapsto$ decision procedure

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)
$F[0] \wedge (\forall x\,(F[x] \rightarrow F[x + 1]) \rightarrow \forall x F[x])$	(induction)
$\forall x\,(x + 0 \approx x)$	(plus zero)
$\forall x, y\,(x + (y + 1) \approx (x + y) + 1)$	(plus successor)
$\forall x, y\,(x * 0 \approx 0)$	(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 $\mathbb{N}/\mathbb{Z}$

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

Axiomatization

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

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

$\mathsf{Th}(\mathbb{Q})$ $\quad$ $\mathbb{Q} = (\mathbb{Q}, \{0, 1, +\}, \{<\})$ 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:

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

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 $v_1 \overset{c_{12}}{\to} v_2 \overset{c_{23}}{\to} \cdots \overset{c_{n-1,n}}{\to} v_n \overset{c_{n1}}{\to} v_1$ be a cycle in $G(S)$.

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

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 $\beta : V \to \mathbb{Z}$ 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:

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

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:

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

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

Satisfiability over $\mathbb{Z}$

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

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:

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

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 $<$.

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

1. $\mathrm{Th}(\mathbb{Z}_+)$ $\quad\quad$ $\mathbb{Z}_+ = (\mathbb{Z}, 0, s, +, <)$ the standard interpretation of integers.
2. **Presburger arithmetic.**

Axiomatization:

$\forall x \neg(x + 1 \approx 0)$	(zero)
$\forall x \forall y \, (x + 1 \approx y + 1 \rightarrow x \approx y$	(successor)
$F[0] \wedge (\forall x \, (F[x] \rightarrow F[x + 1]) \rightarrow \forall x F[x])$	(induction)
$\forall x \, (x + 0 \approx x)$	(plus zero)
$\forall x, y \, (x + (y + 1) \approx (x + y) + 1)$	(plus successor)

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

Presburger arithmetic decidable in 3EXPTIME [Presburger'29]

- automata theoretic method

Linear arithmetic over $\mathbb{Z}$:

check satisfiability of conjunctions of (in)equalities over $\mathbb{Z}$: NP-hard

- Integer linear programming

  use branch-and-bound/cutting planes

- The Omega test – use variable elimination

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

- Th($\mathbb{R}$)
  $\mathbb{R} = (\mathbb{R}, \{0, 1, +\}, \{<\})$ the standard interpretation of real numbers;
- Th($\mathbb{Q}$)
  $\mathbb{Q} = (\mathbb{Q}, \{0, 1, +\}, \{<\})$ the standard interpretation of rational numbers.

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

Axiomatization:

The equational part of linear rational arithmetic is described by the theory of divisible torsion-free abelian groups:

$$\forall x, y, z(x + (y + z) \approx (x + (y + z))) \qquad \text{(associativity)}$$

$$\forall x, y(x + y \approx y + x) \qquad \text{(commutativity)}$$

$$\forall x(x + 0 \approx x) \qquad \text{(identity)}$$

$$\forall x \exists y(x + y \approx 0) \qquad \text{(inverse)}$$

For all $n \geq 1$: $\forall x(\underbrace{x + \cdots + x}_{n \text{ times}} \approx 0 \rightarrow x \approx 0)$ (torsion-freeness)

For all $n \geq 1$: $\forall x \exists y(\underbrace{y + \cdots + y}_{n \text{ times}} \approx x)$ (divisibility)

$$\neg 1 \approx 0 \qquad \text{(non-triviality)}$$

Note: Quantification over natural numbers is not part of our language. We really need infinitely many axioms for torsion-freeness and divisibility.

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

By adding the axioms of a compatible strict total ordering, we define ordered divisible abelian groups:

$$\forall x\ (\neg x < x) \qquad \text{(irreflexivity)}$$
$$\forall x, y, z\ (x < y \wedge y < z \to x < z) \qquad \text{(transitivity)}$$
$$\forall x, y\ (x < y \vee y < x \vee x \approx y) \qquad \text{(totality)}$$
$$\forall x, y, z\ (x < y \to x + z < y + z) \qquad \text{(compatibility)}$$
$$0 < 1 \qquad \text{(non-triviality)}$$

Note: The second non-triviality axiom renders the first one superfluous.

Moreover, as soon as we add the axioms of compatible strict total orderings, torsion-freeness can be omitted.

Every ordered divisible abelian group is obviously torsion-free. In fact the converse holds: Every torsion-free abelian group can be ordered [F.-W. Levi, 1913].

Examples: $\mathbb{Q}, \mathbb{R}, \mathbb{Q}^n, \mathbb{R}^n, \ldots$

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

The signature can be extended by further symbols:

- $\leq/2, >/2, \geq/2, \not\approx/2$: defined using $<$ and $\approx$
- $-/1$: Skolem function for inverse axiom
- $-/2$: defined using $+/2$ and $-/1$
- $\mathrm{div}_n/1$: Skolem functions for divisibility axiom for all $n \geq 1$.
- $\mathrm{mult}_n/1$: defined by $\forall x(\mathrm{mult}_n(x) \approx \underbrace{x + \cdots + x}_{n \text{ times}}$ for all $n \geq 1$.
- $\mathrm{mult}_q/1$: defined using $\mathrm{mult}_n, \mathrm{div}_n$, – for all $q \in \mathbb{Q}$.
  (We usually write $q \cdot t$ or qt instead of $\mathrm{mult}_q(t)$.)
- $q/0$ (for $q \in \mathbb{Q}$): defined by $q \approx q \cdot 1$.

Note: Every formula using the additional symbols is ODAG-equivalent to a formula over the base signature.

When $\cdot$ is considered as a binary operator, (ordered) divisible torsion-free abelian groups correspond to (ordered) rational vector spaces.

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

Theorem.

(1) The satisfiability of any conjunction of (strict and non-strict) linear inequalities can be checked in PTIME [Khakian'79].

(2) The complexity of checking the satisfiability of sets of clauses in linear arithmetic is in NP [Sonntag'85].

Literature

[Khakian'79] L. Khachian. "A polynomial time algorithm for linear programming." *Soviet Math. Dokl.* 20:191-194, 1979.

[Sonntag'85] E.D. Sontag. "Real addition and the polynomial hierarchy". *Inf. Proc. Letters* 20(3):115-120, 1985.

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

Methods The algorithms currently used are not PTIME.

- The simplex method
- The Fourier-Motzkin method – use variable elimination

Fourier-Motzkin Quantifier Elimination

Linear rational arithmetic permits quantifier elimination:
every formula $\exists xF$ or $\forall xF$ in linear rational arithmetic can be converted into an equivalent formula without the variable x.

The method was discovered in 1826 by J. Fourier and re-discovered by T. Motzkin in 1936.

Observation: Every literal over the variables $x, y_1, ..., y_n$ can be converted into an ODAG-equivalent atom $x \sim t[\overline{y}]$ or $0 \sim t[\overline{y}]$, where $\sim \in \{<, >, \leq, \geq, \approx, \not\approx\}$ and $t[\overline{y}]$ has the form $\sum_i q_i \cdot y_i + q_0$.

In other words, we can either eliminate x completely or isolate it on one side of the atom.

Moreover, we can convert every $\not\approx$ atom into an ODAG-equivalent disjunction of two $<$ atoms.

Fourier-Motzkin Quantifier Elimination

We first consider existentially quantified conjunctions of atoms.

(1) **If the conjunction contains an equation $x \approx t[\overline{y}]$,**
we can eliminate the quantifier $\exists x$ by substitution:

$$\exists x(x \approx t[\overline{y}] \wedge F)$$

is equivalent to

$$F\sigma, \text{ where } \sigma = [t[\overline{y}]/x]$$

Fourier-Motzkin Quantifier Elimination

We first consider existentially quantified conjunctions of atoms.

(2) **If x occurs only in inequations, then:**

$$\begin{aligned}\exists x \quad & (\textstyle\bigwedge_i x < s_i(\overline{y}) \wedge \bigwedge_j x \leq t_j(\overline{y}) \wedge \\ & \textstyle\bigwedge_k x > u_k(\overline{y}) \wedge \bigwedge_l x \geq v_l(\overline{y}) \wedge \\ & F(\overline{y}))\end{aligned}$$

is equivalent to:

$$\begin{aligned}&\textstyle\bigwedge_i \bigwedge_k s_i(\overline{y}) > u_k(\overline{y}) \wedge \bigwedge_j \bigwedge_k t_j(\overline{y}) > u_k(\overline{y}) \wedge \\ &\textstyle\bigwedge_i \bigwedge_l s_i(\overline{y}) > v_l(\overline{y}) \wedge \bigwedge_j \bigwedge_l t_j(\overline{y}) \geq v_l(\overline{y}) \wedge \\ &F(\overline{y})\end{aligned}$$

Proof: "$\Rightarrow$" follows by transitivity;

"$\Leftarrow$" Take $\frac{1}{2}(\min\{s_i, t_j\} + \max\{u_k, v_l\})$ as a witness.

Fourier-Motzkin Quantifier Elimination

Extension to arbitrary formulas:

- Transform into prenex formula;
- If innermost quantifier is $\exists$:
  transform matrix into DNF and move $\exists$ into disjunction;
- If innermost quantifier is $\forall$: replace $\forall x F$ by $\neg\exists x \neg F$, then eliminate $\exists$.

Consequences:

(1) Every closed formula over the signature of ODAGs is ODAG-equivalent to either $\top$ or $\bot$.
(2) ODAGs are a complete theory, i.e., every closed formula over the signature of ODAGs is either valid or unsatisfiable w.r.t. ODAGs.
(3) Every closed formula over the signature of ODAGs holds either in all ODAGs or in no ODAG.

ODAGs are indistinguishable by first-order formulas over the signature of ODAGs. (These properties do not hold for extended signatures!)

Fourier-Motzkin: Complexity

- **One FM-step for $\exists$:**

  formula size grows quadratically, therefore $O(n^2)$ runtime.

- **m quantifiers $\exists \ldots \exists$:**

  naive implementation needs $O(n^{2^m})$ runtime;

  It is unknown whether optimized implementation with simply exponential runtime is possible.

- **m quantifiers $\exists\forall\exists\forall \ldots \exists\forall$:**

  CNF/DNF conversion (exponential!) required after each step; therefore non-elementary runtime.

Fourier-Motzkin: Complexity

- **One FM-step for $\exists$:**
  formula size grows quadratically, therefore $O(n^2)$ runtime.
- **m quantifiers $\exists \ldots \exists$:**
  naive implementation needs $O(n^{2^m})$ runtime;

  It is unknown whether optimized implementation with simply exponential runtime is possible.
- **m quantifiers $\exists\forall\exists\forall \ldots \exists\forall$:**
  CNF/DNF conversion (exponential!) required after each step; therefore non-elementary runtime.

Improvement: Loos-Weispfenning Quantifier Elimination

Loos-Weispfenning Quantifier Elimination

A more efficient way to eliminate quantifiers in linear rational arithmetic was developed by R. Loos and V. Weispfenning (1993).

The method is also known as "test point method" or "virtual substitution method".

For simplicity, we consider only one particular ODAG, namely $\mathbb{Q}$ (as we have seen above, the results are the same for all ODAGs).

Loos-Weispfenning Quantifier Elimination

Let $F(x, \overline{y})$ be a positive boolean combination of linear (in-)equations of the form $x \sim_i s_i(\overline{y})$ and $0 \sim_j s_j(\overline{y})$ with $\sim_i, \sim_j \in \{\approx, \not\approx, <, \leq, >, \geq\}$, (i.e. a formula built from linear (in-) equations, $\vee$ and $\wedge$, but without $\neg$).

Goal: Find a finite set T of "test points" so that

$$\exists x F(x, \overline{y}) \models\!\models \bigvee_{t \in T} F(x, \overline{y})[t/x].$$

In other words:

We want to replace the infinite disjunction $\exists x$ by a finite disjunction.

Loos-Weispfenning Quantifier Elimination

If we keep the values of the variables $\overline{y}$ fixed, we can regard F as a function

$$F : \mathbb{Q} \to \{0, 1\} \quad \text{defined by } x \mapsto F(x, \overline{y})$$

Remarks:

(1) The value of each of the atoms $s_i(\overline{y}) \sim_i x$ changes only at $s_i(\overline{y})$,

(2) The value of F can only change if the value of one of its atoms changes.

(3) F is a piecewise constant function; more precisely:
the set of all x with $F(x, \overline{y}) = 1$ is a finite union of intervals.

(The union may be empty, the individual intervals may be finite or infinite and open or closed.)

Let $\delta(\overline{y}) = \min\{|s_i(\overline{y}) - s_j(\overline{y})| \mid s_i(\overline{y}) \neq s_j(\overline{y})\}$.

Each of the intervals has either length 0 (i.e., it consists of one point), or its length is at least $\delta(\overline{y})$.

Loos-Weispfenning Quantifier Elimination

If the set of all x for which $F(x, \overline{y})$ is 1 is non-empty, then

(i) $F(x, \overline{y}) = 1$ for all $x \leq r(\overline{y})$ for some $r(\overline{y}) \in \mathbb{Q}$

(ii) or there is some point where the value of $F(x, \overline{y})$ switches from 0 to 1 when we traverse the real axis from $-\infty$ to $+\infty$.

We use this observation to construct a set of test points.

Loos-Weispfenning Quantifier Elimination

We start with a "sufficiently small" test point $r(\overline{y})$ to take care of case (i).

For case (ii), we observe that $F(x, \overline{y})$ can only switch from 0 to 1 if one of the atoms switches from 0 to 1. (We consider only positive boolean combinations of atoms and $\wedge$ and $\vee$ are monotonic w.r.t. truth values.)

- $x \leq s_i(\overline{y})$ and $x < s_i(\overline{y})$ do not switch from 0 to 1 when x grows.
- $x \geq s_i(\overline{y})$ and $x \approx s_i(\overline{y})$ switch from 0 to 1 at $s_i(\overline{y})$
  $\Rightarrow$ $s_i(\overline{y})$ is a test point.
- $x > s_i(\overline{y})$ and $x \not\approx s_i(\overline{y})$ switch from 0 to 1 "right after" $s_i(\overline{y})$
  $\Rightarrow$ $s_i(\overline{y}) + \epsilon$ (for some $0 < \epsilon < \delta(\overline{y})$) is a test point.

Loos-Weispfenning Quantifier Elimination

We start with a "sufficiently small" test point $r(\overline{y})$ to take care of case (i).

For case (ii), we observe that $F(x, \overline{y})$ can only switch from 0 to 1 if one of the atoms switches from 0 to 1. (We consider only positive boolean combinations of atoms and $\wedge$ and $\vee$ are monotonic w.r.t. truth values.)

- $x \leq s_i(\overline{y})$ and $x < s_i(\overline{y})$ do not switch from 0 to 1 when x grows.
- $x \geq s_i(\overline{y})$ and $x \approx s_i(\overline{y})$ switch from 0 to 1 at $s_i(\overline{y})$
  $\Rightarrow$ $s_i(\overline{y})$ is a test point.
- $x > s_i(\overline{y})$ and $x \not\approx s_i(\overline{y})$ switch from 0 to 1 "right after" $s_i(\overline{y})$
  $\Rightarrow$ $s_i(\overline{y}) + \epsilon$ (for some $0 < \epsilon < \delta(\overline{y})$) is a test point.

If $r(\overline{y})$ is sufficiently small and $0 < \epsilon < \delta(\overline{y})$, then

$$T := \{r(\overline{y})\} \cup \{s_i(\overline{y}) \mid \sim_i \in \{\geq, \approx\}\} \cup \{s_i(\overline{y}) + \epsilon \mid \sim_i \in \{>, \not\approx\}\}.$$

is a set of test points.

Loos-Weispfenning Quantifier Elimination

Problems:

(1) We don't know how small $r(\overline{y})$ has to be for case (i).

(2) We don't know $\delta(\overline{y})$ for case (ii).

Idea: We consider the limits for $r \to -\infty$ and for $\epsilon \to 0$ (but positive), that is, we redefine

$$T := \{-\infty\} \cup \{s_i(\overline{y}) \mid \sim_i \in \{\geq, \approx\}\} \cup \{s_i(\overline{y}) + \epsilon \mid \sim_i \in \{>, \not\approx\}\}.$$

New problem:

How can we eliminate the infinitesimals $-\infty$ and ϵ when we substitute elements of T for x?

Loos-Weispfenning Quantifier Elimination

Virtual substitution:

$$(x < s(\overline{y}))[-\infty/x] := \lim_{r \to -\infty}(r < s(\overline{y})) = \top$$

$$(x \leq s(\overline{y}))[-\infty/x] := \lim_{r \to -\infty}(r \leq s(\overline{y})) = \top$$

$$(x > s(\overline{y}))[-\infty/x] := \lim_{r \to -\infty}(r > s(\overline{y})) = \bot$$

$$(x \geq s(\overline{y}))[-\infty/x] := \lim_{r \to -\infty}(r \geq s(\overline{y})) = \bot$$

$$(x \approx s(\overline{y}))[-\infty/x] := \lim_{r \to -\infty}(r \approx s(\overline{y})) = \bot$$

$$(x \not\approx s(\overline{y}))[-\infty/x] := \lim_{r \to -\infty}(r \not\approx s(\overline{y})) = \top$$

Loos-Weispfenning Quantifier Elimination

Virtual substitution:

$$(x < s(\overline{y}))[u + \epsilon/x] := \lim_{\substack{\epsilon \to 0 \\ \epsilon > 0}} (u + \epsilon < s(\overline{y})) = (u < s(\overline{y}))$$

$$(x \leq s(\overline{y}))[u + \epsilon/x] := \lim_{\substack{\epsilon \to 0 \\ \epsilon > 0}} (u + \epsilon \leq s(\overline{y})) = (u < s(\overline{y}))$$

$$(x > s(\overline{y}))[u + \epsilon/x] := \lim_{\substack{\epsilon \to 0 \\ \epsilon > 0}} (u + \epsilon > s(\overline{y})) = (u \geq s(\overline{y}))$$

$$(x \geq s(\overline{y}))[u + \epsilon/x] := \lim_{\substack{\epsilon \to 0 \\ \epsilon > 0}} (u + \epsilon \geq s(\overline{y})) = (u \geq s(\overline{y}))$$

$$(x \approx s(\overline{y}))[u + \epsilon/x] := \lim_{\substack{\epsilon \to 0 \\ \epsilon > 0}} (u + \epsilon \approx s(\overline{y})) = \bot$$

$$(x \not\approx s(\overline{y}))[u + \epsilon/x] := \lim_{\substack{\epsilon \to 0 \\ \epsilon > 0}} (u + \epsilon \not\approx s(\overline{y})) = \top$$

We have traversed the real axis from $-\infty$ to $+\infty$.

Loos-Weispfenning Quantifier Elimination

Virtual substitution:

Alternatively, we can traverse it from $+\infty$ to $-\infty$.

In this case, the test points are

$$T' := \{+\infty\} \cup \{s_i(\overline{y}) | \sim_i \in \{\leq, \approx\}\} \cup \{s_i(\overline{y}) - \epsilon | \sim_i \in \{<, \not\approx\}\}.$$

Infinitesimals are eliminated in a similar way as before.

In practice: Compute both T and T' and take the smaller set.

For a universally quantified formula $\forall x F$, we replace it by $\neg\exists x \neg F$, push inner negation downwards, and then continue as before.

Note that there is no CNF/DNF transformation required.

Loos-Weispfenning quantifier elimination works on arbitrary positive formulas.

Loos-Weispfenning: Complexity

- **One LW-step for $\exists$ or $\forall$:**

  As the number of test points is at most equal to the number of atoms, the formula size grows quadratically; therefore $O(n^2)$ runtime.

- **Multiple quantifiers of the same kind:**

$$\begin{aligned} & \exists x_2 \exists x_1 . F(x_1, x_2, \overline{y}) \\ \mapsto \quad & \exists x_2 . \bigvee_{t_1 \in T_1} F(x_1, x_2, \overline{y})[t_1/x_1] \\ \mapsto \quad & \bigvee_{t_1 \in T_1} (\exists x_2 . F(x_1, x_2, \overline{y})[t_1/x_1]) \\ \mapsto \quad & \bigvee_{t_1 \in T_1} \bigvee_{t_2 \in T_2} F(x_1, x_2, \overline{y})[t_1/x_1][t_2/x_2] \end{aligned}$$

- **m quantifiers $\exists \ldots \exists$ or $\forall \ldots \forall$:**
  formula size is multiplied by n in each step $\Rightarrow O(n^{m+1})$ runtime.

- **m quantifiers $\exists\forall\exists\forall \ldots \forall$:** doubly exponential runtime.

Note: The formula resulting from a LW-step is usually highly redundant. An efficient implementation must make use of simplification techniques.

Until now

Decidable fragments of first-order logic

Decision procedures for single theories

- UIF
- Numeric domains

  Here:

  Difference logic

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

Next: Reasoning in combinations of theories

Combinations of decision procedures