Automated Reasoning II*

Uwe Waldmann

Summer Term 2014

Topics of the Course

Decision procedures:

equality (congruence closure),
algebraic theories,
combinations.

Satisfiability modulo theories (SMT):

DPLL(T),
dealing with universal quantification.

Superposition:

combining ordered resolution and completion,
optimizations,
integrating theories.

1 Decision Procedures

In general, validity (or unsatisfiability) of first-order formulas is undecidable.

*This document contains the text of the lecture slides (almost verbatim) plus some additional information, mostly proofs of theorems that are presented on the blackboard during the course. It is not a full script and does not contain the examples and additional explanations given during the lecture. Moreover it should not be taken as an example how to write a research paper – neither stylistically nor typographically.

To get decidability results, we have to impose restrictions on

- signatures,
- formulas,
- and/or algebras.

1.1 Theories and Fragments

So far, we have considered the validity or satisfiability of "unstructured" sets of formulas.

We will now split these sets of formulas into two parts: a theory (which we keep fixed) and a set of formulas that we consider relative to the theory.

A *first-order theory* $\mathcal{T}$ is defined by

its signature $\Sigma = (\Omega, \Pi)$

its axioms, that is, a set of closed Σ-formulas.

(We often use the same symbol $\mathcal{T}$ for a theory and its set of axioms.)

Note: This is the *syntactic view* of theories. There is also a *semantic view*, where one specifies a class of Σ-algebras $\mathcal{M}$ and considers $Th(\mathcal{M})$, that is, all closed Σ-formulas that hold in the algebras of $\mathcal{M}$.

A Σ-algebra that satisfies all axioms of $\mathcal{T}$ is called a *$\mathcal{T}$-algebra* (or *$\mathcal{T}$-interpretation*).

$\mathcal{T}$ is called *consistent* if there is at least one $\mathcal{T}$-algebra. (We will only consider consistent theories.)

We can define models, validity, satisfiability, entailment, equivalence, etc., relative to a theory $\mathcal{T}$:

A $\mathcal{T}$-algebra that is a model of a Σ-formula F is also called a $\mathcal{T}$-model of F.

A Σ-formula F is called $\mathcal{T}$-valid, if $\mathcal{A}, \beta \models F$ for all $\mathcal{T}$-algebras $\mathcal{A}$ and assignments β.

A Σ-formula F is called $\mathcal{T}$-satisfiable, if $\mathcal{A}, \beta \models F$ for some $\mathcal{T}$-algebra and assignment β (and otherwise $\mathcal{T}$-unsatisfiable).

($\mathcal{T}$-satisfiability of sets of formulas, $\mathcal{T}$-entailment, $\mathcal{T}$-equivalence: analogously.)

A *fragment* is some syntactically restricted class of Σ-formulas.

Typical restriction: only certain quantifier prefixes are permitted.

1.2 Equality

Theory of equality:

Signature: arbitrary

Axioms: none
(but the equality predicate $\approx$ has a fixed interpretation)

Alternatively:

Signature contains a binary predicate symbol $\sim$ instead of the built-in $\approx$

Axioms: reflexivity, symmetry, transitivity, congruence for $\sim$

In general, satisfiability of first-order formulas w. r. t. equality is undecidable.

However, we will show that it is decidable for *ground* first-order formulas.

Note: It suffices to consider conjunctions of literals. Arbitrary ground formulas can be converted into DNF; a formula in DNF is satisfiable if and only if one of its conjunctions is satisfiable.

Note that our problem can be written in several ways:

An equational clause
$\forall \vec{x}\,(A_1 \lor \ldots \lor A_n \lor \neg B_1 \lor \ldots \lor \neg B_k)$ is $\mathcal{T}$-valid

iff

$\exists \vec{x}\,(\neg A_1 \land \ldots \land \neg A_n \land B_1 \land \ldots \land B_k)$ is $\mathcal{T}$-unsatisfiable

iff

the Skolemized (ground!) formula
$(\neg A_1 \land \ldots \land \neg A_n \land B_1 \land \ldots \land B_k)\{\vec{x} \mapsto \vec{c}\}$ is $\mathcal{T}$-unsatisfiable

iff

$(A_1 \lor \ldots \lor A_n \lor \neg B_1 \lor \ldots \lor \neg B_k)\{\vec{x} \mapsto \vec{c}\}$ is $\mathcal{T}$-valid

Other names:

The theory is also known as *EUF* (equality with uninterpreted function symbols).

The decision procedures for the ground fragment are called *congruence closure* algorithms.

Congruence Closure

Goal: check (un-)satisfiability of a ground conjunction

$$u_1 \approx v_1 \wedge \ldots \wedge u_n \approx v_n \wedge \neg\, s_1 \approx t_1 \wedge \ldots \wedge \neg\, s_k \approx t_k$$

Idea:

transform $E = \{u_1 \approx v_1, \ldots, u_n \approx v_n\}$ into an equivalent convergent TRS R and check whether $s_i{\downarrow_R} = t_i{\downarrow_R}$.

if $s_i{\downarrow_R} = t_i{\downarrow_R}$ for some i:
$s_i{\downarrow_R} = t_i{\downarrow_R} \;\Leftrightarrow\; s_i \leftrightarrow_E^* t_i \;\Leftrightarrow\; E \models s_i \approx t_i \;\Rightarrow\;$ unsat.

if $s_i{\downarrow_R} = t_i{\downarrow_R}$ for no i:
$\mathrm{T}_\Sigma(X)/R = \mathrm{T}_\Sigma(X)/E$ is a model of the conjunction $\;\Rightarrow\;$ sat.

In principle, one could use Knuth-Bendix completion to convert E into an equivalent convergent TRS R.

If done properly (see exercises), Knuth-Bendix completion terminates for ground inputs.

However, for the ground case, one can optimize the general procedure.

First step:

Flatten terms: Introduce new constant symbols $c_1, c_2, \ldots$ for all subterms:

$$g(a, h(h(b))) \approx h(a)$$

is replaced by

$$a \approx c_1 \wedge b \approx c_2 \wedge h(c_2) \approx c_3 \wedge h(c_3) \approx c_4 \wedge g(c_1, c_4) \approx c_5 \wedge h(c_1) \approx c_6 \wedge c_5 \approx c_6$$

Result: only two kinds of equations left.

D-equations: $f(c_{i_1}, \ldots, c_{i_n}) \approx c_{i_0}$ for $f/n \in \Omega$, $n \geq 0$.
C-equations: $c_i \approx c_j$.

$\Rightarrow$ efficient indexing (e. g., using hash tables),
obvious termination for D-equations.

Inference Rules

The congruence closure algorithm is presented as a set of inference rules working on a set of equations E and a set of rules R: $E_0, R_0 \vdash E_1, R_1 \vdash E_2, R_2 \vdash \ldots$

At the beginning, $E = E_0$ is the set of C-equations and $R = R_0$ is the set of D-equations oriented left-to-right. At the end, E should be empty; then R is the result.

Notation: The formula $s \mathrel{\dot{\approx}} t$ denotes either $s \approx t$ or $t \approx s$.

Simplify:

$$\frac{E \cup \{c \mathrel{\dot{\approx}} c'\}, \quad R \cup \{c \to c''\}}{E \cup \{c'' \mathrel{\dot{\approx}} c'\}, \quad R \cup \{c \to c''\}}$$

Delete:

$$\frac{E \cup \{c \approx c\}, \quad R}{E, \quad R}$$

Orient:

$$\frac{E \cup \{c \mathrel{\dot{\approx}} c'\}, \quad R}{E, \quad R \cup \{c \to c'\}} \qquad \text{if } c \succ c'$$

Deduce:

$$\frac{E, \quad R \cup \{t \to c,\, t \to c'\}}{E \cup \{c \approx c'\}, \quad R \cup \{t \to c\}}$$

Collapse:

$$\frac{E, \quad R \cup \{t[c] \to c',\, c \to c''\}}{E, \quad R \cup \{t[c''] \to c',\, c \to c''\}}$$

Note: for ground rewrite rules, critical pair computation does not involve substitution. Therefore, every critical pair computation can be replaced by a simplification, either using Deduce or Collapse.

Strategy

The inference rules are applied according to the following strategy:

(1) If there is an equation in E, use Simplify as long as possible for this equation, then use either Delete or Orient. Repeat until E is empty.

(2) If Collapse is applicable, apply it, if now Deduce is applicable, apply it as well. Repeat until Collapse is no longer applicable.

(3) If E is non-empty, go to (1), otherwise return R.

Implementation

Instead of fixing the ordering $\succ$ in advance, it is preferable to define it on the fly during the algorithm:

If we orient an equation $c \approx c'$ between two constant symbols, we try to make that constant symbol larger that occurs less often in R $\Rightarrow$ fewer Collapse steps.

Additionally:

Use various index data structures so that all the required operations can be performed efficiently.

Use a union-find data structure to represent the equivalence classes encoded by the C-rules.

Average runtime for an implementation using hash tables: $O(m \log m)$, where m is the number of edges in the graph representation of the initial C and D-equations.

Other Predicate Symbols

If the initial ground conjunction contains also non-equational literals $[\neg]\, P(t_1, \dots, t_n)$, treat these like equational literals $[\neg]\, P(t_1, \dots, t_n) \approx \mathit{true}$. Then use the same algorithm as before.

One Small Problem

The inference rules are sound in the usual sense: The conclusions are entailed by the premises, so every $\mathcal{T}$-model of the premises is a $\mathcal{T}$-model of the conclusions.

For the initial flattening, however, we get a weaker result: We have to *extend* the $\mathcal{T}$-models of the original equations to obtain models of the flattened equations. That is, we get a new algebra with the same universe as the old one, with the same interpretations for old functions and predicate symbols, but with appropriately chosen interpretations for the new constants.

Consequently, the relations $\approx_E$ and $\approx_R$ for the original E and the final R are not the same. For instance, $c_3 \approx_E c_7$ does not hold, but $c_3 \approx_R c_7$ may hold.

On the other hand, the model extension preserves the universe and the interpretations for old symbols. Therefore, if s and t are terms over the old symbols, we have $s \approx_E t$ iff $s \approx_R t$.

This is sufficient for our purposes: The terms s_i and t_i that we want to normalize using R do not contain new symbols.

History

Congruence closure algorithms have been published, among others, by Shostak (1978). by Nelson and Oppen (1980), and by Downey, Sethi and Tarjan (1980).

Kapur (1997) showed that Shostak's algorithm can be described as a completion procedure.

Bachmair and Tiwari (2000) did this also for the Nelson/Oppen and the Downey/Sethi/Tarjan algorithm.

The algorithm presented here is the Downey/Sethi/Tarjan algorithm in the presentation of Bachmair and Tiwari.

1.3 Linear Rational Arithmetic

There are several ways to define *linear rational arithmetic.*

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

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

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

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

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

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

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

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

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

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

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

$$\forall x\, (\neg\, x < x) \qquad \text{(irreflexivity)}$$

$$\forall x, y, z\, (x < y \wedge y < z \rightarrow x < z) \qquad \text{(transitivity)}$$

$$\forall x, y\, (x < y \vee y < x \vee x \approx y) \qquad \text{(totality)}$$

$$\forall x, y, z\, (x < y \rightarrow x + z < y + z) \qquad \text{(compatibility)}$$

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

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: $\mathbb{Q}$, $\mathbb{R}$, $\mathbb{Q}^n$, $\mathbb{R}^n$, ...

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 mult_n, 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.

Fourier-Motzkin Quantifier Elimination

Linear rational arithmetic permits *quantifier elimination:* every formula $\exists x\, F$ or $\forall x\, F$ 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 $x, y_1, \dots, y_n$ can be converted into an ODAG-equivalent literal $x \sim t[\vec{y}]$ or $0 \sim t[\vec{y}]$, where $\sim\ \in \{<, >, \leq, \geq, \approx, \not\approx\}$ and $t[\vec{y}]$ has the form $\sum_i q_i \cdot y_i + q_0$.

In other words, we can either eliminate x completely or isolate in on one side of the literal, and we can replace every negative ordering literal by a positive one.

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

We first consider existentially quantified conjunctions of atoms.

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

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

is equivalent to

$$F\{x \mapsto t[\vec{y}]\}$$

If x occurs only in inequations, then

$$\begin{aligned}\exists x \Big(&\textstyle\bigwedge_i x < s_i(\vec{y}) \ \wedge\ \bigwedge_j x \leq t_j(\vec{y}) \\ &\quad \wedge\ \textstyle\bigwedge_k x > u_k(\vec{y}) \ \wedge\ \bigwedge_l x \geq v_l(\vec{y}) \ \wedge\ \bigwedge_m 0 \sim_m w_m(\vec{y})\Big)\end{aligned}$$

is equivalent to

$$\begin{aligned}&\textstyle\bigwedge_i \bigwedge_k s_i(\vec{y}) > u_k(\vec{y}) \ \wedge\ \bigwedge_j \bigwedge_k t_j(\vec{y}) > u_k(\vec{y}) \\ &\quad \wedge\ \textstyle\bigwedge_i \bigwedge_l s_i(\vec{y}) > v_l(\vec{y}) \ \wedge\ \bigwedge_j \bigwedge_l t_j(\vec{y}) \geq v_l(\vec{y}) \\ &\quad \wedge\ \textstyle\bigwedge_m 0 \sim_m w_m(\vec{y}) < 0\end{aligned}$$

Proof: ($\Rightarrow$) by transitivity;
($\Leftarrow$) take $\frac{1}{2}(\min\{s_i, t_j\} + \max\{u_k, v_l\})$ as a witness.

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 $\exists$.

Consequence: every closed formula over the signature of ODAGs is ODAG-equivalent to either $\top$ or $\bot$.

Consequence: ODAGs are a *complete* theory, i. e., every closed formula over the signature of ODAGs is either valid or unsatisfiable w. r. t. ODAGs.

Consequence: 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 $\exists \dots \exists$:

naive implementation needs $O(n^{2^m})$ runtime;
unknown whether optimized implementation with simply exponential runtime is possible.

m quantifiers $\exists\forall\exists\forall \dots \exists$:

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

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).

Let $F(x, \vec{y})$ be a *positive* boolean combination of linear (in-)equations $x \sim_i s_i(\vec{y})$ and $0 \sim_j s'_j(\vec{y})$ with $\sim_i, \sim_j \in \{\approx, \not\approx, <, \leq, >, \geq\}$, that is, a formula built from linear (in-) equations, $\wedge$ and $\vee$ (but without $\neg$).

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

$$\exists x\, F(x, \vec{y}) \quad \models\!\mid \quad \bigvee_{t \in T} F(x, \vec{y})\, \{x \mapsto t\}$$

In other words: We want to replace the infinite disjunction $\exists x$ by a finite disjunction.

If we keep the values of the variables $\vec{y}$ fixed, then we can consider F as a function $F : x \mapsto F(x, \vec{y})$ from $\mathbb{Q}$ to $\{0, 1\}$.

The value of each of the atoms $s_i(\vec{y}) \sim_i x$ changes only at $s_i(\vec{y})$, and the value of F can only change if the value of one of its atoms changes.

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

F is a piecewise constant function; more precisely, the set of all x with $F(x, \vec{y}) = 1$ is a finite union of intervals. (The union may be empty, the individual intervals may be finite or infinite and open or closed.)

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

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

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

(ii) or there is some point where the value of $F(x, \vec{y})$ 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.

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

For case (ii), we observe that $F(x, \vec{y})$ 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.)

$x \leq s_i(\vec{y})$ and $x < s_i(\vec{y})$ do not switch from 0 to 1 when x grows.

$x \geq s_i(\vec{y})$ and $x \approx s_i(\vec{y})$ switch from 0 to 1 at $s_i(\vec{y})$
$\Rightarrow s_i(\vec{y})$ is a test point.

$x > s_i(\vec{y})$ and $x \not\approx s_i(\vec{y})$ switch from 0 to 1 "right after" $s_i(\vec{y})$
$\Rightarrow s_i(\vec{y}) + \varepsilon$ (for some $0 < \varepsilon < \delta(\vec{y})$) is a test point.

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

$$\begin{aligned} T := \{r(\vec{y})\} &\cup \{\, s_i(\vec{y}) \quad\;\;\; \mid \sim_i \in \{\geq, =\}\,\} \\ &\cup \{\, s_i(\vec{y}) + \varepsilon \mid \sim_i \in \{>, \neq\}\,\}. \end{aligned}$$

is a set of test points.

Problem:
We don't know how small $r(\vec{y})$ has to be for case (i), and we don't know $\delta(\vec{y})$ for case (ii).

Idea:
We consider the limits for $r \to -\infty$ and for $\varepsilon \searrow 0$, that is, we redefine

$$\begin{aligned} T := \{-\infty\} &\cup \{\, s_i(\vec{y}) \quad\;\;\; \mid \sim_i \in \{\geq, =\}\,\} \\ &\cup \{\, s_i(\vec{y}) + \varepsilon \mid \sim_i \in \{>, \neq\}\,\}. \end{aligned}$$

How can we eliminate the infinitesimals ∞ and ε when we substitute elements of T for x?

Virtual substitution:

$$
\begin{aligned}
&(x < s(\vec{y}))\,\{x \mapsto -\infty\} := \lim_{r\to-\infty}(r < s(\vec{y})) = \top\\
&(x \leq s(\vec{y}))\,\{x \mapsto -\infty\} := \lim_{r\to-\infty}(r \leq s(\vec{y})) = \top\\
&(x > s(\vec{y}))\,\{x \mapsto -\infty\} := \lim_{r\to-\infty}(r > s(\vec{y})) = \bot\\
&(x \geq s(\vec{y}))\,\{x \mapsto -\infty\} := \lim_{r\to-\infty}(r \geq s(\vec{y})) = \bot\\
&(x \approx s(\vec{y}))\,\{x \mapsto -\infty\} := \lim_{r\to-\infty}(r \approx s(\vec{y})) = \bot\\
&(x \not\approx s(\vec{y}))\,\{x \mapsto -\infty\} := \lim_{r\to-\infty}(r \not\approx s(\vec{y})) = \top
\end{aligned}
$$

$$
\begin{aligned}
&(x < s(\vec{y}))\,\{x \mapsto u+\varepsilon\} := \lim_{\varepsilon\searrow 0}(u+\varepsilon < s(\vec{y})) = (u < s(\vec{y}))\\
&(x \leq s(\vec{y}))\,\{x \mapsto u+\varepsilon\} := \lim_{\varepsilon\searrow 0}(u+\varepsilon \leq s(\vec{y})) = (u < s(\vec{y}))\\
&(x > s(\vec{y}))\,\{x \mapsto u+\varepsilon\} := \lim_{\varepsilon\searrow 0}(u+\varepsilon > s(\vec{y})) = (u \geq s(\vec{y}))\\
&(x \geq s(\vec{y}))\,\{x \mapsto u+\varepsilon\} := \lim_{\varepsilon\searrow 0}(u+\varepsilon \geq s(\vec{y})) = (u \geq s(\vec{y}))\\
&(x \approx s(\vec{y}))\,\{x \mapsto u+\varepsilon\} := \lim_{\varepsilon\searrow 0}(u+\varepsilon \approx s(\vec{y})) = \bot\\
&(x \not\approx s(\vec{y}))\,\{x \mapsto u+\varepsilon\} := \lim_{\varepsilon\searrow 0}(u+\varepsilon \not\approx s(\vec{y})) = \top
\end{aligned}
$$

We have traversed the real axis from $-\infty$ to $+\infty$. Alternatively, we can traverse it from $+\infty$ to $-\infty$. In this case, the test points are

$$
\begin{aligned}
T' := \{+\infty\} &\cup \{\, s_i(\vec{y}) \quad\;\; \mid \sim_i \in \{\leq, =\}\,\}\\
&\cup \{\, s_i(\vec{y}) - \varepsilon \mid \sim_i \in \{<, \neq\}\,\}.
\end{aligned}
$$

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 formulas $\forall x\, F$, 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 half of the number of atoms, the formula size grows quadratically; therefore $O(n^2)$ runtime.

Multiple quantifiers of the same kind:

$$\begin{array}{ll} & \exists x_2\, \exists x_1.\ F(x_1, x_2, \vec{y}) \\ \rightsquigarrow & \exists x_2.\ \big(\bigvee_{t_1 \in T_1} F(x_1, x_2, \vec{y})\,\{x_1 \mapsto t_1\}\big) \\ \rightsquigarrow & \bigvee_{t_1 \in T_1} (\exists x_2.\ F(x_1, x_2, \vec{y})\,\{x_1 \mapsto t_1\}) \\ \rightsquigarrow & \bigvee_{t_1 \in T_1} \bigvee_{t_2 \in T_2} (F(x_1, x_2, \vec{y})\,\{x_1 \mapsto t_1\}\,\{x_2 \mapsto t_2\}) \end{array}$$

m quantifiers $\exists \dots \exists$ or $\forall \dots \forall$:

formula size is multiplied by n in each step, therefore $O(n^{m+1})$ runtime.

m quantifiers $\exists\forall\exists\forall \dots \exists$:

doubly exponential runtime.

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