Automated Reasoning II*

Sophie Tourret, Uwe Waldmann

Summer Term 2018

Topics of the Course

Decision procedures:

equality (congruence closure),
algebraic theories,
combinations.

Satisfiability modulo theories (SMT):

CDCL(T),
dealing with universal quantification.

Superposition:

combining ordered resolution and completion,
optimizations,
integrating theories.

Higher-order Logic:

syntax, classical and Henkin's semantics,
higher-order unification,
constrained resolution

*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.

1 Decision Procedures

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

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

iff

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

iff

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

iff

$(A_1 \vee \ldots \vee A_n \vee \neg B_1 \vee \ldots \vee \neg B_k)\{\vec{x} \mapsto \vec{c}\}$ 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:

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

Collapse:

$$\frac{E, \quad R \cup \{t[c]_p \to c',\, c \to c''\}}{E, \quad R \cup \{t[c'']_p \to c',\, c \to c''\}} \qquad \text{if } p \neq \varepsilon$$

Deduce:

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

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 $[\neg]\ P(t_1, \ldots, t_n)$, treat these like equational literals $[\neg]\ P(t_1, \ldots, t_n) \approx \mathit{true}$. 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.

Literature

Leo Bachmair, Ashish Tiwari: Abstract Congruence Closure and Specializations. Proc. CADE-17, 2000, pp 64–78, LNCS 1831, Springer.

Peter J. Downey, Ravi Sethi, Robert E. Tarjan: Variations on the Common Subexpression Problem. Journal of the ACM, 27(4):758–771, 1980.

Deepak Kapur: Shostak's congruence closure as completion. Proc. 8th RTA, 1997, pp. 23–37, LNCS 1232, Springer.

Greg Nelson, Derek C. Oppen: Fast Decision Procedures Based on Congruence Closure. Journal of the ACM, 27(2):356–364, 1980.

Robert E. Shostak: An algorithm for reasoning about equality. Communications of the ACM, 21(7):583–585, 1978.

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*:

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

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

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

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

For all $n \geq 1$: $\forall x\,(\underbrace{x + \cdots + x}_{n \text{ times}} \approx 0 \to x \approx 0)$ (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.

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

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 $x, y_1, \ldots, y_n$ 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

$$\exists x \Big(\textstyle\bigwedge_i x < s_i(\vec{y}) \ \wedge\ \bigwedge_j x \le t_j(\vec{y}) \\ \wedge\ \bigwedge_k x > u_k(\vec{y}) \ \wedge\ \bigwedge_l x \ge v_l(\vec{y}) \ \wedge\ \bigwedge_m 0 \sim_m w_m(\vec{y})\Big)$$

is equivalent to

$$\textstyle\bigwedge_i \bigwedge_k s_i(\vec{y}) > u_k(\vec{y}) \ \wedge\ \bigwedge_j \bigwedge_k t_j(\vec{y}) > u_k(\vec{y}) \\ \wedge\ \bigwedge_i \bigwedge_l s_i(\vec{y}) > v_l(\vec{y}) \ \wedge\ \bigwedge_j \bigwedge_l t_j(\vec{y}) \ge v_l(\vec{y}) \\ \wedge\ \bigwedge_m 0 \sim_m w_m(\vec{y})$$

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 $\exists \ldots \exists$:

naive implementation produces a doubly exponential number of inequations, therefore needs $O(n^{2^m})$ runtime (the number of *necessary* inequations grows only exponentially, though).

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

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

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

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 one plus the number of atoms (one plus half of the number of atoms, if there are only ordering literals), the formula size grows quadratically; therefore $O(n^2)$ runtime.

Multiple quantifiers of the same kind:

$$\begin{aligned} & \exists x_2\, \exists x_1.\ F(x_1, x_2, \vec{y}) \\ \rightsquigarrow\quad & \exists x_2.\ \left(\bigvee_{t_1 \in T_1} F(x_1, x_2, \vec{y})\,\{x_1 \mapsto t_1\}\right) \\ \rightsquigarrow\quad & \bigvee_{t_1 \in T_1} \left(\exists x_2.\ F(x_1, x_2, \vec{y})\,\{x_1 \mapsto t_1\}\right) \\ \rightsquigarrow\quad & \bigvee_{t_1 \in T_1} \bigvee_{t_2 \in T_2} \left(F(x_1, x_2, \vec{y})\,\{x_1 \mapsto t_1\}\,\{x_2 \mapsto t_2\}\right) \end{aligned}$$

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

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

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

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.

Literature

Andreas Dolzmann: Algorithmic Strategies for Applicable Real Qunantifier Elimination. PhD thesis, Universität Passau, 2000.

Jean-Baptiste Joseph Fourier: Solution d'une question particulière du calcul des inégalités. Nouveau Bulletin des Sciences par la Société philomahique de Paris, 1826.

F. Levi: Arithmetische Gesetze im Gebiete discreter Gruppen. Rendiconti del Circolo Matematico di Palermo, 35:225–236, 1913.

Rüdiger Loos, Volker Weispfenning: Applying Linear Quantifier Elimination. The Computer Journal, 36(5):450–462, 1993.

1.4 Existentially-quantified LRA

So far, we have considered formulas that may contain free, existentially quantified, and universally quantified variables.

For the special case of conjunction of linear inequations in which *all* variables are existentially quantified, there are more efficient methods available.

Main idea: reduce satisfiability problem to optimization problem.

Linear Optimization

Goal:

Solve a linear optimization (also called: linear programming) problem for given numbers $a_{ij}, b_i, c_j \in \mathbb{R}$:

$$\begin{array}{ll} \text{maximize} & \sum_{1 \leq j \leq n} c_j x_j \\ \text{for} & \bigwedge_{1 \leq i \leq m} \sum_{1 \leq j \leq n} a_{ij} x_j \leq b_i \end{array}$$

or in vectorial notation:

$$\begin{array}{ll} \text{maximize} & \vec{c}^{\top} \vec{x} \\ \text{for} & A\vec{x} \leq \vec{b} \end{array}$$

Simplex algorithm:

Developed independently by Kantorovich (1939), Dantzig (1948).

Polynomial-time average-case complexity; worst-case time complexity is exponential, though.

Interior point methods:

First algorithm by Karmarkar (1984).

Polynomial-time worst-case complexity (but large constants).

In practice: no clear winner.

Implementations:

GLPK (GNU Linear Programming Kit),

Gurobi.

Main idea of Simplex:

$A\vec{x} \leq \vec{b}$ describes a convex polyhedron.

Pick one vertex of the polyhedron,
then follow the edges of the polyhedron towards an optimal solution.

By convexity, the local optimum found in this way is also a global optimum.

Details: see special lecture on optimization.

Using an optimization procedure for checking satisfiability:

Goal: Check whether $A\vec{x} \leq \vec{b}$ is satisfiable.

To use the Simplex method, we have to transform the original (possibly empty) polyhedron into another polyhedron that is non-empty and for which we know one initial vertex.

Every real number can be written as the difference of two non-negative real numbers. Use this idea to convert $A\vec{x} \leq \vec{b}$ into an equisatisfiable inequation system $\vec{y} \geq \vec{0}$, $B\vec{y} \leq \vec{b}$ for new variables $\vec{y}$.

Multiply those inequations of the inequation system $B\vec{y} \leq \vec{b}$ in which the number on the right-hand side is negative by -1. We obtain two inequation systems $D_1\vec{y} \leq \vec{g}_1$, $D_2\vec{y} \geq \vec{g}_2$, such that $\vec{g}_1 \geq \vec{0}$, $\vec{g}_2 > 0$.

Now solve

$$\begin{array}{ll} \text{maximize} & \vec{1}^\top(D_2\vec{y} - \vec{z}) \\ \text{for} & \vec{y}, \vec{z} \geq \vec{0} \\ & D_1\vec{y} \leq \vec{g}_1 \\ & D_2\vec{y} - \vec{z} \leq \vec{g}_2 \end{array}$$

where $\vec{z}$ is a vector of new variables with the same size as $\vec{g}_2$.

Observation 1: $\vec{0}$ is a vertex of the polyhedron of this optimization problem.

Observation 2: The maximum is $\vec{1}^\top\vec{g}_2$ if and only if $\vec{y} \geq \vec{0}$, $D_1\vec{y} \leq \vec{g}_1$, $D_2\vec{y} \geq \vec{g}_2$ has a solution.

($\Rightarrow$): If $\vec{1}^\top(D_2\vec{y} - \vec{z}) = \vec{1}^\top\vec{g}_2$ for some $\vec{y}, \vec{z}$ satisfying $D_2\vec{y} - \vec{z} \leq \vec{g}_2$, then $D_2\vec{y} - \vec{z} = \vec{g}_2$, hence $D_2\vec{y} = \vec{g}_2 + \vec{z} \geq \vec{g}_2$.

($\Leftarrow$): $\vec{1}^\top(D_2\vec{y} - \vec{z})$ can never be larger than $\vec{1}^\top\vec{g}_2$. If $\vec{y} \geq \vec{0}$, $D_1\vec{y} \leq \vec{g}_1$, $D_2\vec{y} \geq \vec{g}_2$ has a solution, choose $\vec{z} = D_2\vec{y} - \vec{g}_2$; then $\vec{1}^\top(D_2\vec{y} - \vec{z}) = \vec{1}^\top\vec{g}_2$.

A Simplex variant:

Transform the satisfiability problem into the form

$$A\vec{x} = \vec{0}$$
$$\vec{l} \leq \vec{x} \leq \vec{u}$$

(where l_i may be $-\infty$ and u_i may be $+\infty$).

Relation to optimization problem is obscured.

But: More efficient if one needs an incremental decision procedure, where inequations may be added and retracted (Dutertre and de Moura 2006).

1.5 Non-linear Real Arithmetic

Tarski (1951): Quantifier elimination is possible for *non-linear* real arithmetic (or more generally, for real-closed fields). His algorithm had non-elementary complexity, however.

An improved algorithm by Collins (1975) (with further improvements by Hong) has doubly exponential complexity: Cylindrical algebraic decomposition (CAD).

Implementation: QEPCAD.

Cylindrical Algebraic Decomposition

Given: First-order formula over atoms of the form $f_i(\vec{x}) \sim 0$, where the f_i are polynomials over variables $\vec{x}$.

Goal: Decompose $\mathbb{R}^n$ into a finite number of regions such that all polynomials have invariant sign on every region X:

$$\begin{aligned}\forall i\ (\ &\forall \vec{x} \in X.\ f_i(\vec{x}) < 0 \\ \vee\ &\forall \vec{x} \in X.\ f_i(\vec{x}) = 0 \\ \vee\ &\forall \vec{x} \in X.\ f_i(\vec{x}) > 0\ )\end{aligned}$$

Note: Implementation needs exact arithmetic using algebraic numbers (i. e., zeroes of univariate polynomials with integer coefficients).

1.6 Real Arithmetic incl. Transcendental Functions

Real arithmetic with exp/log: decidability unknown.

Real arithmetic with trigonometric functions: undecidable

The following formula holds exactly if $x \in \mathbb{Z}$:

$$\exists y\ (\sin(y) = 0 \wedge 3 < y \wedge y < 4 \wedge \sin(x \cdot y) = 0)$$

(note that necessarily $y = \pi$).

Consequence: Peano arithmetic (which is undecidable) can be encoded in real arithmetic with trigonometric functions.

However, real arithmetic with transcendental functions is decidable for formulas that are *stable under perturbations*, i. e., whose truth value does not change if numeric constants are modified by some sufficiently small ε.

Example:

Stable under perturbations: $\exists x\ x^2 \leq 5$

Not stable under perturbations: $\exists x\ x^2 \leq 0$
(Formula is true, but if we subtract an arbitrarily small $\varepsilon > 0$ from the right-hand side, it becomes false.)

Unsatisfactory from a mathematical point of view, but sufficient for engineering applications (where stability under perturbations is necessary anyhow).

Approach:

Interval arithmetic + interval bisection if necessary (Ratschan).

Sound for general formulas; complete for formulas that are stable under perturbations; may loop forever if the formula is not stable under perturbations.

1.7 Linear Integer Arithmetic

Linear integer arithmetic = Presburger arithmetic.

Decidable (Presburger, 1929), but quantifier elimination is only possible if additional divisibility operators are present:

$\exists x\ (y = 2x)$ is equivalent to $\text{divides}(2, y)$ but not to any quantifier-free formula over the base signature.

Cooper (1972): Quantifier elimination procedure, triple exponential for arbitrarily quantified formulas.

The Omega Test

Omega test (Pugh, 1991): variant of Fourier–Motzkin for conjunctions of (in-)equations in linear integer arithmetic.

Idea:

- Perform easy transformations, e. g.:
  $3x + 6y \leq 8 \;\mapsto\; 3x + 6y \leq 6 \;\mapsto\; x + 2y \leq 2$
  $3x + 6y = 8 \;\mapsto\; \bot$
  (since $3x + 6y$ must be divisible by 3).
- Eliminate equations
  (easy, if one coefficient is 1; tricky otherwise).
- If only inequations are left:
  no real solutions $\rightarrow$ unsatisfiable for $\mathbb{Z}$
  "sufficiently many" real solutions $\rightarrow$ satisfiable for $\mathbb{Z}$
  otherwise: branch

What does "sufficiently many" mean?

Consider inequations $ax \leq s$ and $bx \geq t$ with $a, b \in \mathbb{N}^{>0}$ and polynomials s, t.

If these inequations have real solutions, the interval of solutions ranges from $\frac{1}{b}t$ to $\frac{1}{a}s$.

The longest possible interval of this kind that does not contain any integer number ranges from $i + \frac{1}{b}$ to $i + 1 - \frac{1}{a}$ for some $i \in \mathbb{Z}$; it has the length $1 - \frac{1}{a} - \frac{1}{b}$.

Consequence:

If $\frac{1}{a}s > \frac{1}{b}t + (1 - \frac{1}{a} - \frac{1}{b})$, or equivalently, $bs \geq at + ab - a - b + 1$ is satisfiable, then the original problem must have integer solutions.

It remains to consider the case that $bs \geq at$ is satisfiable (hence there are real solutions) but $bs \geq at + ab - a - b + 1$ is not (hence the interval of real solutions need not contain an integer).

In the latter case, $bs \leq at + ab - a - b$ holds, hence for every solution of the original problem:

$t \leq bx \leq \frac{b}{a}s \leq t + (b - 1 - \frac{b}{a})$

and if x is an integer, $t \leq bx \leq t + \lfloor b - 1 - \frac{b}{a} \rfloor$

$\Rightarrow$ Branch non-deterministically:
Add one of the equations $bx = t + i$ for $i \in \{0, \ldots, \lfloor b - 1 - \frac{b}{a} \rfloor\}$.

Alternatively, if $b > a$:
Add one of the equations $ax = s - i$ for $i \in \{0, \ldots, \lfloor a - 1 - \frac{a}{b} \rfloor\}$.

Note: Efficiency depends highly on the size of coefficients. In applications from program verification, there is almost always some variable with a very small coefficient. If all coefficients are large, the branching step gets expensive.

Branch-and-Cut

Alternative approach: Reduce satisfiability problem to optimization problem (like Simplex). ILP, MILP: (mixed) integer linear programming.

Two basic approaches:

Branching: If the simplex algorithm finds a solution with $x = 2.7$, add the inequation $x \leq 2$ or the inequation $x \geq 3$.

Cutting planes: Derive an inequation that holds for all real solutions, then round it to obtain an inequation that holds for all integer solutions, but not for the real solution found previously.

Example:

Given:
$$\begin{aligned} 2x - 3y &\leq 1 \\ 2x + 3y &\leq 5 \\ -5x - 4y &\leq -7 \end{aligned}$$

Simplex finds an extremal solution $x = \frac{3}{2}$, $y = \frac{2}{3}$.

From the first two inequations, we see that $4x \leq 6$, hence $x \leq \frac{3}{2}$. If $x \in \mathbb{Z}$, we conclude $x = \lfloor x \rfloor \leq \lfloor \frac{3}{2} \rfloor = 1$.

$\Rightarrow$ Add the inequation $x \leq 1$, which holds for all integer solutions, but cuts off the solution $(\frac{3}{2}, \frac{2}{3})$.

In practice:

Use both: Alternate between branching and cutting steps.
Better performance than the individual approaches.

1.8 Difference Logic

Difference Logic (DL):

Fragment of linear rational or integer arithmetic.

Formulas: conjunctions of atoms $x - y < c$ or $x - y \leq c$,
$x, y \in X$,
$c \in \mathbb{Q}$ (or $c \in \mathbb{Z}$).

One special variable x_0 whose value is fixed to 0 is permitted;
this allows to express atoms like $x < 3$ in the form $x - x_0 < 3$.

Solving difference logic:

Let F be a conjunction in DL.
For simplicity: only non-strict inequalities.

Define a weighted graph G:

Vertices V: Variables in F.

Edges E: $x - y \leq c \rightsquigarrow$ edge (x, y) with weight c.

Theorem: F is unsatisfiable iff G has a negative cycle.

Can be checked in $O(|V| \cdot |E|)$ using the Bellman-Ford algorithm.

1.9 C-Arithmetic

In languages like C: Bounded integer arithmetic (modulo 2^n), in device drivers also combined with bitwise operations.

Bit-Blasting (encode everything as boolean circuits, use CDCL):

Naive encoding: possible, but often too inefficient.

If combined with over-/underapproximation techniques (Bryant, Kroening, et al.): successful.

1.10 Decision Procedures for Data Structures

There are decision procedures for, e. g.,

Arrays (read, write)

Lists (car, cdr, cons)

Sets or multisets with cardinalities

Bitvectors

Note: There are usually restrictions on quantifications. Unrestricted universal quantification can lead to undecidability.

Literature: Further Decision Procedures

Aaron R. Bradley, Zohar Manna: The Calculus of Computation. Springer, 2007.

Aaron R. Bradley, Zohar Manna, Henny B. Sipma: What's decidable about arrays? Verification, Model Checking, and Abstract Interpretation (VMCAI), LNCS 3855, pp. 427-442, Springer, 2006.

Randal E. Bryant, Daniel Kroening, Joël Ouaknine, Sanjit A. Seshia, Ofer Strichman, Bryan Brady: Deciding bit-vector arithmetic with abstraction. 13th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'07), LNCS 4424, pp. 358–372, Springer, 2007.

George E. Collins: Quantifier Elimination for Real Closed Fields by Cylindrical Algebraic Decomposition. 2nd. GI Conf. Automata Theory and Formal Languages, LNCS 33, pp. 134–183, Springer, 1975.

D. C. Cooper: Theorem Proving in Arithmetic Without Multiplication. Machine Intelligence, vol. 7, pp. 91–99. American Elsevier, New York, 1972.

George B. Dantzig: Linear Programming and Extensions. Princeton Univ. Press, 1963.

L. V. Kantorovich: Mathematical Methods in the Organization and Planning of Production. Publication House of the Leningrad State University, 1939. Translated in Management Science, 6:366–422, 1960.

Narendra Karmarkar: A New Polynomial Time Algorithm for Linear Programming. Combinatorica, 4(4):373–395, 1984.

Daniel Kroening, Ofer Strichman: Decision Procedures – An Algorithmic Point of View. Springer, 2008.

Mojżesz Presburger: Über der Vollständigkeit eines gewissen Systems der Arithmetik ganzer Zahlen, in welchem die Addition als einzige Operation hervortritt. Comptes Rendus Premier Congrès des Mathématiciens des Pays Slaves, Warsaw, pp. 92-101, 1929.

William Pugh: The Omega Test: a fast and practical integer programming algorithm for dependence analysis. Comm. of the ACM, 35(8):102-114, 1992.

Stefan Ratschan: Approximate Quantified Constraint Solving by Cylindrical Box Decomposition. Reliable Computing, 8(1):21–42, 2002.

Alfred Tarski: A Decision Method for Elementary Algebra and Geometry. Univ. of California Press, Berkeley, 1951.

1.11 Combining Decision Procedures

Problem:

Let $\mathcal{T}_1$ and $\mathcal{T}_2$ be first-order theories over the signatures Σ_1 and Σ_2.

Assume that we have decision procedures for the satisfiability of existentially quantified formulas (or the validity of universally quantified formulas) w. r. t. $\mathcal{T}_1$ and $\mathcal{T}_2$.

Can we combine them to get a decision procedure for the satisfiability of existentially quantified formulas w. r. t. $\mathcal{T}_1 \cup \mathcal{T}_2$?

General assumption:

Σ_1 and Σ_2 are disjoint.

The only symbol shared by $\mathcal{T}_1$ and $\mathcal{T}_2$ is built-in equality.

We consider only conjunctions of literals.

For general formulas, convert to DNF first and consider each conjunction individually.

Abstraction

To be able to use the individual decision procedures, we have to transform the original formula in such a way that each atom contains only symbols of one of the signatures (plus variables).

This process is known as *variable abstraction* or *purification*.

We apply the following rule as long as possible:

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

if the top symbol of t belongs to Σ_i and t occurs in F directly below a Σ_j-symbol or in a (positive or negative) equation $s \approx t$ where the top symbol of s belongs to Σ_j $(i \neq j)$, and if y is a new variable.

It is easy to see that the original and the purified formula are equivalent.

Stable Infiniteness

Problem:

Even if the Σ_1-formula F_1 and the Σ_2-formula F_2 do not share any symbols (not even variables), and if F_1 is $\mathcal{T}_1$-satisfiable and F_2 is $\mathcal{T}_2$-satisfiable, we cannot conclude that $F_1 \wedge F_2$ is $(\mathcal{T}_1 \cup \mathcal{T}_2)$-satisfiable.

Example:

Consider

$\mathcal{T}_1 = \{\forall x, y, z\,(x \approx y \ \vee \ x \approx z \ \vee \ y \approx z)\}$

and

$\mathcal{T}_2 = \{\exists x, y, z\,(x \not\approx y \ \wedge \ x \not\approx z \ \wedge \ y \not\approx z)\}$.

All $\mathcal{T}_1$-models have at most two elements, and all $\mathcal{T}_2$-models have at least three elements.

Since $\mathcal{T}_1 \cup \mathcal{T}_2$ is contradictory, there are no $(\mathcal{T}_1 \cup \mathcal{T}_2)$-satisfiable formulas.

To ensure that $\mathcal{T}_1$-models and $\mathcal{T}_2$-models can be combined to $(\mathcal{T}_1 \cup \mathcal{T}_2)$-models, we require that both $\mathcal{T}_1$ and $\mathcal{T}_2$ are stably infinite.

A first-order theory $\mathcal{T}$ is called *stably infinite*, if every existentially quantified formula that has a $\mathcal{T}$-model has also a $\mathcal{T}$-model with a (countably) infinite universe.

Note: By the Löwenheim–Skolem theorem, "countable" is redundant here.

Shared Variables

Even if $\exists \vec{x}\, F_1$ is $\mathcal{T}_1$-satisfiable and $\exists \vec{x}\, F_2$ is $\mathcal{T}_2$-satisfiable, it can happen that $\exists \vec{x}\,(F_1 \wedge F_2)$ is not $(\mathcal{T}_1 \cup \mathcal{T}_2)$-satisfiable, for instance because the shared variables x and y must be equal in all $\mathcal{T}_1$-models of $\exists \vec{x}\, F_1$ and different in all $\mathcal{T}_2$-models of $\exists \vec{x}\, F_2$.

Example:

Consider

$F_1 = (x + (-y) \approx 0)$,

and

$F_2 = (f(x) \not\approx f(y))$

where $\mathcal{T}_1$ is linear rational arithmetic and $\mathcal{T}_2$ is EUF.

We must exchange information about shared variables to detect the contradiction.

The Nelson–Oppen Algorithm (Non-deterministic Version)

Suppose that $\exists \vec{x}\, F$ is a purified conjunction of Σ_1 and Σ_2-literals.

Let F_1 be the conjunction of all literals of F that do not contain Σ_2-symbols; let F_2 be the conjunction of all literals of F that do not contain Σ_1-symbols. (Equations between variables are in both F_1 and F_2.)

The Nelson–Oppen algorithm starts with the pair F_1, F_2 and applies the following inference rules.

Unsat:

$$\frac{F_1, F_2}{\bot}$$

if $\exists \vec{x}\, F_i$ is unsatisfiable w. r. t. $\mathcal{T}_i$ for some i.

Branch:

$$\frac{F_1, F_2}{F_1 \wedge (x \approx y), F_2 \wedge (x \approx y) \quad | \quad F_1 \wedge (x \not\approx y), F_2 \wedge (x \not\approx y)}$$

if x and y are two different variables appearing in both F_1 and F_2 such that neither $x \approx y$ nor $x \not\approx y$ occurs in both F_1 and F_2

"|" means non-deterministic (backtracking!) branching of the derivation into two subderivations. Derivations are therefore trees. All branches need to be reduced until termination.

Clearly, all derivation paths are finite since there are only finitely many *shared variables* in F_1 and F_2, therefore the procedure represented by the rules is terminating.

We call a constraint configuration to which no rule applies *irreducible*.

Theorem 1.1 (Soundness) *If "Branch" can be applied to F_1, F_2, then $\exists \vec{x}\,(F_1 \wedge F_2)$ is satisfiable in $\mathcal{T}_1 \cup \mathcal{T}_2$ if and only if one of the successor configurations of F_1, F_2 is satisfiable in $\mathcal{T}_1 \cup \mathcal{T}_2$.*

Corollary 1.2 *If all paths in a derivation tree from F_1, F_2 end in $\bot$, then $\exists \vec{x}\,(F_1 \wedge F_2)$ is unsatisfiable in $\mathcal{T}_1 \cup \mathcal{T}_2$.*

For completeness we need to show that if one branch in a derivation terminates with an irreducible configuration F_1, F_2 (different from $\bot$), then $\exists \vec{x}\,(F_1 \wedge F_2)$ (and, thus, the initial formula of the derivation) is satisfiable in the combined theory.

As $\exists \vec{x}\,(F_1 \wedge F_2)$ is irreducible by "Unsat", the two formulas are satisfiable in their respective component theories, that is, we have $\mathcal{T}_i$-models $\mathcal{A}_i$ of $\exists \vec{x}\, F_i$ for $i \in \{1, 2\}$. We are left with combining the models into a single one that is both a model of the combined theory and of the combined formula. These constructions are called *amalgamations*.

Let F be a Σ_i-formula and let S be a set of variables of F. F is called *compatible* with an equivalence $\sim$ on S if the formula

$$\exists \vec{z} \Big(F \wedge \bigwedge_{x,y \in S,\, x \sim y} x \approx y \wedge \bigwedge_{x,y \in S,\, x \not\sim y} x \not\approx y \Big) \tag{1}$$

is $\mathcal{T}_i$-satisfiable whenever F is $\mathcal{T}_i$-satisfiable. This expresses that F does not contradict equalities between the variables in S as given by $\sim$.

Proposition 1.3 *If F_1, F_2 is a pair of conjunctions over $\mathcal{T}_1$ and $\mathcal{T}_2$, respectively, that is irreducible by "Branch", then both F_1 and F_2 are compatible with some equivalence $\sim$ on the shared variables S of F_1 and F_2.*

Proof. If F_1, F_2 is irreducible by the branching rule, then for each pair of shared variables x and y, both F_1 and F_2 contain either $x \approx y$ or $x \not\approx y$. Choose $\sim$ to be the equivalence given by all (positive) variable equations between shared variables that are contained in F_1.

Lemma 1.4 (Amalgamation Lemma) *Let $\mathcal{T}_1$ and $\mathcal{T}_2$ be two stably infinite theories over disjoint signatures Σ_1 and Σ_2. Furthermore let F_1, F_2 be a pair of conjunctions of literals over $\mathcal{T}_1$ and $\mathcal{T}_2$, respectively, both compatible with some equivalence $\sim$ on the shared variables of F_1 and F_2. Then $F_1 \wedge F_2$ is $(\mathcal{T}_1 \cup \mathcal{T}_2)$-satisfiable if and only if each F_i is $\mathcal{T}_i$-satisfiable.*

Proof. The "only if" part is obvious.

For the "if" part, assume that each of the F_i is $\mathcal{T}_i$-satisfiable. That is, there exist models $\mathcal{A}_i$ of $\mathcal{T}_i$ and variable assignments β_i such that $\mathcal{A}_i, \beta_i \models F_i$. As the F_i are compatible with an equivalence $\sim$ on their shared variables, we may assume that the β_i also satisfy the extended conjunctions in (1) with S the set of shared variables. In particular, whenever we have two shared variables x and y, $\beta_1(x) = \beta_1(y)$ if and only if $\beta_2(x) = \beta_2(y)$. Since the theories are stably infinite we may additionally assume that the $\mathcal{A}_i$ have countably infinite universes, hence there are bijections ρ_i from the domain of $\mathcal{A}_i$ to $\mathbb{N}$ such that $\rho_1(\beta_1(x))) = \rho_2(\beta_2(x))$ for each shared variable x. Now define $\mathcal{A}$ to be the algebra having $\mathbb{N}$ as its domain; for f or P in Σ_i define $f_{\mathcal{A}}(n_1, \ldots, n_k) = \rho_i(f_{\mathcal{A}_i}(\rho_i^{-1}(n_1), \ldots, \rho_i^{-1}(n_k)))$ and $P_{\mathcal{A}}(n_1, \ldots, n_k) \Leftrightarrow P_{\mathcal{A}_i}(\rho_i^{-1}(n_1), \ldots, \rho_i^{-1}(n_k))$. Define $\beta(x) = \rho_i(\beta_i(x))$ if x is a variable occurring in F_i. By construction of the ρ_i this definition is independent of the choice of i. Clearly $\mathcal{A}|_{\Sigma_i}, \beta \models F_i$, for $i = 1, 2$, hence $\mathcal{A}, \beta \models F_1 \wedge F_2$. Moreover, the reducts $\mathcal{A}|_{\Sigma_i}$ are isomorphic (via ρ_i) to $\mathcal{A}_i$ and thus are models of $\mathcal{T}_i$, so that $\mathcal{A}$ is a model of $\mathcal{T}_1 \cup \mathcal{T}_2$ as required.

Theorem 1.5 *The non-deterministic Nelson–Oppen algorithm is terminating and complete for deciding satisfiability of pure conjunctions of literals F_1 and F_2 over $\mathcal{T}_1 \cup \mathcal{T}_2$ for signature-disjoint, stably infinite theories $\mathcal{T}_1$ and $\mathcal{T}_2$.*

Proof. Suppose that F_1, F_2 is irreducible by the inference rules of the Nelson–Oppen algorithm. Applying the amalgamation lemma in combination with Prop. 1.3 we infer that F_1, F_2 is satisfiable w. r. t. $\mathcal{T}_1 \cup \mathcal{T}_2$.

Convexity

The number of possible equivalences of shared variables grows superexponentially with the number of shared variables, so enumerating all possible equivalences non-deterministically is going to be inefficient.

A much faster variant of the Nelson–Oppen algorithm exists for convex theories.

A first-order theory $\mathcal{T}$ is called *convex w. r. t. equations*, if for every conjunction Γ of Σ-equations and non-equational Σ-literals and for all Σ-equations A_i $(1 \leq i \leq n)$, whenever $\mathcal{T} \models \forall \vec{x} \, (\Gamma \rightarrow A_1 \vee \ldots \vee A_n)$, then there exists some index j such that $\mathcal{T} \models \forall \vec{x} \, (\Gamma \rightarrow A_j)$.

Theorem 1.6 *If a first-order theory $\mathcal{T}$ is convex w. r. t. equations and has no trivial models (i. e., models with only one element), then $\mathcal{T}$ is stably infinite.*

Proof. We shall prove the contrapositive of the statement. Suppose $\mathcal{T}$ is not stably infinite. Then there exists a satisfiable conjunction of literals $\exists \vec{x} \, F$ that has only finite models w. r. t. $\mathcal{T}$. We split F into two conjunctions F^+ and F^-, such that F^- contains the negative equational literals in F and F^+ contains the rest. As $\mathcal{T}$ is a first-order theory, it is compact, hence all models of F are bounded in cardinality by some number m. Now consider the clause $C = F^+ \rightarrow \neg F^- \vee \bigvee_{1 \leq i < j \leq m+1} y_i \approx y_j$, with fresh variables $y_1, \ldots, y_{m+1}$ not occurring in F. $\mathcal{T} \models \forall \vec{x}, \vec{y} \, C$, as the clause exactly expresses that all models of F have size less than or equal to m. However, $\mathcal{T} \not\models \forall \vec{x}, \vec{y} \, (F^+ \rightarrow A)$, for any literal A of $\neg F^-$ (as otherwise F would not be satisfiable), and also $\mathcal{T} \not\models \forall \vec{x}, \vec{y} \, (F^+ \rightarrow y_i \approx y_j)$, for each i, j, as otherwise $\mathcal{T}$ would have trivial models, which we have excluded.

Lemma 1.7 *Suppose $\mathcal{T}$ is convex, F a conjunction of literals, and S a subset of its variables. Let, for any pair of variables x_i and x_j in S, $x_i \sim x_j$ if and only if $\mathcal{T} \models \forall\vec{x}\,(F \to x_i \approx x_j)$. Then F is compatible with $\sim$.*

Proof. We show that with this choice of $\sim$ the constraint (1) is satisfiable in $\mathcal{T}$ whenever F is. Suppose, to the contrary, that F is satisfiable but (1) is not, that is,

$$\mathcal{T} \models \forall\vec{z}\Big(F \to \bigvee_{x,y\in S,\, x\sim y} x \not\approx y \;\vee \bigvee_{x,y\in S,\, x\not\sim y} x \approx y\Big)$$

or, equivalently,

$$\mathcal{T} \models \forall\vec{z}\Big(F^+ \wedge \bigwedge_{x,y\in S,\, x\sim y} x \approx y \;\to\; \neg F^- \vee \bigvee_{x,y\in S,\, x\not\sim y} x \approx y\Big).$$

By convexity of $\mathcal{T}$, the antecedent implies one of the equations of the succedent. Since the equations $x \approx y$, with $x \sim y$, are entailed by F and since F is satisfiable, this means that this equation must come from the last disjunct. In other words, there exists a pair of different variables x' and y' in S such that $x' \not\sim y'$ and

$$\mathcal{T} \models \forall\vec{z}\Big(F^+ \wedge \bigwedge_{x,y\in S,\, x\sim y} x \approx y \;\to\; x' \approx y'\Big).$$

Since

$$\mathcal{T} \models \forall\vec{z}\Big(F \to \bigwedge_{x,y\in S,\, x\sim y} x \approx y\Big),$$

we derive $\mathcal{T} \models \forall\vec{z}\Big(F \;\to\; x' \approx y'\Big)$, which is impossible.

The Nelson–Oppen Algorithm (Deterministic Version for Convex Theories)

Unsat:

$$\frac{F_1, F_2}{\bot}$$

if $\exists\vec{x}\,F_i$ is unsatisfiable w. r. t. $\mathcal{T}_i$ for some i.

Propagate:

$$\frac{F_1, F_2}{F_1 \wedge (x \approx y), F_2 \wedge (x \approx y)}$$

if x and y are two different variables appearing in
both F_1 and F_2 such that
$\mathcal{T}_1 \models \forall\vec{x}\,(F_1 \to x \approx y)$ and $\mathcal{T}_2 \not\models \forall\vec{x}\,(F_2 \to x \approx y)$
or $\mathcal{T}_2 \models \forall\vec{x}\,(F_2 \to x \approx y)$ and $\mathcal{T}_1 \not\models \forall\vec{x}\,(F_1 \to x \approx y)$.

Theorem 1.8 *If $\mathcal{T}_1$ and $\mathcal{T}_2$ are signature-disjoint theories that are convex w. r. t. equations and have no trivial models, then the deterministic Nelson–Oppen algorithm is terminating, sound and complete for deciding satisfiability of pure conjunctions of literals F_1 and F_2 over $\mathcal{T}_1 \cup \mathcal{T}_2$.*

Proof. Termination and soundness are obvious: there are only finitely many different equations that can be added, and each of them is entailed by given formulas.

For completeness, we have to show that every configuration that is irreducible by "Unsat" and "Propagate" is satisfiable w. r. t.. $\mathcal{T}_1 \cup \mathcal{T}_2$: Let F_1, F_2 be such a configuration. As it is irreducible by "Propagate", we have, for every equation $x \approx y$ between shared variables, $\mathcal{T}_1 \models \forall \vec{x}\,(F_1 \to x \approx y)$ if and only if $\mathcal{T}_2 \models \forall \vec{x}\,(F_2 \to x \approx y)$. Consequently, F_1 and F_2 are compatible with the same equivalence on the shared variables of F_1 and F_2. Moreover, each of the formulas F_i is $\mathcal{T}_i$-satisfiable, and since convexity implies stable infiniteness, F_i has a $\mathcal{T}_i$-model with a countably infinite universe. Hence, by the amalgamation lemma, $F_1 \wedge F_2$ is $(\mathcal{T}_1 \cup \mathcal{T}_2)$-satisfiable.

Corollary 1.9 *The deterministic Nelson–Oppen algorithm for convex theories requires at most $O(n^3)$ calls to the individual decision procedures for the component theories, where n is the number of shared variables.*

Iterating Nelson–Oppen

The Nelson–Oppen combination procedures can be iterated to work with more than two component theories by virtue of the following observations where signature disjointness is assumed:

Theorem 1.10 *If $\mathcal{T}_1$ and $\mathcal{T}_2$ are stably infinite, then so is $\mathcal{T}_1 \cup \mathcal{T}_2$.*

Proof. The non-deterministic Nelson–Oppen algorithm is sound and complete for $\mathcal{T}_1 \cup \mathcal{T}_2$, that is, an existentially quantified conjunction F over $\Sigma_1 \cup \Sigma_2$ is satisfiable if and only if in every derivation from the purified form of F there exists a branch leading to some irreducible constraint F_1, F_2 entailing F. The amalgamation lemma 1.4 constructs a model with a countably infinite universe for F from the models of F_1 and F_2.

Lemma 1.11 *A first-order theory $\mathcal{T}$ is convex w. r. t. equations if and only if for every conjunction Γ of Σ-equations and non-equational Σ-literals and for all equations $x_i \approx x'_i$ $(1 \leq i \leq n)$, whenever $\mathcal{T} \models \forall \vec{x}\,(\Gamma \to x_1 \approx x'_1 \vee \ldots \vee x_n \approx x'_n)$, then there exists some index j such that $\mathcal{T} \models \forall \vec{x}\,(\Gamma \to x_j \approx x'_j)$.*

Lemma 1.12 *Let $\mathcal{T}$ be a first-order theory that is convex w. r. t. equations. Let F is a conjunction of literals; let F^- be the conjunction of all negative equational literals in F and let F^+ be the conjunction of all remaining literals in F. If $\mathcal{T} \models \forall \vec{x}\,(F \to x \approx y)$, then $\exists \vec{x}\, F$ is $\mathcal{T}$-unsatisfiable or $\mathcal{T} \models \forall \vec{x}\,(F^+ \to x \approx y)$.*

Proof. $\mathcal{T} \models \forall \vec{x}\,(F \to x \approx y)$ is equivalent to $\mathcal{T} \models \forall \vec{x}\,(F^+ \to (\neg F^- \vee x \approx y))$. By convexity of $\mathcal{T}$ we know that $\mathcal{T} \models \forall \vec{x}\,(F^+ \to x \approx y)$ or $\mathcal{T} \models \forall \vec{x}\,(F^+ \to A)$ for some literal $\neg A$ in F^-. In the latter case, $\exists \vec{x}\,(F^+ \wedge \neg A)$ is $\mathcal{T}$-unsatisfiable; hence $\exists \vec{x}\, F$, that is, $\exists \vec{x}\,(F^+ \wedge F^-)$ is $\mathcal{T}$-unsatisfiable as well.

Theorem 1.13 *If $\mathcal{T}_1$ and $\mathcal{T}_2$ are convex w. r. t. equations and do not have trivial models, then so is $\mathcal{T}_1 \cup \mathcal{T}_2$.*

Proof. Suppose that $\mathcal{T}_1$ and $\mathcal{T}_2$ are convex w. r. t. equations and do not have trivial models. Assume furthermore that $\mathcal{T} \models \forall \vec{x}\,(\Gamma \to x_1 \approx x'_1 \vee \ldots \vee x_n \approx x'_n)$ for some conjunction Γ of $(\Sigma_1 \cup \Sigma_2)$-equations and non-equational $(\Sigma_1 \cup \Sigma_2)$-literals. Then $\exists \vec{x}\,(\Gamma \wedge x_1 \not\approx x'_1 \wedge \ldots \wedge x_n \not\approx x'_n)$ is $\mathcal{T}$-unsatisfiable, and we can detect this by some run of the deterministic Nelson–Oppen algorithm starting with $\exists \vec{x}, \vec{y}\,(\Gamma_1 \wedge \Gamma_2 \wedge x_1 \not\approx x'_1 \wedge \ldots \wedge x_n \not\approx x'_n)$, where $\Gamma_1 \wedge \Gamma_2$ is the result of purifying Γ. This run consists of a sequence of "Propagate" steps followed by a final "Unsat" step, and without loss of generality, we use the "Propagate" rule only if "Unsat" cannot be applied. Consequently, whenever we add an equation $x \approx y$ that is entailed by F_1 w. r. t. $\mathcal{T}_1$ or by F_2 w. r. t. $\mathcal{T}_2$, then it is by Lemma 1.12 already entailed by the positive and the non-equational literals in F_1 or F_2. Furthermore, due to the convexity of $\mathcal{T}_1$ and $\mathcal{T}_2$, the final "Unsat" step depends on at most one negative equational literal in F_1 or F_2. We can therefore construct a similar Nelson–Oppen derivation that starts with only the positive and the non-equational literals in Γ_1 and Γ_2, plus at most one negative equational literal that may be needed for the "Unsat" step. If a negative equational literal is needed, it is one of the $x_j \not\approx x'_j$; then $\exists \vec{x}\,(\Gamma \wedge x_j \not\approx x'_j)$ is $\mathcal{T}$-unsatisfiable and $\forall \vec{x}\,(\Gamma \to x_j \approx x'_j)$ is $\mathcal{T}$-valid; if no negative equational literal is needed at all, then $\exists \vec{x}\,\Gamma$ is $\mathcal{T}$-unsatisfiable, so $\forall \vec{x}\,(\Gamma \to x_j \approx x'_j)$ is $\mathcal{T}$-valid for every j.

Extensions

Many-sorted logics:

$read/2$ becomes $read : array \times int \to data.$
$write/3$ becomes $write : array \times int \times data \to array.$
Variables: $x : data$

Only one declaration per function/predicate/variable symbol.
All terms, atoms, substitutions must be well-sorted.

Algebras:

Instead of universe $U_{\mathcal{A}}$, one set per sort: $array_{\mathcal{A}}$, $int_{\mathcal{A}}$.

Interpretations of function and predicate symbols correspond to their declarations: $read_{\mathcal{A}} : array_{\mathcal{A}} \times int_{\mathcal{A}} \to data_{\mathcal{A}}$

If we consider combinations of theories with shared sorts but disjoint function and predicate symbols, then we get essentially the same combination results as before.

However, stable infiniteness and/or convexity are only required for the shared sorts.

Non-stably infinite theories:

If we impose stronger conditions on one theory, we can relax the conditions on the other one.

For instance, EUF can be combined with any other theory; stable infiniteness is not required.

Non-disjoint combinations:

Have to ensure that both decision procedures interpret shared symbols in a compatible way.

Some results, e. g. by Ghilardi, using strong model theoretical conditions on the theories.

Another Combination Method

Shostak's method:

Applicable to combinations of EUF and *solvable* theories.

A Σ-theory $\mathcal{T}$ is called *solvable*, if there exists an effectively computable function *solve* such that, for any $\mathcal{T}$-equation $s \approx t$:

(A) $solve(s \approx t) = \bot$ if and only if $\mathcal{T} \models \forall \vec{x}\,(s \not\approx t)$;

(B) $solve(s \approx t) = \emptyset$ if and only if $\mathcal{T} \models \forall \vec{x}\,(s \approx t)$; and otherwise

(C) $solve(s \approx t) = \{x_1 \approx u_1, \ldots, x_n \approx u_n\}$, where

- the x_i are pairwise different variables occurring in $s \approx t$;
- the x_i do not occur in the u_j; and
- $\mathcal{T} \models \forall \vec{x}\,((s \approx t) \leftrightarrow \exists \vec{y}\,(x_1 \approx u_1 \wedge \ldots \wedge x_n \approx u_n))$, where $\vec{y}$ are the variables occurring in one of the u_j but not in $s \approx t$, and $\vec{x} \cap \vec{y} = \emptyset$.

Additionally useful (but not required):

A canonizer, that is, a function that simplifies terms by computing some unique normal form

Main idea of the procedure:

If $s \approx t$ is a positive equation and $solve(s \approx t) = \{x_1 \approx u_1, \ldots, x_n \approx u_n\}$, replace $s \approx t$ by $x_1 \approx u_1 \wedge \ldots \wedge x_n \approx u_n$ and use these equations to eliminate the x_i elsewhere.

Practical problem:

Solvability is a rather restrictive condition.

Literature

Harald Ganzinger: Shostak Light. Automated Deduction, CADE-18, LNCS 2392, pp 332–346, Springer, 2002.

Silvio Ghilardi: Model Theoretic Methods in Combined Constraint Satisfiability. Journal of Automated Reasoning, 33(3–4):221–249, 2005.

Greg Nelson, Derek C. Oppen: Simplification by Cooperating Decision Procedures. ACM Transactions on Programming Languages and Systems, 1(2):245–257, 1979.

Robert E. Shostak: Deciding Combinations of Theories. Journal of the ACM, 31(1):1–12, 1984.

2 Satisfiability Modulo Theories (SMT)

So far:

decision procedures for satisfiability for various fragments of first-order theories;

often only for ground conjunctions of literals.

Goals:

extend decision procedures efficiently to ground CNF formulas;

later: extend to non-ground formulas (we will often lose completeness, however).

2.1 The CDCL(T) Procedure

Goal:

Given a propositional formula in CNF (or alternatively, a finite set N of clauses), where the atoms represent ground formulas over some theory $\mathcal{T}$, check whether it is satisfiable in $\mathcal{T}$ (and optionally: output *one* solution, if it is satisfiable).

Assumption:

As in the propositional case, clauses contain neither duplicated literals nor complementary literals.

For propositional CDCL ("Conflict-Driven Clause Learning"), we have considered partial valuations, i. e., partial mappings from propositional variables to truth values.

A partial valuation $\mathcal{A}$ corresponds to a set M of literals that does not contain complementary literals, and vice versa:

$\mathcal{A}(L)$ is true, if $L \in M$.

$\mathcal{A}(L)$ is false, if $\overline{L} \in M$.

$\mathcal{A}(L)$ is undefined, if neither $L \in M$ nor $\overline{L} \in M$.

We will now consider partial mappings from ground $\mathcal{T}$-atoms to truth values (which correspond to sets of $\mathcal{T}$-literals).

In order to check whether a (partial) valuation is permissible, we identify the valuation $\mathcal{A}$ or the set M with the conjunction of all literals in M:

The valuation $\mathcal{A}$ or the set M is called $\mathcal{T}$-satisfiable, if the literals in M have a $\mathcal{T}$-model.

Since the elements of M can be interpreted both as propositional variables and as ground $\mathcal{T}$-formulas, we have to distinguish between two notions of entailment:

We write $M \models F$ if F is entailed by M propositionally. We write $M \models_{\mathcal{T}} F$ if the ground $\mathcal{T}$-formulas represented by M entail F.

M is called a $\mathcal{T}$-model of F, if it is $\mathcal{T}$-satisfiable and $M \models F$.

We write $F \models_{\mathcal{T}} G$, if the formula F entails G w. r. t. $\mathcal{T}$, that is, if every $\mathcal{T}$-model of F is also a model of G.

Idea

Naive Approach:

Use CDCL to find a propositionally satisfying valuation.

If the valuation found is $\mathcal{T}$-satisfiable, stop; otherwise continue CDCL search.

Note: The CDCL procedure may *not* use "pure literal" checks.

Improvements:

Check already partial valuations for $\mathcal{T}$-satisfiability.

If $\mathcal{T}$-decision procedure yields explanations, use them for non-chronological backjumping.

If $\mathcal{T}$-decision procedure can provide $\mathcal{T}$-entailed literals, use them for propagation.

Since $\mathcal{T}$-satisfiability checks may be costly, learn clauses that incorporate useful $\mathcal{T}$-knowledge, in particular explanations for backjumping.

CDCL(T)

The "CDCL Modulo Theories" procedure is modelled by a transition relation $\Rightarrow_{\mathrm{CDCL}(\mathcal{T})}$ on a set of states.

States:

- *fail*
- $M \parallel N$,

where M is a *list of annotated literals ("trail")* and N is a set of clauses.

Annotated literal:

- L: deduced literal, due to propagation.
- L^{d}: decision literal (guessed literal).

CDCL(T) Rules from CDCL

Unit Propagate:

$M \parallel N \cup \{C \vee L\} \Rightarrow_{\mathrm{CDCL}(\mathcal{T})} M\ L \parallel N \cup \{C \vee L\}$

if C is false under M and L is undefined under M.

Decide:

$M \parallel N \Rightarrow_{\mathrm{CDCL}(\mathcal{T})} M\ L^{\mathrm{d}} \parallel N$

if L is undefined under M.

Fail:

$M \parallel N \cup \{C\} \Rightarrow_{\mathrm{CDCL}(\mathcal{T})} \mathit{fail}$

if C is false under M and M contains no decision literals.

Specific CDCL(T) Rules

$\mathcal{T}$-Learn:

$M \parallel N \Rightarrow_{\mathrm{CDCL}(\mathcal{T})} M \parallel N \cup \{C\}$

if $N \models_{\mathcal{T}} C$ and each atom of C occurs in N or M.

$\mathcal{T}$-Forget:

$M \parallel N \cup \{C\} \Rightarrow_{\mathrm{CDCL}(\mathcal{T})} M \parallel N$

if $N \models_{\mathcal{T}} C$.

$\mathcal{T}$-Propagate:

$M \parallel N \Rightarrow_{\mathrm{CDCL}(\mathcal{T})} M\ L \parallel N$

if $M \models_{\mathcal{T}} L$ where L is undefined in M, and L or $\overline{L}$ occurs in N.

$\mathcal{T}$-Backjump:

$M'\ L^{\mathrm{d}}\ M'' \parallel N \Rightarrow_{\mathrm{CDCL}(\mathcal{T})} M'\ L' \parallel N$

if $M'\ L^{\mathrm{d}}\ M'' \models \neg C$ for some $C \in N$
and if there is some "backjump clause" $C' \vee L'$ such that
$N \models_{\mathcal{T}} C' \vee L'$ and $M' \models \neg C'$,
L' is undefined under M', and
L' or $\overline{L'}$ occurs in N or in $M'\ L^{\mathrm{d}}\ M''$.

Note: We don't need a special rule to handle the case that $M'\ L^{\mathrm{d}}\ M'' \models_{\mathcal{T}} \bot$. If the trail contains a $\mathcal{T}$-inconsistent subset, we can always add the negation of that subset using $\mathcal{T}$-Learn and apply $\mathcal{T}$-Backjump afterwards.

CDCL(T) Properties

The system CDCL($\mathcal{T}$) consists of the rules Decide, Fail, Unit Propagate, $\mathcal{T}$-Propagate, $\mathcal{T}$-Backjump, $\mathcal{T}$-Learn and $\mathcal{T}$-Forget.

Lemma 2.1 *If we reach a state $M \parallel N$ starting from $\emptyset \parallel N$, then:*

(1) M does not contain complementary literals.

(2) Every deduced literal L in M follows from $\mathcal{T}$, N, and decision literals occurring before L in M.

Proof. By induction on the length of the derivation. □

Lemma 2.2 *If no clause is learned infinitely often, then every derivation starting from $\emptyset \parallel N$ terminates.*

Proof. Similar to the propositional case.

Lemma 2.3 *If $\emptyset \parallel N \ \Rightarrow^*_{\mathrm{CDCL}(\mathcal{T})}\ M \parallel N'$ and there is some conflicting clause in $M \parallel N'$, that is, $M \models \neg C$ for some clause C in N', then either Fail or $\mathcal{T}$-Backjump applies to $M \parallel N'$.*

Proof. Similar to the propositional case. □

Lemma 2.4 *If $\emptyset \parallel N \ \Rightarrow^*_{\mathrm{CDCL}(\mathcal{T})}\ M \parallel N'$ and M is $\mathcal{T}$-unsatisfiable, then either there is a conflicting clause in $M \parallel N'$, or else $\mathcal{T}$-Learn applies to $M \parallel N'$, generating a conflicting clause.*

Proof. If M is $\mathcal{T}$-unsatisfiable, then there are literals $L_1, \ldots, L_n$ in M such that $\emptyset \models_{\mathcal{T}} \overline{L_1} \vee \ldots \vee \overline{L_n}$. Hence the conflicting clause $\overline{L_1} \vee \ldots \vee \overline{L_n}$ is either in $M \parallel N'$, or else it can be learned by one $\mathcal{T}$-Learn step. □

Theorem 2.5 *Consider a derivation $\emptyset \parallel N \ \Rightarrow^*_{\mathrm{CDCL}(\mathcal{T})}\ S$, where no more rules of the CDCL(T) procedure are applicable to S except $\mathcal{T}$-Learn or $\mathcal{T}$-Forget, and if S has the form $M \parallel N'$ then M is $\mathcal{T}$-satisfiable. Then*

(1) S is fail iff N is $\mathcal{T}$-unsatisfiable.

(2) If S has the form $M \parallel N'$, then M is a $\mathcal{T}$-model of N.

The Solver Interface

The general CDCL($\mathcal{T}$) procedure has to be connected to a "Solver" for $\mathcal{T}$, a theory module that performs *at least* $\mathcal{T}$-satisfiability checks.

The solver is initialized with a list of all literals occurring in the input of the CDCL($\mathcal{T}$) procedure.

Internally, it keeps a stack I of theory literals that is initially empty. The solver performs the following operations on I:

SetTrue(L: $\mathcal{T}$-Literal):

Check whether $I \cup \{L\}$ is $\mathcal{T}$-satisfiable.

If no: return an explanation for $\overline{L}$, that is, a subset J of I such that $J \models_{\mathcal{T}} \overline{L}$.

If yes: push L on I.

Optionally: Return a list of literals that are $\mathcal{T}$-consequences of $I \cup \{L\}$ (and have not yet been detected before).

Note: Depending on $\mathcal{T}$, detecting (all) $\mathcal{T}$-consequences may be very cheap or very expensive.

Backtrack(n: $\mathbb{N}$):

Pop n literals from I.

Explanation(L: $\mathcal{T}$-Literal):

Return an explanation for L, that is, a subset J of I such that $J \models_{\mathcal{T}} L$.

We assume that L has been returned previously as a result of some SetTrue(L') operation. No literal of J may occur in I after L'.

Computing Backjump Clauses

Backjump clauses for a conflict can then be computed as in the propositional case:

Start with the conflicting clause.

Resolve with the clauses used for Unit Propagate or the explanations produced by the solver until a backjump clause (or $\bot$) is found.

2.2 Heuristic Instantiation

CDCL(T) is limited to ground (or existentially quantified) formulas. Even if we have decidability for more than the ground fragment of a theory $\mathcal{T}$, we cannot use this in CDCL(T).

Most current SMT implementations offer a limited support for universally quantified formulas by heuristic instantiation.

Goal:

> Create potentially useful ground instances of universally quantified clauses and add them to the given ground clauses.

Idea (Detlefs, Nelson, Saxe: Simplify):

> Select subset of the terms (or atoms) in $\forall \vec{x}\, C$ as "trigger" (automatically, but can be overridden manually).
>
> If there is a ground instance $C\theta$ of $\forall \vec{x}\, C$ such that $t\theta$ occurs (modulo congruence) in the current set of ground clauses for every $t \in \mathit{trigger}(C)$, add $C\theta$ to the set of ground clauses (incrementally).

Conditions for trigger terms (or atoms):

(1) Every quantified variable of the clause occurs in some trigger term (therefore more than one trigger term may be necessary).

(2) A trigger term is not a variable itself.

(3) A trigger is not explicitly forbidden by the user.

(4) There is no larger instance of the term in the formula:
(If $f(x)$ were selected as a trigger in $\forall x\, P(f(x), f(g(x)))$, a ground term $f(a)$ would produce an instance $P(f(a), f(g(a)))$, which would produce an instance $P(f(g(a)), f(g(g(a))))$, and so on.)

(5) No proper subterm satisfies (1)–(4).

Also possible (but expensive, therefore only in restricted form): Theory matching

> The ground atom $P(a)$ is not an instance of the trigger atom $P(x+1)$; it is however equivalent (in linear algebra) to $P((a-1)+1)$, which *is* an instance and may therefore produce a new ground clause.

Heuristic instantiation is obviously incomplete

> e. g., it does not find the contradiction for $f(x, a) \approx x$, $f(b, y) \approx y$, $a \not\approx b$

but it is quite useful in practice:

> modern implementations: CVC, Yices, Z3.

2.3 Local Theory Extensions

Under certain circumstances, instantiating universally quantified variables with "known" ground terms is sufficient for completeness.

Scenario:

$\Sigma_0 = (\Omega_0, \Pi_0)$: base signature;
$\mathcal{T}_0$: Σ_0-theory.

$\Sigma_1 = (\Omega_0 \cup \Omega_1, \Pi_0)$: signature extension;
K: universally quantified Σ_1-clauses;
G: ground clauses.

Assumption: clauses in G are Σ_1-flat and Σ_1-linear:

only constants as arguments of Ω_1-symbols,

if a constant occurs in two terms below an Ω_1-symbol, then the two terms are identical,

no term contains the same constant twice below an Ω_1-symbol.

Example: Monotonic functions over $\mathbb{Z}$.

$\mathcal{T}_0$: Linear integer arithmetic.

$\Omega_1 = \{f/1\}$.
$K = \{\ \forall x, y\ (\neg x \leq y \vee f(x) \leq f(y))\ \}$.

$G = \{\ f(3) \geq 6,\ f(5) \leq 9\ \}$.

Observation: If we choose interpretations for $f(3)$ and $f(5)$ that satisfy the G and monotonicity axiom, then it is always possible to define f for all remaining integers such that the monotonicity axiom is satisfied.

Example: Strictly monotonic functions over $\mathbb{Z}$.

$\mathcal{T}_0$: Linear integer arithmetic.

$\Omega_1 = \{f/1\}$.
$K = \{\ \forall x, y\ (\neg x < y \vee f(x) < f(y))\ \}$.

$G = \{\ f(3) > 6,\ f(5) < 9\ \}$.

Observation: Even though we can choose interpretations for $f(3)$ and $f(5)$ that satisfy G and the strict monotonicity axiom (map $f(3)$ to 7 and $f(5)$ to 8), we cannot define $f(4)$ such that the strict monotonicity axiom is satisfied.

To formalize the idea, we need partial algebras:

like (usual) total algebras, but $f_{\mathcal{A}}$ may be a partial function.

There are several ways to define equality in partial algebras (strong equality, Evans equality, weak equality, etc.). Here we use weak equality:

an equation $s \approx t$ holds w. r. t. $\mathcal{A}$ and β if both $\mathcal{A}(\beta)(s)$ and $\mathcal{A}(\beta)(t)$ are defined and equal or if at least one of them is undefined;

a negated equation $s \not\approx t$ holds w. r. t. $\mathcal{A}$ and β if both $\mathcal{A}(\beta)(s)$ and $\mathcal{A}(\beta)(t)$ are defined and different or if at least one of them is undefined.

If a partial algebra $\mathcal{A}$ satisfies a set of formulas N w. r. t. weak equality, it is called a weak partial model of N.

A partial algebra $\mathcal{A}$ embeds weakly into a partial algebra $\mathcal{B}$ if there is an injective total mapping $h : U_{\mathcal{A}} \to U_{\mathcal{B}}$ such that if $f_{\mathcal{A}}(a_1, \ldots, a_n)$ is defined in $\mathcal{A}$ then $f_{\mathcal{B}}(h(a_1), \ldots, h(a_n))$ is defined in $\mathcal{B}$ and equal to $h(f_{\mathcal{A}}(a_1, \ldots, a_n))$.

A theory extension $\mathcal{T}_0 \subseteq \mathcal{T}_0 \cup K$ is called *local*, if for every set G, $\mathcal{T}_0 \cup K \cup G$ is satisfiable if and only if $\mathcal{T}_0 \cup K[G] \cup G$ has a (partial) model, where $K[G]$ is the set of instances of clauses in K in which all terms starting with an Ω_1-symbol are ground terms occurring in K or G.

If every weak partial model of $\mathcal{T}_0 \cup K$ can be embedded into a a total model, then the theory extension $\mathcal{T}_0 \subseteq \mathcal{T}_0 \cup K$ is local (Sofronie-Stokkermans 2005).

Note: There are many variants of partial models and embeddings corresponding to different kinds of locality.

Examples of local theory extensions:

free functions, constructors/selectors, monotonic functions, Lipschitz functions.

2.4 Goal-driven Instantiation

Instantiation is used to refute the current model discovered by the ground solver.

Rather than a fast but loosely guided instantiation technique, we can search for the most suitable instance if it exists.

Scenario:

M: a model of the ground formula returned by the ground SMT solver.

$\mathcal{Q}$: the set of universally quantified clauses contained in the original input.

Problem:

Find a clause $\forall x\, C \in \mathcal{Q}$ and a grounding substitution σ such that $M \cup C\sigma$ is unsatisfiable, if it exists.

E-ground (Dis)unification Problem

Given

E: a set of ground equality literals,
N: a set of equality literals,

find σ such that $E \models N\sigma$.

The E-ground (dis)unification problem can be used to encode the goal-driven instantiation problem:

For M and each $\forall x\, C \in \mathcal{Q}$, try to solve the E-ground (dis)unification problem $M \models (\neg C)\sigma$.

Congruence Closure with Free Variables

CCFV (Barbosa et al, 2017) decomposes N into sets of smaller constraints by replacing terms with equivalent smaller ones until either

1. a variable assignment is possible, and the decomposition restarts afterwards,
2. a contradiction occurs, and the corresponding search branch is closed,
3. a substitution satisfying the problem is found.

CCFV is sound, complete and terminating for the E-ground (dis)unification problem.

Modern implementations: CVC4, VeriT.

Literature

Haniel Barbosa, Pascal Fontaine, Andrew Reynolds: Congruence Closure with Free Variables. Tools and Algorithms for the Construction and Analysis of Systems, TACAS 2017, LNCS 10206, pp. 214-230, Springer, 2017.

David Detlefs, Greg Nelson, James B. Saxe: Simplify: A Theorem Prover for Program Checking. Journal of the ACM, 52(3):365–473, 2005.

Yeting Ge, Leonardo de Moura: Complete instantiation for quantified formulas in Satisfiabiliby Modulo Theories. International Conference on Computer Aided Verification, CAV 2009 LNCS 5643, pp. 306–320, Springer, 2009.

Leonardo de Moura, Nikolaj Bjørner: Efficient E-Matching for SMT solvers. Automated Deduction, CADE-21, LNAI 4603, pp. 183–198, Springer, 2007.

Robert Nieuwenhuis, Albert Oliveras, Cesare Tinelli: Solving SAT and SAT Modulo Theories: From an Abstract Davis-Putnam-Logemann-Loveland Procedure to DPLL(T). Journal of the ACM, 53(6):937–977, 2006.

Viorica Sofronie-Stokkermans: Hierarchic reasoning in local theory extensions. Automated Deduction, CADE-20, LNAI 3632, pp. 219–234, Springer, 2005.

3 Superposition

First-order calculi considered so far:

Resolution: for first-order clauses without equality.

(Unfailing) Knuth-Bendix Completion: for unit equations.

Goal:

Combine the ideas of ordered resolution (overlap maximal literals in a clause) and Knuth-Bendix completion (overlap maximal sides of equations) to get a calculus for equational clauses.

3.1 Recapitulation

First-order logic:

Atom: either $P(s_1, \ldots, s_m)$ with $P \in \Pi$ or $s \approx t$.

Literal: Atom or negated atom.

Clause: (possibly empty) disjunction of literals (all variables implicitly universally quantified).

Refutational theorem proving:

For refutational theorem proving, it is sufficient to consider sets of clauses: every first-order formula F can be translated into a set of clauses N such that F is unsatisfiable if and only if N is unsatisfiable.

In the non-equational case, unsatisfiability can for instance be checked using the (ordered) resolution calculus.

(Ordered) resolution: inference rules:

	Ground case:	Non-ground case:
Resolution:	$\dfrac{D' \vee A \qquad C' \vee \neg A}{D' \vee C'}$	$\dfrac{D' \vee A \qquad C' \vee \neg A'}{(D' \vee C')\sigma}$ where $\sigma = \mathrm{mgu}(A, A')$.
Factoring:	$\dfrac{C' \vee A \vee A}{C' \vee A}$	$\dfrac{C' \vee A \vee A'}{(C' \vee A)\sigma}$ where $\sigma = \mathrm{mgu}(A, A')$.

Ordering restrictions:

Let $\succ$ be a well-founded and total ordering on ground atoms.

Literal ordering $\succ_L$: compares literals by comparing lexicographically first the respective atoms using $\succ$ and then their polarities (negative $>$ positive).

Clause ordering $\succ_C$: compares clauses by comparing their multisets of literals using the multiset extension of $\succ_L$.

Ordering restrictions (ground case):

Inference are necessary only if the following conditions are satisfied:

- The left premise of a Resolution inference is not larger than or equal to the right premise.
- The literals that are involved in the inferences ($[\neg]\ A$) are maximal in the respective clauses (strictly maximal for the left premise of Resolution).

Ordering restrictions (non-ground case):

Define the atom ordering $\succ$ also for non-ground atoms.

Need stability under substitutions: $A \succ B$ implies $A\sigma \succ B\sigma$.

Note: $\succ$ cannot be total on non-ground atoms.

For literals involved in inferences we have the same maximality requirements as in the ground case.

Resolution is (even with ordering restrictions) refutationally complete:

Dynamic view of refutational completeness:

If N is unsatisfiable ($N \models \bot$) then *fair* derivations from N produce $\bot$.

Static view of refutational completeness:

If N is *saturated*, then N is unsatisfiable if and only if $\bot \in N$.

Proving refutational completeness for the ground case:

We have to show:

If N is saturated (i. e., if sufficiently many inferences have been computed), and $\bot \notin N$, then N is satisfiable (i. e., has a model).

Constructing a candidate interpretation:

Suppose that N be saturated and $\bot \notin N$. We inspect all clauses in N in ascending order and construct a sequence of Herbrand interpretations (starting with the empty interpretation: all atoms are false).

If a clause C is false in the current interpretation, and has a positive and strictly maximal literal A, then extend the current interpretation such that C becomes true: add A to the current interpretation. (Then C is called *productive*.)

Otherwise, leave the current interpretation unchanged.

The sequence of interpretations has the following properties:

(1) If an atom is true in some interpretation, then it remains true in all future interpretations.

(2) If a clause is true at the time where it is inspected, then it remains true in all future interpretations.

(3) If a clause $C = C' \vee A$ is productive, then C remains true and C' remains false in all future interpretations.

Show by induction: if N is saturated and $\bot \notin N$, then every clause in N is either true at the time where it is inspected or productive.

Note:
For the induction proof, it is not necessary that the conclusion of an inference is contained in N. It is sufficient that it is redundant w. r. t. N.

N is called *saturated up to redundancy* if the conclusion of every inference from clauses in $N \setminus Red(N)$ is contained in $N \cup Red(N)$.

Proving refutational completeness for the non-ground case:

If $C_i\theta$ is a ground instance of the clause C_i for $i \in \{0, \ldots, n\}$ and

$$\frac{C_n, \ldots, C_1}{C_0}$$

and

$$\frac{C_n\theta, \ldots, C_1\theta}{C_0\theta}$$

are inferences, then the latter inference is called a *ground instance* of the former.

For a set N of clauses, let $G_\Sigma(N)$ be the set of all ground instances of clauses in N.

Construct the interpretation from the set $G_\Sigma(N)$ of all ground instances of clauses in N:

N is saturated and does not contain $\bot$
$\Rightarrow$ $G_\Sigma(N)$ is saturated and does not contain $\bot$
$\Rightarrow$ $G_\Sigma(N)$ has a Herbrand model I
$\Rightarrow$ I is a model of N.

It is possible to encode an arbitrary predicate P using a function f_P and a new constant *true*:

$$\begin{array}{rcl} P(t_1,\ldots,t_n) & \rightsquigarrow & f_P(t_1,\ldots,t_n) \approx \mathit{true} \\ \neg\, P(t_1,\ldots,t_n) & \rightsquigarrow & \neg\, f_P(t_1,\ldots,t_n) \approx \mathit{true} \end{array}$$

In equational logic it is therefore sufficient to consider the case that $\Pi = \emptyset$, i. e., equality is the only predicate symbol.

Abbreviation: $s \not\approx t$ instead of $\neg\, s \approx t$.

3.2 The Superposition Calculus – Informally

Conventions:

From now on: $\Pi = \emptyset$ (equality is the only predicate).

Inference rules are to be read modulo symmetry of the equality symbol.

We will first explain the ideas and motivations behind the superposition calculus and its completeness proof. Precise definitions will be given later.

Ground inference rules:

Pos. Superposition:
$$\frac{D' \vee t \approx t' \qquad C' \vee s[t] \approx s'}{D' \vee C' \vee s[t'] \approx s'}$$

Neg. Superposition:
$$\frac{D' \vee t \approx t' \qquad C' \vee s[t] \not\approx s'}{D' \vee C' \vee s[t'] \not\approx s'}$$

Equality Resolution:
$$\frac{C' \vee s \not\approx s}{C'}$$

(Note: We will need one further inference rule.)

Ordering wishlist:

Like in resolution, we want to perform only inferences between (strictly) maximal literals.

Like in completion, we want to perform only inferences between (strictly) maximal sides of literals.

Like in resolution, in inferences with two premises, the left premise should not be larger than the right one.

Like in resolution and completion, the conclusion should then be smaller than the larger premise.

The ordering should be total on ground literals.

Consequences:

The literal ordering must depend primarily on the larger term of an equation.

As in the resolution case, negative literals must be a bit larger than the corresponding positive literals.

Additionally, we need the following property: If $s \succ t \succ u$, then $s \not\approx u$ must be larger than $s \approx t$. In other words, we must compare first the larger term, then the polarity, and finally the smaller term.

The following construction has the required properties:

Let $\succ$ be a *reduction ordering that is total on ground terms.*

To a positive literal $s \approx t$, we assign the multiset $\{s, t\}$, to a negative literal $s \not\approx t$ the multiset $\{s, s, t, t\}$. The *literal ordering* $\succ_L$ compares these multisets using the multiset extension of $\succ$.

The *clause ordering* $\succ_C$ compares clauses by comparing their multisets of literals using the multiset extension of $\succ_L$.

Constructing a candidate interpretation:

We want to use roughly the same ideas as in the completeness proof for resolution.

But: a Herbrand interpretation does not work for equality: The equality symbol $\approx$ must be interpreted by equality in the interpretation.

Solution: Productive clauses contribute ground rewrite rules to a TRS R.

The interpretation has the universe $\mathrm{T}_\Sigma(\emptyset)/R = \mathrm{T}_\Sigma(\emptyset)/{\approx_R}$; a ground atom $s \approx t$ holds in the interpretation, if and only if $s \approx_R t$ if and only if $s \leftrightarrow_R^* t$.

We will construct R in such a way that it is terminating and confluent. In this case, $s \approx_R t$ if and only if $s \downarrow_R t$.

One problem:

The completeness proof for the resolution calculus depends on the following property:

> If $C = C' \vee A$ with a strictly maximal and positive literal A is false in the current interpretation, then adding A to the current interpretation cannot make any literal of C' true.

This property does not hold for superposition:

> Let $b \succ c \succ d$. Assume that the current rewrite system (representing the current interpretation) contains the rule $c \to d$. Now consider the clause $b \approx d \vee b \approx c$.

We need a further inference rule to deal with clauses of this kind, either the "Merging Paramodulation" rule of Bachmair and Ganzinger or the following "Equality Factoring" rule due to Nieuwenhuis:

Equality Factoring:
$$\frac{C' \vee s \approx t' \vee s \approx t}{C' \vee t \not\approx t' \vee s \approx t'}$$

Note: This inference rule subsumes the usual factoring rule.

How do the non-ground versions of the inference rules for superposition look like?

> Main idea as in the resolution calculus:
>
> Replace identity by unifiability. Apply the mgu to the resulting clause. In the ordering restrictions, use $\not\preceq$ instead of $\succ$.

However:

> As in Knuth-Bendix completion, we do not want to consider overlaps at or below a variable position.
>
> Consequence: there are inferences between ground instances $D\theta$ and $C\theta$ of clauses D and C which are *not* ground instances of inferences between D and C.
>
> Such inferences have to be treated in a special way in the completeness proof.

3.3 The Superposition Calculus – Formally

Until now, we have seen most of the ideas behind the superposition calculus and its completeness proof.

We will now start again from the beginning giving precise definitions and proofs.

Inference rules:

Pos. Superposition:
$$\frac{D' \vee t \approx t' \qquad C' \vee s[u] \approx s'}{(D' \vee C' \vee s[t'] \approx s')\sigma}$$
where $\sigma = \mathrm{mgu}(t, u)$ and u is not a variable.

Neg. Superposition:
$$\frac{D' \vee t \approx t' \qquad C' \vee s[u] \not\approx s'}{(D' \vee C' \vee s[t'] \not\approx s')\sigma}$$
where $\sigma = \mathrm{mgu}(t, u)$ and u is not a variable.

Equality Resolution:
$$\frac{C' \vee s \not\approx s'}{C'\sigma}$$
where $\sigma = \mathrm{mgu}(s, s')$.

Equality Factoring:
$$\frac{C' \vee s' \approx t' \vee s \approx t}{(C' \vee t \not\approx t' \vee s \approx t')\sigma}$$
where $\sigma = \mathrm{mgu}(s, s')$.

Theorem 3.1 *All inference rules of the superposition calculus are* correct, *i. e., for every rule*
$$\frac{C_n, \ldots, C_1}{C_0}$$
we have $\{C_1, \ldots, C_n\} \models C_0$.

Proof. Exercise. □

Orderings:

Let $\succ$ be a *reduction ordering that is total on ground terms.*

To a positive literal $s \approx t$, we assign the multiset $\{s, t\}$, to a negative literal $s \not\approx t$ the multiset $\{s, s, t, t\}$. The *literal ordering* $\succ_L$ compares these multisets using the multiset extension of $\succ$.

The *clause ordering* $\succ_C$ compares clauses by comparing their multisets of literals using the multiset extension of $\succ_L$.

Inferences have to be computed only if the following ordering restrictions are satisfied (after applying the unifier to the premises):

- In superposition inferences, the left premise is not greater than or equal to the right one.
- The last literal in each premise is maximal in the respective premise, i. e., there exists no greater literal (strictly maximal for positive literals in superposition inferences, i. e., there exists no greater or equal literal).
- In these literals, the lhs is neither smaller nor equal than the rhs (except in equality resolution inferences).

A ground clause C is called *redundant w. r. t. a set of ground clauses* N, if it follows from clauses in N that are smaller than C.

A clause is *redundant w. r. t. a set of clauses* N, if all its ground instances are redundant w. r. t. $G_\Sigma(N)$.

The set of all clauses that are redundant w. r. t. N is denoted by $Red(N)$.

N is called *saturated up to redundancy*, if the conclusion of every inference from clauses in $N \setminus Red(N)$ is contained in $N \cup Red(N)$.

3.4 Superposition: Refutational Completeness

For a set E of ground equations, $\mathrm{T}_\Sigma(\emptyset)/E$ is an E-interpretation (or E-algebra) with universe $\{\, [t] \mid t \in \mathrm{T}_\Sigma(\emptyset) \,\}$.

One can show (similar to the proof of Birkhoff's Theorem) that for every *ground* equation $s \approx t$ we have $\mathrm{T}_\Sigma(\emptyset)/E \models s \approx t$ if and only if $s \leftrightarrow_E^* t$.

In particular, if E is a convergent set of rewrite rules R and $s \approx t$ is a ground equation, then $\mathrm{T}_\Sigma(\emptyset)/R \models s \approx t$ if and only if $s \downarrow_R t$. By abuse of terminology, we say that an equation or clause is valid (or true) in R if and only if it is true in $\mathrm{T}_\Sigma(\emptyset)/R$.

Construction of candidate interpretations (Bachmair & Ganzinger 1990):

Let N be a set of clauses not containing $\bot$. Using induction on the clause ordering we define sets of rewrite rules E_C and R_C for all $C \in G_\Sigma(N)$ as follows:

Assume that E_D has already been defined for all $D \in G_\Sigma(N)$ with $D \prec_C C$. Then $R_C = \bigcup_{D \prec_C C} E_D$.

The set E_C contains the rewrite rule $s \to t$, if

(a) $C = C' \vee s \approx t$.

(b) $s \approx t$ is strictly maximal in C.

(c) $s \succ t$.

(d) C is false in R_C.

(e) C' is false in $R_C \cup \{s \to t\}$.

(f) s is irreducible w. r. t. R_C.

In this case, C is called *productive*. Otherwise $E_C = \emptyset$.

Finally, $R_\infty = \bigcup_{D \in G_\Sigma(N)} E_D$.

Lemma 3.2 *If $E_C = \{s \to t\}$ and $E_D = \{u \to v\}$, then $s \succ u$ if and only if $C \succ_C D$.*

Proof. ($\Rightarrow$): By condition (b), $s \approx t$ is strictly maximal in C and $u \approx v$ is strictly maximal in D, and since the literal ordering is total on ground literals, this implies that all other literals in C or in D are actually smaller than $s \approx t$ or $u \approx v$, respectively.

Moreover, $s \succ t$ and $u \succ v$ by condition (c). Therefore $s \succ u$ implies $\{s, t\} \succ_{\text{mul}} \{u, v\}$. Hence $s \approx t \succ_L u \approx v \succeq_L L$ for every literal L of D, and thus $C \succ_C D$.

($\Leftarrow$): Let $C \succ_C D$, then $E_D \subseteq R_C$. By condition (f), s must be irreducible w. r. t. R_C, so $s \neq u$.

Assume that $s \not\succ u$. By totality, this implies $s \preceq u$, and since $s \neq u$, we obtain $s \prec u$. But then $C \prec_C D$ can be shown in the same way as in the ($\Rightarrow$)-part, contradicting the assumption. □

Corollary 3.3 *The rewrite systems R_C and R_∞ are convergent (i. e., terminating and confluent).*

Proof. By condition (c), $s \succ t$ for all rules $s \to t$ in R_C and R_∞, so R_C and R_∞ are terminating.

Furthermore, it is easy to check that there are no critical pairs between any two rules: Assume that there are rules $u \to v$ in E_D and $s \to t$ in E_C such that u is a subterm of s. As $\succ$ is a reduction ordering that is total on ground terms, we get $u \prec s$ and therefore $D \prec_C C$ and $E_D \subseteq R_C$. But then s would be reducible by R_C, contradicting condition (f).

Now the absence of critical pairs implies local confluence, and termination and local confluence imply confluence. □

Lemma 3.4 *If $D \preceq_C C$ and $E_C = \{s \to t\}$, then $s \succ u$ for every term u occurring in a negative literal in D and $s \succeq u$ for every term u occurring in a positive literal in D.*

Proof. If $s \preceq u$ for some term u occurring in a negative literal $u \not\approx v$ in D, then $\{u, u, v, v\} \succ_{\text{mul}} \{s, t\}$. So $u \not\approx v \succ_L s \approx t \succeq_L L$ for every literal L of C, and therefore $D \succ_C C$.

Similarly, if $s \prec u$ for some term u occurring in a positive literal $u \approx v$ in D, then $\{u, v\} \succ_{\text{mul}} \{s, t\}$. So $u \approx v \succ_L s \approx t \succeq_L L$ for every literal L of C, and therefore $D \succ_C C$. □

Corollary 3.5 *If $D \in G_\Sigma(N)$ is true in R_D, then D is true in R_∞ and R_C for all $C \succ_C D$.*

Proof. If a positive literal of D is true in R_D, then this is obvious.

Otherwise, some negative literal $s \not\approx t$ of D must be true in R_D, hence $s \not\downarrow_{R_D} t$. As the rules in $R_\infty \setminus R_D$ have left-hand sides that are larger than s and t, they cannot be used in a rewrite proof of $s \downarrow t$, hence $s \not\downarrow_{R_C} t$ and $s \not\downarrow_{R_\infty} t$. □

Corollary 3.6 *If $D = D' \vee u \approx v$ is productive, then D' is false and D is true in R_∞ and R_C for all $C \succ_C D$.*

Proof. Obviously, D is true in R_∞ and R_C for all $C \succ_C D$.

Since all negative literals of D' are false in R_D, it is clear that they are false in R_∞ and R_C. For the positive literals $u' \approx v'$ of D', condition (e) ensures that they are false in $R_D \cup \{u \to v\}$. Since $u' \preceq u$ and $v' \preceq u$ and all rules in $R_\infty \setminus R_D$ have left-hand sides that are larger than u, these rules cannot be used in a rewrite proof of $u' \downarrow v'$, hence $u' \not\downarrow_{R_C} v'$ and $u' \not\downarrow_{R_\infty} v'$. □

Lemma 3.7 ("Lifting Lemma") *Let C be a clause and let θ be a substitution such that $C\theta$ is ground. Then every equality resolution or equality factoring inference from $C\theta$ is a ground instance of an inference from C.*

Proof. Exercise. □

Lemma 3.8 ("Lifting Lemma") *Let $D = D' \vee u \approx v$ and $C = C' \vee [\neg]\, s \approx t$ be two clauses (without common variables) and let θ be a substitution such that $D\theta$ and $C\theta$ are ground.*

If there is a superposition inference between $D\theta$ and $C\theta$ where $u\theta$ and some subterm of $s\theta$ are overlapped, and $u\theta$ does not occur in $s\theta$ at or below a variable position of s, then the inference is a ground instance of a superposition inference from D and C.

Proof. Exercise. □

Theorem 3.9 ("Model Construction") *Let N be a set of clauses that is saturated up to redundancy and does not contain the empty clause. Then we have for every ground clause $C\theta \in G_\Sigma(N)$:*

(i) $E_{C\theta} = \emptyset$ if and only if $C\theta$ is true in $R_{C\theta}$.

(ii) If $C\theta$ is redundant w. r. t. $G_\Sigma(N)$, then it is true in $R_{C\theta}$.

(iii) $C\theta$ is true in R_∞ and in R_D for every $D \in G_\Sigma(N)$ with $D \succ_C C\theta$.

Proof. We use induction on the clause ordering $\succ_{\mathrm{C}}$ and assume that (i)–(iii) are already satisfied for all clauses in $G_\Sigma(N)$ that are smaller than $C\theta$. Note that the "if" part of (i) is obvious from the construction and that condition (iii) follows immediately from (i) and Corollaries 3.5 and 3.6. So it remains to show (ii) and the "only if" part of (i).

Case 1: $C\theta$ is redundant w. r. t. $G_\Sigma(N)$.

If $C\theta$ is redundant w. r. t. $G_\Sigma(N)$, then it follows from clauses in $G_\Sigma(N)$ that are smaller than $C\theta$. By part (iii) of the induction hypothesis, these clauses are true in $R_{C\theta}$. Hence $C\theta$ is true in $R_{C\theta}$.

Case 2: $x\theta$ is reducible by $R_{C\theta}$.

Suppose there is a variable x occurring in C such that $x\theta$ is reducible by $R_{C\theta}$, say $x\theta \to_{R_{C\theta}} w$. Let the substitution θ' be defined by $x\theta' = w$ and $y\theta' = y\theta$ for every variable $y \neq x$. The clause $C\theta'$ is smaller than $C\theta$. By part (iii) of the induction hypothesis, it is true in $R_{C\theta}$. By congruence, every literal of $C\theta$ is true in $R_{C\theta}$ if and only if the corresponding literal of $C\theta'$ is true in $R_{C\theta}$; hence $C\theta$ is true in $R_{C\theta}$.

Case 3: $C\theta$ contains a maximal negative literal.

Suppose that $C\theta$ does not fall into Case 1 or 2 and that $C\theta = C'\theta \vee s\theta \not\approx s'\theta$, where $s\theta \not\approx s'\theta$ is maximal in $C\theta$. If $s\theta \approx s'\theta$ is false in $R_{C\theta}$, then $C\theta$ is clearly true in $R_{C\theta}$ and we are done. So assume that $s\theta \approx s'\theta$ is true in $R_{C\theta}$, that is, $s\theta \downarrow_{R_{C\theta}} s'\theta$. Without loss of generality, $s\theta \succeq s'\theta$.

Case 3.1: $s\theta = s'\theta$.

If $s\theta = s'\theta$, then there is an *equality resolution* inference

$$\frac{C'\theta \vee s\theta \not\approx s'\theta}{C'\theta} .$$

As shown in the Lifting Lemma, this is an instance of an *equality resolution* inference

$$\frac{C' \vee s \not\approx s'}{C'\sigma}$$

where $C = C' \vee s \not\approx s'$ is contained in N and $\theta = \rho \circ \sigma$. (Without loss of generality, σ is idempotent, therefore $C'\theta = C'\sigma\rho = C'\sigma\sigma\rho = C'\sigma\theta$, so $C'\theta$ is a ground instance of $C'\sigma$.) Since $C\theta$ is not redundant w. r. t. $G_\Sigma(N)$, C is not redundant w. r. t. N. As N is saturated up to redundancy, the conclusion $C'\sigma$ of the inference from C is contained in $N \cup Red(N)$. Therefore, $C'\theta$ is either contained in $G_\Sigma(N)$ and smaller than $C\theta$, or it follows from clauses in $G_\Sigma(N)$ that are smaller than itself (and therefore smaller than $C\theta$). By the induction hypothesis, clauses in $G_\Sigma(N)$ that are smaller than $C\theta$ are true in $R_{C\theta}$, thus $C'\theta$ and $C\theta$ are true in $R_{C\theta}$.

Case 3.2: $s\theta \succ s'\theta$.

If $s\theta \downarrow_{R_{C\theta}} s'\theta$ and $s\theta \succ s'\theta$, then $s\theta$ must be reducible by some rule in some $E_{D\theta} \subseteq R_{C\theta}$. (Without loss of generality we assume that C and D are variable disjoint; so we can use the same substitution θ.) Let $D\theta = D'\theta \vee t\theta \approx t'\theta$ with $E_{D\theta} = \{t\theta \to t'\theta\}$. Since $D\theta$ is productive, $D'\theta$ is false in $R_{C\theta}$. Besides, by part (ii) of the induction hypothesis, $D\theta$ is not redundant w. r. t. $G_\Sigma(N)$, so D is not redundant w. r. t. N. Note that $t\theta$ cannot occur in $s\theta$ at or below a variable position of s, say $x\theta = w[t\theta]$, since otherwise $C\theta$ would be subject to Case 2 above. Consequently, the *negative superposition* inference

$$\frac{D'\theta \vee t\theta \approx t'\theta \qquad C'\theta \vee s\theta[t\theta] \not\approx s'\theta}{D'\theta \vee C'\theta \vee s\theta[t'\theta] \not\approx s'\theta}$$

is a ground instance of a *negative superposition* inference from D and C. By saturation up to redundancy, its conclusion is either contained in $G_\Sigma(N)$ and smaller than $C\theta$, or it follows from clauses in $G_\Sigma(N)$ that are smaller than itself (and therefore smaller than $C\theta$). By the induction hypothesis, these clauses are true in $R_{C\theta}$, thus $D'\theta \vee C'\theta \vee s\theta[t'\theta] \not\approx s'\theta$ is true in $R_{C\theta}$. Since $D'\theta$ and $s\theta[t'\theta] \not\approx s'\theta$ are false in $R_{C\theta}$, both $C'\theta$ and $C\theta$ must be true.

Case 4: $C\theta$ does not contain a maximal negative literal.

Suppose that $C\theta$ does not fall into Cases 1 to 3. Then $C\theta$ can be written as $C'\theta \vee s\theta \approx s'\theta$, where $s\theta \approx s'\theta$ is a maximal literal of $C\theta$. If $E_{C\theta} = \{s\theta \to s'\theta\}$ or $C'\theta$ is true in $R_{C\theta}$ or $s\theta = s'\theta$, then there is nothing to show, so assume that $E_{C\theta} = \emptyset$ and that $C'\theta$ is false in $R_{C\theta}$. Without loss of generality, $s\theta \succ s'\theta$.

Case 4.1: $s\theta \approx s'\theta$ is maximal in $C\theta$, but not strictly maximal.

If $s\theta \approx s'\theta$ is maximal in $C\theta$, but not strictly maximal, then $C\theta$ can be written as $C''\theta \vee t\theta \approx t'\theta \vee s\theta \approx s'\theta$, where $t\theta = s\theta$ and $t'\theta = s'\theta$. In this case, there is a *equality factoring* inference

$$\frac{C''\theta \vee t\theta \approx t'\theta \vee s\theta \approx s'\theta}{C''\theta \vee t'\theta \not\approx s'\theta \vee t\theta \approx t'\theta}$$

This inference is a ground instance of an inference from C. By saturation, its conclusion is true in $R_{C\theta}$. Trivially, $t'\theta = s'\theta$ implies $t'\theta \downarrow_{R_{C\theta}} s'\theta$, so $t'\theta \not\approx s'\theta$ must be false and $C\theta$ must be true in $R_{C\theta}$.

Case 4.2: $s\theta \approx s'\theta$ is strictly maximal in $C\theta$ and $s\theta$ is reducible.

Suppose that $s\theta \approx s'\theta$ is strictly maximal in $C\theta$ and $s\theta$ is reducible by some rule in $E_{D\theta} \subseteq R_{C\theta}$. Let $D\theta = D'\theta \vee t\theta \approx t'\theta$ and $E_{D\theta} = \{t\theta \to t'\theta\}$. Since $D\theta$ is productive, $D\theta$ is not redundant and $D'\theta$ is false in $R_{C\theta}$. We can now proceed in essentially the same way as in Case 3.2: If $t\theta$ occurred in $s\theta$ at or below a variable position of s, say $x\theta = w[t\theta]$, then $C\theta$ would be subject to Case 2 above. Otherwise, the *positive superposition* inference

$$\frac{D'\theta \vee t\theta \approx t'\theta \qquad C'\theta \vee s\theta[t\theta] \approx s'\theta}{D'\theta \vee C'\theta \vee s\theta[t'\theta] \approx s'\theta}$$

is a ground instance of a *positive superposition* inference from D and C. By saturation up to redundancy, its conclusion is true in $R_{C\theta}$. Since $D'\theta$ and $C'\theta$ are false in $R_{C\theta}$, $s\theta[t'\theta] \approx s'\theta$ must be true in $R_{C\theta}$. On the other hand, $t\theta \approx t'\theta$ is true in $R_{C\theta}$, so by congruence, $s\theta[t\theta] \approx s'\theta$ and $C\theta$ are true in $R_{C\theta}$.

Case 4.3: $s\theta \approx s'\theta$ is strictly maximal in $C\theta$ and $s\theta$ is irreducible.

Suppose that $s\theta \approx s'\theta$ is strictly maximal in $C\theta$ and $s\theta$ is irreducible by $R_{C\theta}$. Then there are three possibilities: $C\theta$ can be true in $R_{C\theta}$, or $C'\theta$ can be true in $R_{C\theta} \cup \{s\theta \to s'\theta\}$, or $E_{C\theta} = \{s\theta \to s'\theta\}$. In the first and the third case, there is nothing to show. Let us therefore assume that $C\theta$ is false in $R_{C\theta}$ and $C'\theta$ is true in $R_{C\theta} \cup \{s\theta \to s'\theta\}$. Then $C'\theta = C''\theta \vee t\theta \approx t'\theta$, where the literal $t\theta \approx t'\theta$ is true in $R_{C\theta} \cup \{s\theta \to s'\theta\}$ and false in $R_{C\theta}$. In other words, $t\theta \downarrow_{R_{C\theta} \cup \{s\theta \to s'\theta\}} t'\theta$, but not $t\theta \downarrow_{R_{C\theta}} t'\theta$. Consequently, there is a rewrite proof of $t\theta \to^* u \leftarrow^* t'\theta$ by $R_{C\theta} \cup \{s\theta \to s'\theta\}$ in which the rule $s\theta \to s'\theta$ is used at least once. Without loss of generality we assume that $t\theta \succeq t'\theta$. Since $s\theta \approx s'\theta \succ_{\mathrm{L}} t\theta \approx t'\theta$ and $s\theta \succ s'\theta$ we can conclude that $s\theta \succeq t\theta \succ t'\theta$. But then there is only one possibility how the rule $s\theta \to s'\theta$ can be used in the rewrite proof: We must have $s\theta = t\theta$ and the rewrite proof must have the form $t\theta \to s'\theta \to^* u \leftarrow^* t'\theta$, where the first step uses $s\theta \to s'\theta$ and all other steps use rules from $R_{C\theta}$. Consequently, $s'\theta \approx t'\theta$ is true in $R_{C\theta}$. Now observe that there is an *equality factoring* inference

$$\frac{C''\theta \vee t\theta \approx t'\theta \vee s\theta \approx s'\theta}{C''\theta \vee t'\theta \not\approx s'\theta \vee t\theta \approx t'\theta}$$

whose conclusion is true in $R_{C\theta}$ by saturation. Since the literal $t'\theta \not\approx s'\theta$ must be false in $R_{C\theta}$, the rest of the clause must be true in $R_{C\theta}$, and therefore $C\theta$ must be true in $R_{C\theta}$, contradicting our assumption. This concludes the proof of the theorem. □

A Σ-interpretation $\mathcal{A}$ is called *term-generated,* if for every $b \in U_{\mathcal{A}}$ there is a ground term $t \in \mathrm{T}_\Sigma(\emptyset)$ such that $b = \mathcal{A}(\beta)(t)$.

Lemma 3.10 *Let N be a set of (universally quantified) Σ-clauses and let $\mathcal{A}$ be a term-generated Σ-interpretation. Then $\mathcal{A}$ is a model of $G_\Sigma(N)$ if and only if it is a model of N.*

Proof. ($\Rightarrow$): Let $\mathcal{A} \models G_\Sigma(N)$; let $(\forall \vec{x}\, C) \in N$. Then $\mathcal{A} \models \forall \vec{x}\, C$ iff $\mathcal{A}(\gamma[x_i \mapsto a_i])(C) = 1$ for all γ and a_i. Choose ground terms t_i such that $\mathcal{A}(\gamma)(t_i) = a_i$; define θ such that $x_i\theta = t_i$, then $\mathcal{A}(\gamma[x_i \mapsto a_i])(C) = \mathcal{A}(\gamma \circ \theta)(C) = \mathcal{A}(\gamma)(C\theta) = 1$ since $C\theta \in G_\Sigma(N)$.

($\Leftarrow$): Let $\mathcal{A}$ be a model of N; let $\forall \vec{x}\, C \in N$ and $C\theta \in G_\Sigma(N)$. Then $\mathcal{A} \models \forall \vec{x}\, C$ and therefore $\mathcal{A} \models C$. Consequently $\mathcal{A}(\gamma)(C\theta) = \mathcal{A}(\gamma \circ \theta)(C) = 1$. □

Theorem 3.11 (Refutational Completeness: Static View) *Let N be a set of clauses that is saturated up to redundancy. Then N has a model if and only if N does not contain the empty clause.*

Proof. If $\bot \in N$, then obviously N does not have a model. If $\bot \notin N$, then the interpretation R_∞ (that is, $\mathrm{T}_\Sigma(\emptyset)/R_\infty$) is a model of all ground instances in $G_\Sigma(N)$ according to part (iii) of the model construction theorem. As $\mathrm{T}_\Sigma(\emptyset)/R_\infty$ is term-generated, it is a model of N. □

So far, we have considered only inference rules that add new clauses to the current set of clauses (corresponding to the *Deduce* rule of Knuth-Bendix Completion).

In other words, we have derivations of the form $N_0 \vdash N_1 \vdash N_2 \vdash \ldots$, where each N_{i+1} is obtained from N_i by adding the consequence of some inference from clauses in N_i.

Under which circumstances are we allowed to delete (or simplify) a clause during the derivation?

A *run* of the superposition calculus is a sequence $N_0 \vdash N_1 \vdash N_2 \vdash \ldots$, such that
(i) $N_i \models N_{i+1}$, and
(ii) all clauses in $N_i \setminus N_{i+1}$ are redundant w. r. t. N_{i+1}.

In other words, during a run we may add a new clause if it follows from the old ones, and we may delete a clause, if it is redundant w. r. t. the remaining ones.

For a run, $N_\infty = \bigcup_{i \geq 0} N_i$ and $N_* = \bigcup_{i \geq 0} \bigcap_{j \geq i} N_j$. The set N_* of all *persistent* clauses is called the *limit* of the run.

Lemma 3.12 *If $N \subseteq N'$, then $Red(N) \subseteq Red(N')$.*

Proof. Obvious. □

Lemma 3.13 *If $N' \subseteq Red(N)$, then $Red(N) \subseteq Red(N \setminus N')$.*

Proof. Follows from the compactness of first-order logic and the well-foundedness of the multiset extension of the clause ordering. □

Lemma 3.14 *Let $N_0 \vdash N_1 \vdash N_2 \vdash \ldots$ be a run. Then $Red(N_i) \subseteq Red(N_\infty)$ and $Red(N_i) \subseteq Red(N_*)$ for every i.*

Proof. Exercise. □

Corollary 3.15 *$N_i \subseteq N_* \cup Red(N_*)$ for every i.*

Proof. If $C \in N_i \setminus N_*$, then there is a $k \geq i$ such that $C \in N_k \setminus N_{k+1}$, so C must be redundant w. r. t. N_{k+1}. Consequently, C is redundant w. r. t. N_*. □

A run is called *fair*, if the conclusion of every inference from clauses in $N_* \setminus \mathit{Red}(N_*)$ is contained in some $N_i \cup \mathit{Red}(N_i)$.

Lemma 3.16 *If a run is fair, then its limit is saturated up to redundancy.*

Proof. If the run is fair, then the conclusion of every inference from non-redundant clauses in N_* is contained in some $N_i \cup \mathit{Red}(N_i)$, and therefore contained in $N_* \cup \mathit{Red}(N_*)$. Hence N_* is saturated up to redundancy. □

Theorem 3.17 (Refutational Completeness: Dynamic View) *Let $N_0 \vdash N_1 \vdash N_2 \vdash \ldots$ be a fair run, let N_* be its limit. Then N_0 has a model if and only if $\bot \notin N_*$.*

Proof. ($\Leftarrow$): By fairness, N_* is saturated up to redundancy. If $\bot \notin N_*$, then it has a term-generated model. Since every clause in N_0 is contained in N_* or redundant w. r. t. N_*, this model is also a model of $G_\Sigma(N_0)$ and therefore a model of N_0.

($\Rightarrow$): Obvious, since $N_0 \models N_*$. □

3.5 Improvements and Refinements

The superposition calculus as described so far can be improved and refined in several ways.

Concrete Redundancy and Simplification Criteria

Redundancy is undecidable.

Even decidable approximations are often expensive (experimental evaluations are needed to see what pays off in practice).

Often a clause can be *made* redundant by adding another clause that is entailed by the existing ones.

This process is called *simplification*.

Examples:

Subsumption:
If N contains clauses D and $C = C' \vee D\sigma$, where C' is non-empty, then D subsumes C and C is redundant.
Example: $f(x) \approx g(x)$ subsumes $f(y) \approx a \vee f(h(y)) \approx g(h(y))$.

Trivial literal elimination:
Duplicated literals and trivially false literals can be deleted: A clause $C' \vee L \vee L$ can be simplified to $C' \vee L$; a clause $C' \vee s \not\approx s$ can be simplified to C'.

Condensation:
If we obtain a clause D from C by applying a substitution, followed by deletion of duplicated literals, and if D subsumes C, then C can be simplified to D.
Example: By applying $\{y \to g(x)\}$ to $C = f(g(x)) \approx a \vee f(y) \approx a$ and deleting the duplicated literal, we obtain $f(g(x)) \approx a$, which subsumes C.

Semantic tautology deletion:
Every clause that is a tautology is redundant. Note that in the non-equational case, a clause is a tautology if and only if it contains two complementary literals, whereas in the equational case we need a congruence closure algorithm to detect that a clause like $x \not\approx y \vee f(x) \approx f(y)$ is tautological.

Rewriting:
If N contains a unit clause $D = s \approx t$ and a clause $C[s\sigma]$, such that $s\sigma \succ t\sigma$ and $C \succ_C D\sigma$, then C can be simplified to $C[t\sigma]$.
Example: If $D = f(x, x) \approx g(x)$ and $C = h(f(g(y), g(y))) \approx h(y)$, and $\succ$ is an LPO with $h > f > g$, then C can be simplified to $h(g(g(y))) \approx h(y)$.

Selection Functions

Like the ordered resolution calculus, superposition can be used with a selection function that overrides the ordering restrictions for negative literals.

A *selection function* is a mapping

$$S : C \quad \mapsto \quad \text{set of occurrences of } \textit{negative} \text{ literals in } C$$

We indicate selected literals by a box:

$$\boxed{\neg f(x) \approx a} \;\vee\; g(x, y) \approx g(x, z)$$

The second ordering condition for inferences is replaced by

- The last literal in each premise is either selected, or there is no selected literal in the premise and the literal is maximal in the premise (strictly maximal for positive literals in superposition inferences).

In particular, clauses with selected literals can only be used in equality resolution inferences and as the second premise in negative superposition inferences.

Refutational completeness is proved essentially as before:

We assume that each ground clause in $G_\Sigma(N)$ inherits the selection of one of the clauses in N of which it is a ground instance (there may be several ones!).

In the proof of the model construction theorem, we replace case 3 by "$C\theta$ contains a selected or maximal negative literal" and case 4 by "$C\theta$ contains neither a selected nor a maximal negative literal".

In addition, for the induction proof of this theorem we need one more property, namely: (iv) If $C\theta$ has selected literals then $E_{C\theta} = \emptyset$.

Redundant Inferences

So far, we have defined saturation in terms of redundant clauses:

N is *saturated up to redundancy,* if the conclusion of every inference from clauses in $N \setminus \mathit{Red}(N)$ is contained in $N \cup \mathit{Red}(N)$.

This definition ensures that in the proof of the model construction theorem, the conclusion $C_0\theta$ of a ground inference follows from clauses in $G_\Sigma(N)$ that are smaller than or equal to itself, hence they are smaller than the premise $C\theta$ of the inference, hence they are true in $R_{C\theta}$ by induction.

However, a closer inspection of the proof shows that it is actually sufficient that the clauses from which $C_0\theta$ follows are smaller than $C\theta$ – it is *not* necessary that they are smaller than $C_0\theta$ itself. This motivates the following definition of redundant *inferences*:

A ground inference with conclusion C_0 and right (or only) premise C is called *redundant w. r. t. a set of ground clauses* N, if one of its premises is redundant w. r. t. N, or if C_0 follows from clauses in N that are smaller than C.

An inference is *redundant w. r. t. a set of clauses* N, if all its ground instances are redundant w. r. t. $G_\Sigma(N)$.

Recall that a clause can be redundant w. r. t. N without being contained in N. Analogously, an inference can be redundant w. r. t. N without being an inference from clauses in N.

The set of all inferences that are redundant w. r. t. N is denoted by $\mathit{RedInf}(N)$.

Saturation is then redefined in the following way:

N is *saturated up to redundancy*, if every inference from clauses in N is redundant w. r. t. N.

Using this definition, the model construction theorem can be proved essentially as before.

The connection between redundant inferences and clauses is given by the following lemmas. They are proved in the same way as the corresponding lemmas for redundant clauses:

Lemma 3.18 *If $N \subseteq N'$, then $\mathit{RedInf}(N) \subseteq \mathit{RedInf}(N')$.*

Lemma 3.19 *If $N' \subseteq \mathit{Red}(N)$, then $\mathit{RedInf}(N) \subseteq \mathit{RedInf}(N \setminus N')$.*

Literature

Leo Bachmair, Harald Ganzinger: Completion of First-Order Clauses with Equality by Strict Superposition (Extended Abstract). Conditional and Typed Rewriting Systems, 2nd International Workshop, LNCS 516, pp. 162–180, Springer, 1990.

Leo Bachmair, Harald Ganzinger: Rewrite-based Equational Theorem Proving with Selection and Simplification. Journal of Logic and Computation, 4(3):217–247, 1994.

Leo Bachmair, Harald Ganzinger: Resolution Theorem Proving. Handbook of Automated Reasoning, Vol. 1, Ch. 2, pp. 19–99, Elsevier Science B.V., 2001.

Christoph Weidenbach: Combining Superposition, Sorts and Splitting. Handbook of Automated Reasoning, Vol. 2, Ch. 27, pp. 1965–2013, Elsevier Science B.V., 2001.

3.6 Splitting

Motivation:

A clause like $f(x) \approx a \ \vee \ g(y) \approx b$ has rather undesirable properties in the superposition calculus: It does not have negative literals that one could select; it does not have a unique maximal literal; moreover, after performing a superposition inference with this clause, the conclusion often does not have a unique maximal literal either.

On the other hand, the two unit clauses $f(x) \approx a$ and $g(y) \approx b$ have much nicer properties.

Splitting with Backtracking

If a clause $\forall \vec{x}, \vec{y}\ C_1(\vec{x}) \vee C_2(\vec{y})$ consists of two non-empty variable-disjoint subclauses, then it is equivalent to the disjunction $\big(\forall \vec{x}\ C_1(\vec{x})\big) \ \vee \ \big(\forall \vec{y}\ C_2(\vec{y})\big)$.

In this case, superposition derivations can branch in a tableau-like manner:

Splitting: $$\frac{N \cup \{C_1 \vee C_2\}}{N \cup \{C_1\} \quad | \quad N \cup \{C_2\}}$$
where C_1 and C_2 do not have common variables.

If $\bot$ is found on the left branch, backtrack to the right one.

If C_1 is ground, the general rule can be improved:

Splitting: $$\frac{N \cup \{C_1 \vee C_2\}}{N \cup \{C_1\} \quad | \quad N \cup \{C_2\} \cup \{\neg C_1\}}$$
where C_1 is ground.

Note: $\neg C_1$ denotes the conjunction of all negations of literals in C_1.

In practice: most useful if both subclauses contain at least one positive literal.

Implementing Splitting

Most clauses that are derived after a splitting step do *not* depend on the split clause.

It is unpractical to delete them as soon as one branch is closed and to recompute them in the other branch afterwards.

Solution: Associate a label set $\mathcal{L}$ to every clause C that indicates on which splits it depends.

Inferences: $$\frac{C_2 \leftarrow \mathcal{L}_2 \qquad C_1 \leftarrow \mathcal{L}_1}{C_0 \leftarrow \mathcal{L}_2 \cup \mathcal{L}_1}$$

If we derive $\bot \leftarrow \mathcal{L}$ in one branch:

Determine the last split in $\mathcal{L}$.

Backtrack to the corresponding right branch.

Keep those clauses that are still valid on the right branch.

Restore clauses that have been simplified if the simplifying clause is no longer valid on the right branch.

Additionally: Delete splittings that did not contribute to the contradiction (branch condensation).

AVATAR

Superposition with splitting has some similarity with CDCL.

Can we actually use CDCL?

Encoding splitting components:

Use propositional literals as labels for splitting components:

non-ground component $C \rightarrow$ propositional variable P_C
positive ground component $C \rightarrow$ propositional variable P_C
negative ground component $C \rightarrow$ negated propositional variable $\neg P_C$

Therefore: splittable clauses $\rightarrow$ propositional clauses.

Implementation:

Combine a CDCL solver and a superposition prover.

The superposition prover passes splittable clauses and labelled empty clauses to the CDCL solver.

If the CDCL solver finds contradiction: input contradictory.

Otherwise the CDCL solver extracts a boolean model and passes the associated labelled clauses to the superposition prover.

Literature

Andrei Voronkov: AVATAR: The Architecture for First-Order Theorem Provers. Int. Conf. on Computer-Aided Verification, CAV, LNCS 8559, pp. 696–710, Springer, 2014.

Christoph Weidenbach: Combining Superposition, Sorts and Splitting. Handbook of Automated Reasoning, Vol. 2, Ch. 27, pp. 1965–2013, Elsevier Science B.V., 2001.

3.7 Constraint Superposition

So far:

Refutational completeness proof for superposition is based on the analysis of inferences between ground instances of clauses.

Inferences between ground instances must be covered by inferences between original clauses.

Non-ground clauses represent the set of all their ground instances.

Do we really need *all* ground instances?

Constrained Clauses

A *constrained clause* is a pair (C, K), usually written as $C \,[\![K]\!]$, where C is a Σ-clause and K is a formula (called *constraint*).

Often: K is a boolean combination of *ordering literals* $s \succ t$ with Σ-terms s, t.
(also possible: comparisons between literals or clauses).

Intuition: $C \,[\![K]\!]$ represents the set of all ground clauses $C\theta$ for which $K\theta$ evaluates to true for some fixed term ordering. Such a $C\theta$ is called a ground instance of $C \,[\![K]\!]$.

A clause C without constraint is identified with $C \,[\![\top]\!]$.

A constrained clause $C \,[\![\bot]\!]$ with an unsatisfiable constraint represents no ground instances; it can be discarded.

Constraint Superposition

Inference rules for constrained clauses:

Pos. Superposition:

$$\frac{D' \vee t \approx t' \,[\![K_2]\!] \qquad C' \vee s[u] \approx s' \,[\![K_1]\!]}{(D' \vee C' \vee s[t'] \approx s')\sigma \,[\![(K_2 \wedge K_1 \wedge K)\sigma]\!]}$$

where $\sigma = \mathrm{mgu}(t, u)$ and
u is not a variable and
$K = (t \succ t' \;\wedge\; s[u] \succ s'$
$\wedge\; (t \approx t') \succ_C D'$
$\wedge\; (s[u] \approx s') \succ_C C'$
$\wedge\; (s[u] \approx s') \succ_L (t \approx t'))$

The other inference rules are modified analogously.

To work effectively with constrained clauses in a calculus, we need methods to check the satisfiability of constraints:

Possible for LPO, KBO, but expensive.

If constraints become too large, we may delete some conjuncts of the constraint. (Note that the calculus remains sound, if constraints are replaced by implied constraints.)

Refutational Completeness

The refutational completeness proof for constraint superposition looks mostly like in Sect. 3.4.

Lifting works as before, so every ground infererence that is required in the proof is an instance of some inference from the corresponding constrained clauses. (Easy.)

There is one significant problem, though.

Case 2 in the proof of Thm. 3.9 does not work for constrained clauses:

If we have a ground instance $C\theta$ where $x\theta$ is reducible by $R_{C\theta}$, we can no longer conclude that $C\theta$ is true because it follows from some rule in $R_{C\theta}$ and some smaller ground instance $C\theta'$.

Example: Let $C\,[\![K]\!]$ be the clause $f(x) \approx a\ [\![x \succ a]\!]$, let $\theta = \{x \mapsto b\}$, and assume that $R_{C\theta}$ contains the rule $b \to a$.
Then θ satisfies K, but $\theta' = \{x \mapsto a\}$ does not, so $C\theta'$ is *not* a ground instance of $C\,[\![K]\!]$.

Solution:

Assumption: We start the saturation with a set N_0 of *unconstrained* clauses; the limit N_* contains constrained clauses, though.

During the model construction, we ignore ground instances $C\theta$ of clauses in N_* for which $x\theta$ is reducible by $R_{C\theta}$.

We obtain a model R_∞ of all *variable irreducible* ground instances of clauses in N_*.

R_∞ is also a model of all *variable irreducible* ground instances of clauses in N_0.

Since all clauses in N_0 are unconstrained, every ground instance of a clause in N_0 follows from some rule in R_∞ and some smaller ground instance; so it is true in R_∞.

Consequently, R_∞ is a model of *all* ground instances of clauses in N_0.

Other Constraints

The approach also works for other kinds of constraints.

In particular, we can replace unification by equality constraints ($\rightsquigarrow$ "basic superposition"):

Pos. Superposition:
$$\frac{D' \vee t \approx t' \,[\![K_2]\!] \qquad C' \vee s[u] \approx s' \,[\![K_1]\!]}{D' \vee C' \vee s[t'] \approx s' \,[\![K_2 \wedge K_1 \wedge K]\!]}$$
where u is not a variable and
$K = (t = u)$

Note: In contrast to ordering constraints, these constraints are essential for soundness.

The Drawback

Constraints reduce the number of required inferences; however, they are detrimental to redundancy:

Since we consider only *variable irreducible* ground instances during the model construction, we may use only such instances for redundancy:

A clause is redundant, if all its variable irreducible ground instances follow from smaller variable irreducible ground instances.

Even worse, since we don't know R_∞ in advance, we must consider variable irreducibility w. r. t. arbitrary rewrite systems.

Consequence: Not every subsumed clause is redundant!

Literature

Robert Nieuwenhuis, Albert Rubio: Paramodulation-Based Theorem Proving. Handbook of Automated Reasoning, Vol. 1, Ch. 7, pp. 371–443, Elsevier Science B.V., 2001.

3.8 Hierarchic Superposition

The superposition calculus is a powerful tool to deal with formulas in *uninterpreted* first-order logic.

What can we do if some symbols have a *fixed interpretation*?

Can we combine superposition with decision procedures, e. g., for linear rational arithmetic? Can we integrate the decision procedure as a "black box"?

Sorted Logic

It is useful to treat this problem in sorted logic (cf. Sect. 1.11, page 31).

A many-sorted signature $\Sigma = (\Xi, \Omega, \Pi)$ fixes an alphabet of non-logical symbols, where

- Ξ is a set of sort symbols,
- Ω is a sets of function symbols,
- Π is a set of predicate symbols.

Each function symbol $f \in \Omega$ has a unique declaration $f : \xi_1 \times \cdots \times \xi_n \to \xi_0$; each predicate symbol $P \in \Pi$ has a unique declaration $P : \xi_1 \times \cdots \times \xi_n$ with $\xi_i \in \Xi$.

In addition, each variable x has a unique declaration $x : \xi$.

We assume that all terms, atoms, substitutions are well-sorted.

A many-sorted algebra $\mathcal{A}$ consists of

- a non-empty set $\xi_\mathcal{A}$ for each $\xi \in \Xi$,
- a function $f_\mathcal{A} : \xi_{1,\mathcal{A}} \times \cdots \times \xi_{n,\mathcal{A}} \to \xi_{0,\mathcal{A}}$ for each $f : \xi_1 \times \cdots \times \xi_n \to \xi_0 \in \Omega$,
- a subset $P_\mathcal{A} \subseteq \xi_{1,\mathcal{A}} \times \cdots \times \xi_{n,\mathcal{A}}$ for each $P : \xi_1 \times \cdots \times \xi_n \in \Pi$.

Hierarchic Specifications

A *specification* $SP = (\Sigma, \mathcal{C})$ consists of

- a signature $\Sigma = (\Xi, \Omega, \Pi)$,
- a class of term-generated Σ-algebras $\mathcal{C}$ closed under isomorphisms.

If $\mathcal{C}$ consists of *all* term-generated Σ-algebras satisfying the set of Σ-formulas N, we write $SP = (\Sigma, N)$.

A *hierarchic specification* $HSP = (SP, SP')$ consists of

- a base specification $SP = (\Sigma, \mathcal{C})$,
- an extension $SP' = (\Sigma', N')$,

where $\Sigma = (\Xi, \Omega, \Pi)$, $\Sigma' = (\Xi', \Omega', \Pi')$, $\Xi \subseteq \Xi'$, $\Omega \subseteq \Omega'$, and $\Pi \subseteq \Pi'$.

A Σ'-algebra $\mathcal{A}$ is called a model of $HSP = (SP, SP')$, if $\mathcal{A}$ is a model of N' and $\mathcal{A}|_\Sigma \in \mathcal{C}$, where the reduct $\mathcal{A}|_\Sigma$ is defined as $((\xi_\mathcal{A})_{\xi \in \Xi}, (f_\mathcal{A})_{f \in \Omega}, (P_\mathcal{A})_{P \in \Pi})$.

Note:

- no confusion: models of HSP may not identify elements that are different in the base models.
- no junk: models of HSP may not add new elements to the interpretations of base sorts.

Example:

Base specification: $((\Xi, \Omega, \Pi), \mathcal{C})$, where

$\Xi = \{ int \}$

$\Omega = \{ 0, 1, -1, 2, -2, \ldots : \to int,$
$\quad - : int \to int,$
$\quad + : int \times int \to int \}$

$\Pi = \{ \geq : int \times int,$
$\quad > : int \times int \}$

$\mathcal{C} =$ isomorphy class of $\mathbb{Z}$

Extension: $((\Xi', \Omega', \Pi'), N')$, where

$\Xi' = \Xi \cup \{ list \}$

$\Omega' = \Omega \cup \{ cons : int \times list \to list,$
$\quad length : list \to int,$
$\quad empty : \to list,$
$\quad a : \to list \}$

$\Pi' = \Pi$

$N' = \{ length(a) \geq 1,$
$\quad length(cons(x, y)) \approx length(y) + 1 \}$

Goal:

Check whether N' has a model in which the sort int is interpreted by $\mathbb{Z}$ and the symbols from Ω and Π accordingly.

Hierarchic Superposition

In order to use a prover for the base theory, we must preprocess the clauses:

A term that consists only of base symbols and variables of base sort is called a base term (analogously for atoms, literals, clauses).

A clause C is called *weakly abstracted*, if every base term that occurs in C as a subterm of a non-base term (or non-base non-equational literal) is a variable.

Every clause can be transformed into an equivalent weakly abstracted clause. We assume that all input clauses are weakly abstracted.

A substitution is called simple, if it maps every variable of a base sort to a base term.

The inference rules of the hierarchic superposition calculus correspond to the rules of of the standard superposition calculus with the following modifications:

- The term ordering $\succ$ must have the property that every base ground term (or non-equational literal) is smaller than every non-base ground term (or non-equational literal).
- We consider only simple substitutions as unifiers.
- We perform only inferences on non-base terms (or non-base non-equational literals).
- If the conclusion of an inference is not weakly abstracted, we transform it into an equivalent weakly abstracted clause.

While clauses that contain non-base literals are manipulated using superposition rules, base clauses have to be passed to the base prover.

This yields one more inference rule:

Constraint Refutation: $\dfrac{M}{\bot}$

where M is a set of base clauses
that is inconsistent w. r. t. $\mathcal{C}$.

Problems

There are two potential problems that are harmful to refutational completeness:

- We can only apply the constraint refutation rule to finite sets M. If $\mathcal{C}$ is not compact, this is not sufficient.
- Since we only consider simple substitutions, we will only obtain a model of all *simple ground instances*.

  To show that we have a model of *all* instances, we need an additional condition called *sufficient completeness w. r. t. simple instances*.

A set N of clauses is called *sufficiently complete with respect to simple instances*, if for every model $\mathcal{A}'$ of the set of simple ground instances of N and every ground non-base term t of a base sort there exists a ground base term t such that $t' \approx t$ is true in $\mathcal{A}'$.

Note: Sufficient completeness w. r. t. simple instances ensures the absence of junk.

If the base signature contains Skolem constants, we can sometimes enforce sufficient completeness by equating ground extension terms with a base sort to Skolem constants.

Skolem constants may harmful to compactness, though.

Completeness of Hierarchic Superposition

If the base theory is compact, the hierarchic superposition calculus is refutationally complete for sets of clauses that are sufficiently complete with respect to simple instances (Bachmair, Ganzinger, Waldmann, 1994; Baumgartner, Waldmann 2013).

Main proof idea:

If the set of base clauses in N has some base model, represent this model by a set E of convergent ground equations and a set D of ground disequations.

Then show: If N is saturated w. r. t. hierarchic superposition, then $E \cup D \cup \tilde{N}$ is saturated w. r. t. standard superposition, where $\tilde{N}$ is the set of simple ground instances of clauses in N that are reduced w. r. t. E.

A Refinement

In practice, a base signature often contains *domain elements*, that is, constant symbols that are

- guaranteed to be different from each other in every base model, and
- minimal w. r. t. $\succ$ in their equivalent class.

Typical example for domain elements: number constants $0, 1, -1, 2, -2, \ldots$

If the base signature contains *domain elements*, then weak abstraction can be redefined as follows:

A clause C is called *weakly abstracted*, if every base term that occurs in C as a subterm of a non-base term (or non-base non-equational literal) is a variable *or a domain element*.

Why does that work?

Literature

Leo Bachmair, Harald Ganzinger. Uwe Waldmann: Refutational Theorem Proving for Hierarchic First-Order Theories. Applicable Algebra in Engineering, Communication and Computing, 5(3/4):193–212, 1994.

Peter Baumgartner, Uwe Waldmann: Hierarchic Superposition With Weak Abstraction. Automated Deduction, CADE-24, LNAI 7898, pp. 39–57, Springer, 2013.

3.9 Integrating Theories I: E-Unification

Dealing with mathematical theories naively in a superposition prover is difficult:

Some axioms (e. g., commutativity) cannot be oriented w. r. t. a reduction ordering.
⇒ Provers compute many equivalent copies of a formula.

Some axiom sets (e. g., torsion-freeness, divisibility) are infinite.
⇒ Can we tell which axioms are really needed?

Hierarchic ("black-box") superposition is easy to implement, but conditions like compactness and sufficient completeness are rather restrictive.

Can we integrate theories directly into theorem proving calculi ("white-box" integration)?

Idea:

In order to avoid enumerating entire congruence classes w. r. t. an equational theory E, treat formulas as *representatives* of their congruence classes.

Compute an inference between formula C and D if an inference between some clause represented by C and some clause represented by D would be possible.

Consequence: We have to check whether there are substitutions that make terms s and t equal w. r. t. E.
⇒ Unification is replaced by E-unification.

E-Unification

E-unification (unification modulo an equational theory E):

For a set of equality problems $\{s_1 \approx t_1, \ldots, s_n \approx t_n\}$, an E-unifier is a substitution σ such that for all $i \in \{1, \ldots, n\}$: $s_i\sigma \approx_E t_i\sigma$.

Recall: $s_i\sigma \approx_E t_i\sigma$ means $E \models s_i\sigma \approx t_i\sigma$.

In general, there are infinitely many (E-)unifiers.
What about most general unifiers?

Frequent cases: $E = \emptyset$, $E = \mathrm{AC}$, $E = \mathrm{ACU}$:

$$
\begin{array}{rcll}
x + (y + z) & \approx & (x + y) + z & (\text{associativity} = \mathrm{A}) \\
x + y & \approx & y + x & (\text{commutativity} = \mathrm{C}) \\
x + 0 & \approx & x & (\text{identity (unit)} = \mathrm{U})
\end{array}
$$

The identity axiom is also abbreviated by "1", in particular, if the binary operation is denoted by $*$. (ACU = AC1).

Example:

$x + y$ and c are ACU-unifiable with $\{x \mapsto c,\ y \mapsto 0\}$ and $\{x \mapsto 0,\ y \mapsto c\}$.

$x + y$ and $x' + y'$ are ACU-unifiable with $\{x \mapsto z_1 + z_2,\ y \mapsto z_3 + z_4,\ x' \mapsto z_1 + z_3,\ y' \mapsto z_2 + z_4\}$ (among others).

More general substitutions:

Let X be a set of variables.
A substitution σ is more general modulo E than a substitution σ' on X, if there exists a substitution ρ such that $x\sigma\rho \approx_E x\sigma'$ for all $x \in X$.

Notation: $\sigma \lesssim_E^X \sigma'$.

(Why X? Because we cannot restrict to idempotent substitutions.)

Complete sets of unifiers:

Let S be an E-unification problem, let $X = Var(S)$.
A set C of E-unifiers of S is called complete (CSU),
if for every E-unifier σ' of S there exists a $\sigma \in C$
with $\sigma \lesssim_E^X \sigma'$.

A complete set of E-unifiers C is called minimal (μCSU),
if for all $\sigma, \sigma' \in C$, $\sigma \lesssim_E^X \sigma'$ implies $\sigma = \sigma'$.

Note: every E-unification problem has a CSU. (Why?)

The set of equations E is of unification type

unitary, if every E-unification problem has a μCSU with cardinality ≤ 1 (e. g.: $E = \emptyset$);

finitary, if every E-unification problem has a finite μCSU (e. g.: $E =$ ACU, $E =$ AC, $E =$ C);

infinitary, if every E-unification problem has a μCSU and some E-unification problem has an infinite μCSU (e. g.: $E =$ A);

zero (or nullary), if some E-unification problem does not have a μCSU (e. g.: $E =$ A $\cup \{x + x \approx x\}$).

Unification modulo ACU

Let us first consider elementary ACU-unification:
the terms to be unified contain only variables and the function symbols from $\Sigma = (\{+/2,\, 0/0\}, \emptyset)$.

Since parentheses and the order of summands don't matter, every term over $X_n = \{x_1, \ldots, x_n\}$ can be written as a sum $\sum_{i=1}^{n} a_i\, x_i$.

The ACU-equivalence class of a term $t = \sum_{i=1}^{n} a_i\, x_i \in T_\Sigma(X_n)$ is uniquely determined by the vector $\vec{v}_n(t) = (a_1, \ldots, a_n)$.

Analogously, a substitution $\sigma = \{\, x_i \to \sum_{j=1}^{m} b_{ij}\, x_j \mid 1 \le i \le n \,\}$ is uniquely determined by the matrix

$$M_{n,m}(\sigma) = \begin{pmatrix} b_{11} & \cdots & b_{1m} \\ \vdots & & \vdots \\ b_{n1} & \cdots & b_{nm} \end{pmatrix}$$

Let $t = \sum_{i=1}^{n} a_i\, x_i$ and $\sigma = \{\, x_i \to \sum_{j=1}^{m} b_{ij}\, x_j \mid 1 \le i \le n \,\}$.

$$\begin{aligned} \text{Then } t\sigma &= \textstyle\sum_{i=1}^{n} a_i\, (\sum_{j=1}^{m} b_{ij}\, x_j) \\ &= \textstyle\sum_{i=1}^{n} \sum_{j=1}^{m} a_i\, b_{ij}\, x_j \\ &= \textstyle\sum_{j=1}^{m} \sum_{i=1}^{n} a_i\, b_{ij}\, x_j \\ &= \textstyle\sum_{j=1}^{m} (\sum_{i=1}^{n} a_i\, b_{ij})\, x_j. \end{aligned}$$

Consequence:

$$\vec{v}_m(t\sigma) = \vec{v}_n(t) \cdot M_{n,m}(\sigma).$$

Let $S = \{s_1 \approx t_1, \ldots, s_k \approx t_k\}$ be a set of equality problems over $T_\Sigma(X_n)$.

Then the following properties are equivalent:

(a) σ is an ACU-unifier of S from $X_n \to T_\Sigma(X_m)$.

(b) $\vec{v}_m(s_i\sigma) = \vec{v}_m(t_i\sigma)$ for all $i \in \{1, \ldots, k\}$.

(c) $\vec{v}_n(s_i) \cdot M_{n,m}(\sigma) = \vec{v}_n(t_i) \cdot M_{n,m}(\sigma)$ for all $i \in \{1, \ldots, k\}$.

(d) $(\vec{v}_n(s_i) - \vec{v}_n(t_i)) \cdot M_{n,m}(\sigma) = \vec{0}_m$ for all $i \in \{1, \ldots, k\}$.

(e) $M_{k,n}(S) \cdot M_{n,m}(\sigma) = \vec{0}_{k,m}$.
where $M_{k,n}(S)$ is the $k \times n$ matrix whose rows are the vectors $\vec{v}_n(s_i) - \vec{v}_n(t_i)$.

(f) The columns of $M_{n,m}(\sigma)$ are non-negative integer solutions of the system of homogeneous linear diophantine equations $DE(S)$:
$$M_{k,n}(S) \cdot \begin{pmatrix} y_1 \\ \vdots \\ y_n \end{pmatrix} = \begin{pmatrix} 0 \\ \vdots \\ 0 \end{pmatrix}$$

Computing unifiers:

Obviously: if $\vec{y}_1, \ldots, \vec{y}_r$ are solutions of $DE(S)$ and $a_1, \ldots, a_r$ are natural numbers, then $a_1\vec{y}_1 + \cdots + a_r\vec{y}_r$ is also a solution. (In particular, the zero vector is a solution!)

In fact, one can compute a finite set of solutions $\vec{y}_1, \ldots, \vec{y}_r$, such that every solution of $DE(S)$ can be represented as such a linear combination.

Moreover, if we combine these column vectors $\vec{y}_1, \ldots, \vec{y}_r$ to an $n \times r$ matrix, this matrix represents a most general unifier of S. (Proof: see Baader/Nipkow.)

From ACU to AC

A complete set of AC-unifiers for elementary AC-unification problems can be computed from a most general ACU-unifier by some postprocessing.

Elementary AC-unification is finitary and the elementary unifiability problem is solvable in polynomial time.

But that does not mean that minimal complete sets of AC-unifiers can be computed efficiently.

E. Domenjoud has computed the exact size of AC-μCSUs for unification problems of the following kind:

$$m\,x_1 + \cdots + m\,x_p \approx n\,y_1 + \cdots + n\,y_q$$

where $gcd(m, n) = 1$.

The number of unifiers is

$$(-1)^{p+q} \sum_{i=0}^{p} \sum_{j=0}^{q} (-1)^{i+j} \binom{p}{i} \binom{q}{j} 2^{\binom{m+j-1}{m}\binom{n+i-1}{n}}$$

For $p = m = 1$ and $q = n = 4$, that is, for the equation

$$4\,x \approx y_1 + y_2 + y_3 + y_4$$

this is

$$34\,359\,607\,481.$$

Consequence:

If possible, avoid the enumeration of AC-μCSUs
(which may have doubly exponential size).

Rather: only check AC-unifiability.

Or: use ACU instead.

Unification with Constants

So far:

Elementary unification:
terms over variables and $\{+, 0\}$ or $\{+\}$.

Step 2:

Additional free constants.

Step 3:

Additional arbitrary free function symbols.
$\rightsquigarrow$ Unification in the union of disjoint equational theories.

Unification with constants:

We can treat constants a_i like variables x_i that *must* be mapped to themselves.

Consequence: The algorithm is similar to the one we have seen before, but we have to deal with homogeneous and inhomogeneous linear diophantine equations.

Some complexity bounds change, however:

Unification type:

elementary ACU-unification: unitary;
ACU-unification with constants: finitary.

Checking unifiability:

elementary ACU-unification: trivial;
ACU-unification with constants: NP-complete.

Combining Unification Procedures

The Baader–Schulz combination procedure allows to combine unification procedures for disjoint theories (e. g., ACU and the free theory).

Basic idea (as usual): Use abstraction to convert the combined unification problem into a union of two pure unification problems; solve them individually; combine the results.

Problem 1:

The individual unification procedures might map the same variable to different terms, e. g., $\{x \mapsto y + z\}$ and $\{x \mapsto f(w)\}$.

Solution: Guess for each variable non-deterministically which procedure treats it like a constant.

Problem 2:

Combining the results might produce cycles, e. g., $\{x \mapsto y + z\}$ and $\{y \mapsto f(x)\}$.

Solution: Guess an ordering of the variables non-deterministically; each individual unifier that is computed must respect the ordering.

Note: This is a non-trivial extension that may be impossible for some unification procedures (but it is possible for *regular* equational theories, i. e., theories where for each equation $u \approx v$ the terms u and v contain the same variables).

Literature

Franz Baader, Tobias Nipkow: Term Rewriting and All That. Cambridge University Press, 1998.

Franz Baader, Klaus Schulz: Unification in the union of disjoint equational theories: Combining decision procedures. Automated Deduction, CADE-11, LNCS 607, pp. 50–65, Springer, 1992.

Eric Domenjoud: A technical note on AC-unification. The number of minimal unifiers of the equation $\alpha x_1 + \cdots + \alpha x_p \doteq_{\mathrm{AC}} \beta y_1 + \cdots + \beta y_q$. Journal of Automated Reasoning, 8(1):39–44, 1992.

François Fages: Associative-commutative unification. Automated Deduction, CADE-7, LNCS 170, pp. 194–208, Springer, 1984.

Mike Livesey, Jörg H. Siekmann: Unification of AC-terms (bags) and ACI-terms (sets). Internal report, University of Essex, 1975.

Gordon Plotkin: Building-in equational theories. Machine Intelligence, 7:73–90, American Elsevier, 1972.

Manfred Schmidt-Schauß: Unification under associativity and idempotence is of type nullary. Journal of Automated Reasoning, 2:277–282, 1986.

3.10 Integrating Theories II: Calculi

We can replace syntactic unification by E-unification in the superposition calculus.

Moreover, it is usually necessary to choose a term ordering in such a way that all terms in an E-congruence class behave in the same way in comparisons (E-compatible ordering).

However, this is usually not sufficient.

AC and ACU

Example: Let $E = \mathrm{AC}$. The clauses

$$\begin{aligned} a + b &\approx d \\ b + c &\approx e \\ c + d &\not\approx a + e \end{aligned}$$

are contradictory w. r. t. AC, but if $a \succ b \succ c \succ d \succ e$, then the maximal sides of these clauses are not AC-unifiable.

We have to compute inferences if some part of a maximal sum overlaps with a part of another maximal sum (the constant b in the example above).

Technically, we can do this in such a way that we first replace positive literals $s \approx t$ by $s + x \approx t + x$, and then unify maximal sides w. r. t. AC or ACU (Peterson and Stickel 1981, Wertz 1992, Bachmair and Ganzinger 1994).

However, it turns out that even if we integrate AC or ACU in such a way into superposition, the resulting calculus is not particularly efficient – not even for ground formulas.

This is not surprising: The uniform word problem for AC or ACU is EXPSPACE-complete (Cardoza, Lipton, and Meyer 1976, Mayr and Meyer 1982).

Abelian Groups

Working in Abelian groups is easier:

If we integrate also the inverse axiom, it is sufficient to compute inferences if the maximal part of a maximal sum overlaps with the maximal part of another maximal sum (like in Gaussian elimination).

Intuitively, in Abelian groups we can always isolate the maximal part of a sum on one side of an equation.

What does that mean for the non-ground case?

Example:

$$g(y) + x \not\approx 2z \quad \vee \quad f(x) + z \approx 2y$$

Shielded variables (x, y):

occur below a free function symbol,
$\rightsquigarrow$ cannot be mapped to a maximal term,
$\rightsquigarrow$ are not involved in inferences.

Unshielded variables (z):

can be instantiated with $m \cdot u + s$, where u is maximal,
$\rightsquigarrow$ must be considered in inferences,
$\rightsquigarrow$ variable overlaps (similar to ACU).

Variable overlaps are ugly:

If we want to derive a contradiction from

$$\begin{aligned} 2a &\approx c \\ 2b &\approx d \\ 2x &\not\approx c + d \end{aligned}$$

and $a \succ b \succ c \succ d$, we have to map x to a sum of two variables $x' + x''$, unify x' with a and x'' with b.

Divisible Torsion-free Abelian Groups

Working in divisible torsion-free Abelian groups is still easier:

DTAGs permit variable elimination.

Every clause can be converted into a DTAG-equivalent clause without *unshielded* variables.

Since only overlaps of maximal parts of maximal sums have to be computed, variable overlaps become unnecessary.

Moreover, if abstraction is performed eagerly, terms to be unified do not contain $+$, so ACU-unification can be replaced by standard unification.

Other Theories

A similar case: Chaining calculus for orderings.

$$\frac{D' \vee t' < t \qquad C' \vee s < s'}{(D' \vee C' \vee t' < s')\sigma}$$

where σ is a most general unifier of t and s.

Avoids explicit inferences with transitivity.
Only maximal sides of ordering literals have to be overlapped.
But unshielded variables can be maximal.

In dense linear orderings without endpoints, all unshielded variables can be eliminated.

DTAG-superposition and chaining can be combined to get a calculus for ordered divisible Abelian groups. Again, all unshielded variables can be eliminated.

Conclusion

Integrating theory axioms into superposition can become easier by integrating more axioms:

Easier unification problem (AC $\rightarrow$ ACU).

More restrictive inference rules (ACU $\rightarrow$ AG).

Fewer (or no) variable overlaps (AG $\rightarrow$ DTAG).

Main drawback of all theory integration methods:

For each theory, we have to start from scratch, both for the completeness proof and the implementation.

Literature

Leo Bachmair, Harald Ganzinger: Rewrite techniques for transitive relations. IEEE Symposium on Logic in Computer Science, LICS-9, pp. 384–393, 1994.

Leo Bachmair, Harald Ganzinger: Ordered chaining for total orderings. Automated Deduction, CADE-12, LNAI 814, pp. 435–450, Springer, 1994.

Leo Bachmair, Harald Ganzinger: Associative-commutative superposition. Conditional and Typed Rewriting Systems, CTRS-94, LNCS 968, pp. 1–14, Springer, 1994.

E. Cardoza, R. Lipton, A. R. Meyer: Exponential space complete problems for Petri nets and commutative semigroups: preliminary report. Eighth Annual ACM Symposium on Theory of Computing, STOC, pp. 50–54, 1976.

Guillem Godoy, Robert Nieuwenhuis: Paramodulation with built-in Abelian groups. IEEE Symposium on Logic in Computer Science, LICS-15, pp. 413–424, 2000.

Ernst W. Mayr, Albert R. Meyer: The complexity of the word problems for commutative semigroups and polynomial ideals. Advances in Mathematics, 46(3):305–329, 1982.

Gerald E. Peterson, Mark E. Stickel: Complete sets of reductions for some equational theories. Journal of the ACM, 28(2):233–264, 1981.

Uwe Waldmann: Cancellative abelian monoids and related structures in refutational theorem proving (Part I & II). Journal of Symbolic Computation, 33(6):777–829/831–861, 2002.

Uwe Waldmann: Superposition and chaining for totally ordered divisible abelian groups. Technical report MPI-I-2001-2-001, Max-Planck-Institut für Informatik, Saarbrücken, 2001.

Ulrich Wertz: First-order theorem proving modulo equations. Technical report MPI-I-92-216, Max-Planck-Institut für Informatik, Saarbrücken, 1992.

4 Higher-Order Logic

Higher-Order logic

- extends first-order logic with quantification over functions and predicates
- is *very* expressive (natural numbers, uncountable sets...)
- is the preferred language of most mathematicians

Higher-order logic is also called *simple type theory*.

4.1 History

Higher-order quantification

Unrestricted quantification is first considered by Frege (1879).

It contains several paradoxical statements, such as Russell's paradox, which motivated the creation of *ramified type theory* by Russell (1908).

A later simplification of this theory by Church (1940) was denoted a *simple type theory*, or HOL.

4.2 Syntax

Syntax choices:

explicit function symbols

explicit predicate abstraction

quantifiers and connectives as constants of the language

with extensionality

Types

Types are defined recursively:

o is the type of Booleans, of order 0.

ι is the type of individuals, of order 1.

if τ_1 and τ_2 are types then $\tau_1 \to \tau_2$ is a type, of order $\max(\text{order}(\tau_1) + 1, \text{order}(\tau_2))$

We also use the notation $\tau_1, \ldots, \tau_n \to \tau$ to denote $\tau_1 \to (\cdots \to (\tau_n \to \tau) \ldots)$.

Terms

Given a non-empty set of constants and a collection of non-empty sets of variables for each type,

constants are terms

variables are terms

if t_1 and t_2 are terms then $(t_1 t_2)$ is a term

if x is a variable and t is a term then $\lambda x.\, t$ is a term

Types of Terms

Given a non-empty set S of individuals and a collection of non-empty sets of variables for each type, the term t is of type

o if $t \in \{\top, \bot\}$

ι if $t \in S$

τ if $t = x_{(\tau)}$ is a variable of type τ

$\tau_1 \to \tau_2$ if $t = \lambda x_{(\tau_1)}.\, t_{1(\tau_2)}$

τ_2 if $t = (t_{1(\tau_1 \to \tau_2)}\ t_{2(\tau_1)})$

A term is well-typed if a type can be associated to it according to the previous definition. We only consider well-typed terms in what follows.

4.3 Semantics

A well-founded formula is a term of type o.

How to evaluate the truth of such a formula?

Classical Model

Let D be a non-empty set, for each type τ we define the following collection, denoted as the *frame* of the type

the frame of $\tau = o$ is $[\![o, D]\!] = \{\top, \bot\}$

the frame of $\tau = \iota$ is $[\![\iota, D]\!] = D$

the frame of $\tau = \tau_1 \to \tau_2$ is $[\![\tau_1 \to \tau_2, D]\!]$, the collection of all functions mapping $[\![\tau_1, D]\!]$ into $[\![\tau_2, D]\!]$

A higher-order *classical model* is a structure $\mathcal{M} = \langle D, \mathcal{I} \rangle$ where D is a non-empty set called the *domain* of the model and $\mathcal{I}$ is the *interpretation* of the model, a mapping such that

if $a_{(\tau)}$ is a constant then $\mathcal{I}(a) \in [\![\tau, D]\!]$,

$\mathcal{I}(=_{(\tau \to \tau \to o)})$ is the equality relation on $[\![\tau, D]\!]$.

By adding a *valuation* function α such that for any variable $x_{(\tau)}$, $\alpha(x) \in [\![\tau, D]\!]$, it becomes possible to evaluate the truth-value of higher-order formulas as in first-order logic.

The *evaluation* $\mathcal{V}_{\mathcal{M},\alpha}(t)$ of a term t given a model $\mathcal{M} = \langle D, \mathcal{I} \rangle$ and a valuation α is recursively defined as

$\mathcal{I}(a)$ if t is a constant a

$\alpha(x)$ if t is a variable x

the function from $[\![\tau_1, D]\!]$ to $[\![\tau_2, D]\!]$ such that for all $a \in [\![\tau_1, D]\!]$, $(\mathcal{V}_{\mathcal{M},\alpha}(\lambda x.\, t))(a) = \mathcal{V}_{\mathcal{M},\alpha}(t[a/x])$ if $t = \lambda x_{(\tau_1)}.\, t_{(\tau_2)}$

$(\mathcal{V}_{\mathcal{M},\alpha}(t_1))(\mathcal{V}_{\mathcal{M},\alpha}(t_2))$ if $t = (t_{1(\tau_1 \to \tau_2)}\ t_{2(\tau_1)})$

Truth evaluation:

Given a model $\mathcal{M} = \langle D, \mathcal{I} \rangle$ and a valuation α, a well-founded formula ϕ is true in $\mathcal{M}$ with respect to α, denoted as $\mathcal{M}, \alpha \models \phi$ iff $\mathcal{V}_{\mathcal{M},\alpha}(\phi) = \{\top\}$

ϕ is satisfiable in $\mathcal{M}$ iff there exist a valuation α such that $\mathcal{M}, \alpha \models \phi$

ϕ is valid in $\mathcal{M}$, denoted $\mathcal{M} \models \phi$ iff for all valuations α, $\mathcal{M}, \alpha \models \phi$

ϕ is valid, denoted $\models \phi$ iff for all models $\mathcal{M}$, $\mathcal{M} \models \phi$

These notions extend straightforwardly to sets of formulas.

Problems with the classical semantic

- Loss of compactness: in FOL, every unsatisfiable set of formulas has a finite unsatisfiable subset. This is no longer the case in HOL with classical semantics (cHOL).
- Loss of strong completeness: no proof procedure able to derive all consequences of a set of formulas can exist in cHOL.
- Loss of weak completeness: no proof procedure able to derive all valid sets of formulas can exist in cHOL.
- And even worse: the status of validity of some formulas is unclear.

Henkin Semantics

To solve the previously mentioned issues, it is possible to generalize the notion of a model by relaxing the notion of a frame into that of a Henkin frame. Given a non-empty set D,

$[\![o, D]\!] = \{\top, \bot\}$

$[\![\iota, D]\!] = D$

$[\![\tau_1 \to \tau_2, D]\!]$ is the collection of ~~all~~ *some* functions mapping $[\![\tau_1, D]\!]$ into $[\![\tau_2, D]\!]$ *with some additional closure conditions*.

Henkin vs Classical Semantics

- Any formula true in all Henkin models is true in all classical models.
- There are formulas true in all classical models that are not true in all Henkin models.
- There are (weak) complete proof procedures for HOL with Henkin semantics.

4.4 Higher-Order Term Unification

In FOL, there exist a unique m.g.u. for two terms.

This is no longer true in HOL.

For example, consider $t_1 = f\ x$ and $t_2 = a$ where f, x are variables and a is a constant. The unifiers of t_1 and t_2 are $\{f \mapsto \lambda y.\, a\}$ and $\{f \mapsto \lambda y.\, y, x \mapsto a\}$.

Some equations even have an infinite number of m.g.u's.

Even worse, the higher-order unification problem is *undecidable*.

Huet's Unification Algorithm

Given:

E, a unification problem, i.e. a finite set of equations.

Goal:

find a substitution σ such that $E\sigma$ contains only syntactically equal equations.

Idea:

Test if the head symbols of the two sides of equations can be unified or not to restrict the search space.

Rigid and Flexible Terms

A term is *rigid* if its head symbol is a constant or a bound variable. Otherwise its head symbol is a free variable and the term is *flexible*.

Rigid-Rigid Equations

Two rules can be applied depending on the head symbols in the rigid-rigid equation.

Fail:

$$\frac{E \cup \{\lambda x_1 \dots x_n.\, f\ u_1 \dots u_p \approx \lambda x_1 \dots x_n.\, g\ v_1 \dots v_q\}}{\bot}$$

Simplify:

$$\frac{E \cup \{\lambda x_1 \dots x_n.\, f\ u_1 \dots u_p \approx \lambda x_1 \dots x_n.\, f\ v_1 \dots v_p\}}{E \cup \{\lambda x_1 \dots x_n.\, u_1 \approx \lambda x_1 \dots x_n.\, v_1, \dots, \lambda x_1 \dots x_n.\, u_p \approx \lambda x_1 \dots x_n.\, v_p\}}$$

Flexible-Rigid Equations

There is only one rule to handle such terms, but it can generate many results.

Generate:

$$\frac{E}{E\sigma}$$

where $\sigma = \{X \mapsto \lambda y_1, \dots, y_p.\, h\ (H_1\ y_1 \dots y_p) \dots (H_r\ y_1 \dots y_p)\}$ and $(\lambda x_1 \dots x_n.\, X\ u_1 \dots u_p \approx \lambda x_1 \dots x_n.\, f\ \ v_1 \dots v_q) \in E$ such that $h \in \{f, y_1, \dots, y_p\}$ if f is a constant and $h \in \{y_1, \dots, y_p\}$ otherwise.

Flexible-Flexible Equations

The following result, also by Huet, handles flexible-flexible equations.

Proposition 4.1 *A unification problem E containing only flexible-flexible equations has always a solution.*

Proof. Consider any flexible-flexible equation

$$e = (\lambda x_1 \dots x_n.\, X_{(\tau)}\ u_1 \dots u_p \approx \lambda x_1 \dots x_n.\, Y\ v_1 \dots v_q).$$

Since there are no empty types, there exists a constant $a_{(\tau)}$ for each type τ. Let θ_τ be the substitution that maps all variables of type τ to this constant a. Then $e\theta = (\lambda x_1 \dots x_n.\, a \approx \lambda x_1, \dots, x_n.\, a)$ thus θ is a unifier of e.

We say that a unification problem with only flexible-flexible equations is a *solved* unification problem.

The Whole Procedure

A reasonable strategy consists in applying Fail and Simplify eagerly, and Generate only when there is no rigid-rigid equation left.

Generate is non-deterministic, making this procedure branching.

Theorem 4.2 *The procedure made of the rules Fail, Simplify and Generate is sound and complete.*

Soundness

Proposition 4.3 *If a unification problem E can be transformed into a solved problem E' by applying Fail, Simplify and Generate then E has a solution.*

The proof is by induction on the size of the derivation from E to E'.

- If E is a solved problem then Prop. 4.1 applies.
- If the first rule applied on E is Fail then $E' = \bot$ is not a solved problem, a contradiction.

- If the first rule applied on E is Simplify, resulting in E_1, then by the induction hypothesis, E_1 has a solution σ and

$$E = E_0 \cup \{\lambda x_1 \dots x_n.\ f\ u_1 \dots u_p \approx \lambda x_1 \dots x_n.\ f\ v_1 \dots v_p\}$$
$$\text{and } E_1 = E_0 \cup \{\lambda x_1 \dots x_n.\ u_1 \approx \lambda x_1 \dots x_n.\ v_1, \dots, \lambda x_1 \dots x_n.\ u_p \approx \lambda x_1 \dots x_n.\ v_p\}$$

  such that $\sigma(u_i) = \sigma(v_i)$ for all $i \in \{1, \dots, p\}$ and $E_0\sigma$ contains only trivial equations. Thus $E\sigma = E_0\sigma \cup \{(\lambda x_1 \dots x_n.\ f\ u_1 \dots u_p \approx \lambda x_1 \dots x_n.\ f\ v_1 \dots v_p)\sigma\}$. Since $(f\ u_1 \dots u_p)\sigma = f\ u_1\sigma \dots u_p\sigma = f\ v_1\sigma \dots v_p\sigma = (f\ v_1 \dots v_p)\sigma$, the substitution σ is a solution to the unification problem E.

- If the first rule applied on E is Generate and E_1 is the resulting unification problem then there exists a solution σ to E_1 by the induction hypothesis, and $E_1 = E\theta$ where $\theta = \{X \mapsto \lambda y_1 \dots y_p.\ h\ (H_1\ y_1 \dots y_p) \dots (H_r\ y_1 \dots y_p)\}$ and E contains an equation $\lambda x_1 \dots x_n.\ X\ u_1 \dots u_p \approx \lambda x_1 \dots x_n.\ f\ v_1 \dots v_q$ Let $\sigma' = \sigma \circ \theta$. Since $E\sigma' = E(\sigma \circ \theta) = (E\theta)\sigma = E_1\sigma$, the substitution σ' is a solution of E.

Completeness

Proposition 4.4 *If a unification problem E has a solution σ then we can derive a solved problem E' from E using the rules Fail, Simplify and Generate.*

Proof. This proof is done by induction of the complexity of σ. First we must define this measure. Let us consider an arbitrary term $t = \lambda x_1 \dots x_n.\ h\ u_1 \dots u_p$. The number of applications used in t, denoted $\pi(t)$ is computed in the following way: $\pi(t) = p + \sum_{i=1}^{p} \pi(u_i)$. This function is used to compute the complexity of a substitution. Let σ be a substitution that maps the variables x_1,...,x_k to the terms t_1,...,t_k such that x_i and t_i are distinct, and that maps all other variables to themselves. The complexity $\mathcal{C}(\sigma)$ of σ is $\mathcal{C}(\sigma) = k + \sum_{i=1}^{k} \pi(t_i)$. We can now use this measure to perform an induction in the following way.

- If E contains rigid-rigid equations then it is possible to get rid of them by repeatedly applying the rule Simplify. This process terminates, which can be proved by induction on the size of terms in E. Fail can never be applied during this process, otherwise it would contradict the fact that there exist a solution of E. If the resulting set of equations E_s is solved, we have a derivation to a solved problem.

- Otherwise E_s contains no rigid-rigid equations, but at least one flexible-rigid one, i.e., $\lambda x_1 \dots x_n.\ X\ u_1 \dots u_p \approx \lambda x_1 \dots x_n.\ f\ v_1 \dots v_q \in E_s$. Since σ is a solution of E_s, $(\lambda x_1 \dots x_n.\ X\ u_1 \dots u_p)\sigma = (\lambda x_1 \dots x_n.\ f\ v_1 \dots v_q)\sigma$, thus $\sigma = \sigma_0 \cup \theta$ where $\theta = \{X \mapsto \lambda y_1 \dots y_p.\ h\ w_1 \dots w_r\}$, and $h \in \{y_1, \dots, y_p, f\}$ if f is a constant or $h \in \{y_1, \dots, y_p\}$ otherwise. Then, we can use Generate with the function h occurring in θ on E_s to generate $E'_s = E_s\theta'$ where $\theta' = X \mapsto \lambda y_1, \dots, y_p.\ h\ (H_1\ y_1 \dots y_p) \dots$

$(H_r\ y_1 \ldots y_p)$. Let $\gamma = \{H_1 \mapsto \lambda y_1 \ldots y_p.\, w_1, \ldots, H_r \mapsto \lambda y_1 \ldots y_p.\, w_r\}$. The complexity of σ is $\mathcal{C}(\sigma) = \mathcal{C}(\sigma_0 \cup \theta) = \mathcal{C}(\sigma_0) + 1 + r + \sum_{i=1}^{r} \pi(w_i)$, and that of σ' is $\mathcal{C}(\sigma') = \mathcal{C}(\sigma_0 \cup \gamma) = \mathcal{C}(\sigma_0) + r + \sum_{i=1}^{r} \pi(w_i)$. Thus $\mathcal{C}(\sigma') < \mathcal{C}(\sigma)$. By the induction hypothesis, there exists a derivation from E'_s to a solved unification problem E' and E'_s was obtained by derivation from E hence we can conclude.

Termination?

Higher-order unification is only semi-decidable.

When solutions exist, Huet's algorithm will find one and terminate, but when there is no solution, it may loop forever.

4.5 Resolution in Higher-Order Logic

In first-order logic, resolution for general clauses has two rules:

Resolution:
$$\frac{D \vee B \qquad C \vee \neg A}{(D \vee C)\sigma}$$
where $\sigma = \mathrm{mgu}(A, B)$.

Factoring:
$$\frac{C \vee A \vee B}{(C \vee A)\sigma}$$
where $\sigma = \mathrm{mgu}(A, B)$.

In higher-order logic, a first problem is that m.g.u's need not exist and unification is undecidable.

Example 4.5 *Given $D \vee B$ and $C \vee \neg A$ where A and B are unifiable but without m.g.u., there may exist infinitely many σ_1, σ_2,... unifiers of A and B generating distinc resolvents $(D \vee C)\sigma_1$, $(D \vee C)\sigma_2$,... and in general there is no way to know which one is needed to prove the given theorem.*

Huet proposes to delay the computation of unifiers (when no m.g.u. exists) by using constraints storing the corresponding unification problems.

Once a contradiction has been derived, the corresponding unification problem can then be solved using Huet's algorithm.

Resolution: $$\frac{D \vee B[\![X]\!] \qquad C \vee \neg A[\![Y]\!]}{D \vee C[\![X \wedge Y \wedge A \approx B]\!]}$$

Factoring: $$\frac{C \vee A \vee B[\![X]\!]}{C \vee A[\![X \wedge A \approx B]\!]}$$

Another problem in HOL is that it is not always possible to guess the necessary substitution based on the available terms.

Example 4.6 *Consider the formula $\neg X_{(o)}$ where X is a Boolean variable. The set $\{\neg X\}$ is saturated by resolution, but still the formula $\neg X$ is unsatisfiable. However, we can guess the substitution $\sigma = \{X \mapsto \neg Y\}$. Then $(\neg X)\sigma = \neg(\neg Y) = Y$ and resolution can now derive the empty clause from $\neg X$ and Y.*

To overcome this issue, Huet introduces additional *splitting rules*.

$$\frac{C \vee A[\![X]\!]}{C \vee \neg x_{(o)}[\![X \wedge A \approx \neg x]\!]}$$

$$\frac{C \vee A[\![X]\!]}{C \vee x_{(o)} \vee y_{(o)}[\![X \wedge A \approx (x \vee y)]\!]}$$

$$\frac{C \vee A[\![X]\!]}{C \vee P_{(\tau \to o)} x_{(\tau)}[\![X \wedge A \approx \Pi_{((\tau \to o) \to o)} P]\!]}$$

$\Pi_{((\tau \to o) \to o)}$ is the function that associates $\top$ to any set of type $\tau \to o$ that contains all elements of type τ.

$$\frac{C \vee \neg A[\![X]\!]}{C \vee x_{(o)}[\![X \wedge A \approx \neg x]\!]}$$

$$\frac{C \vee \neg A[\![X]\!]}{C \vee x_{(o)}[\![X \wedge A \approx (x \vee y_{(o)})]\!] \text{ and } C \vee y[\![X \wedge A \approx (x \vee y)]\!]}$$

$$\frac{C \vee \neg A[\![X]\!]}{C \vee \neg P_{(\tau \to o)}(\mathrm{sk}_{((\tau \to o) \to \tau)} P)[\![X \wedge A \approx \Pi_{((\tau \to o) \to o)} P]\!]}$$

sk is the skolem constant such that $\neg \Pi_{((\tau \to o) \to o)} P = \neg P_{(\tau \to o)}(\mathrm{sk}_{((\tau \to o) \to \tau)} P)$.

Huet proved that resolution with these splitting rules is sound and complete (but not terminating).

In practice, several improvements are possible.

As soon as a constraint becomes unsatisfiable, delete the corresponding clause.

If a constraint has a small enough set of solutions, generate all applied clauses to replace the constrained original one.

4.6 Superposition in Higher-Order Logic

In HOL, existing automated solvers rely on:

- Tableau (Satallax)
- Resolution (Leo III)
- Applicative encoding to first-order logic (Sledgehammer)
- ...

Currently there exists no efficient version of Superposition for full higher-order logic.

There are many theoretical problems to lifting Superposition to HOL (unification,...)

Superposition in λ-free Higher-Order Logic

Things get easier in λ-free higher-order logic (i.e. no λ-terms and no predicate variables).

This fragment can be encoded in FOL using a binary function *app* (application).

If the ordering has all standard properties of reduction orderings plus compatibility with arguments, the extension of Superposition to this fragment is straightforward.

There is only one known ordering with these properties: KBO.

There are applications where standard KBO is not optimal.

There are other orderings that one would like to use (LPO, KBO with multipliers) but one loses at least one of the desired properties (e.g. compatibility with arguments).

There are workarounds that allow to recover from the loss of compatibility with arguments, e.g. by:

redefining redundancy so that $g\ x \approx f\ x$ is not redundant to $g \approx f$, and

adding a rule that adds context to an equation (generate $g\ x \approx f\ x$ from $g \approx f$), and

relaxing the variable constraint in the superposition rules (no superposition at or under a variable, except if...), and

adding a layer to the completeness proof.

Our current goal is to extend this calculus to predicate-free HOL (including λ-terms) and then to full HOL.

Literature

Peter B. Andrews: An introduction to mathematical logic and type theory - to truth through proof. Computer science and applied mathematics, Academic Press, ISBN 978-0-12-058535-9, pp. I-XV, 1-304, 1986.

Peter B. Andrews: Classical Type Theory. Handbook of Automated Reasoning: 965-1007, 2001.

Heiko Becker, Jasmin Christian Blanchette, Uwe Waldmann, Daniel Wand: A Transfinite Knuth-Bendix Order for Lambda-Free Higher-Order Terms. CADE: 432-453, 2017.

Alexander Bentkamp, Jasmin Christian Blanchette, Simon Cruanes, and Uwe Waldmann: Superposition for lambda-free higher-order logic. IJCAR: (to appear), 2018.

Christoph Benzmüller, Dale Miller: Automation of Higher-Order Logic. Computational Logic: 215-254, 2014.

Jasmin Christian Blanchette, Uwe Waldmann, Daniel Wand: A Lambda-Free Higher-Order Recursive Path Order. FoSSaCS: 461-479, 2017.

Gilles Dowek: Higher-Order Unification and Matching. Handbook of Automated Reasoning: 1009-1062, 2001.

Melvin Fitting: Types Tableaus and Gödel's God. Studia Logica 81(3): 425-427, 2005.

Gérard P. Huet: A Mechanization of Type Theory. IJCAI: 139-146, 1973.