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 call a ground instance $C\theta$ *variable irreducible* w. r. t. a ground TRS R, if for every variable x occurring in a literal L of C, $x\theta$ is irreducible by all rules in R that are smaller than $L\theta$.

The construction yields a TRS R_∞ that is a model of all R_∞-*variable irreducible* ground instances of clauses in N_*.

R_∞ is also a model of all R_∞-*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 rules in R_∞ and some smaller or equal 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 R_∞-*variable irreducible* ground instances during the model construction, we may use only such instances for redundancy:

A clause is redundant, if all its R_∞-variable irreducible ground instances follow from smaller R_∞-variable irreducible ground instances and smaller rules in R_∞.

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 32).

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.