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 $h(g(g(y))) \approx h(y)$.

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 $N \setminus Red(N)$ is contained in $N \cup Red(N)$.

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 $RedInf(N)$.

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 $RedInf(N) \subseteq RedInf(N')$.*

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

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.

Static 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 $E_{C\theta} = \emptyset$.

For dynamic refutational completeness, there is a problem, however:

In the static refutational completeness proof, the selection function *gsel* for ground clauses depends on the selection function *sel* for general clauses and on the saturated set N_∞ itself.

N_∞ is the limit of a run, therefore it depends on *RedInf*.

RedInf depends on what counts as a ground instance of an inference and what does not, and thus on the set of ground inferences.

The set of ground inferences depends of *gsel*, though!

How can we break this cycle?

Solution: In the definition of *RedInf*, we have to quantify over all possible ground selection functions *gsel*:

> An inference ι is redundant, if for every ground selection function *gsel* corresponding to *sel*, all *gsel*-ground instances of ι are redundant.

Result:

> Worst-case analysis: When we check whether some inference involving a clause $C \in N$ is redundant, we must assume that every ground instance D of C inherits the selection of C (even though D might also be a ground instance of another clause $C' \in N$ with a different selection).

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

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

Splitting: $$\frac{N \cup \{C_1 \lor C_2\}}{N \cup \{C_1\} \qquad | \qquad N \cup \{C_2\}}$$
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: $$\frac{N \cup \{C_1 \lor C_2\}}{N \cup \{C_1\} \qquad | \qquad N \cup \{C_2\} \cup \{\neg C_1\}}$$
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
$$\begin{aligned} K = (t \succ t' \ \wedge\ & s[u] \succ s' \\ \wedge\ & (t \approx t') \succ_C D' \\ \wedge\ & (s[u] \approx s') \succ_C C' \\ \wedge\ & (s[u] \approx s') \succ_L (t \approx t')) \end{aligned}$$

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:
$$\frac{D' \lor t \approx t' \, [\![K_2]\!] \qquad C' \lor s[u] \approx s' \, [\![K_1]\!]}{D' \lor C' \lor s[t'] \approx s' \, [\![K_2 \land K_1 \land K]\!]}$$
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

$$\Xi = \{\mathit{int}\}$$

$$\begin{aligned}\Omega = \{\, & 0, 1, -1, 2, -2, \ldots : \to \mathit{int}, \\ & - : \mathit{int} \to \mathit{int}, \\ & + : \mathit{int} \times \mathit{int} \to \mathit{int}\,\}\end{aligned}$$

$$\begin{aligned}\Pi = \{\, & \geq : \mathit{int} \times \mathit{int}, \\ & > : \mathit{int} \times \mathit{int}\,\}\end{aligned}$$

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

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

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

$$\begin{aligned}\Omega' = \Omega \cup \{\, & \mathit{cons} : \mathit{int} \times \mathit{list} \to \mathit{list}, \\ & \mathit{length} : \mathit{list} \to \mathit{int}, \\ & \mathit{empty} : \to \mathit{list}, \\ & a : \to \mathit{list}\,\}\end{aligned}$$

$$\Pi' = \Pi$$

$$\begin{aligned}N' = \{\, & \mathit{length}(a) \geq 1, \\ & \mathit{length}(\mathit{cons}(x, y)) \approx \mathit{length}(y) + 1\,\}\end{aligned}$$

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.