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 $N_* \setminus Red(N_*)$ is contained in some $N_i \cup Red(N_i)$.

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 $N_i \cup Red(N_i)$, and therefore contained in $N_* \cup Red(N_*)$. Hence N_* is saturated up to redundancy. $\Box$

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 $N_0 \models N_*$. $\Box$

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 $\bigl(\forall \vec{x}\ C_1(\vec{x})\bigr) \ \vee \ \bigl(\forall \vec{y}\ C_2(\vec{y})\bigr)$.

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

Splitting: $$\frac{N \cup \{C_1 \vee C_2\}}{N \cup \{C_1\} \qquad | \qquad N \cup \{C_2\}}$$

where C_1 and C_2 do not have common variables.

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

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

Splitting: $$\frac{N \cup \{C_1 \vee C_2\}}{N \cup \{C_1\} \qquad | \qquad N \cup \{C_2\} \cup \{\neg C_1\}}$$

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 $\perp \leftarrow \mathcal{L}$ 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 $C \to$ propositional variable P_C
positive ground component $C \to$ propositional variable P_C
negative ground component $C \to$ negated propositional variable $\neg P_C$

Therefore: splittable clauses $\to$ 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.