6.3 Reduction Pairs and Argument Filterings

Goal: Show the non-existence of K-minimal infinite rewrite sequences

$$t_1 \rightarrow_R^* u_1 \rightarrow_K t_2 \rightarrow_R^* u_2 \rightarrow_K \dots$$

using well-founded orderings.

We observe that the requirements for the orderings used here are less restrictive than for reduction orderings:

K-rules are only used at the top, so we need stability under substitutions, but compatibility with contexts is unnecessary.

While $\rightarrow_K$-steps should be decreasing, for $\rightarrow_R$-steps it would be sufficient to show that they are not increasing.

Reduction Pairs and Argument Filterings

This motivates the following definitions:

Rewrite quasi-ordering $\succsim$:

reflexive and transitive binary relation, stable under substitutions, compatible with contexts.

Reduction pair $(\succsim, \succ)$:

$\succsim$ is a rewrite quasi-ordering.

$\succ$ is a well-founded ordering that is stable under substitutions.

$\succsim$ and $\succ$ are compatible: $\succsim \circ \succ \subseteq \succ$ or $\succ \circ \succsim \subseteq \succ$.

(In practice, $\succ$ is almost always the strict part of the quasi-ordering $\succsim$.)

Reduction Pairs and Argument Filterings

Clearly, for any reduction ordering $\succ$, $(\succeq, \succ)$ is a reduction pair. More general reduction pairs can be obtained using argument filterings:

Argument filtering π:

$$\pi : \Omega \cup \Omega^\sharp \to \mathbb{N} \cup \text{list of } \mathbb{N}$$

$$\pi(f) = \begin{cases} i \in \{1, \dots, \mathit{arity}(f)\}, \text{ or} \\ [i_1, \dots, i_k], \text{ where } 1 \le i_1 < \cdots < i_k \le \mathit{arity}(f), \\ \qquad\qquad 0 \le k \le \mathit{arity}(f) \end{cases}$$

Reduction Pairs and Argument Filterings

Extension to terms:

$$\pi(x) = x$$

$\pi(f(t_1, \ldots, t_n)) = \pi(t_i)$, if $\pi(f) = i$

$\pi(f(t_1, \ldots, t_n)) = f'(\pi(t_{i_1}), \ldots, \pi(t_{i_k}))$, if $\pi(f) = [i_1, \ldots, i_k]$, where f'/k is a new function symbol.

Reduction Pairs and Argument Filterings

Let $\succ$ be a reduction ordering, let π be an argument filtering. Define $s \succ_\pi t$ iff $\pi(s) \succ \pi(t)$ and $s \succsim_\pi t$ iff $\pi(s) \succeq \pi(t)$.

Lemma 6.2:
$(\succsim_\pi, \succ_\pi)$ is a reduction pair.

Proof:
Follows from the following two properties:

$\pi(s\sigma) = \pi(s)\sigma_\pi$, where $\sigma_\pi(x) := \pi(\sigma(x))$.

$$\pi(s[u]_p) = \begin{cases} \pi(s), \text{if } p \text{ does not correspond to any position in } \pi(s) \\ \pi(s)[\pi(u)]_q, \text{if } p \text{ corresponds to } q \text{ in } \pi(s) \end{cases}$$

$\square$

Reduction Pairs and Argument Filterings

For interpretation-based orderings (such as polynomial orderings) the idea of “cutting out” certain subterms can be included directly in the definition of the ordering:

Reduction Pairs and Argument Filterings

Reduction pairs by interpretation:

Let $\mathcal{A}$ be a Σ-algebra; let $\succ$ be a well-founded strict partial ordering on its universe.

Assume that all interpretations $f_{\mathcal{A}}$ of function symbols are weakly monotone, i. e., $a_i \succeq b_i$ implies $f(a_1, \dots, , a_n) \succeq f(b_1, \dots, b_n)$ for all $a_i, b_i \in U_{\mathcal{A}}$.

Define $s \succsim_{\mathcal{A}} t$ iff $\mathcal{A}(\beta)(s) \succeq \mathcal{A}(\beta)(t)$ for all assignments $\beta : X \to U_{\mathcal{A}}$; define $s \succ_{\mathcal{A}} t$ iff $\mathcal{A}(\beta)(s) \succ \mathcal{A}(\beta)(t)$ for all assignments $\beta : X \to U_{\mathcal{A}}$.

Then $(\succsim_{\mathcal{A}}, \succ_{\mathcal{A}})$ is a reduction pair.

Reduction Pairs and Argument Filterings

For polynomial orderings, this definition permits interpretations of function symbols where some variable does not occur at all (e. g., $P_f(X, Y) = 2X + 1$ for a *binary* function symbol). It is no longer required that *every* variable must occur with some positive coefficient.

Reduction Pairs and Argument Filterings

Theorem 6.3 (Arts and Giesl):
Let K be a cycle in the dependency graph of the TRS R. If there is a reduction pair $(\succsim, \succ)$ such that

- $l \succsim r$ for all $l \to r \in R$,
- $l \succsim r$ or $l \succ r$ for all $l \to r \in K$,
- $l \succ r$ for at least one $l \to r \in K$,

then there is no K-minimal infinite sequence.

Reduction Pairs and Argument Filterings

Proof:

Assume that $t_1 \to_R^* u_1 \to_K t_2 \to_R^* u_2 \to_K \dots$ is a K-minimal infinite rewrite sequence.

As $l \succsim r$ for all $l \to r \in R$, we obtain $t_i \succsim u_i$ by stability under substitutions, compatibility with contexts, reflexivity and transitivity.

As $l \succsim r$ or $l \succ r$ for all $l \to r \in K$, we obtain $u_i \; ({\succsim} \cup {\succ}) \; t_{i+1}$ by stability under substitutions.

So we get an infinite $({\succsim} \cup {\succ})$-sequence containing infinitely many $\succ$-steps (since every DP in K, in particular the one for which $l \succ r$ holds, is used infinitely often).

By compatibility of $\succsim$ and $\succ$, we can transform this into an infinite $\succ$-sequence, contradicting well-foundedness. $\square$

Reduction Pairs and Argument Filterings

The idea can be extended to SCCs in the same way as for the subterm criterion:

Search for a reduction pair $(\succsim, \succ)$ such that $l \succsim r$ for all $l \to r \in R$ and $l \succsim r$ or $l \succ r$ for all DPs $l \to r$ in the SCC. Delete all DPs in the SCC for which $l \succ r$. Then re-compute SCCs for the remaining graph and re-start.

Reduction Pairs and Argument Filterings

Example: Consider the following TRS R from [Arts and Giesl]:

$$minus(x, 0) \to x \tag{1}$$

$$minus(s(x), s(y)) \to minus(x, y) \tag{2}$$

$$quot(0, s(y)) \to 0 \tag{3}$$

$$quot(s(x), s(y)) \to s(quot(minus(x, y), s(y))) \tag{4}$$

(R is not contained in any simplification ordering, since the left-hand side of rule (4) is embedded in the right-hand side after instantiating y by $s(x)$.)

Reduction Pairs and Argument Filterings

R has three dependency pairs:

$$minus^\sharp(s(x), s(y)) \to minus^\sharp(x, y) \tag{5}$$

$$quot^\sharp(s(x), s(y)) \to quot^\sharp(minus(x, y), s(y)) \tag{6}$$

$$quot^\sharp(s(x), s(y)) \to minus^\sharp(x, y) \tag{7}$$

The dependency graph of R is

$$\begin{tikzcd}
(5) \arrow[loop below] & (7) \arrow[l] & (6) \arrow[l] \arrow[loop below]
\end{tikzcd}$$

Reduction Pairs and Argument Filterings

There are exactly two SCCs (and also two cycles). The cycle at (5) can be handled using the subterm criterion with $\pi(\mathit{minus}^\sharp) = 1$. For the cycle at (6) we can use an argument filtering π that maps *minus* to 1 and leaves all other function symbols unchanged (that is, $\pi(g) = [1, \ldots, \mathit{arity}(g)]$ for every g different from *minus*.) After applying the argument filtering, we compare left and right-hand sides using an LPO with precedence $\mathit{quot} > s$ (the precedence of other symbols is irrelevant). We obtain $l \succ r$ for (6) and $l \succsim r$ for (1), (2), (3), (4), so the previous theorem can be applied.

DP Processors

The methods described so far are particular cases of DP processors:

A DP processor

$$\frac{(G, R)}{(G_1, R_1), \quad \ldots, \quad (G_n, R_n)}$$

takes a graph G and a TRS R as input and produces a set of pairs consisting of a graph and a TRS.

It is sound and complete if there are K-minimal infinite sequences for G and R if and only if there are K-minimal infinite sequences for at least one of the pairs (G_i, R_i).

DP Processors

Examples:

$$\frac{(G, R)}{(SCC_1, R), \ \ \ldots, \ (SCC_n, R)}$$

where $SCC_1, \ldots, SCC_n$ are the strongly conn. components of G.

$$\frac{(G, R)}{(G \setminus N, R)}$$

if there is an SCC of G and a simple projection π such that $\pi(l) \trianglerighteq \pi(r)$ for all DPs $l \to r$ in the SCC, and N is the set of DPs of the SCC for which $\pi(l) \triangleright \pi(r)$.

(and analogously for reduction pairs)

Innermost Termination

The dependency method can also be used for proving termination of innermost rewriting: $s \xrightarrow{\mathrm{i}}_R t$ if $s \to_R t$ at position p and no rule of R can be applied at a position strictly below p. (DP processors for innermost termination are more powerful than for ordinary termination, and for program analysis, innermost termination is usually sufficient.)

6.4 Superposition

Goal:

Combine the ideas of superposition for first-order logic without equality (overlap maximal literals in a clause) and Knuth-Bendix completion (overlap maximal sides of equations) to get a calculus for equational clauses.

Observation

It is possible to encode an arbitrary predicate p using a function f_p and a new constant tt:

$$\begin{array}{rcr} P(t_1, \ldots, t_n) & \rightsquigarrow & f_P(t_1, \ldots, t_n) \approx tt \\ \neg\, P(t_1, \ldots, t_n) & \rightsquigarrow & \neg\, f_P(t_1, \ldots, t_n) \approx tt \end{array}$$

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 $\neg\, s \approx t$.

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.

The Superposition Calculus – Informally

Ground inference rules:

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

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

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

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

The Superposition Calculus – Informally

Ordering restrictions:

Some considerations:

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 Superposition Calculus – Informally

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 $\succ_L$.

The Superposition Calculus – Informally

Ordering restrictions:

Ground inferences are necessary only if the following conditions are satisfied:

- In superposition inferences, the left premise is smaller than the right premise.
- The literals that are involved in the inferences are maximal in the respective clauses
  (strictly maximal for positive literals in superposition inferences).
- In these literals, the lhs is greater than or equal to the rhs (in superposition inferences: greater than the rhs).

The Superposition Calculus – Informally

Model construction:

We want to use roughly the same ideas as in the completenes proof for superposition on first-order without equality.

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

The Superposition Calculus – Informally

Solution: Define a set E of ground equations and take $\mathrm{T}_\Sigma(\emptyset)/E = \mathrm{T}_\Sigma(\emptyset)/{\approx_E}$ as the universe.

Then two ground terms s and t are equal in the interpretation, if and only if $s \approx_E t$.

If E is a terminating and confluent rewrite system R, then two ground terms s and t are equal in the interpretation, if and only if $s \downarrow_R t$.

The Superposition Calculus – Informally

One problem:

In the completeness proof for the resolution calculus, the following property holds:

> 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 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 $b \approx c \vee b \approx d$.

The Superposition Calculus – Informally

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:
$$\frac{C' \lor s \approx t' \lor s \approx t}{C' \lor t \not\approx t' \lor s \approx t'}$$

Note: This inference rule subsumes the usual factoring rule.

The Superposition Calculus – Informally

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

Main idea as in non-equational first-order case:

Replace identity by unifiability.
Apply the mgu to the resulting clause.
In the ordering restrictions, replace $\succ$ by $\not\preceq$.

The Superposition Calculus – Informally

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.

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 are applied with respect to the commutativity of equality $\approx$.

The Superposition Calculus – Formally

Inference rules (part 1):

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.

The Superposition Calculus – Formally

Inference rules (part 2):

Equality Resolution:

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

where $\sigma = \mathrm{mgu}(s, s')$.

Equality Factoring:

$$\frac{C' \lor s' \approx t' \lor s \approx t}{(C' \lor t \not\approx t' \lor s \approx t')\sigma}$$

where $\sigma = \mathrm{mgu}(s, s')$.

The Superposition Calculus – Formally

Theorem 6.4:
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. $\square$

The Superposition Calculus – Formally

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 $\succ_L$.

The Superposition Calculus – Formally

Inferences have to be computed only if the following ordering restrictions are satisfied:

– In superposition inferences, after applying the unifier to both premises, 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 not smaller than the rhs
(in superposition inferences: neither smaller nor equal).

The Superposition Calculus – Formally

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 $N \cup Red(N)$.