Multisets

A multiset is a set that can distinguish how often an element occurs in it. (Alternatively, it is a list that cannot distinguish the order of its elements.)

We write multisets with square brackets:

$$[a, b, c], \quad [\ ], \quad [a, a, a, b, b], \quad [b, a, a, b, a].$$

The last two multisets are equal.

Formally, a multiset is a function S from its domain to $\mathcal{N}$, so one can write $S(n)$ for the number of occurrences of n in S.

A multiset is finite if $\sum_{d \in D} S(d)$ is finite.

Operations on Multisets

Let D be the common domain of the multisets.

Define

$$A \cup B = \lambda d \in D : \; A(d) + B(d),$$

$$A \cap B = \lambda d \in D : \; \min(\; A(d), B(d) \;),$$

$$A \backslash B = \lambda d \in D : \; A(d) - B(d).$$

In the definition of $A \backslash B$, it is assumed that for all $d \in D : \; A(d) \geq B(d)$.

Resolution with Multisets

Using multisets, one can define resolution as follows:

Resolution From $[A] \cup R_1$ and $[\neg A] \cup R_2$ derive $R_1 \cup R_2$.

Factoring From $[A, A] \cup R$ derive $[A] \cup R$.

It is possible to restrict resolution and factoring by a total order $>$ as follows: In resolution, it must be the case $A > R_1$, and $\neg A > R_2$. With factoring, it mus be the case that $A > R$.

The resulting calculus is still complete. Try it out!

Multiset Order

Let D be a domain for the multisets. Let $<$ be a well-order on D.

The multiset extension $<<$ of $<$ to finite multisets is defined as follows:

$S_1 << S_2$ iff for every $d \in D$ with $S_1(d) > S_2(d)$, there is a d', s.t. $d < d'$ and $S_1(d') < S_2(d')$.

Alternatively, one can say that for the $<$-maximal element for which $S_1(d) \neq S_2(d)$, it must be the case that $S_1(d) < S_2(d)$.

The maximal element exists because S_1 and S_2 are finite.

There is also the following definition: Pick one element d from S_2 and replace it by an aribtrary, finite multiset of elements $[d_1, \ldots, d_n]$ with $d_i < d$. Call the resulting set S_1. Then $S_1 << S_2$.

Multiset Order (2)

One can view a multiset as a number, where the digits are taken from $\mathcal{N}$ and D are the possible positions:

Let $N_1 = (d_m, \ldots, d_0)$ and $N_2 = (d'_{m'}, \ldots, d'_0)$ be numbers : N_1 is bigger than N_2 if on the first position where N_1 and N_2 differ, the digit in N_1 is bigger than the digit in N_2.

If N_1 is a number, then a smaller number N_2 can be obtained by decreasing one digit, while at the same time increasing arbitrary digits at lower positions.

For example:

$$(1, 0, 0, 0, 0, 0, 0) > (0, 9, 9, 9, 9, 9, 9).$$

Superposition

Superposition is a combination of resolution and Knuth-Bendix completion.

A clause is a finite set of ground equalities or negations of ground equalities:

$$[\ f(a) \approx b, \quad g(b) \approx b\ ]$$

$$[\ a \not\approx b,\ a \approx c\ ]$$

$$[\ a \not\approx b,\ s(a) \approx s(b)\ ]$$

$$[\ a \not\approx b,\ a \approx f(a)\ ]$$

Ordering Equalities

Let $\succ$ be a simplification order. We extend $\succ$ to positive and negative equalities as follows:

$$\begin{array}{rcl} S(t_1 \approx t_2) & = & [t_1, t_2], \\ S(t_1 \not\approx t_2) & = & [t_1, t_1, t_2, t_2]. \end{array}$$

Then $A \succ B$ iff $S(A) \succ\succ S(B)$.

Now we can use the order to direct equalities, and to sort equalities within a clause.

In order to allow non-equality predicates, we assume a constant term $\mathbf{t}$, with the property $t \succ \mathbf{t}$ for all other terms t.

$$[A \approx \mathbf{t},\ A \approx \mathbf{t}]$$

$$[A \not\approx \mathbf{t},\ A \not\approx \mathbf{t}]$$

Positive Superposition

Assume that $[\ t_1 \approx t_2\ ] \cup R_1$ and $[\ u_1 \approx u_2\ ] \cup R_2$ are derived clauses. If

1. $t_1 \succ t_2$,
2. $(t_1 \approx t_2) \succ\!\succ R_1$,
3. $u_1 \succ u_2$,
4. $(u_1 \approx u_2) \succ\!\succ R_2$,
5. u_1 contains t_1 as subterm,

then derive $[\ u_1[\ t_1 \Rightarrow t_2\ ] \approx u_2\ ] \cup R_1 \cup R_2$.

Negative Superposition

Assume that $[\ t_1 \approx t_2\ ] \cup R_1$ and $[\ u_1 \not\approx u_2\ ] \cup R_2$ are derived clauses. If

1. $t_1 \succ t_2$,
2. $(t_1 \approx t_2) \succ\succ R_1$,
3. $u_1 \succ u_2$,
4. $(u_1 \not\approx u_2) \succeq\succeq R_2$,
5. u_1 contains t_1 as subterm,

then derive $[\ u_1[\ t_1 \Rightarrow t_2\ ] \not\approx u_2\ ] \cup R_1 \cup R_2$.

Equality Reflexivity

Assume that $[\ t \not\approx t\ ] \cup R$ is a derived clause. If

1. $(t \not\approx t) \succeq\succeq R$,

then derive R.

Equality Factoring

Assume that $[\ t \approx u_1,\ t \approx u_2\ ] \cup R$ is a derived clause. If

1. $t \succ u_1$,
2. $u_1 \succeq u_2$,
3. $(t \approx u_1) \succeq\!\succeq R$,

then derive $[\ t \approx u_1,\ u_1 \not\approx u_2\ ] \cup R$.

In order to see the correctness of this rule, do a case split on $u_1 \approx u_2$.

Note: One could also use $[\ t \approx u_2,\ u_1 \not\approx u_2\ ] \cup R$, but it is better to use $[\ t \approx u_1,\ u_1 \not\approx u_2\ ] \cup R$ because this clause is $\succ\!\succ$-bigger, and therefore has a slightly better chance of being redundant.)

Algorithm Based on Superposition

In order to check whether a set S of clauses is satisfiable, one can use the following algorithm:

As long as there exists a clause c, which can be derived from S by positive/negative superposition, equality reflexivity, or equality factoring, add c to S.

As soon as S contains the empty clause, we know that S is unsatisfiable.

If no more clauses can be added, we know that S is satisfiable.

Soundness and Completeness

Theorem: If it is possible to derive the empty clause [ ] by repeated applications of positive/negative superposition, equality reflexivity and equality factoring from a clause set S, then the clause set S is unsatisfiable.

Proof: As usual, this is the easy part. It is sufficient to inspect the rules, and observe that they are logically sound.

Theorem: If a clause set S is not satisfiable, then it is possible to derive the empty clause by repeated applications of positive/negative superposition, equality reflexivity and equality factoring.

Proof: We call a clause set S saturated if every clause c that can be derived from S by a single application of positive/negative superposition, equality reflexivity, or equality factoring, is already present in S.

We will prove completeness by showing that every saturated clause set has a model, but we first introduce redundancy.

Redundancy is a very important optimization, which tells that not all derivable clauses have to be present in a saturated set. Because of this, it is possible to delete clauses from a saturated set without losing completeness.

Redundancy

We have already seen some forms of redundancy:

1. If we have two clauses c_1 and c_2 with $c_1 \subset c_2$ in S, then one would like to delete c_2 from S. One says that c_1 subsumes c_2.

2. If we have a clause $[\ t_1 \approx t_2\ ] \cup R_1$ and another clause $[\ u[t_1] \approx u[t_2]\ ] \cup R_2$ with $R_1 \subseteq R_2$ in S, then one would like to delete the second clause $[\ u[t_1] \approx u[t_2]\ ] \cup R_2$ from S.

Redundancy (2)

Let c be a clause. Let S be a clause set. We way that c is redundant in S if there exist clauses $c_1, \ldots, c_n \in S$, with the following properties:

1. $c_1, \ldots, c_n$ logically imply c, and
2. for each c_i, $\quad c_i \preceq\preceq c$.

Here $\preceq\preceq$ is the extension of $\preceq$ to clauses. (In theory, one should write four times $\preceq$, but I think two is enough.)

Since $c_i \preceq\preceq c$ is equivalent to $c_i \prec\prec c$ or $c_i = c$, we can reformulate the definition of redundancy as

1. either $c \in S$, or
2. there exist $c_1, \ldots, c_n$ with $c_i \prec\prec c$, which logically imply c.

Saturated Clause Set

Let S be a clause set. We call S saturated if every clause c that can be derived from clauses in S with a single step of positive/negative superposition, equality resolution, or equality factoring, is redundant in S.

We call S a saturation of of a clause set C if every clause c in C is redundant in S.

Three Ways to Use Redundancy

Redundancy can be used in three possible ways:

1. If a new clause is obtained by one of the four rules, then check its redundancy. If it is redundant, then don't keep it. This is called forward redundancy checking.
2. If we derive a new clause, then check if any existing clauses become redundant. If this is the case, then delete these clauses. This is called backward redundancy checking.
3. Try to make logically correct inferences, (while completely ignoring the superposition calculus) that make existing clauses redundant. This is called simplification.

Examples of Simplification

- Merging of repleated equalities is always possible: Every clause of form $[\ t \approx u,\ \ t \approx u\ ] \cup R$ can be replaced by $[\ t \approx u\ ] \cup R$. Similarly, every clause of form $[\ t \not\approx u,\ \ t \not\approx u\ ] \cup R$ can be replaced by $[\ t \not\approx u\ ] \cup R$.

  In both cases, the merging is logically sound, and the result is $\prec\!\prec$-smaller.

- A clause of form $[\ t \not\approx u,\ \ f(t) \approx c\ ] \cup R$ can be replaced by $[\ t \not\approx u,\ \ f(u) \approx c\ ] \cup R$ if $t \succ u$.

  The replacement is logically sound, and the result is $\prec\!\prec$-smaller.

Model Construction

Let S be a saturated set of clauses. We will show that S has a model.

The model will be represented by a rewrite system I without critical pairs, and ordered by $\succ$.

Using properties of strongly normalizing, confluent rewrite systems, a clause c is true in this model iff either

1. c contains a positive equality $t_1 \approx t_2$, for which t_1 and t_2 have the same normal form in I.
2. c contains a negative equality $t_1 \not\approx t_2$, for which t_1 and t_2 have different normal forms in I.

Model Construction (2)

Let $\succ$ be our simplification order on terms. We first extend $\succ$ to pairs of terms as follows:

$$(t_1, t_2) \succ_2 (u_1, u_2) \text{ iff } t_1 \succ u_1 \text{ or } (t_1 = u_1 \text{ and } t_2 \succ u_2).$$

If $\succ$ has ordinal length α, then $\succ_2$ has ordinal length $\alpha \times \alpha$.

For an ordinal λ with $0 \leq \lambda < \alpha \times \alpha$, let π_λ be the pair that has index λ.

Given a saturated set S, we iterate through the pairs π_λ, and decide if π_λ should be added to the interpretation.

Model Construction (3)

Let S be a saturated set. For $\lambda \leq \alpha \times \alpha$, we define:

- For a limit ordinal λ, put

$$I_\lambda = \bigcup_{\lambda' < \lambda} I_{\lambda'}.$$

- In order to define I for a successor ordinal, we specify how to obtain $I_{\lambda+1}$ from I_λ: First write $\pi_\lambda = (t_1, t_2)$.

  If (1) $t_1 \succ t_2$, and (2) I_λ does not contain a rule that can rewrite t_1, and (3) there exists a clause of form $[\ t_1 \approx t_2\ ] \cup R$ in S with $(t_1 \approx t_2) \succeq\succeq R$, and (4) this clause $[\ t_1 \approx t_2\ ] \cup R$ is false in I_λ, and (5) R would be still false in $I_\lambda \cup \{t_1 \to t_2\}$, then put $I_{\lambda+1} = I_\lambda \cup \{t_1 \to t_2\}$. Otherwise, put $I_{\lambda+1} = I_\lambda$.

Since 0 is a limit ordinal, and there are no ordinals below 0, we have $I_0 = \{\}$.

We assumed that I was defined for all $\lambda' < \lambda$ to obtain a value for I_λ.

In the case of a successor ordinal, we looked only at the previous value. In the case of a limit ordinal, we used all preceeding values.

In order to prove that the function I_λ is well-defined, one has to remember that every successor ordinal has a unique predecessor, which follows from well-foundedness.

Theorem: Every I_λ is strongly normalizing.

Proof: By construction we have $t_1 \succ t_2$ for every rule $(t_1 \to t_2)$ in I_λ, and $\succ$ is a reduction order.

Theorem: No I_λ has a critical pair.

Proof: Supppose there were one, we would have distinct rules $(t_1 \to t_2), (u_1 \to u_2) \in I_\lambda$ with t_1 a subterm of u_1.

Assume that $(t_1, t_2) = \pi_{\lambda_1}$ and $(u_1, u_2) = \pi_{\lambda_2}$. If $t_1 \neq u_1$, then it must be the case that $t_1 \prec u_1$, so that by definition of $\prec_2$, we have $\lambda_1 < \lambda_2$. If $t_1 = u_1$, we can assume without loss of generality that $t_1 \prec u_2$, so that in that case, we can also have $\lambda_1 < \lambda_2$.

At level λ_2 of the construction, condition (2) would have been false, so that $(u_1 \to u_2)$ would not have been added.

Theorem MAXANDONLY: For every rewrite rule $t_1 \to t_2$ in $I_{\alpha \times \alpha}$, there is a clause c of form $[\ t_1 \approx t_2\ ] \cup R$ in S, such that

1. $t_1 \succ t_2$,
2. $(t_1 \approx t_2) \succ\succ R$,
3. R is false in $I_{\alpha \times \alpha}$.

Proof: Let λ be the ordinal for which $\pi_\lambda = (t_1, t_2)$. The fact that $(t_1 \to t_2) \in I_{\alpha \times \alpha}$ implies that some clause of form $[\ t_1 \approx t_2\ ] \cup R$ met the conditions (1,2,3,4,5) of the model construction at stage λ. It follows from condition (1) that $t_1 \succ t_2$. Condition (3) implies that $(t_1 \approx t_2) \succeq\succeq R$. Because R is false in $I_\lambda \cup \{t_1 \to t_2\}$, R cannot contain another instance of $t_1 \approx t_2$. It follows that $(t_1 \approx t_2) \succ\succ R$.

It follows from condition (5) that R is false in I_λ. We show on the next slide that R is still false in $I_{\alpha \times \alpha}$.

Proof of MAXANDONLY (2)

For a negative literal $(u_1 \not\approx u_2) \in R$, the fact that it is false in $I_{\lambda+1}$ implies that $I_{\lambda+1}$ merges u_1 with u_2. Since $I_{\lambda+1} \subseteq I_{\alpha\times\alpha}, \quad I_{\alpha\times\alpha}$ still merges u_1 with u_2. As a consequence, $(u_1 \not\approx u_2)$ is still false in $I_{\alpha\times\alpha}$.

Proof of MAXANDONLY (3)

For a positive literal $(u_1 \approx u_2)$, suppose that $I_{\alpha\times\alpha}$ merges u_1 with u_2. We show that this contradicts the fact that $I_{\lambda+1}$ does not merge u_1 with u_2.

If $u_1 = u_2$, then obviously $I_{\lambda+1}$ would merge u_1 with u_2, so that this is a contradiction.

Since $I_{\lambda+1}$ does not merge u_1 with u_2, there must exist a rule $(w_1 \rightarrow w_2) \in (I_{\alpha\times\alpha}\backslash I_{\lambda+1})$, that is used when merging u_1 with u_2. If $w_1 = t_1$, then this would result in a critical pair, because of the rule $(t_1 \rightarrow t_2) \in I_{\lambda+1}$.

It follows that $(w_1, w_2) \succ_2 (t_1, t_2)$, so that $w_1 \succ t_1$. Since every rewrite sequence is $\succ$-decreasing, it must be the case that $u_1 \succeq w_1$ or $u_2 \succeq w_1$. Since $t_1 \succ t_2$, this would imply that $(u_1 \approx u_2) \succ (t_1 \approx t_2)$, which contradicts condition (3).

All clauses in S are true in $I_{\alpha \times \alpha}$.

We will prove this by transfinite induction, using the order $\prec\prec$ on clauses.

We will assume that all clauses c' with $c' \prec\prec c$ are true in $I_{\alpha \times \alpha}$, and use this fact to show that c is true in $I_{\alpha \times \alpha}$.

The proof consists of many cases, which depend on the form of the maximal element in c.

Maximal Element is Negative

Assume that c has form $[\ t \not\approx t\ ] \cup R$ with $(t \not\approx t) \succeq\succeq R$.

Since S is saturated, there are clauses $c_1, \ldots, c_n \in S$, with $c_i \preceq\preceq R$, which logically imply R. By induction, these clauses are true in $I_{\alpha\times\alpha}$, so that R is true in $I_{\alpha\times\alpha}$.

This implies that $[\ t \not\approx t\ ] \cup R$ is also true in $I_{\alpha\times\alpha}$.

Maximal Element is Negative (2)

Assume that c has form $[\ t_1 \not\approx t_2\ ] \cup R$ with $(t_1 \not\approx t_2) \succeq\succeq R$ and $t_1 \succ t_2$.

If there is no rule $(u_1 \rightarrow u_2) \in I_{\alpha\times\alpha}$ that can rewrite t_1, then $t_1 \not\approx t_2$ must be true in $I_{\alpha\times\alpha}$, so that c is true in $I_{\alpha\times\alpha}$.

So assume there is a rule $(u_1 \rightarrow u_2) \in I_{\alpha\times\alpha}$, that can rewrite t_1.

By MAXANDONLY, there is a clause $[\ u_1 \approx u_2\ ] \cup R'$ in S, s.t. $u_1 \succ u_2$, $(u_1 \approx u_2) \succ\succ R'$, and R' is false in $I_{\alpha\times\alpha}$. Since u_1 is a subterm of t_1, one can apply negative superposition with c and obtain the clause $[\ t_1[u_1 \Rightarrow u_2] \not\approx t_2\ ] \cup R \cup R'$. This clause is $\prec\prec$-smaller than c. Using the fact that S is saturated in the same way as on the previous slide, we see that $[\ t_1[u_1 \Rightarrow u_2] \not\approx t_2\ ] \cup R \cup R'$ is true in $I_{\alpha\times_\alpha}$. Since $I_{\alpha\times\alpha}$ still merges $[\ t_1[u_1 \Rightarrow u_2]$ with t_2, and R' is false in $I_{\alpha\times\alpha}$, it follows that R must be true in $I_{\alpha\times\alpha}$. This implies that c is true in $I_{\alpha\times\alpha}$.

Maximal Element is Positive

The cases where the maximal element is positive are similar, but much trickier.

If the maximal element is positive, then c can be written in the form

$$[\ t_1 \approx t_2\ ] \cup R \text{ with } t_1 \succeq t_2 \text{ and } (t_1 \approx t_2) \succeq\succeq R.$$

There exists a λ with $0 \leq \lambda < \alpha \times \alpha$, s.t. $\pi_\lambda = (t_1, t_2)$.

In order to show that c is true, we need to make a very big case distinction. We check the cases on the following slides.

If $t_1 = t_2$, then c is obviously true.

If $t_1 \neq t_2$, then $t_1 \succ t_2$.

We first cover the case where there are no further occurrences of t_1 in c. In this case, c can be written in the form $[\ t_1 \approx t_2\ ] \cup R$, where R does not contain t_1. It is easily checked that $(t_1 \approx t_2) \succ\!\succ R$.

If $I_{\alpha\times\alpha}$ contains a rule $u_1 \rightarrow u_2$ that can rewrite t_1, then by MAXANDONLY, there is a clause of form $[\ u_1 \approx u_2\ ] \cup R'$ in S, s.t. $u_1 \succ u_2,\ \ (u_1 \approx u_2) \succ\!\succ R'$, and R' is false in $I_{\alpha\times\alpha}$.

We can apply positive superposition with c and obtain $[\ t_1[u_1 \Rightarrow u_2] \approx t_2\ ] \cup R \cup R'$. This clause is $\prec\!\prec$-smaller than c, so that we can assume that it is true by the reasoning that we have already seen before. Since R' is false in $I_{\alpha\times\alpha}$, it follows that either R is true in $I_{\alpha\times\alpha}$, or $I_{\alpha\times\alpha}$ merges $t_1[u_1 \Rightarrow u_2]$ with t_2. In the latter case, since $I_{\alpha\times\alpha}$ contains $u_1 \rightarrow u_2$, it also merges t_1 with t_2.

We now check the case where $t_1 \succ t_2$, there are no occurrences of t_1 in R, there is no rule $(u_1 \rightarrow u_2)$ in $I_{\alpha\times\alpha}$ that can rewrite t_1, and R was true in I_λ.

Assume that R contains a positive equality $u_1 \approx u_2$, such that I_λ merges u_1 and u_2. Since $I_\lambda \subseteq I_{\alpha\times\alpha}$, the terms u_1 and u_2 are also merged by $I_{\alpha\times\alpha}$, so that R is true in $I_{\alpha\times\alpha}$.

Assume that R contains a negative equality $u_1 \not\approx u_2$, s.t. I_λ does not merge u_1 with u_2. Since neither u_1, u_2 contains t_1, $I_{\lambda+1}$ will also not merge u_1 with u_2. Now assume that $I_{\alpha\times\alpha}$ merges u_1 with u_2.

This implies that $I_{\alpha\times\alpha}\backslash I_{\lambda+1}$ contains a rule $(w_1 \rightarrow w_2)$ that played a role in merging u_1 or u_2. It follows that either $u_1 \succeq w_1$ or $u_2 \succeq w_1$, which implies $t_1 \succ w_1$. This contradicts the fact that $(w_1 \rightarrow w_2) \in (I_{\alpha\times\alpha}\backslash I_{\lambda+1})$.

We check the case where $t_1 \succ t_2$, there are no occurrences of t_1 in R, there is no rule in $I_{\alpha \times \alpha}$ that can rewrite t_1, and R is false in I_λ.

If I_λ merges t_1 with t_2, then because $I_\lambda \subseteq I_{\alpha \times \alpha}, \quad I_{\alpha \times \alpha}$ also merges t_1 with t_2, so that c is true in $I_{\alpha \times \alpha}$.

If I_λ does not merge t_1 with t_2, then the clause $[\ t_1 \approx t_2\ ] \cup R$ is false in I_λ.

If R would be true in $I_\lambda \cup \{t_1 \rightarrow t_2\}$, then the true element in R cannot have form $u_1 \not\approx u_2$: If $I_\lambda \cup \{t_1 \rightarrow t_2\}$ does not merge u_1 with u_2, then certainly, also I_λ does not merge u_1 with u_2.

If it has form $u_1 \approx u_2$, and is true in $I_\lambda \cup \{t_1 \rightarrow t_2\}$, but not in I_λ, this can only happen when u_1 or u_2 contains t_1, which contradicts the assumption that R does not contain t_1.

As a consequence $I_{\lambda+1}$ contains $t_1 \rightarrow t_2$, which implies that $I_{\alpha \times \alpha}$ contains $t_1 \rightarrow t_2$, which makes c true.

(Finally, the model construction has done some useful work!)

It remains to check the cases where $t_1 \succ t_2$, and there are other occurrences of t_1 in R. The occurrences cannot be in negative equalities, because that would imply that $R \succ\succ (t_1 \approx t_2)$.

It follows that R can be written in the form

$$[\ t_1 \approx u_1,\ t_1 \approx u_2, \ldots t_1 \approx u_n\ ] \cup R',$$

where R' does not contain t_1, and each $t_1 \succ u_j$. (It is possible that $u_j = t_2$.)

If $I_{\alpha \times \alpha}$ merges t_2 with one of the u_j, we may assume without loss of generality that it is u_1. One can form the equality factor $[\ t_1 \approx t_2,\ t_2 \not\approx u_1,\ t_1 \approx u_2, \ldots, t_1 \approx u_n\ ] \cup R'$ from c. Since the equality factor is $\prec\prec$-smaller than c, we can assume by the reasoning that we have already seen before that it is true in $I_{\alpha \times \alpha}$.

Since we assumed that $I_{\alpha \times \alpha}$ merges t_2 with u_1, it follows that $t_2 \not\approx u_1$ is not true in I_α, which implies that c is true in I_α.

(and now we have seen the equality factoring rule in action!)

It remains to check the case where $I_{\alpha\times\alpha}$ does not merge t_2 with any of the u_j.

If I_λ merges t_1 with t_2, then c is obviously true in $I_{\alpha\times\alpha}$.

If I_λ does not merge t_1 with t_2, then c meets the conditions (1,2,3,4) in the model construction. By the argument from three slides ago, no element in R' will be true in $I_\lambda \cup \{t_1 \rightarrow t_2\}$.

If one of $t_1 \approx u_j$ would be true in $I_\lambda \cup \{t_1 \rightarrow t_2\}$, this would imply that $I_{\lambda+1}$ merges t_1 with t_2, and also t_1 with u_j. This would imply that $I_{\lambda+1}$ also merges t_2 with u_j, which contradicts the fact that we are checking here the case where $I_{\alpha\times\alpha}$ does not merge merge t_2 with any of the u_j. It follows that condition (5) is met, so that $I_{\lambda+1}$ contains $t_1 \rightarrow t_2$, which implies that $I_{\alpha\times\alpha}$ makes c true.

Selection Functions

The superposition calculus can be further extended with selection functions:

A selection function is a function from clauses to sets of negative equalities. It must be the case that $\Sigma(c) \subseteq c$.

This means that $\Sigma(c) = \emptyset$, when c has no negative equalities.

Use of Selection Functions

- A positive equality occurring in a clause $c = [\, t_1 \approx t_2 \,] \cup R$ can be used in a rule application if $(t_1 \approx t_2) \succ\succ R$ (or $(t_1 \approx t_2) \succeq\succeq R$, dependent on the rule) and $\Sigma(c) = \emptyset$.
- A negative equality occurring in a clause $c = [\, t_1 \not\approx t_2 \,] \cup R$ can be used in a rule application if either $\Sigma(c) = \emptyset$ and $(t_1 \not\approx t_2) \succeq\succeq R$, or $\Sigma(c)$ is not empty and $(t_1 \not\approx t_2) \in \Sigma(c)$.

Completeness with Selection Functions

The model construction stays almost the same: Condition (3) has to be replaced by: there exists a clause of form $[\ t_1 \approx t_2\ ] \cup R$ in S with $(t_1 \approx t_2) \succeq\succeq R$, and $\Sigma(\ [\ t_1 \approx t_2\ ] \cup R\ ) = \emptyset$.

As far as I see, the proof that all clauses are true in $I_{\alpha \times \alpha}$ stays the same.

Conclusions

The completeness proof is due to Leo Bachmair and
Harald Ganzinger 1994.

It created order in a big chaos of combinations of optimizations of paramodulation, with different completeness proofs.

It is the basis of many theorem provers. (More about this later.)