1

Handout for lecture 12:

Predicate and Equational Logic

1. Recap: predicate logic

1.1 Providing validity

2

Recap

Goal:

- There is a student that is awake during all lectures.
- During all boring lectures no student keeps awake.

$\Rightarrow$ Then there are no boring lectures.

Steps:

- **Specify** the goal in predicate logic.

$$(\exists x[\mathtt{S}(x) \wedge \forall y[\mathtt{L}(y) \rightarrow \mathtt{A}(x,y)]]) \wedge$$
$$(\forall x[(\mathtt{L}(x) \wedge \mathtt{B}(x)) \rightarrow \neg\exists y[\mathtt{S}(y) \wedge \mathtt{A}(y,x)]]) \rightarrow$$
$$\neg\exists x[\mathtt{L}(x) \wedge \mathtt{B}(x)]$$

- **Negate** it (so we can prove that its negation is not satisfiable).

$$(\exists x[\mathtt{S}(x) \wedge \forall y[\mathtt{L}(y) \rightarrow \mathtt{A}(x,y)]]) \wedge$$
$$(\forall x[(\mathtt{L}(x) \wedge \mathtt{B}(x)) \rightarrow \neg\exists y[\mathtt{S}(y) \wedge \mathtt{A}(y,x)]]) \wedge$$
$$\exists x[\mathtt{L}(x) \wedge \mathtt{B}(x)]$$

- Transform the formula into **Prenex normal form**

$$\exists x\exists v\forall y\forall z\forall u[\mathtt{S}(x) \wedge (\neg\mathtt{L}(y) \vee \mathtt{A}(x,y)) \wedge$$
$$(\neg\mathtt{L}(z) \vee \neg\mathtt{B}(z)) \vee (\neg\mathtt{S}(u) \vee \neg\mathtt{A}(u,z)) \wedge$$
$$\mathtt{L}(v) \wedge \mathtt{B}(v)]$$

- Use **Skolemization** to replace existential variables by function symbols

$$\forall y\forall z\forall u[\mathtt{S}(\mathtt{a}) \wedge (\neg\mathtt{L}(y) \vee \mathtt{A}(\mathtt{a},y)) \wedge$$
$$(\neg\mathtt{L}(z) \vee \neg\mathtt{B}(z) \vee \neg\mathtt{S}(u) \vee \neg\mathtt{A}(u,z)) \wedge$$
$$\mathtt{L}(\mathtt{e}) \wedge \mathtt{B}(\mathtt{e})]$$

- Use **resolution** to obtain a contradiction.

1	$\mathtt{S}(\mathtt{a})$	
2	$\neg\mathtt{L}(y) \vee \mathtt{A}(\mathtt{a}, y)$	
3	$\neg\mathtt{L}(z) \vee \neg\mathtt{B}(z) \vee \neg\mathtt{S}(u) \vee \neg\mathtt{A}(u, z)$	
4	$\mathtt{L}(\mathtt{e})$	
5	$\mathtt{B}(\mathtt{e})$	
6	$\mathtt{A}(\mathtt{a}, \mathtt{e})$	$(2, 4)$
7	$\neg\mathtt{B}(\mathtt{e}) \vee \neg\mathtt{S}(u) \vee \neg\mathtt{A}(u, \mathtt{e})$	$(3, 4)$
8	$\neg\mathtt{S}(u) \vee \neg\mathtt{A}(u, \mathtt{e})$	$(5, 7)$
9	$\neg\mathtt{A}(\mathtt{a}, \mathtt{e})$	$(1, 8)$
	$\bot$	$(6, 9)$

Recall: models in predicate logic

Idea:

- give a meaning to function symbols and relation symbols
  - $[\![\mathtt{f}]\!] : \mathcal{M}^n \to \mathcal{M}$ for every function symbol with arity n
  - $[\![\mathtt{P}]\!] : \mathcal{M}^n \to \{\mathit{false}, \mathit{true}\}$ for every relation symbol with arity n
- give a meaning to terms and formulas accordingly
  - given: $\alpha : \mathcal{X} \to \mathcal{M}$
  - $[\![s]\!]_\alpha$ maps terms to $\mathcal{M}$
  - $[\![P]\!]_\alpha$ maps formulas to $\{\mathit{false}, \mathit{true}\}$
- Let $\mathcal{M} \Vdash P$ if $[\![P]\!]_\alpha = \mathit{true}$ for any α
- A formula P is **valid** if $\mathcal{M} \Vdash P$ for all models $\mathcal{M}$

2. (Im-)proving predicate logic

2.1 Proving refutation-completeness

4

Refutation-completeness

Theorem

(refutation-completeness of resolution)

A predicate in CNF is equivalent to *false* if and only if there is a sequence of resolution steps ending in the empty clause.

That is:

there is a sequence of resolution steps ending in the empty clause
$$\iff$$
there is no **model** $\mathcal{M}$ with $\mathcal{M} \Vdash P$

How does one prove such a thing?

5

Proof outline

There are two sides to prove!

Let $\mathcal{X}$ be a set of clauses.

($\Leftarrow$) If there is a sequence of resolution steps ending in the empty clause
then there *is no model* that makes all clauses in $\mathcal{X}$ true.

($\Rightarrow$) If there is no sequence of resolution steps ending in the empty clause
then there *is a model* that makes all clauses in $\mathcal{X}$ true.

In this handout, I will provide a *proof sketch*; some details are omitted, but I hope that this will give a good idea of why this theorem holds. You do not need to know this for the exam, so feel free to skip it; but it provides context that some of you may find interesting.

6

$\Longleftarrow$

One side is relatively easy!

Goal: If there is a sequence of resolution steps ending in the empty clause then there *is no model* that makes all clauses in $\mathcal{X}$ true.

Proof. Suppose (towards a contradiction) that there is a model $\mathcal{M} := (M, [\![\cdot]\!]_\alpha)$ that makes all clauses in $\mathcal{X}$ true. ($**^1$)

Recall that clauses are implicitly *universally quantified*. Hence, a clause C holds in C if $[\![\forall \vec{x}[C]]\!] = \mathit{true}$, where $\vec{x}$ lists the variables occurring bound in C.
By definition of $[\![\cdot]\!]_\alpha$, this is exactly the case if $[\![C]\!]_\alpha = \mathit{true}$ for all α.

So for C a clause with free variables, we will write $\mathcal{M} \Vdash C$ to mean: for all α: $[\![C]\!]_\alpha = \mathit{true}$.

Hence, ($**^1$) can be rewritten as: $\mathcal{M} \Vdash C$ for all $C \in \mathcal{X}$. ($**^2$)

We can see: if $\mathcal{M} \Vdash C$ then $\mathcal{M} \Vdash C\sigma$ for all substitutions σ: ($**^3$)

> $\mathcal{M} \Vdash C\sigma$ holds if $[\![C\sigma]\!]_\alpha$ holds for all α.
>
> Let $\alpha' = [x \mapsto [\![\sigma(x)]\!]_\alpha]$.
>
> Then $\mathcal{M} \Vdash C$ gives that $[\![C]\!]_{\alpha'}$ holds.
>
> By induction on the definition of $[\![C]\!]_{\alpha'}$ we see that $[\![C]\!]_{\alpha'} = [\![C\sigma]\!]_\alpha$, so the latter is *true* as well.

Now consider: what does a resolution step do?

$$\frac{P \vee V \qquad \neg Q \vee W}{V\tau \vee W\tau} \text{ for } \tau \text{ an mgu such that } P\tau = Q\tau$$

If $\mathcal{M} \Vdash P \vee V$ and $\mathcal{M} \Vdash \neg Q \vee W$, then also $\mathcal{M} \Vdash P\tau \vee V\tau$ and $\mathcal{M} \Vdash \neg Q\tau \vee W\tau$ by ($**^3$).

Recall that $\mathcal{M} \Vdash P\tau \vee V\tau$ means that for all α, $[\![P\tau]\!]_\alpha$ holds or $[\![V\tau]\!]_\alpha$ holds. We similarly have that, for all α: $[\![\neg Q\tau]\!]_\alpha$ or $[\![W\tau]\!]_\alpha$.

Since $P\tau = Q\tau$, we cannot both have $[\![P\tau]\!]_\alpha$ and $[\![\neg Q\tau]\!]_\alpha$. Hence, for all α, $[\![V\tau]\!]_\alpha$ or $[\![W\tau]\!]_\alpha$ must hold.

This exactly tells us that $\mathcal{M} \Vdash V\tau \vee W\tau$.

That is: if $\mathcal{M} \Vdash A$ and $\mathcal{M} \Vdash B$ and C follows from A and B by resolution, then $\mathcal{M} \Vdash C$ ($**^4$).

Suppose now that there is a series of resolution steps starting with clauses in $\mathcal{X}$ and ending in $\bot$. By ($**^1$) and ($**^4$) together, we have $\mathcal{M} \Vdash \bot$.

As this is impossible by the definition of $[\![\cdot]\!]_\alpha$, we have the required contradiction! □

$\Longrightarrow$

But how to prove the other side of our lemma?

Goal: If there is no sequence of resolution steps ending in the empty clause then there *is a model* that makes all clauses in $\mathcal{X}$ true.

Answer: By constructing a model!

The basic idea: $\mathcal{M}$ is a set of *ground terms* (that is, terms without variables):

- infinitely many constants $\mathtt{c}_1, \mathtt{c}_2, \ldots$ are in $\mathcal{M}$;
- if $\mathtt{f}$ is a function (not predicate!) of arity n, and $s_1, \ldots, s_n \in \mathcal{M}$, then $\mathtt{f}(s_1, \ldots, s_n) \in \mathcal{M}$.

We choose the value of all atomic formulas $\mathtt{P}(s_1, \ldots, s_n)$ step by step, to avoid a contradiction.

8

$\Longrightarrow$

The overall outline is as follows:

Preparation:

- Start with $\mathcal{M} := \emptyset$.(We will add to it later.)
- Let $\mathcal{N} := \{$all ground instances of clauses that can be obtained from $\mathcal{X}$ using resolution$\}$
- Choose a **well-founded, total** ordering $\succ$ on ground terms. (For example: lexicographic comparison)

Result: we obtain a total ordering on **ground clauses**.

Idea of the proof: we go over all ground clauses $(\neg)A_1 \vee \cdots \vee (\neg)A_n$ in $\mathcal{N}$, in order of $\succ$

- If the clause is true in $\mathcal{M}$ as it is, do nothing.
- Otherwise, include the largest A_i into $\mathcal{M}$.

The tricky part: the largest A_i does not occur negatively since $\mathcal{N}$ is closed under resolution!

The full proof follows below. This is only provided for those who are interested in the mechanics; it will not be on the exam, and you are totally free to skip it!
Let $\succ$ be a *well-founded total order* on all ground atomic formulas $\mathtt{P}(s_1, \ldots, s_n)$ (for example the lexicographic order).

We can now order literals A and $\neg B$, where A, B are ground atoms:

$$\neg A \succ_L A$$

$$(\neg)A \succ_L (\neg)B \text{ if } A \succ B$$

(That is, the negated version of an atom is larger than its non-negated version, but otherwise, we only compare the atoms in the original order $\succ$.) We easily see that $\succ_L$ is also a well-founded total ordering (on literals).

By totality, clauses can be written as $L_1 \vee \cdots \vee L_n$ with $L_1 \succ_L \cdots \succ_L L_n$.

We use this to define an ordering on clauses: $L_1 \vee \cdots \vee L_n \succ_C R_1 \vee \cdots \vee R_m$ if there is $i \geq m, n$ such that:

- $L_1 = R_1, \ldots, L_i = R_i$;
- $i < n$;
- either $i = m$ or $L_{i+1} \succ_L R_{i+1}$.

Since $\succ_L$ is a total and well-founded ordering, also $\succ_C$ is total and well-founded.[1]

Now suppose that $\mathcal{X}$ is a set of predicates such that there is no resolution sequence ending in the empty clause.

Let $\mathcal{X}'$ be the smallest set of clauses that includes $\mathcal{X}$ and is closed under resolution; that is, $\mathcal{X}'$ contains all clauses that can be obtained from $\mathcal{X}$ using zero or more resolution steps. Let $\mathcal{N}$ be the set of all *ground instances* of clauses in $\mathcal{X}'$. Note that $\mathcal{N}$ is totally ordered by $\succ_C$.

Note that $\mathcal{N}$ is closed under the basic (ground) resolution rule:

$$\frac{P \vee V \qquad \neg P \vee W}{V \vee W}$$

(This holds because $P \vee V$ can be written $P'\sigma \vee V'\sigma$ with $P' \vee V'$ in $\mathcal{X}'$, and $\neg P \vee W$ can be written $\neg Q'\sigma \vee W'\sigma$ with $\neg Q' \vee W' \in \mathcal{X}'$, where σ unifies P' and Q'. For τ the most general unifier of P' and Q', then $V'\tau \vee W'\tau \in \mathcal{X}'$ (as $\mathcal{X}'$ is closed under resolution). By definition of the mgu, there exists σ_0 such that $\sigma = \tau\sigma_0$. Then $V \vee W = (V'\tau \vee W'\tau)\sigma_0 \in \mathcal{N}$.)

We will construct, by induction on the set $\mathcal{N}$, a set $\mathcal{I}$ such that:

- $\mathcal{I}$ contains atoms $\mathtt{P}(s_1, \ldots, s_n)$ where all s_i are ground;
- after updating $\mathcal{I}$ for clause $U \in \mathcal{N}$: all clauses $V \in \mathcal{N}$ with $U \succeq_C V$ hold in $\mathcal{I}$;
- (invariant) for all literals L in a clause $U \in \mathcal{N}$:

  if L holds in $\mathcal{I}$ after updating $\mathcal{I}$ for U,
  then L stays true throughout the construction.

Formally, we define $[\mathtt{P}](s_1, \ldots, s_n) = \mathit{true}$ if and only if $\mathtt{P}(s_1, \ldots, s_n) \in \mathcal{I}$.

To make the idea above concrete, we define a set $\mathcal{I}_U$ for each $U \in \mathcal{N}$, as follows:

- Write $U = L_1 \vee \cdots \vee L_n$ (with $L_1 \succ_L L_2 \succ_L \cdots \succ_L L_n$).
- Let $\mathsf{Previous}_U = \bigcup\{\mathcal{I}_V \mid V \in \mathcal{N} \wedge U \succ_C V\}$.
- If some L_i is already true in $\mathsf{Previous}_U$ then $\mathcal{I}_U := \mathsf{Previous}_U$.

[1] The details for this claim are omitted, but we can see this because $\succ_C$ is essentially the multiset extension of $\succ_L$. We can also prove it directly: totality folows from a case analysis and well-foundedness because, given an infinite $\succ_C$-decreasing sequence of clauses, we can construct an infinite $\succ_L$-decreasing sequence of literals.

- Otherwise, **necessarily L_1 is an atom** (so not negated). Let $\mathcal{I}_U = \mathsf{Previous}_U \cup \{L_1\}$. (We will prove the part in bold on the next slide.)

We let $\mathcal{I} = \bigcup_{C \in \mathcal{N}} \mathcal{I}_C$.

The invariant is formally stated as follows:

(INV) if $V = L_1 \vee \cdots \vee L_n \in \mathcal{N}$ and L_i is true in $\mathcal{I}_V$, then L_i is true in $\mathcal{I}_U$ for all $U \succ_C V$.

We prove the invariant by induction on $U \in \mathcal{N}$:

- If L_i is an atom B then $B \in \mathcal{I}_V \subseteq \mathsf{Previous}_U \subseteq \mathcal{I}_U$.
- If L_i is a negated atom $\neg B$, then $B \notin \mathcal{I}_V$, so $B \notin \mathcal{I}_W$ for any $W \in \mathcal{N}$ with $V \succeq_C W$; by the induction hypothesis also $B \notin \mathcal{I}_W$ for any $W \in \mathcal{N}$ with $U \succ_C W \succ_C V$. Hence, $B \notin \mathsf{Previous}_U$.

  If $\mathcal{I}_U = \mathsf{Previous}_U$, then also $B \notin \mathcal{I}_U$.

  If $\mathcal{I}_U = \mathsf{Previous}_U \cup \{A\}$, then $U = A \vee R_1 \cdots \vee R_m$, and since $U \succ V$ we have $A \succeq_L L_1 \succeq \neg B \succ B$; hence $B \neq A$. So also $B \notin \mathcal{I}_U$.

The other two criteria listed above also hold:

- Clearly U holds in $\mathcal{I}_U$.
- If $U \succ V$ and V holds in $\mathcal{I}_V$ because $V = L_1 \vee \cdots \vee L_n$ holds and L_i holds in $\mathcal{I}_V$, then L_i still holds in $\mathcal{I}_U$ by (INV). Hence, also V still holds in $\mathcal{I}_U$.

Only one important step is missing: why can we assume that $U = L_1 \vee \cdots \vee L_n$ with L_1 positive if U is not true in $\mathsf{Previous}_P$?

First note that U cannot be $\bot$: we assumed that $\bot$ could not be obtained from $\mathcal{X}$ using resolution, so $\bot \notin \mathcal{X}'$, and therefore $\bot \notin \mathcal{N}$.

The only alternative is that $U = \neg A \vee L_2 \vee \cdots \vee L_n$. Towards a contradiction, assume that U has this form. Since U is not already true, necessarily $A \in \mathsf{Previous}_U$.

(Note that $L_2, \ldots, L_n$ cannot be A, since then U would be true in $\mathsf{Previous}_U$. Hence, $A \succ L_i$ for all i.)

We can only have $A \in \mathsf{Previous}_U$ if there exists $V = A \vee R_1 \vee \cdots \vee R_m$ with $U \succ_C V$ such that $R_1, \ldots, R_m$ do not hold in $\mathsf{Previous}_V$.

Since $\mathcal{N}$ is closed under resolution, $\mathcal{N}$ contains the clause $W := L_2 \vee \cdots \vee L_n \vee R_1 \vee \cdots \vee R_m$ (suitably reordered to respect $\succ_L$).

Since A is maximal in V and $\neg A$ in U, necessarily all L_i, R_j are smaller than A. Hence, $V \succ_C W$, so W is true in $\mathcal{I}_W$. This means some L_i or some R_j is true in $\mathcal{I}_W$.

If some R_i is true in $\mathcal{I}_W$, then by (INV) it is true in $\mathsf{Previous}_V$. If some L_i is true in $\mathcal{I}_W$, then by (INV) it is true in $\mathsf{Previous}_U$. Either way we obtain a contradiction with previous assumptions. □

2.2 An improvement to automate the resolution rule

9

Restricting resolution

An important insight from the proof is that we only used resolution in a very specific way:

- $Q = A \vee R_1 \vee \cdots \vee R_m$ with $A \succ_L R_i$ for all i
- $P = \neg A \vee L_1 \vee \cdots \vee L_n$ with $A \succ L_1 \vee \cdots \vee L_n$

This was enough to obtain refutation-completeness, which allows us to be more restrictive in how we use resolution!

Hope:

$$\frac{P \vee V \qquad \neg Q \vee W}{V\tau \vee W\tau} \quad \begin{array}{l} \tau \text{ an mgu such that } P\tau = Q\tau \\ P \succ L \text{ for all literals } L \text{ in } V \\ Q \succ L \text{ for all literals } L \text{ in } W \end{array}$$

Problem: we defined $\succ$ as a total well-founded ordering on **ground** atoms.

Solution: to handle non-ground terms, we need $\succ$ such that:

$$\text{If } P \succ Q \text{ then } P\sigma \succ Q\sigma \text{ for substitutions } \sigma.$$

10

Restricting resolution

To be precise, let $\succ$ be an ordering on predicates such that:

- $\succ$ is **stable**: if $P \succ Q$ then $P\sigma \succ Q\sigma$ for all σ;
- $\succ$ is **well-founded**: there is no infinite decreasing sequence $s_1 \succ s_2 \succ \ldots$;
- $\succ$ is **ground-total**: if P, Q are ground, then $P \succ Q$ or $Q \succ P$ or $P = Q$.

An example of such an ordering is LPO.

11

Ordered resolution

Let **ordered resolution** be given by:

$$\frac{P \vee V \qquad \neg Q \vee W}{V\tau \vee W\tau} \quad \begin{array}{l} \tau \text{ an mgu such that } P\tau = Q\tau \\ \text{for every atom } L \text{ occurring in } V\text{: } L \not\succ P \\ \text{for every atom } L \text{ occurring in } W\text{: } L \not\succ Q \end{array}$$

Theorem

(refutation-completeness of ordered resolution)

A predicate in CNF is equivalent to *false* if and only if there is a sequence of ordered resolution steps ending in the empty clause.

Ordered resolution: example

Let's try it out! Recall the boring lectures problem.

1	$\mathtt{S}(\mathtt{a})$
2	$\neg\mathtt{L}(x) \vee \mathtt{A}(\mathtt{a}, x)$
3	$\neg\mathtt{L}(y) \vee \neg\mathtt{B}(y) \vee \neg\mathtt{S}(z) \vee \neg\mathtt{A}(z, y)$
4	$\mathtt{L}(\mathtt{e})$
5	$\mathtt{B}(\mathtt{e})$

We arbitrarily set: $\mathtt{A} \rhd \mathtt{B} \rhd \mathtt{L} \rhd \mathtt{S}$.

1	$\mathtt{S}(\mathtt{a})$	
2	$\mathtt{A}(\mathtt{a}, x) \vee \neg\mathtt{L}(x)$	
3	$\neg\mathtt{A}(z, y) \vee \neg\mathtt{B}(y) \vee \neg\mathtt{L}(y) \vee \neg\mathtt{S}(z)$	
4	$\mathtt{L}(\mathtt{e})$	
5	$\mathtt{B}(\mathtt{e})$	
6	$\neg\mathtt{B}(y) \vee \neg\mathtt{L}(y) \vee \neg\mathtt{S}(\mathtt{a})$	$(2, 3)$
7	$\neg\mathtt{L}(\mathtt{e}) \vee \neg\mathtt{S}(\mathtt{a})$	$(5, 6)$
8	$\neg\mathtt{S}(\mathtt{a})$	$(4, 7)$
9	$\bot$	$(1, 8)$

3. Horn Clauses

13

Horn Clauses

Even with ordered resolution, there is often still a lot of choice when searching for a proof.

This choice is more restricted if every clause contains at most one positive literal, making search for (normal or ordered) resolution much simpler.

A **Horn clause** is a clause with at most one positive literal.

Usually a Horn clause $C \vee \neg C_1 \vee \cdots \vee \neg C_n$ is written as

$$C \leftarrow C_1, \ldots, C_n$$

or as

$$C \texttt{ :- } C_1, \ldots, C_n.$$

The positive literal C is called the **head**.

A clause without a head is called a **goal**.

A clause consisting only of a head is called a **fact**;
it is written as `C.` instead of `C :- .`

14

Resolution between Horn clauses

A resolution between two Horn clauses again yields a Horn clause.

$$\frac{P \vee \neg V_1 \vee \cdots \vee \neg V_n \qquad Q \vee \neg P' \vee \neg W_1 \vee \ldots \neg \vee W_m}{(Q \vee \neg W_1 \vee \cdots \vee \neg W_m \vee \neg V_1 \vee \ldots \neg \vee V_n)\sigma} \quad \begin{array}{c}\sigma \text{ the mgu} \\ \text{of } P, P'\end{array}$$

This format leads to much more efficient proofs.

A **Prolog program** is a set of Horn clauses in this notation.

Prolog is a standard programming language in artificial intelligence.

15

Example

```
arrow(a,b).
arrow(a,c).
arrow(b,c).
arrow(c,d).
path(X,Y) :- arrow(X,Y).
path(X,Y) :- arrow(X,Z),path(Z,Y).
```

The first four clauses define a directed graph on four nodes.

By the last two clauses the notion of a path in a graph is defined.

If we wonder whether a path exists from `a` to `d`, then we add the goal
`:- path(a,d)`
being the clause ¬`path(a,d)`.

Systematical depth first search for a resolution proof starting from the goal yields:

16

Resolution for Horn clauses

1	$\texttt{arrow}(\texttt{a},\texttt{b})$	
2	$\texttt{arrow}(\texttt{a},\texttt{c})$	
3	$\texttt{arrow}(\texttt{b},\texttt{c})$	
4	$\texttt{arrow}(\texttt{c},\texttt{d})$	
5	$\texttt{path}(x,y) \vee \neg\texttt{arrow}(x,y)$	
6	$\texttt{path}(x,y) \vee \neg\texttt{arrow}(x,z) \vee \neg\texttt{path}(z,y)$	
7	$\neg\texttt{path}(\texttt{a},\texttt{d})$	
8	$\neg\texttt{arrow}(\texttt{a},z) \vee \neg\texttt{path}(z,\texttt{d})$	$(6,7)$
9	$\neg\texttt{path}(\texttt{b},\texttt{d})$	$(1,8)$
10	$\neg\texttt{arrow}(\texttt{b},z) \vee \neg\texttt{path}(z,\texttt{d})$	$(6,9)$
11	$\neg\texttt{path}(\texttt{c},\texttt{d})$	$(3,10)$
12	$\neg\texttt{arrow}(\texttt{c},\texttt{d})$	$(5,11)$
13	$\bot$	$(4,12)$

This kind of search for a resolution proof is called **SLD-resolution**.

17

Ordered resolution for Horn clauses

Choose: $\texttt{path} \rhd \texttt{arrow}$:

1	$\texttt{arrow}(\texttt{a},\texttt{b})$	
2	$\texttt{arrow}(\texttt{a},\texttt{c})$	
3	$\texttt{arrow}(\texttt{b},\texttt{c})$	
4	$\texttt{arrow}(\texttt{c},\texttt{d})$	
5	$\texttt{path}(x,y) \vee \neg\texttt{arrow}(x,y)$	
6	$\texttt{path}(x,y) \vee \neg\texttt{path}(z,y) \vee \neg\texttt{arrow}(x,z)$	
7	$\neg\texttt{path}(\texttt{a},\texttt{d})$	
8	$\neg\texttt{path}(z,\texttt{d}) \vee \neg\texttt{arrow}(\texttt{a},z)$	$(6,7)$
9	$\neg\texttt{arrow}(z,\texttt{d}) \vee \neg\texttt{arrow}(\texttt{a},z)$	$(5,8)$
10	$\neg\texttt{arrow}(\texttt{a},\texttt{c})$	$(4,9)$
11	$\bot$	$(2,10)$

The found resolution sequence ending in $\bot$ is called a **refutation**.

In the example it was proved automatically that a path from a to d exists, while the input is nothing more than

- the definition of a graph
- the definition of the notion 'path'
- the question: 'is there a path from a to d?'

18

Prolog

This mechanism also applies for goals containing variables.

For instance,
`:- path(a,X)`
will yield a refutation, but
`:- path(d,X)`
will not.

This is quite subtle: sometimes search can go on forever.

Prolog is a **declarative programming language**, it is a kind of **logic programming**.

In many Prolog implementations occur check is omitted for efficiency reasons, by which
`p(X,X).`
`:- p(X,f(X)).`
yielding the CNF $\mathtt{p}(X,X) \wedge \neg \mathtt{p}(X,\mathtt{f}(X))$ gives rise to an infinite computation.

19

Prover9

The Prolog graph example can also be done by `Prover9`:

```
formulas(assumptions).
arrow(a,b).
arrow(a,c).
arrow(b,c).
arrow(c,d).
arrow(x,y) -> path(x,y).
(arrow(x,z) & path(z,y)) -> path(x,y).
end_of_list.
formulas(goals).
path(a,d).
end_of_list.
```

By default, in `Prover9` x, y, z, u, v are universally quantified variables, and other names are constants, by which declarations may be omitted.

4. Equational Logic

4.1 Definition

20

We have hinted at it before, but never properly defined it: one critical feature is still missing from our discussions of predicate logic:

equality

In most applications of predicate logic, it is desirable to have an equality symbol.

- Reasoning over natural numbers:

$$\forall x[\forall y[\mathtt{suc}(x) = \mathtt{suc}(y) \rightarrow x = y]]$$

- Our students / lectures example:

$$\exists x[\exists y[x \neq y \wedge \mathtt{Favourite}(x) = \mathtt{AR} \wedge \mathtt{Favourite}(y) = \mathtt{AR}]]$$

21

Can we already do this?

$$\forall x[\mathtt{EQ}(x, x)]$$

$$\forall x \forall y[\mathtt{EQ}(x, y) \rightarrow \mathtt{EQ}(y, x)]$$

$$\forall x \forall y \forall z[\mathtt{EQ}(x, y) \wedge \mathtt{EQ}(y, z) \rightarrow \mathtt{EQ}(x, z)]$$

$$\exists x[\exists y[\neg \mathtt{EQ}(x, y) \wedge \mathtt{EQ}(\mathtt{Favourite}(x), \mathtt{AR}) \wedge \mathtt{EQ}(\mathtt{Favourite}(y), \mathtt{AR})]]$$

This is not sufficient! The following formula is satisfiable:

$$\mathtt{P}(\mathtt{Alice}) \wedge \neg \mathtt{P}(\mathtt{Bob}) \wedge \mathtt{EQ}(\mathtt{Alice}, \mathtt{Bob})$$

And also:

$$\exists x[\mathtt{EQ}(x, \mathtt{Alice}) \wedge \neg \mathtt{EQ}(\mathtt{Favourite}(x), \mathtt{Favourite}(\mathtt{Alice}))]$$

22

So to make this work, we will need a bit more:

$$\forall x[\texttt{EQ}(x,x)]$$

$$\forall x \forall y[\texttt{EQ}(x,y) \rightarrow \texttt{EQ}(y,x)]$$

$$\forall x \forall y \forall z[\texttt{EQ}(x,y) \wedge \texttt{EQ}(y,z) \rightarrow \texttt{EQ}(x,z)]$$

For all predicates $\texttt{P}$ of arity n and all $i \in \{1,\ldots,n\}$:

$$\forall x_1 \ldots \forall x_n \forall y_i$$
$$[\texttt{EQ}(x_i,y_i) \rightarrow (\texttt{P}(x_1,\ldots,x_i,\ldots,x_n) \leftrightarrow \texttt{P}(x_1,\ldots,y_i,\ldots,x_n))]$$

For all functions $\texttt{f}$ of arity n and all $i \in \{1,\ldots,n\}$:

$$\forall x_1 \ldots \forall x_n \forall y_i$$
$$[\texttt{EQ}(x_i,y_i) \rightarrow \texttt{EQ}(\texttt{f}(x_1,\ldots,x_i,\ldots,x_n),\texttt{f}(x_1,\ldots,y_i,\ldots,x_n)]$$

23

Supporting equality directly

It is much easier for tools to support equality directly.

(For example Prover9 already does this!)

Recall: formula P in predicate logic is true in model $(M,[\![\cdot]\!]_\alpha)$ if $[\![P]\!]_\alpha = \mathit{true}$.

To define truth in the extension, we simply add:

$$[\![s=t]\!]_\alpha = \mathit{true} \text{ if and only if } [\![s]\!]_\alpha = [\![t]\!]_\alpha$$

The transformation to Prenex normal form and Skolemization stay exactly the same.

24

Question: Do we need predicates other than $=$?

Answer: No! Replace

$$\texttt{P}(s_1,\ldots,s_n)$$

by

$$\texttt{P}(s_1,\ldots,s_n) = \mathit{true}$$

Hence we arrive at **equational logic**:

> Formulas built from $\wedge, \vee, \neg, \rightarrow, \leftrightarrow$ and atoms $s = t$,
> where s and t are terms (that may contain variables).

Variables are implicitly universally quantified.

Functions (and constants) are implicitly existentially qualified.

We have seen that predicate logic can be reduced to equational logic.

4.2 Equational logic and term rewriting

25

Term rewriting

Rules can be seen as **oriented equations**.

The rules

$$\begin{array}{rcl} \mathtt{add}(\mathtt{0}, y) & \to & y \\ \mathtt{add}(\mathtt{s}(x), y) & \to & \mathtt{s}(\mathtt{add}(x, y)) \end{array}$$

define the equations:

$$\begin{array}{rrcl} \forall y. & \mathtt{add}(\mathtt{0}, y) & = & y \\ \forall x \forall y. & \mathtt{add}(\mathtt{s}(x), y) & = & \mathtt{s}(\mathtt{add}(x, y)) \end{array}$$

26

Term rewriting

Theorem

Given equations $\mathcal{E} = \{\ell_1 = r_1, \ldots, \ldots, \ell_n = r_2\}$
And rules $\mathcal{R} = \{\ell_1 \to r_1, \ldots, \ell_n \to r_n\}$
For any model $(\mathcal{M}, [\![\cdot]\!]_\alpha)$ with $\mathcal{M} \Vdash \ell_i = r_i$ for all i:
If $s \to t$ then $[\![s]\!]_\alpha = [\![t]\!]_\alpha$ for all α.

Proof: By induction on the size of s.

Corollary: if $s \leftrightarrow_\mathcal{R} t$, then $s = t$ follows from equations in $\mathcal{E}$.

27

Completion

Question:

> Given equations $\mathcal{E} = \{s_1 = t_1, \ldots, s_n = t_n\}$,
> And given another equation $u = v$.
> Does $u = v$ follow from $\mathcal{E}$?

(For example: if $\mathcal{E}$ contains the axioms for addition, does it follow that always $\mathtt{add}(x, y) = \mathtt{add}(y, x)$?)

Method:

- Use completion to find a **terminating**, **confluent** TRS $\mathcal{R}$ such that $\leftrightarrow_\mathcal{R}$ is exactly $=_\mathcal{E}$.
- Then see if $u \downarrow_\mathcal{R}$ and $v \downarrow_\mathcal{R}$ are the same! (That is, the unique normal forms of u and v.)

5. Superposition

28

The general case

Knuth-Bendix Completion provides a way to prove:

> If $s_1 = t_1$ and ... and $s_n = t_n$
> then also $u = v$.

In mathematics, we often encounter problems of this form.

But to prove validity in equational logic in general, we need:

> If φ_1 and ... and φ_n
> then also $u = v$.

where all φ_i are **clauses** $L_1 \vee \cdots \vee L_k$ with each L_i either an equality $s = t$ or an inequality $s \neq t$.

Put differently: we should be able to prove unsatisfiablity of formulas:

$$\varphi_1 \wedge \cdots \wedge \varphi_n \wedge \psi$$

where ψ and all φ_i are clauses.

Using Skolemization and translating to Prenex normal form, this would let us prove validity of any (true) formula.

29

Resolution?

For problems of this nature, we would normally use **resolution**.

However, equality comes with some implicit axioms that should be considered.

To use resolution we would have to include clauses like:

$$x = x$$

$$x \neq y \vee y = x$$

$$x \neq y \vee y \neq z \vee x = z$$

and

$$x_i \neq y_i \vee \mathtt{f}(x_1, \ldots, x_i, \ldots, x_n) = \mathtt{f}(x_1, \ldots, y_i, \ldots, x_n)$$

for all $\mathtt{f}$ and argument positions i.

As observed before, it would likely be more effective to use dedicated methods for equational logic!

5.1 Intuition

30

Goal: a form of resolution for equational logic!

What should hold? Let us consider *ground* equations:

$$\frac{s = t \vee V \qquad s \neq t \vee W}{V \vee W}$$

But we'll also need:

$$\frac{s = t \vee V \qquad \mathtt{f}(\dots, s, \dots) \neq \mathtt{f}(\dots, t, \dots) \vee W}{V \vee W}$$

We can combine this through a *context*: a term with a hole $\Box$.

So resolution could become something like:

$$\frac{s = t \vee V \qquad C[s] \neq C[t] \vee W}{V \vee W}$$

For all contexts C.

31

Reflexivity and symmetry

Reflexivity:

$$\frac{s \neq s \vee V}{V}$$

Symmetry:

$$\frac{s = t \vee V}{t = s \vee V} \qquad \frac{s \neq t \vee V}{t \neq s \vee V}$$

Better: we just agree that we can freely swap order when applying rules: $s = t$ is considered the same as $t = s$, and $s \neq t$ the same as $t \neq s$.

32

Transitivity

We should be able to derive:

$$\frac{s = t \vee U \qquad t = q \vee V \qquad s \neq q \vee W}{U \vee V \vee W}$$

And also:

$$\frac{s = t \vee V_1 \qquad \mathtt{f}(t) = \mathtt{f}(q) \vee V_2 \qquad \mathtt{f}(q) = \mathtt{f}(u) \vee V_3 \qquad \mathtt{g}(\mathtt{f}(s), x) \neq \mathtt{g}(\mathtt{f}(u), x) \vee W}{V_1 \vee V_2 \vee V_3 \vee W}$$

And also:

$$\frac{s = t \vee U \qquad u = v \vee V \qquad \mathtt{f}(s, u) \neq \mathtt{f}(t, v) \vee W}{U \vee V \vee W}$$

Idea:

$$\frac{s = t \vee V \qquad C[s] = q \vee W}{C[t] = q \vee V \vee W} \qquad \frac{s = t \vee V \qquad C[s] \neq q \vee W}{C[t] \neq q \vee V \vee W}$$

33

Observation: we don't need resolution anymore!

$$\frac{s = t \vee V \qquad C[s] \neq C[t] \vee W}{V \vee W}$$

This can be derived in two steps by:

$$\frac{s = t \vee V \qquad C[s] \neq q \vee W}{C[t] \neq q \vee V \vee W} \qquad \frac{q \neq q \vee U}{U}$$

For $q = C[t]$ and $U = V \vee W$.

34

Equality factoring

It turns out we need one more rule (if we want refutation-completeness, which we do):

$$\frac{s = t \vee s = u \vee V}{s = t \vee t \neq u \vee V}$$

This holds because if the premise holds then either:

- $s = t$ holds; then the consequent also holds;
- $s = t$ does not hold but $s = u$ holds: then $t = u$ cannot hold (as this would imply $s = t$);
- neither of the equalities holds but V holds; then the consequent also holds.

35

Overview (for ground equations)

In summary, we arrive at the following rules for the **superposition calculus** on ground equations:

Equality resolution:

$$\frac{s \neq s \vee V}{V}$$

Positive superposition

$$\frac{s = t \vee V \qquad C[s] = q \vee W}{C[t] = q \vee V \vee W}$$

Negative superposition

$$\frac{s = t \vee V \qquad C[s] \neq q \vee W}{C[t] \neq q \vee V \vee W}$$

Equality factoring

$$\frac{s = t \vee s = u \vee V}{s = t \vee t \neq u \vee V}$$

where the two components of an equality or inequality can be reordered, just like the literals in a clause can be reordered.

5.2 Full definition

36

Superposition calculus for general equations

Considering general equations rather than merely ground equations, the superposition calculus is given by:

Equality resolution:

$$\frac{s \neq t \vee V}{V\sigma}\ \sigma = mgu(s,t)$$

Positive superposition

$$\frac{s = t \vee V \qquad C[u] = q \vee W}{(C[t] = q \vee V \vee W)\sigma}\quad \begin{array}{l}\sigma = mgu(s,u) \text{ and} \\ u \text{ is not a variable}\end{array}$$

Negative superposition

$$\frac{s = t \vee V \qquad C[u] \neq q \vee W}{(C[t] \neq q \vee V \vee W)\sigma}\quad \begin{array}{l}\sigma = mgu(s,u) \text{ and} \\ u \text{ is not a variable}\end{array}$$

Equality factoring

$$\frac{s = t \vee q = u \vee V}{(s = t \vee t \neq u \vee V)\sigma}\ \sigma = mgu(s, q)$$

37

Example:

We want to prove:

- If $\forall x, y[\texttt{leq}(x, y) \vee \texttt{leq}(y, x)]$
- and $\forall x, y[\texttt{leq}(x, y) \rightarrow \texttt{max}(x, y) = y]$
- and $\forall x, y[\texttt{leq}(y, x) \rightarrow \texttt{max}(x, y) = x]$
- and $\forall x, y, z[\texttt{max}(\texttt{max}(x, y), z) = \texttt{max}(x, \texttt{max}(y, z))]$
- then $\forall x, y, z[\texttt{leq}(x, y) \wedge \texttt{leq}(y, z) \rightarrow \texttt{leq}(x, z)]$

To do this, we will try to refute the **negation** of the above implication.

If we replace predicate symbols by function symbols, and translate implications $a \rightarrow b$ by $\neg a \vee b$, this exactly gives:

$$\begin{aligned}
\forall x, y[\texttt{leq}(x, y) = \texttt{true} \vee \texttt{leq}(y, x) = \texttt{true}] \quad & \wedge \\
\forall x, y[\texttt{leq}(x, y) \neq \texttt{true} \vee \texttt{max}(x, y) = y] \quad & \wedge \\
\forall x, y[\texttt{leq}(x, y) \neq \texttt{true} \vee \texttt{max}(x, y) = x] \quad & \wedge \\
\forall x, y, z[\texttt{max}(\texttt{max}(x, y), z) = \texttt{max}(x, \texttt{max}(y, z))] \quad & \wedge \\
\exists x, y, z[\texttt{leq}(x, y) = \texttt{true} \wedge \texttt{leq}(y, z) = \texttt{true} \wedge \texttt{leq}(y, z) \neq \texttt{true}] &
\end{aligned}$$

After Skolemization, we obtain a CNF which we can do superposition on!

38

1	$\mathtt{leq}(x, y) = \mathtt{true} \vee \mathtt{leq}(y, x) = \mathtt{true}$	
2	$\mathtt{leq}(x, y) \neq \mathtt{true} \vee \mathtt{max}(x, y) = y$	
3	$\mathtt{leq}(y, x) \neq \mathtt{true} \vee \mathtt{max}(x, y) = x$	
4	$\mathtt{max}(\mathtt{max}(x, y), z) = \mathtt{max}(x, \mathtt{max}(y, z))$	
5	$\mathtt{leq}(\mathtt{a}, \mathtt{b}) = \mathtt{true}$	
6	$\mathtt{leq}(\mathtt{b}, \mathtt{c}) = \mathtt{true}$	
7	$\mathtt{leq}(\mathtt{a}, \mathtt{c}) \neq \mathtt{true}$	
8	$\mathtt{true} \neq \mathtt{true} \vee \mathtt{max}(\mathtt{a}, \mathtt{b}) = \mathtt{b}$	$(5, 2, \text{neg.sup})$
9	$\mathtt{max}(\mathtt{a}, \mathtt{b}) = \mathtt{b}$	$(9, \text{eq.res})$
10	$\mathtt{true} \neq \mathtt{true} \vee \mathtt{max}(\mathtt{b}, \mathtt{c}) = \mathtt{c}$	$(6, 2, \text{neg.sup})$
11	$\mathtt{max}(\mathtt{b}, \mathtt{c}) = \mathtt{c}$	$(10, \text{eq.res})$
12	$\mathtt{true} \neq \mathtt{true} \vee \mathtt{leq}(x, y) = \mathtt{true} \vee \mathtt{max}(x, y) = x$	$(1, 3, \text{neg.sup})$
13	$\mathtt{leq}(x, y) = \mathtt{true} \vee \mathtt{max}(x, y) = x$	$(12, \text{eq.res})$
14	$\mathtt{true} \neq \mathtt{true} \vee \mathtt{max}(\mathtt{a}, \mathtt{c}) = \mathtt{a}$	$(13, 7, \text{neg.sup})$
15	$\mathtt{max}(\mathtt{a}, \mathtt{c}) = \mathtt{a}$	$(14, \text{equ.res})$
16	$\mathtt{max}(\mathtt{b}, z) = \mathtt{max}(\mathtt{a}, \mathtt{max}(\mathtt{b}, z))$	$(9, 4, \text{pos.sup})$
17	$\mathtt{max}(\mathtt{b}, \mathtt{c}) = \mathtt{max}(\mathtt{a}, \mathtt{c})$	$(11, 16, \text{pos.sup})$
18	$\mathtt{c} = \mathtt{max}(\mathtt{a}, \mathtt{c})$	$(11, 17, \text{pos.sup})$
19	$\mathtt{c} = \mathtt{a}$	$(15, 18, \text{pos.sup})$
20	$\mathtt{leq}(\mathtt{a}, \mathtt{a}) \neq \mathtt{true}$	$(19, 7, \text{neg.sup})$
21	$\mathtt{true} \neq \mathtt{true} \vee \mathtt{leq}(\mathtt{a}, \mathtt{a}) = \mathtt{true}$	$(1, 20, \text{neg.sup})$
22	$\mathtt{true} \neq \mathtt{true} \vee \mathtt{true} \neq \mathtt{true}$	$(20, 21, \text{neg.sup})$
23	$\bot$	$(22, \text{eq.res})$

39

Combining steps

Note that while superposition is **sufficient**, it is in practice **useful** to also have resolution as shorthand for negative superposition + equality resolution.

$$\frac{s = t \vee V \qquad C[u] \neq q \vee W}{(V \vee W)\sigma} \quad \begin{array}{l}\sigma = mgu(s, u) \text{ and} \\ u \text{ is not a variable} \\ \text{and } C[t]\sigma = q\sigma\end{array}$$

However, superposition is not typically done by hand so this is not really an issue.

5.3 Ordered superposition

40

Result

Theorem

refutation-completeness of superposition

A CNF of equational clauses is equivalent to *false* if and only if there is a sequence of superposition steps ending in the empty clause.

To give a broad idea of the proof, recall the proof idea of **resolution**. For the difficult step ($\Rightarrow$):

41

Resolution proof idea

- We assume that there is no such sequence, and generate a model that makes all clauses true.
- $\mathcal{M}$ will be a set of base predicates $\mathtt{P}(s_1, \ldots, s_n)$.
- We use a *ground total*, *well-founded* ordering $\succ$ on ground predicates $\mathtt{P}(s_1, \ldots, s_n)$. This is extended to an ordering on literals and clauses.
- We use induction on clauses using $\succ$:

  We may mark $\mathtt{P}(s_1, \ldots, s_n)$ as true when $\mathtt{P}(s)$ is the largest atom in a clause that is not already true

- By construction, every atom that is marked as true is $\succ$-larger than the atoms that were marked before.
- Because the set of clauses we consider is closed under the resolution rule, we do not have to worry about literals $\neg\mathtt{P}(s)$.

42

Superposition proof idea

- We assume that there is no such sequence, and generate a model that makes all clauses true.
- This model will be defined by a **confluent and terminating TRS** $\mathcal{R}$:
  - the set $\mathcal{M}$ consists of the normal forms of $\mathcal{R}$;
  - for ground terms s, t, $[\![s = t]\!]$ holds if and only if s and t have the same normal form.

- to ensure this, rules will contain:
  - $\mathcal{R}$ will contain ground rules $s \to t$ with $s \succ t$
  - there will be no critical pairs between any rule

- To build this model, we use a *ground total*, *well-founded* ordering $\succ$ on ground **terms** s, which is also **monotonic** and has **the subterm property**. (For example: LPO!)

43

Superposition proof idea (continued)

- $\succ$ is extended to an ordering on literals $s = t$ or $s \neq t$, and to clauses.

- Then by induction on clauses using $\succ$ we stepwise add ground rules to R, as follows:

  when we have to make $s = t \vee V_1 \cdots \vee V_n$ true with $s = t$ the maximum literal in the clause, and $s \succ t$

  and the clause is not already true

  then we add a rule $s \to t$ to the set $\mathcal{R}$

- By construction, both s and t are in normal form with respect to $\mathcal{R}$ when $s \to t$ is added. Because $\succ$ has the subterm property, adding $s \to t$ does not cause extra critical pairs.

- Because the set of clauses we consider is closed under the superposition rules, we do not have to worry about literals $s \neq t$.

44

Ordered superposition

As a consequence, we can limit interest to **ordered superposition**:

- when using $s = t \vee V$ or $s \neq t \vee V$ in any of the rules except equality resolution, $t \not\succeq s$;
- when we consider a clause $s = t \vee V$ in one of the superposition rules, there is no literal L in V with $L \succ_L s = t$;
- when we consider a clause $s \neq t \vee V$ in one of the superposition rules, there is no literal L in V with $L \succeq_L s \neq t$;
- when using $s = t \vee V$ and $C[u] = q \vee W$ in positive superposition to conclude $(C[t] = q \vee V \vee W)\sigma$, not $s = t \succeq_L C[u] = q$;
- when using $s = t \vee V$ and $C[u] = q \vee W$ in negative superposition to conclude $(C[t] = q \vee V \vee W)\sigma$, not $s = t \succeq_L C[u] = q$.

6. Functional programs

45

Functional program analysis

An important application area for term rewriting is program analysis.

In particular for functional programming languages, rewriting is a natural match.

For instance the following Ocaml code:

```
let rec append a b =
    match a with
        | [] -> b
        | h ::  t -> h ::  append t b;;
```

corresponds exactly to the TRS rules:

$$\begin{array}{rcl} \mathtt{append}(\mathtt{nil}, b) & \to & b \\ \mathtt{append}(\mathtt{cons}(h, t)) & \to & \mathtt{cons}(h, \mathtt{append}(t, b)) \end{array}$$

46

Struggles of functional program analysis

However, functional languages have features which do not appear in term rewriting:

- integers, floating point numbers, etc.
- higher-order functions (i.e., where a function can be passed as an argument)
- side effects such as storing values in a global variable
- ...

47

Analysing functional programs anyway:

Hence, program analysis using rewriting can esentially be done in three ways:

- **translate** functional programs to TRSs, while retaining some desirable properties

```
let rec length l =
    match l with
        | [] -> 0
```

```
        | _ ::  t -> 1 + length t;;
```

becomes:

$$\begin{aligned} \texttt{length}(\texttt{nil}) &\to \texttt{0} \\ \texttt{length}(\texttt{cons}(x,t)) &\to \texttt{s}(\texttt{length}(t)) \end{aligned}$$

- **transpose** methods from term rewriting to functional programs

  Ideas such as dependency pairs and montonic algebras can be defined directly for OCaml programs, and critical pairs are irrelevant as OCaml rules are non-overlapping by their nature.

- define **extensions** of basic TRSs, transpose methods from term rewriting to them, and translate functional programs to these more fitting extensions

$$\begin{aligned} \texttt{length}(\texttt{nil}) &\to \texttt{0} \\ \texttt{length}(\texttt{cons}(x,t)) &\to \texttt{length}(t) + 1 \end{aligned}$$

This is an active research area, with the last approach gaining in popularity.

7. Quiz

48

Quiz

1. In ordered resolution, we require that $L \not\succ P$. Why do we not just require $P \succeq L$?
2. What is a Horn clause?
3. Why is it more efficient to use resolution with Horn clauses than in general?
4. Why do we need equational logic when we already have predicate logic?
5. Give the positive superposition and negative superposition rule, and an example of how they might be applied (just one step; no need to give a full superposition proof).