Lecture 5

Herbrand Models,

Unification, Resolution

Overview

Where are we with classical logic?

- Repitition of FOL, models, interpretation, satisfiability
- We learned a proof method called natural deduction
- We saw that it is hard to automatize…
(completeness proof)

Today:

- Automated Theorem proving, Logic Programming
- Herbrand Interpretations & Herbrand Models
- An automatic proof method: Resolution
  - Transformation to clausal form
  - Substitutions
  - Unification
  - Resolution

Motivation: We want to build an <u>automatic</u> proof procedure…

- For proofing a goal G from a set of formula S in a special normal form called clausal form

- We will see, that we can do this by showing:

  $S \cup \{\neg G\}$ is unsatisfiable.

- Problem: remember, for doing this automatically, we would need to consider ALL possible interpretations

- Furthermore: If G or S are not in clausal form, we need to transform them to clausal form.

Automated theorem proving?

- Imagine a (set of) logical formula(e):
  - Program clauses:

$$\begin{aligned}\phi = & father(Sepp, Hans) \wedge \\ & father(Hans, Karl) \wedge \\ & \forall x \forall y \forall z \ (grandpa(x,y) \leftarrow father(x,z) \wedge father(z,y))\end{aligned}$$

  - Goal:

$$\psi = \exists x \ \ grandpa(sepp, x)$$

  - Finding a solution for the question
  - “Who is Sepp’s grandpa?” amounts to finding a proof for:

$$\phi \models \psi$$

But: Usual proof procedures do not return the substitutions, we need slight modifications.

Clauses

- **Universally closed disjunctions of literals** are called **clauses**:

$$\forall x_1 \ldots \forall x_n (L_1 \vee \ldots \vee L_m)$$

(no other variables except $x_1,\ldots,x_n$ occur!)

- Special notation for clauses: The clause

$$\forall x_1 \ldots \forall x_n (A_1 \vee \ldots \vee A_k \vee \neg B_1 \vee \ldots \vee \neg B_l)$$

(where A_i, B_j are atoms) is written:

$$\underbrace{A_1, \ldots, A_k}_{\text{head}} \leftarrow \underbrace{B_1, \ldots, B_l}_{\text{body}}$$

- universal quantification is implicit in this notation
- *","* in the head stands for disjunction
- *","* in the body stands for conjunction

We will deal a lot with formulae in clausal form!

Clauses and Logic Programming:

- Clauses with exactly one positive literal (i.e. one head literal) are called definite clauses.
- Clauses with at most one positive literal (i.e. at most one head literal) are called **Horn clauses**.
- A **logic program** is a set of clauses.
- A **Horn program** (or definite logic program) is a program which consists only of Horn clauses (or only definite clauses).
- A clause with an empty body

  $$A \leftarrow$$

  is also called **unit clause** (or **fact**).
- A clause with an empty head (this is also a Horn clause, but not definite!)

  $$\leftarrow B_1, \ldots, B_n$$

  is also called **goal** (sometimes also "constraint", or "query").
- The empty clause, written $\emptyset$, stands for contradiction.

Logic Programming, what is this?

- Basically based on Kowalsky, Colmerauer, 1972:
- "*Logic can be used as a programming Language*" by interpreting clauses of the form

$$A \leftarrow B_1, B_2, \ldots, B_n$$

as "procedures"… how?

Logic Programming in one slide:

- A logic program is a set of clauses ($\approx$ rules):

$$\underbrace{A}_{\text{head}} \leftarrow \underbrace{B_1, \ldots, B_n}_{\text{body}}$$

- A program run is given an initial goal:

$$\leftarrow C_1, C_2, \ldots, C_m$$

- If the current goal is $\leftarrow C_1, C_2, \ldots, C_m$ a computation step involves
  - unifying some C_i with the head A of some clause $A \leftarrow B_1, B_2, \ldots, B_n$ and
  - Reducing the current goal to

    $$\leftarrow (C_1, \ldots, C_{i-1}, B_1, B_2, \ldots, B_n, C_{j+1}, \ldots, C_m)\theta$$

    where θ is the unifying substitution.
  - The computation ends when the empty goal is produced.

- Variable substitutions for the variables in the original goal mark "solutions".

A proof for our simple example:

Program clauses:

c_1: *father(sepp,hans)* $\leftarrow$

c_2: *father(hans,franz)* $\leftarrow$

c_3: *grandpa*$(x_1,y_1) \leftarrow$ *father*(x_1,z_1),*father*(z_1,y_1)

Goal: $\leftarrow$ *grandpa(sepp,* x*)*

	$\leftarrow$ *grandpa(sepp,* x*)*	$\theta = \{\}$
c_3	$\leftarrow$ *father(sepp,* z_1*)*,*father*(z_1,x)	$\theta = \{x_1/sepp\ ,\ y_1/x\}$
c_1	$\leftarrow$ *father(hans,* x*)*	$\theta = \{z_1/hans\}$
c_2	$\leftarrow$	$\theta = \{x/franz\}$

Solution: franz

Motivation: We want to build an <u>automatic</u> proof procedure…

- For proofing a **goal** G from a **set of formula S** in a special normal form called clausal form

- **We will see, that we can do this by showing:**

  $$S \cup \{\neg G\} \text{ is unsatisfiable.}$$

- Problem: remember, for doing this automatically, we would need to consider ALL possible interpretations

- Furthermore: If G or S are not in clausal form, we need to transform them to clausal form.

Observation:

- The notions of interpretations, models, satifiability and validity, discussed in Lecture 2 can be expanded to sets of (closed) formulae (i.e. to sets of clauses) straightforwardly:

- A set of closed formulae $S = \{F_1, \ldots, F_n\}$ is then simply viewed as the conjunction

$$F_1 \wedge \ldots \wedge F_n$$

Crucial Idea behind the evaluation of Logic Programs: Proof by refutation!

in our case: a set of clauses…

- **Proposition 1:** Let S be a set of closed formulae. Then F is a logical consequence of S iff $S \cup \{\neg F\}$ is unsatisfiable, i.e.

  $S \models F$ iff $S \cup \{\neg F\}$ has no model.

- Recall the example from the beginning:

$$\begin{aligned}\phi = & father(sepp, hans) \wedge \\ & father(hans, karl) \wedge \\ & \forall x \forall y \forall z \; grandpa(x, y) \leftarrow father(x, z) \wedge father(z, y)\end{aligned}$$

$$\psi = \exists x \; grandpa(sepp, x)$$

$$\phi \models \psi \quad ?$$

show that:

$$\phi \cup \forall x \neg grandpa(sepp, x)$$

has no model.

$\leftarrow$ *grandpa(sepp,x)*

Proof of Proposition 1:

$S \models F$ iff $S \cup \{\neg F\}$ is unsatisfiable.

$\Rightarrow$ Suppose F is a logical consequence of S. Now let $\mathcal{I}$ be an interpretation and suppose $\mathcal{I}$ is a model for S. Then $\mathcal{I}$ is also a model for F. Hence, $\mathcal{I}$ is not a model of $S \cup \{\neg F\}$. Therefore, no model for $S \cup \{\neg F\}$ can exist and thus $S \cup \{\neg F\}$ is unsatisfiable.

$\Leftarrow$ Suppose $S \cup \{\neg F\}$ is unsatisfiable. Now let $\mathcal{I}$ be any interpretation. Suppose $\mathcal{I}$ is a model for S, then, since $S \cup \{\neg F\}$ is unsatisfiable it cannot be a model for $\neg F$. Thus, $\mathcal{I}$ is a model for F and therefore F is a logical consequence of S $\square$

Problem:

- Showing unsatisfiability is not easy:

  We have to consider EVERY possible interpretation!

- However, it turns out that for showing unsatisfiability,we only have to consider a subset of all possible interpretations: So-called Herbrand Interpretations

Motivation: We want to build an <u>automatic</u> proof procedure…

- For proofing a goal G from a set of formula S in a special normal form called clausal form

- We will see, that we can do this by showing:

  $S \cup \{\neg G\}$ is unsatisfiable.

- **Problem: remember, for doing this automatically, we would need to consider ALL possible interpretations**

… or can we restrict ourselves to special interpretations?

- Furthermore: If G or S are not in clausal form, we need to transform them to clausal form.

Jacques Herbrand

(1908-1931)

- Idea: only consider interpretations where each constant and each ground term is interpreted as itself…

Ground Terms, ground atoms and the Herbrand Universe

- terms not containing any variables ar called *ground terms*
- Similarly, a ground atom is an atom not containing variables

For a first order language (alphabet) L, the Herbrand Universe U_L is the set of all ground terms which can be formed out of the constants and function symbols appearing in L.

For a first order language (alphabet) L, the Herbrand Base B_L is the set of all ground atoms which can be build from the predicate symbols in L using ground terms from U_L as arguments.

Example:

- Language/alphabet *L*:

  Function symbols and constants: f/2, g/1,a/0.

  Predicate symbols: p/1, q/2, variable symbols *x*,*y*,*z*.

  How does U_L *look like?*

  How does B_L *look like?*

Herbrand Interpretation

- Recall: interpretation consists of domain D, assignments to D for each constant, mappings for each function symbol and relations for each predicate symbol.

- A Herbrand Interpretation for a FO language L is an interpretation as follows:
  - The domain of a Herbrand interpretation is the Herbrand Universe U_L
  - Constants are assigned themselves
  - If f is an n-ary function symbol (n>0) in L then then the mapping from $(U_L)^n$ to U_L defined by
    $(t_1, \ldots, t_n) \rightarrow f(t_1, \ldots, t_n)$ is assigned to f

- Basically, **each Herbrand Interpretation $\mathcal{I}$ can be viewed as a subset of the Herbrand base** , where each predicate symbol p is interpreted as the subset of $\mathcal{I}$ corresponding to p.

- **Herbrand Model**s are defined anaolgously to normal Models for a set S of closed Formulae, i.e. a Herbrand interpretation which is a model is called Herbrand Model

For unsatisfiability it is enough to consider Herbrand interpretations:

Proposition 2: *Let S be a set of clauses and suppose S has a model. Then S has a Herbrand model.*

- ***Proof:*** *Let $\mathcal{I}$ be an interpretation of S. We define a Herbrand interpretation $\mathcal{I}'$ of S as follows:*

  $\mathcal{I}' = \{ p(t_1, \ldots, t_n) \in B_S \mid p(t_1, \ldots, t_n) \text{ is true wrt } \mathcal{I}\}$

  It is obvious that if $\mathcal{I}$ is a model, so is $\mathcal{I}'$ $\square$

From this and Proposition 1 it follows immediately that:

Proposition 3: *A set of clauses S is unsatisfiable iff S has no Herbrand models*

Attention!

- Note that this only follows if the formulae in S are clauses!!

Example: $S = \{p(a), \exists x \neg p(x)\}$

Has clearly a model, e.g. take D *the natural numbers* p *for "odd" and 1 assigned to* a.

However, the only Herbrand interpretations are $\emptyset$ and {p(a)}, but neither is a model.

We'll hear more about this when we speak about skolemization…

Logical equivalence vs. equi-satisfiability

- We say two formulae F and G are logically equivalent iff $(\forall)\ F \leftrightarrow G$ is valid.
- In other words: Two formulae are logically equivalent iff they have the same models.
- We say that two (sets of) formulae F and G are equi-satisfiable, written $F \sim_e G$, iff whenever F has a model then also G has a model.
- Our goal: Given a set S of arbitrary closed FO formulae, construct an equi-satisfiable set of clauses.
- Remember: Since we want to prove UNSATISFIABILITY, this is sufficient.

Note: It is not always possible to create a logically equivalent set of clauses:

- Example: $S = \{p(a), \exists x\ \neg p(x)$

  $S' = \{p(a), \neg p(f(a))\}$

  But: $S \sim_e S'$

Motivation: We want to build an <u>automatic</u> proof procedure…

- For proofing a goal G from a set of formula S in a special normal form called clausal form

- We will see, that we can do this by showing:

  $S \cup \{\neg G\}$ is unsatisfiable.

- For doing this automatically, we need to consider Herbrand interpretations

- **If G or S are not in clausal form, we need to transform them to** equi-satisfiable **clausal form.**

Transformation to clausal form

- Given a closed Formula F:*
- **Step 1:** Transform F in a (logically equivalent) form F' where
  - variables are renamed to avoid ambiguities
  - $\neg$ occurs only in front of atoms
  - $\leftarrow$, $\rightarrow$ and $\leftrightarrow$ are eliminated
- **Step 2:** Transform F' into an equisatisfiable form F'', eliminating all $\exists$-quantifiers by introducing constant and function symbols (Skolemization)
- **Step 3:** Transform F'' into quantor-free conjunctive normal form, i.e. a set of clauses

* For sets of formulae we proceed by taking the conjunction of the formulae

Step 1a: Rename variables

- Transform F such that no different occurrences of Quantifiers F in bind the same variable

- This can always be achieved through simple variable renaming…

Examples:

$$\forall\, x\; p(x) \leftarrow \exists\, x\; q(x)$$
$$\Rightarrow \forall\, x_1\; p(x_1) \leftarrow \exists\, x_2\; q(x_2)$$
$$\forall\, y\; \exists\, x\; (s(x,y) \wedge \forall\, x\; p(x))$$
$$\Rightarrow \forall\, x_1\; \exists\, x_2\; (s(x_2,x_1) \wedge \forall\, x_3\; p(x_3))$$

Again equivalence obviously retains…

Step 1b: Eliminate implications and "push" negation in front of atoms

$$\neg\neg F \Rightarrow F$$

$$\neg\exists F \Rightarrow \forall\neg F$$

$$\neg\forall F \Rightarrow \exists\neg F$$

$$\neg(F \vee G) \Rightarrow \neg F \wedge \neg G \quad \text{(de Morgan)}$$

$$\neg(F \wedge G) \Rightarrow \neg F \vee \neg G \quad \text{(de Morgan)}$$

$$F \rightarrow G \Rightarrow \neg F \vee G$$

$$F \leftrightarrow G \Rightarrow (F \rightarrow G) \wedge (G \rightarrow F)$$

Only basic logic transformations which retain equivalence. Obviously, by iterative application you reach a form where $\neg$ only occurs in front of atoms.

Step 2: Eliminate ∃ quantifiers

Thoralf Skolem

(1887-1963)

Step 2: Eliminate ∃ quantifiers

while F' contains ∃ quantifiers, repeat:
{
pick the first (from left to right) existential quantifier $\exists\, x$:

- **Case 1:** $\exists\, x$ is in not the scope of any all-quantifiers:
  - Replace $\exists\, x\; G$ by $G[x/a]$ where a is a **new** constant symbol, not occurring in F
- **Case 2:** $\exists\, x$ is in the scope of all-quantifiers $\forall\, y_1 \ldots \forall\, y_k$:
  - Replace $\exists\, x\; G$ by $G[x/f(y_1,\ldots, y_k)]$ where f is a **new** k-ary function symbol, not occurring in F'

}

Remark: This elimination is called "Skolemization" after its inventor Thoralf Skolem, a and f are often called Skolem-constant or Skolem-function, respectively.

The ∃-free formula F'' obtained in this step is called Skolemform of F'

G[x/t] … stands for the formula obtained from G by substituting the variable x with the term t

Properties of Skolemization:

- Skolemization does not preserve logical equivalence, but preserves equi-satisfiability:

Proposition 4: If F' is a closed formula obtained from **Step 1** and F'' is the Skolemform of F', then $F' \sim_e F''$

Proof (sketch): Enough to show that each step in the Skolemization preserves equi-satisfiability, since $\sim_e$ is an equivalence relation.

Let us consider the (more interesting) **Case 2**:

Since F' is in the form of Step 1 (no negation in front of quantifiers, no implications, variables renamed), quantifiers can be moved in front.

So, if $\exists\, x\, G$ is in the scope of all-quantifiers $\forall\, y_1 \ldots \forall\, y_k$ we can rewrite F' to

$\forall\, y_1 \ldots \forall\, y_k \exists\, x\, F$.

Now informal: This stands for *"for all* $y_1 \ldots y_k$ *there exists an* x*, such that* F *is valid"*

So, for any model, we chose a respective function $\phi : D^k \rightarrow D$ which makes exactly the assignment from $y_1 \ldots y_k$ *to x.*

Now extend the model *M* to *M'* such that a new function symbol f is assigned this function ϕ.

Then *M'* is a model for F'' which is obtained from *F'* by replacing $\exists\, x\, G$ *with* $G[x/f(y_1,\ldots, y_k)]$

Step 3:

- Since F'' does not contain $\exists$ quantors, negation only occurs in front of atoms, and variables have been renamed, $\forall$ quantors can be moved to the front.

  We obtain a formula of the form:

$$\forall y_1 \ \forall y_k \ G$$

- Finally, by applying the distributive laws on $\wedge$ and $\vee$ bring the formula G to conjunctive normal form.

➔we obtain a conjunction of Clauses!

Example:

$$\neg(((\forall x\ \exists y\ p(x,y) \wedge \forall x\ \forall y\ (p(x,y) \rightarrow r(y))) \rightarrow \forall z R(z))$$

After **Step 1a:** *

$$\neg(((\forall x\ \exists y\ p(x,y) \wedge \forall u\ \forall v\ (p(u,v) \rightarrow r(v))) \rightarrow \forall z R(z))$$

After **Step 1b:**

$$(\forall x\ \exists y\ p(x,y) \wedge \forall u\ \forall v\ (\neg p(u,v) \vee r(v))) \wedge \exists z\ \neg r(z)$$

After **Step 2:**

$$(\forall x\ \ p(x, f(x)) \wedge \forall u\ \ \forall v\ \ (\neg p(u,v) \vee r(v))) \wedge \neg r(a)$$

After **Step 3:**

$$\forall (p(x, f(x)) \wedge (\neg p(u,v) \vee r(v))) \wedge \neg r(a))$$

This corresponds to a set of clauses**:**

$$S = \{p(x, f(x)) \leftarrow, r(v) \leftarrow p(u,v)), \leftarrow r(a)\}$$

* A mechanical translation would maybe rather create something like:

$$\neg(((\forall x_1\ \exists x_2\ p(x_1,x_2) \wedge \forall x_3\ \forall x_4\ (p(x_3,x_4) \rightarrow r(x_4))) \rightarrow \forall x_5 R(x_5))$$

Automatic Proof Generation for sets of clauses by resolution

- A proof system needs inference rules
- Idea: We will prove a goal G from a set of clauses S by proving contradiction (i.e. unsatisfiability) from $S \cup \{\neg G\}$.

- Assume that your goal has the form:

  $G = \exists(A)$ where A is an atom:

  Then $\neg G$ is the clause $\leftarrow A$

- We will now learn some inference rule which can allows to prove contradiction, called resolution.

Example:

- All birds are animals $a(x) \leftarrow b(x)$
- All eagles are birds $b(x) \leftarrow e(x)$
- Karl loves (all) animals $l(karl,x) \leftarrow a(x)$
- Franzi is a bird $b(franz) \leftarrow$
- Hansi is an Eagle $e(hansi) \leftarrow$

- Goal/Query: Does Karl love Hansi?

  $\neg$ G $\equiv \neg\, l(karl,hansi)$

  $\Rightarrow \qquad \leftarrow l(karl,hansi)$

☺ let's try it out in Prolog!

Resolution:
Cut + Substitution

Resolution is a combination of an inference rule called **cut rule** …

$$\frac{\mathcal{C}_1 \vee A \vee \mathcal{C}_2 \qquad \mathcal{C}_3 \vee \neg A \vee \mathcal{C}_4}{\mathcal{C}_1 \vee \mathcal{C}_2 \vee \mathcal{C}_3 \vee \mathcal{C}_4} \qquad \text{(R)} \; *$$

…and another one called **substitution rule**:

$$\frac{\forall x_1, \ldots \forall x_n F}{\forall (F[x_1/t_1, \ldots, x_n/t_n])} \qquad \text{(S)}$$

Exercise: Use these two rules to prove contradiction from the example from the last slide!

* Could for clauses also be written as:

$$\frac{A_1, \ldots, \mathbf{A}, \ldots, A_k \leftarrow B_1, \ldots, B_l \qquad C_1, \ldots, C_m \leftarrow D_1, \ldots, \mathbf{A}, \ldots, D_n}{A_1, \ldots, A_k, C_1, \ldots, C_m \leftarrow B_1, \ldots, B_l, D_1, \ldots, D_n}$$

Let's try to prove contradiction using (R) and (S)

S: $a(x) \leftarrow b(x)$, $b(x) \leftarrow e(x)$, $l(karl,x) \leftarrow a(x)$, $b(franz) \leftarrow$, $e(hansi) \leftarrow$,

G: $\leftarrow l(karl,hansi)$

S: $\forall(a(x) \vee \neg b(x))$, $\forall(b(x) \vee \neg e(x))$, $\forall(l(karl,x) \vee \neg a(x))$, $\forall(b(franz))$, $\forall(e(hansi))$

$$\forall(\neg l(karl,hansi)) \qquad \frac{\forall(l(karl,x) \vee \neg a(x))}{\forall(l(karl,hansi) \vee \neg a(hansi))}\ \text{(S)}$$

(R)

$$\forall(\neg a(hansi)) \qquad \frac{\forall(a(x) \vee \neg b(x))}{\forall(a(hansi) \vee \neg b(hansi))}\ \text{(S)}$$

(R)

$$\forall(\neg b(hansi)) \qquad \frac{\forall(b(x) \vee \neg e(x))}{\forall(b(hansi) \vee \neg e(hansi))}\ \text{(S)}$$

(R)

$$\forall(\neg e(hansi)) \qquad \forall(e(hansi))$$

(R)

$$\emptyset$$

Contradiction!

Resolution rule:

- In the refutation in the last slide, a substitution always served as "preparation step" before the cut rule was applied.
- The derivation is "goal-oriented"
- Idea: Combine substitution and cut rules by using a canonical substitution, the so called most general unifier (mgu), which can be found automatically.
- This combined rule serves as a basis for an automatic refutation procedure.
- First we have to define **substitutions** and **unifiers…**

Subsitutions:

- A substitution is a finite set of the form:

  $\theta = \{v_1/t_1, \ldots, v_n/t_n\}$ where:
  - the v_i are variables and the t_i are terms
  - each t_i is distinct from v_i
  - the variables $v_1, \ldots, v_n$ are distinct
- If all $t_1, \ldots, t_n$ are ground terms then θ is called called a *ground substitution*
- If all $t_1, \ldots, t_n$ are variables then θ is called called a *variable-pure substitution*
- For a quantifier-free formula F

  $F[v_1/t_1, \ldots, v_n/t_n]$ *can be written* $F\theta$

Example: $p(x,y,f(a))$ $\theta = \{x/b, y/x\}$

Subsitutions and Unifiers:

- Composition of substitutions:

  $\theta = \{u_1/s_1, \ldots, u_m/s_m\}$

  $\sigma = \{v_1/t_1, \ldots, v_n/t_n\}$

- $\theta\sigma$ is obtained from $\{u_1/s_1\sigma, \ldots, u_m/s_m\sigma, \; v_1/t_1, \ldots, v_n/t_n\}$ by:
  - deleting any binding where $u_i = s_i\sigma$
  - deleting any binding v_i/t_i where $v_j \in \{u_1, \ldots, u_m\}$
- Two atomic formulae P and Q with the same predicate symbol are called unifiable if there exists a substitution θ such that

  $P\theta = Q\theta$

- A unifier θ of P and Q is called **most general unifier** (mgu), if for each unifier σ of P and Q a substitution γ exists such that $\sigma = \theta\gamma$

Examples:

- *p(f(x),a)*, *p(y,f(w))* are not unifiable because the second argument cannot be unified.
- *p(f(x),z), p(y,a)* are unifiable:

  σ = {y/f(a), x/a, z/a} is a unifier

  θ = {y/f(x), z/a} is a mgu

- How to compute an mgu automatically?

A unification algorithm:

The **disagreement** of P and Q is defined as follows: Find the leftmost position in P and Q where there is a different symbol and extract from P and Q the pair of terms (t_P, t_Q) beginning at that position.

Input: Two atomic formulae P and Q with the same predicate symbol.
Output: *fail* or an mgu θ_k

- Set $\theta_0 := \{\}$, $k := 0$
- If $P\theta_k = Q\theta_k$ return θ_k (θ_k is an mgu).
  Otherwise: find disagreement (t_P, t_Q) of $P\theta_k$ and $Q\theta_k$
- If t_P, t_Q are both non-variable terms return *fail*
- If t_P, is a variable not occurring in t_Q
  set $\theta_{k+1} := \theta_k\{t_P/t_Q\}$, $k := k+1$,
  else if t_Q, is a variable not occurring in t_P
  set $\theta_{k+1} := \theta_k\{t_Q/t_P\}$, $k := k+1$,
  and go to 2.

Remark: This "occur-check" Is not done by all Prolog systems !

Remark: There is also a general unification algorithm for sets of literals, but for our form of resolution this one is sufficient…

An example:

$P = p(a,x,f(g(y)))$ $\quad$ $Q= p(z,h(z,w),f(w))$

$\theta_0 = \{\}$

$P\ \theta_0 = p(a,x,f(g(y)))$ $\quad$ $Q\ \theta_0= p(z,h(z,w),f(w))$

$\theta_1 = \theta_0\{z/a\}=\{z/a\}$

$P\ \theta_1 = p(a,x,f(g(y)))$ $\quad$ $Q\ \theta_1= p(a,h(a,w),f(w))$

$\theta_2 = \theta_1\{x/h(a,w)\}=\{z/a, x/h(a,w)\}$

$P\ \theta_2 = p(a,h(a,w),f(g(y)))$ $Q\ \theta_2= p(a,h(a,w),f(w))$

$\theta_3 = \theta_2\{w/g(y)\}=\{z/a, x/h(a,g/y)), w/g(y)\}$

$P\ \theta_3 = p(a,h(a,g(y)),f(g(y)))$ $Q\ \theta_3= p(a,h(a,g(y)),f(g(y)))$

θ_3 is an mgu!

Resolution:

- Let C be a Horn clause

  $$A \leftarrow A_1, \ldots A_n$$

  and G be a goal

  $$\leftarrow B_1, \ldots, B_m$$

  where G and C have no variables in common.

  Let further θ be an mgu of A and B_i for some $1 \leq i \leq m$

  Then the goal

  $$\leftarrow B_1\theta, \ldots B_{i-1}\theta, A_1\theta, \ldots A_n\theta, B_{i+1}\theta, B_m\theta$$

  is called resolvent of G and C.

Remark: There is also a general resolution for full Clause logic, but for Horn programs the above is sufficient…

A refutation proof by resolution

$S = \{a(x) \leftarrow b(x),\ b(x) \leftarrow e(x),\ l(karl,x) \leftarrow a(x),\ b(franz) \leftarrow,\ \ e(hansi) \leftarrow\}$

- Does Karl love Hansi?

G: $\leftarrow l(karl, hansi)$ $\qquad l(karl, x) \leftarrow a(x)$

mgu $= \{x/hansi\}$

$\leftarrow a(hansi)$ $\qquad a(x) \leftarrow b(x)$

mgu $= \{x/hansi\}$

$\leftarrow b(hansi)$ $\qquad b(x) \leftarrow e(x)$

mgu $= \{x/hansi\}$

$\leftarrow e(hansi)$ $\qquad e(hansi) \leftarrow$

mgu $= \{\}$

$\emptyset$

Attention:

Why goal and clause may not have variables in common:

- try resolving:

$$G : \leftarrow p(x, f(x)) \qquad\qquad C : \; p(f(x), y) \leftarrow q(x, y)$$

Not unifiable!

This can however always be solved by renaming the variables in the clause to resolve:

$$G : \leftarrow p(x, f(x)) \qquad\qquad C : \; p(f(x_1), y_1) \leftarrow q(x_1, y_1)$$

$$\text{mgu} = \{x/f(x_1), y_1/f(f(x_1))\}$$

$$\leftarrow q(x_1, f(f(x_1)))$$

Answer Substitutions!

- However: What about more complex questions:
  - Who does Karl love?
  - Which Eagles does Karl love?

- Try resolution! There are possibly several possible refutations, each of which "contains" an answer substitution.

- More: next time!

Exercises:

- See separate sheet! Will be put on the lecture web site today.

Next Lecture:

- How to compute all answer substitutions
- Why does resolution work?
  - Correspondence between consequences and the minimal Herbrand model
  - Soundness & completeness
- SLD-Resolution + Prolog
  - 1. Why is the occur-check so expensive?
  - 2. Problems with termination in Prolog's procedural depth-first search! Left- and right recursion, selection-strategy…
  - Programming with Prolog! ☺