UNIVERSITY OF CALGARY

Computer Science

Prolog

Resolution & Unification

UNIVERSITY OF CALGARY

Resolution (for beginners?)

- Resolution is a theorem proving method developed by Robinson based on representing logical formulas as clauses
- Clauses are build out of a set V of variables, F of function symbols and P of predicate symbols.
- The set Term of terms is defined by
  - $V \subseteq Term$
  - If $f \in F$ with arity n and $t_1,\ldots,t_n \in Term$, then $f(t_1,\ldots,t_n) \in Term$

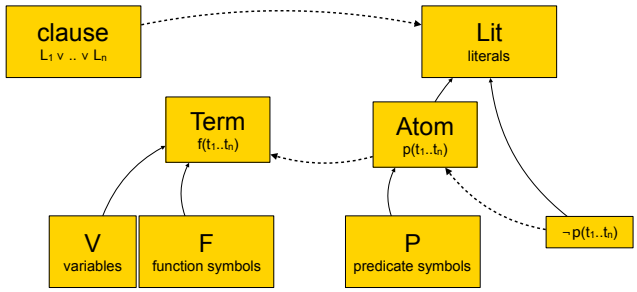
UNIVERSITY OF CALGARY

Resolution (cont.)

- The set Atom of atoms is defined by:
  If $p \in P$ with arity n and $t_1,\ldots,t_n \in Term$, then $p(t_1,\ldots,t_n) \in Atom$
- The set Lit of literals is defined by
  - $Atom \subseteq Lit$
  - If $p(t_1,\ldots,t_n) \in Atom$ then $\neg p(t_1,\ldots,t_n) \in Lit$
- If $L_1,\ldots,L_m \in Lit$, then $L_1 \vee \ldots \vee L_m$ is a clause
- Every set of formulae of first-order predicate calculus can be transformed into a set of clauses that is satisfiable exactly if the original set is satisfiable!

UNIVERSITY OF CALGARY

Resolution (cont.)



UNIVERSITY OF CALGARY

Resolution (still cont.)

- The resolution rule is as follows:
  if we have two clauses of the form $A \vee C_1$ and $\neg A' \vee C_2$ (C_1, C_2 clauses, A,A' atoms) and a substitution σ so that $\sigma(A) = \sigma(A')$, then we can generate the new clause $\sigma(C_1 \vee C_2)$
- This is essentially an extension of the modus ponens rule
- Usually, σ is the most general unifier of the two atoms and a function that assigns to each variable in the clauses a term. In $\sigma(A)$ we copy A, but replace every variable by the term assigned by σ

UNIVERSITY OF CALGARY

Unification

- In order to unify two terms $s = f(s_1,\ldots,s_n)$ and $t = g(t_1,\ldots,t_m)$, f has to be equal to g, and n=m.
- Then we unify sequentially s_1 and t_1, …, s_n and t_n
- If either s_i or t_i is a variable, then
  - We assign to this variable the other term, if the variable has not been substituted so far (and the variable does not occur in the other term)
  - If it is already substituted, then we unify the other term with the term substituted for the variable

Examples (I)

Unification:

- s = $f(x_1,g(f(x_1,a)))$ t = $f(g(b),y_1)$ [a,b are constants]
- s = $f(x_1,g(f(x_1,y_1)))$ t = $f(g(b),y_1)$
- s = $f(x_1,g(f(x_1,a)))$ t = $f(x_1,g(f(x_1,b)))$
- s = f(x,b) t = f(g(b),y)
- s = f(z,f(x,b)) t = f(y,f(a,y))

Examples (I)

Unification:

- s = $f(x_1,g(f(x_1,a)))$ t = $f(g(b),y_1)$
  - ☞ $\sigma = \{x_1/g(b), y_1/g(f(g(b),a))\}$
- s = $f(x_1,g(f(x_1,y_1)))$ t = $f(g(b),y_1)$
  - ☞ not unifiable (occur test fails)
- s = $f(x_1,g(f(x_1,a)))$ t = $f(x_1,g(f(x_1,b)))$
  - ☞ not unifiable (a is not equal to b, a and b being constants)
- s = f(x,b) t = f(g(b),y)
  - ☞ $\sigma = \{x/g(b), y/b\}$
- s = f(z,f(x,b)) t = f(y,f(a,y))
  - ☞ $\sigma = \{x/a, y/b, z/b\}$

Examples (II)

Resolution:

- $p \vee q, \neg p \vee s$
- $P(x,a) \vee R(x), \neg P(b,x) \vee S(x)$
- $P(f(x),b) \vee P(g(x),c), \neg P(y,z) \vee R(y,z)$
- $P(x), \neg P(x,y)$
- $P(x,a), \neg P(x,y)$

Examples (II)

Resolution:

- $p \vee q, \neg p \vee s$ ☞ $q \vee s$ (no unification needed)
- $P(x,a) \vee R(x), \neg P(b,x) \vee S(x)$ ☞ $R(b) \vee S(a)$

  Note: variables in different clauses are assumed to be different!
- $P(f(x),b) \vee P(g(x),c), \neg P(y,z) \vee R(y,z)$ ☞

  $P(g(x),c) \vee R(f(x),b)$ and $P(f(x),b) \vee R(g(x),c)$
- $P(x), \neg P(x,y)$ ☞ resolution not applicable, P/1 and P/2 are considered different predicate symbols
- $P(x,a), \neg P(x,y)$ ☞ □

Some facts on resolution

- The resolution rule (together with factorization) allows to generate the empty clause □ out of a set of clauses that is unsatisfiable
- Usually, in a given set of clauses there are many applications of the resolution rule possible (sometimes there are even several applications for just two clauses, see example)
- The control of the rule applications is very important for the efficiency of the process of generating □ (from very quick to impossible)

From resolution to SLD-resolution

- Full resolution is not something you can build a programming language on
- We need
  - Restrictions on the clauses
    - Horn-clauses
  - A fixed control
  - Higher-order predicates, build-in predicates
  - IO

☞ Use SLD-resolution (Selected, Linear, Definite)

SLD-resolution

- Works on Horn-clauses: clauses that contain only one literal that is an atom (= positive literal)
- Only one clause: one literal followed by only negative literals is allowed: the goal
- Transforms Horn-clauses into rules:
  $A_1 \vee \neg A_2 \vee \ldots \vee \neg A_n \Rightarrow A_1 :- A_2,\ldots,A_n$
- Provides a clear control strategy of applications of resolution rule based on sequence of rules in memory (data base), sequence of atoms in rule and goal that we want to prove (including backtracking)

SLD-resolution (explained)

- $X \rightarrow Y \equiv \neg X \vee Y$
- We reverse the implication to put the consequent on the left:
  $Y \leftarrow X \equiv Y \vee \neg X$, which is a Horn clause
- We write this as rule with the consequent on the left:
  $Y :- X$ (which means exactly the same as the above, "X implies Y")

and for multiple antecedents

- $X_1 \wedge X_2 \rightarrow Y \equiv \neg(X_1 \wedge X_2) \vee Y \equiv \neg X_1 \vee \neg X_2 \vee Y$
- Reverse (as above)
  $Y \vee \neg X_1 \vee \neg X_2$, which is a Horn clause
- And as a rule we write:
  $Y :- X_1, X_2$ (so we read "X_1 and X_2 implies Y")

Control

- Refute goal using data base of clauses and backtracking

?- $G_1,G_2,\ldots,G_m$
↓ $\sigma_1(G_1) = \sigma_1(A_1)$
?- $\sigma_1(B_{11}),\ldots,\sigma_1(B_{1n1}),\sigma_1(G_2),\ldots,\sigma_1(G_m)$
⋮ $\sigma_1(B_{ij})$not solvable
☞ backtrack
?- $G_1,G_2,\ldots,G_m$
↓ $\sigma_2(G_1) = \sigma_2(A_2)$
?- $\sigma_2(B_{21}),\ldots,\sigma_2(B_{2n2}),\sigma_2(G_2),\ldots,\sigma_2(G_m)$

Data base:
...
A_1:-$B_{11},\ldots,B_{1n1}$.
...
A_k:-$B_{k1},\ldots,B_{knk}$.
...

σ_l mgus

Control (animated)

?- $G_1,G_2,\ldots,G_m$

Data base:
...
A_1:-$B_{11},\ldots,B_{1n1}$.
...
A_k:-$B_{k1},\ldots,B_{knk}$.
...
σ_l = mgus

Control (animated)

?- $G_1,G_2,\ldots,G_m$ ← $\sigma_1(G_1) = \sigma_1(A_1)$

Data base:
...
A_1:-$B_{11},\ldots,B_{1n1}$.
...
A_k:-$B_{k1},\ldots,B_{knk}$.
...
σ_l = mgus

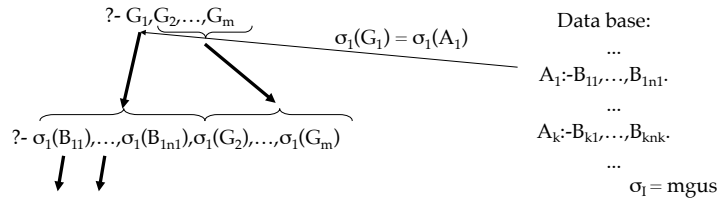
Control (animated)

?- $G_1,G_2,\ldots,G_m$ → $\sigma_1(G_1) = \sigma_1(A_1)$

?- $\sigma_1(B_{11}),\ldots,\sigma_1(B_{1n1}),\sigma_1(G_2),\ldots,\sigma_1(G_m)$

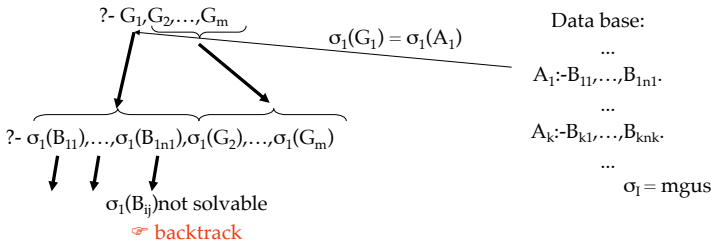
Data base:
...
A_1:-$B_{11},\ldots,B_{1n1}$.
...
A_k:-$B_{k1},\ldots,B_{knk}$.
...
σ_l = mgus

Control (animated)



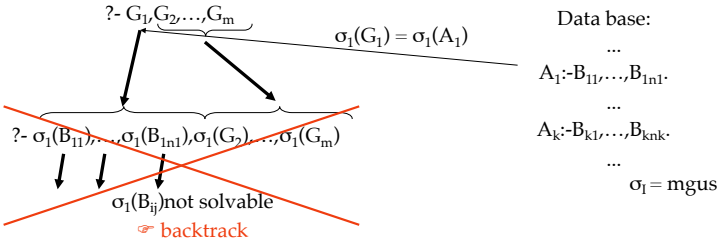
Data base:
...
A_1:-$B_{11},\ldots,B_{1n1}$.
...
A_k:-$B_{k1},\ldots,B_{knk}$.
...
σ_l = mgus

Control (animated)



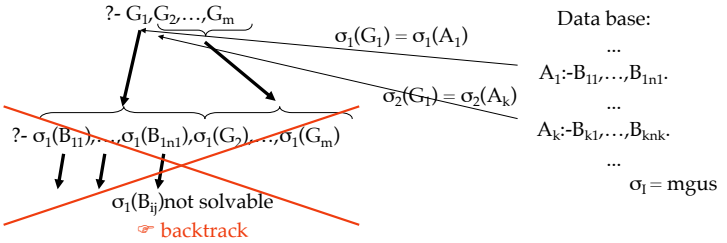
Data base:
...
A_1:-$B_{11},\ldots,B_{1n1}$.
...
A_k:-$B_{k1},\ldots,B_{knk}$.
...
σ_l = mgus

Control (animated)



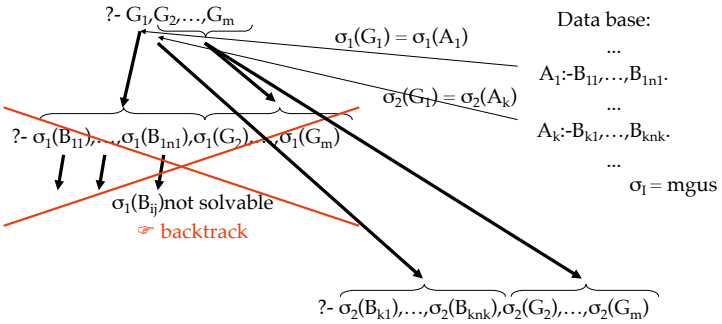
Data base:
...
A_1:-$B_{11},\ldots,B_{1n1}$.
...
A_k:-$B_{k1},\ldots,B_{knk}$.
...
σ_l = mgus

Control (animated)



Data base:
...
A_1:-$B_{11},\ldots,B_{1n1}$.
...
A_k:-$B_{k1},\ldots,B_{knk}$.
...
σ_l = mgus

Control (animated)



Data base:
...
A_1:-$B_{11},\ldots,B_{1n1}$.
...
A_k:-$B_{k1},\ldots,B_{knk}$.
...
σ_l = mgus

Solutions

- Solution: if goal stack gets empty
  - ➜ collect substitutions that fulfilled original goals
  - ➜ use as answer
- Next solution: initiate backtrack
- No solution:
  if no clause in data base solves a particular subgoal G_i for all solutions to $G_1,\ldots,G_{i-1}$

UNIVERSITY OF CALGARY

Example (just truth of a conjecture)

Father(abraham, isaac).
Father(haran, lot).
Father(haran, milcah).
Father(haran, yiscah).

Male(isaac).
Male(lot).
Female(milcah).
Female(yiscah).

Son(X,Y) ← Father(Y,X), Male(X).
Daughter(X,Y) ← Father(Y,X), Female(X).

Son(lot, haran)?

UNIVERSITY OF CALGARY

Example (just truth of a conjecture)

Father(abraham, isaac).
Father(haran, lot).
Father(haran, milcah).
Father(haran, yiscah).

Male(isaac).
Male(lot).
Female(milcah).
Female(yiscah).

Son(X,Y) ← Father(Y,X), Male(X).
Daughter(X,Y) ← Father(Y,X), Female(X).

Son(lot, haran)?

2016/01/19 CPSC 449 288

UNIVERSITY OF CALGARY

Example (just truth of a conjecture)

Father(abraham, isaac).
Father(haran, lot).
Father(haran, milcah).
Father(haran, yiscah).

Male(isaac).
Male(lot).
Female(milcah).
Female(yiscah).

Son(X,Y) ← Father(Y,X), Male(X).
Daughter(X,Y) ← Father(Y,X), Female(X).

Son(lot, haran)?

2016/01/19 CPSC 449 288

UNIVERSITY OF CALGARY

Example (just truth of a conjecture)

Father(abraham, isaac).
Father(haran, lot).
Father(haran, milcah).
Father(haran, yiscah).

Male(isaac).
Male(lot).
Female(milcah).
Female(yiscah).

Son(X,Y) ← Father(Y,X), Male(X).
Daughter(X,Y) ← Father(Y,X), Female(X).

Son(lot, haran)?

2016/01/19 CPSC 449 288

UNIVERSITY OF CALGARY

Example (just truth of a conjecture)

Father(abraham, isaac).
Father(haran, lot).
Father(haran, milcah).
Father(haran, yiscah).

Male(isaac).
Male(lot).
Female(milcah).
Female(yiscah).

Son(X,Y) ← Father(Y,X), Male(X).
Daughter(X,Y) ← Father(Y,X), Female(X).

Son(lot, haran)?

2016/01/19 CPSC 449 288

UNIVERSITY OF CALGARY

Example (just truth of a conjecture)

Father(abraham, isaac).
Father(haran, lot).
Father(haran, milcah).
Father(haran, yiscah).

Male(isaac).
Male(lot).
Female(milcah).
Female(yiscah).

Son(X,Y) ← Father(Y,X), Male(X).
Daughter(X,Y) ← Father(Y,X), Female(X).

Son(lot, haran)?

2016/01/19 CPSC 449 288

Example (just truth of a conjecture)

Father(abraham, isaac). Male(isaac).
Father(haran, lot). Male(lot).
Father(haran, milcah). Female(milcah).
Father(haran, yiscah). Female(yiscah).

Son(X,Y) ← Father(Y,X), Male(X).
Daughter(X,Y) ← Father(Y,X), Female(X).

Son(lot, haran)?

UNIVERSITY OF CALGARY

Example (just truth of a conjecture)

Father(abraham, isaac). Male(isaac).
Father(haran, lot). Male(lot).
Father(haran, milcah). Female(milcah).
Father(haran, yiscah). Female(yiscah).

Son(X,Y) ← Father(Y,X), Male(X).
Daughter(X,Y) ← Father(Y,X), Female(X).

Son(lot, haran)?

2016/01/19 CPSC 449 288

UNIVERSITY OF CALGARY

Example (just truth of a conjecture)

Father(abraham, isaac). Male(isaac).
Father(haran, lot). Male(lot).
Father(haran, milcah). Female(milcah).
Father(haran, yiscah). Female(yiscah).

Son(X,Y) ← Father(Y,X), Male(X).
Daughter(X,Y) ← Father(Y,X), Female(X).

Son(lot, haran)?

2016/01/19 CPSC 449 288

UNIVERSITY OF CALGARY

Example (just truth of a conjecture)

Father(abraham, isaac). Male(isaac).
Father(haran, lot). Male(lot).
Father(haran, milcah). Female(milcah).
Father(haran, yiscah). Female(yiscah).

Son(X,Y) ← Father(Y,X), Male(X).
Daughter(X,Y) ← Father(Y,X), Female(X).

Son(lot, haran)?

2016/01/19 CPSC 449 288

UNIVERSITY OF CALGARY

Example (just truth of a conjecture)

Father(abraham, isaac). Male(isaac).
Father(haran, lot). Male(lot).
Father(haran, milcah). Female(milcah).
Father(haran, yiscah). Female(yiscah).

Son(X,Y) ← Father(Y,X), Male(X).
Daughter(X,Y) ← Father(Y,X), Female(X).

Son(lot, haran)? ✓

Example (filling in variables)

- mother(anna,peter).
  mother(anna,clara).
  father(joe,peter).
  father(jim,clara).
  mother(mary,anna).
  mother(mary,joe).
  grandmother(X,Y) :- mother(X,Z), mother(Z,Y).
  grandmother(X,Y) :- mother(X,Z), father(Z,Y).
- Answer: ?- grandmother(mary,X).
  ?- grandmother(X,peter).

PROLOG in short

- Stands for PROgramming in LOGic
- Rather old language (starting in 1970s)
- No strong type concept (what types?)
- Unification without occur check
- Suffers a little bit from the LISP-version syndrome
- Key concepts:
  - Negation as failure
  - Closed world assumption
  - Knowledge (data) base control in language
  - No distinction between input and output