Resolution and Logic Programming

- ★ Ground resolution
- ★ Unification and occur check
- ★ General Resolution
- ★ Logic Programming
- ★ SLD-resolution
- ★ The programming language Prolog
  - ⇒ Syntax
  - ⇒ Arithmetic
  - ⇒ Lists

Motivation (1)

- We want to show $\Phi \models \Psi$, for two propositional formulas Φ, Ψ.
- Assume Φ is $\Phi_1 \wedge \cdots \wedge \Phi_n$, in CNF, and Ψ is $L_1 \wedge \cdots \wedge L_m$, a conjunction of literals.
- Showing $\Phi \models \Psi$ is equivalent with showing that the set of formulas $\{\Phi_1, \ldots, \Phi_n, \neg\Psi\}$ is unsatisfiable.
- ***Resolution:*** a procedure $\mathbf{Res}(\chi_1, \chi_2)$ applied to two formulas, and returning a (simpler) formula χ, such that, if $\{\chi_1, \chi_2, \chi\}$ is unsatisfiable, then so is $\{\chi_1, \chi_2\}$.

Motivation (2)

- We hope to produce the iteration

  $\{\Phi_1, \ldots, \Phi_n, \neg\Psi\}$
  $\{\Phi_1, \ldots, \Phi_n, \neg\Psi, \mathbf{Res}(\neg\Psi, \Phi_{k_1}) = \chi_1\}$
  $\{\Phi_1, \ldots, \Phi_n, \neg\Psi, \chi_1, \mathbf{Res}(\chi_1, \Phi_{k_2}) = \chi_2\}$
  ....
  $\{\Phi_1, \ldots, \Phi_n, \neg\Psi, \chi_1, \ldots \chi_{l-1}, \mathbf{Res}(\chi_{l-1}, \Phi_{k_l}) = \bot\}$ — unsatisfiable

  where $1 \leq k_i \leq n$, $1 \leq i \leq l$.
- According to the property on the previous slide, if the last set is unsatisfiable, then so is the first set.
- A procedure showing that a set of formulas is unsatisfiable is called a ***refutation procedure***.

CNF and Clausal Form (1)

- Given the CNF propositional formula $\Phi \equiv \Phi_1 \wedge \Phi_n$, where Φ_i are disjuncts, $1 \leq i \leq n$
- For each i, $1 \leq i \leq n$, $\Phi_i \equiv \neg p_{i1} \vee \neg p_{i2} \vee \cdots \vee \neg p_{ik_i} \vee q_{i1} \vee \cdots \vee q_{il_i}$
- Φ_i is equivalent to $p_{i1} \wedge \cdots \wedge p_{ik_i} \to q_{i1} \vee \cdots \vee q_{il_i}$ which we call ***a clause***.
- We represent the clause by $p_{i1}, \ldots, p_{ik_i} \to q_{i1}, \ldots, q_{il_i}$
- We represent Φ as the set of clauses

  $$\{(p_{i1}, \ldots, p_{ik_i} \to q_{i1}, \ldots, q_{il_i}), \ldots, () | 1 \leq i \leq n\}$$

  which we call the ***clausal form*** of Φ.

CNF and Clausal Form (2)

$\neg(p_1 \wedge \cdots \wedge p_k)$ can be written as $p_1 \wedge \cdots \wedge p_k \to \top$, or as $p_1, \ldots, p_k \to$

$q_1 \vee \cdots \vee q_l$ can be written as $\bot \to q_1, \ldots, q_l$, or as $\to q_1, \ldots, q_l$

$\bot$ can be written as $\bot \to \top$, and is denoted by $\square$ (empty clause).

Ground Resolution

Given two clauses

$$\chi_1 : p_1, \ldots, p_k, \ldots p_{m_1} \to q_1, \ldots q_{n_1}$$
$$\chi_2 : r_1, \ldots r_{m_2} \to s_1, \ldots s_l \ldots s_{n_2}$$

If p_k and s_l are the same propositional symbol, then $\mathbf{Res}(\chi_1, \chi_2)$ is

$$p_1, \ldots, p_{k-1}, p_{k+1}, \ldots, p_{m_1} r_1, \ldots, r_{m_2} \to q_1, \ldots, q_{n_1}, s_1, \ldots, s_{l-1}, s_{l+1}, \ldots, s_{n_2}$$

This is similar to the following cancelling rule in arithmetic.

$$\frac{\begin{array}{rcl} a+b & = & c \\ c & = & d+e \end{array}}{a+b+\not{c} \; = \; \not{c}+d+e}$$

Ground Resolution Example

$$\begin{array}{lllll} \Phi_1 & \text{is} & p \wedge q & \rightarrow & r \\ \Phi_2 & \text{is} & & \rightarrow & p \\ \Phi_3 & \text{is} & & \rightarrow & q \\ \Psi & \text{is} & r & \rightarrow & \end{array}$$

$$\begin{array}{lll} \chi_1 = \mathbf{Res}(\Phi_1, \Psi) & \text{is} & p, q \rightarrow \\ \chi_2 = \mathbf{Res}(\chi_1, \Phi_2) & \text{is} & q \rightarrow \\ \chi_3 = \mathbf{Res}(\chi_2, \Phi_3) & \text{is} & \square \end{array}$$

Alternatively

$$\begin{array}{lll} \chi_1 = \mathbf{Res}(\Phi_1, \Phi_2) & \text{is} & q \rightarrow r \\ \chi_2 = \mathbf{Res}(\chi_1, \Phi_3) & \text{is} & \rightarrow r \\ \chi_3 = \mathbf{Res}(\chi_2, \Psi) & \text{is} & \square \end{array}$$

Predicate Logic Clauses

A predicate logic clause:

$$p(x,y), q(f(x),z) \rightarrow r(y,z,w), s(g(z),w)$$

Meaning:

$$\forall x \forall y \forall z \exists w\, (p(x,y) \wedge q(f(x),z) \rightarrow r(y,z,w) \vee s(g(z),w))$$

- First order clauses are a subset of predicate logic: not all predicate logic formulas can be expressed as clauses.
- They are more general than a Turing machine: can specify all possible computations.

Non-Ground Resolution

Consider the following first order clauses.

$$\chi_1 : A_1, \ldots, A_k, \ldots, A_{m_1} \rightarrow B_1, \ldots, B_{n_1}$$
$$\chi_2 : C_1, \ldots, C_{m_2} \rightarrow D_1, \ldots, D_l, \ldots, D_{n_2}$$

where the As, Bs, Cs, and Ds are first order atoms. Assume there exists a substitution θ such that $A_k\theta = D_l\theta$. We call θ a ***unifier***. Then $\mathbf{Res}(\chi_1\theta, \chi_2\theta)$ is

$$A_1\theta, \ldots, A_{k-1}\theta, A_{k+1}\theta, \ldots, A_{m_1}\theta, C_1\theta, \ldots, C_{n_1}\theta \rightarrow B_1\theta, \ldots, B_{m_2}\theta, D_1\theta, \ldots, D_{l-1}\theta, D_{l+1}\theta, \ldots, D_{n_2}\theta$$

which is the same as

$$(A_1, \ldots, A_{k-1}, A_{k+1}, \ldots, A_{m_1}, C_1, \ldots, C_{n_1} \rightarrow B_1, \ldots, B_{m_2}, D_1, \ldots, D_{l-1}, D_{l+1}, \ldots, D_{n_2})\theta$$

Non-Ground Resolution Example

$$\begin{array}{rcl} \chi_1 & : & p(x,y) \rightarrow q(y,z) \\ \chi_2 & : & q(f(w),v) \rightarrow r(v) \\ \theta & : & [f(w)/y, z/v] \\ \chi_1\theta & : & p(x,f(w)) \rightarrow q(f(w),z) \\ \chi_2\theta & : & q(f(w),z) \rightarrow r(z) \\ \mathbf{Res}(\chi_1\theta, \chi_2\theta) & : & p(x,f(w)) \rightarrow r(z) \end{array}$$

Unification, MGU

Given two atoms, A, B, can we find a unifying substitution θ, such that $A\theta = B\theta$? Answer: YES.

A ***most general unifier (mgu)*** is a unifying substitution θ such that for every other unifier θ', there exists a substitution σ such that

$$A\theta' = (A\theta)\sigma$$
$$B\theta' = (A\theta)\sigma$$

Unification Algorithm

The following algorithm computes the mgu of two atoms A and B, or returns ***"no solution"*** if no such mgu exists.

1. If the predicate symbols of A and B are not identical, return "no solution".
2. From $p(t_1, \ldots, t_k) = p(t'_1, \ldots, t'_k)$ derive the set of equations $\{t_1 = t'_1, \ldots, t_k = t'_k\}$.
3. Erase all equations of the form $x = x$, where x is a variable.
4. Transform all equations of the form $t = x$, where t is not a variable, into $x = t$.
5. Let $t' = t''$ be an equation where t' and t'' are not variables. If the function symbols of t' and t'' are not identical, return "no solution." Otherwise, replace the equation $f(t'_1, \ldots, t'_k) = f(t''_1, \ldots, t''_k)$ by the equations $t'_1 = t''_1$, …, $t'_k = t''_k$.
6. Let $x = t$ be an equation such that x has another occurrence in the set of equations. If t contains x, return "no solution." Otherwise replace all other occurrences of x by t.

Repeat steps 4, 5, and 6 until it is no longer possible. If the "no solution" answer has not been produced yet, all equations are of the form $x = t$, where t does not contain x. The mgu contains all the bindings t/x, where $x = t$ is an equation in our set.

Example of Applying the Unification Algorithm

Unify the atoms

$$p(x, f(x, h(x), y)) \text{ and } p(g(y), f(g(z), w, z))$$

First derive the equations:

$$(1) \quad x = g(y)$$
$$(2) \quad f(x, h(x), y) = f(g(z), w, z)$$

Apply step 5 and replace (2) by

$$(3) \quad x = g(z)$$
$$(4) \quad h(x) = w$$
$$(5) \quad y = z$$

Apply step 4 and replace (4) by

$$(6) \quad w = h(x)$$

Example (2)

Current set:

$$(1') \quad x = g(y)$$
$$(2') \quad x = g(z)$$
$$(3') \quad w = h(x)$$
$$(4') \quad y = z$$

Apply step 6 and use (1') in (2') and (3')

$$(1'') \quad x = g(y)$$
$$(2'') \quad g(y) = g(z)$$
$$(3'') \quad w = h(g(y))$$
$$(4'') \quad y = z$$

Replace $(2'')$ by

$y = z$ ← already in the set

Use $(4'')$ in $(1'')$ and $(3'')$. The set is now:

$$x = g(z)$$
$$w = h(g(z))$$
$$y = z$$

Substitution:

$$[g(z)/x, h(g(z))/w, z/y]$$

Example (3)

$p(x, f(x, h(x), y))[g(z)/x, h(g(z))/w, z/y]$ is

$$p(g(z), f(g(z), h(g(z)), z))$$

$p(g(y), f(g(z), w, z))[g(z)/x, h(g(z))/w, z/y]$ is

$$p(g(z), f(g(z), h(g(z)), z))$$

Occur Check

Step 6 in the unification algorithm can be very expensive.

Consider unifying

$$p(x_1, x_2, \ldots, x_n, x_0) \text{ and } p(f(x_0, x_0), f(x_1, x_1), \ldots, f(x_n, x_n))$$

This produces:

$$\begin{aligned} x_1 &= f(x_0, x_0) \\ x_2 &= f(f(x_0, x_0), f(x_0, x_0)) \\ x_3 &= f(f(f(x_0, x_0), f(x_0, x_0)), f(f(x_0, x_0), f(x_0, x_0))) \\ &\cdots\cdots \\ x_n &= \text{term with } 2^n \text{ occurrences of } x_0 \\ x_0 &= \text{term with } 2^{n+1} \text{ occurrences of } x_0 \end{aligned}$$

Using step 6, we must return "no solution" ; detecting the fact that x_0 occurs in the right hand side of last equation may require exponential time.

General Resolution

Consider the following first order clauses.

$$\chi_1 : A_1, \ldots, A_k, \ldots, A_{m_1} \to B_1, \ldots, B_{n_1}$$
$$\chi_2 : C_1, \ldots, C_{m_2} \to D_1, \ldots, D_l, \ldots, D_{n_2}$$

where the As, Bs, Cs, and Ds are first order atoms. Denote by θ the mgu of A_k and D_l. Then $\mathbf{Res}(\chi_1, \chi_2)$ is

$$(A_1, \ldots, A_{k-1}, A_{k+1}, \ldots, A_{m_1}, C_1, \ldots, C_{n_1} \to B_1, \ldots, B_{m_2}, D_1, \ldots, D_{l-1}, D_{l+1}, \ldots, D_{n_2})\theta$$

If there exist no two unifiable atoms A_k and D_l, then the resolution rule is undefined.

Resolution procedure: Let S be a set of clauses and define $S_0 = S$. Assume that S_i has been constructed. Choose two clauses χ_1, $\chi_2 \in S_i$ such that $\mathbf{Res}(\chi_1, \chi_2)$ is defined. If $\mathbf{Res}(\chi_1, \chi_2) = \Box$, the original set S is unsatisfiable. Otherwise, construct $S_{i+1} = S_i \cup \mathbf{Res}(\chi_1, \chi_2)$. If $S_{i+1} = S_i$ for all possible pairs χ_1 and χ_2, then S is satisfiable.

Example of General Resolution

Original set:

1. $p(x) \to q(x), r(x, f(x))$
2. $p(x) \to q(x), s(f(x))$
3. $\to t(a)$
4. $\to p(a)$
5. $r(a, y) \to t(y)$
6. $t(x), q(x) \to$
7. $t(x), s(x) \to$

Application of the resolution procedure:

8.	$q(a) \to$	$[a/x]$	3,6
9.	$\to q(a), s(f(a))$	$[a/x]$	2,4
10.	$\to s(f(a))$		8,9
11.	$\to q(a), r(a, f(a))$	$[a/x]$	1,4
12.	$\to r(a, f(a))$		8,11
13.	$\to t(f(a))$	$[f(a)/y]$	5,12
14.	$s(f(a)) \to$	$[f(a)/x]$	7,13
15.	$\Box$		10,14

Soundness and Completeness of Resolution

Soundness: If the unsatisfiable clause $\Box$ is derived during the general resolution procedure, then the original set of clauses is unsatisfiable.

Completeness: If a set of clauses is unsatisfiable, then the empty clause $\Box$ can be derived by the resolution procedure.

Logic Programming

From now on, instead of writing clauses as

$$A_1,\ldots,A_m \rightarrow B_1,\ldots,B_n$$

we shall prefer to write clauses as

$$B_1,\ldots,B_n \leftarrow A_1,\ldots,A_m$$

For $n=1$ we have ***Horn clauses***, typically denoted as

$$H \leftarrow A_1,\ldots,A_m$$

H — the head, $A_1,\ldots,A_m$ — the body
If $n=0$, the clause is a ***goal***.
If $n=1$ and $m=0$ (body is empty), we have a ***fact***.
A ***logic program*** is a set of Horn clauses.

Resolution for Logic Programs

In what follows, we shall introduce restrictions for the resolution procedure that would make it more computationally efficient.

Definition: A ***computation rule*** is a rule for choosing literals in a goal clause. A ***search rule*** is a rule for choosing clauses to resolve with the chosen literal in a goal clause.

Typical computation rule: leftmost atom in a goal Γ.
Typical search rule: clauses are tried in the order in which they are written.

Example of Resolution for Logic Programs

Logic program:

1. $q(x,y) \leftarrow p(x,y)$
2. $q(x,y) \leftarrow p(x,z), q(z,y)$
3. $p(b,a) \leftarrow$
4. $p(c,a) \leftarrow$
5. $p(d,b) \leftarrow$
6. Goal: $\leftarrow q(d,a)$

Applying the resolution procedure, with computation and search rules.

7.	$\leftarrow p(d,a)$	$[d/x, a/y]$	6,1
8.	$\leftarrow p(d,z), q(z,a)$	$[d/x, a/y]$	7,2
9.	$\leftarrow q(b,a)$	$[b/z]$	8,5
10.	$\leftarrow p(b,a)$	$[b/x, a/y]$	9,1
11.	$\Box$		10,3

The Programming Language Prolog

A Prolog program is, in its most basic form, a set of Horn clauses. Given a goal, the execution of the program and the goal is achieved by applyin the resolution procedure with the following rules:

Computation rule: choose literals from left to right in the goal.

Search rule: Choose clauses top-to-bottom as they are written in the program text.

The resolution procedure augmented with these rules is called ***SLD-resolution***.

Syntax:

- Predicate and function symbols start with lowercase letters.
- Variables start with uppercase letters or underscore.
- The arrow is represented by the `:-` operator.
- The dot `.` acts as a clause separator.

Prolog Example

```
ancestor(X,Y) :- parent(X,Y).
ancestor(X,Y) :- parent(X,Z),ancestor(Z,Y).

parent(bob,allen).
parent(catherine,allen).
parent(dave,bob).
parent(ellen,bob).
parent(fred,dave).
parent(harry,george).
parent(ida,george).
parent(joe,harry).
```

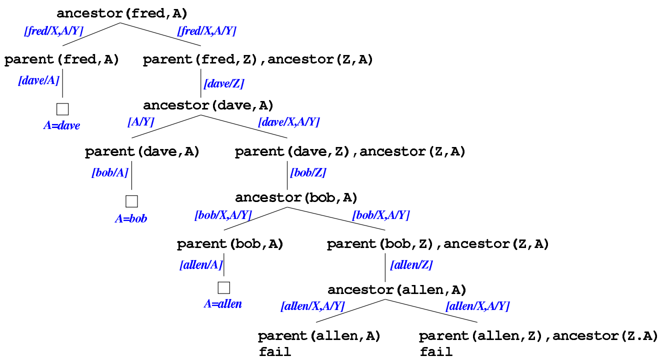
Goal: `ancestor(fred,bob)`
Answer: `Yes`

Goal: `ancestor(fred,A)`
Answer: `A=dave`
`A=bob`
`A=allen`

Goal: `ancestor(A,allen)`

Goal: `ancestor(A,B)`

Execution of Prolog Programs. SLD-Tree.



Free and Bound Variables

When a substitution is computed, a pair x/t is called a ***binding***.

If t is a variable, then x is called ***free***.

If t is a non-variable term, then x is called ***bound***.

Prolog uses special predicates for arithmetic, accessing files, etc. Such predicates have restrictions on using free variables.

Arithmetic Predicates

The predicate `is`:

`?- X is 2+3.`
Answer: `X=5`

`?- 5 is 2+3.`
Answer: `Yes`

`?- 5 is 2+X.`
Error! Free variable not allowed on the right side of `is`

"Less then" predicate:

`?- 0 < 1.`
Answer: `Yes`

`?- X = 0, X < 1.`
Answer: `Yes`

`?- X < 1, X = 0.`
Error! Free variable not allowed on the right side of `is`

A Factorial Program

Correct program:

```
factorial(0,1).
factorial(N,X) :-
   N > 0, N1 is N-1, factorial(N1,X1), X is X1*N.
```

Goal: `?- factorial(5,X).`
Answer: `X=120`

Wrong program:

```
factorial(0,1).
factorial(N,X) :-
   N > 0, N1 is N-1, X is X1*N, factorial(N1,X1).
```

Goal: `?- factorial(5,X).`
Error!!!

Lists (By Example)

Examples of lists:

- `[1,2,3,4]`
- `[]` — empty list.
- `[1|[2,3,4]]` — same as `[1,2,3,4]`, same as `|(1,|(2,|(3,|(4,nil))))`

`?- [H|T] = [1,2,3,4].`
Answer: `H=1, T=[2,3,4]`

`?- H=a, T=[b,c,d], X=[H|T].`
Answer: `H=a, T=[b,c,d], X=[a,b,c,d]`

Warning:

`?- H=[a,b,c], T=[d,e,f], X=[H|T]`
Answer: `X=[[a,b,c],d,e,f]`

`[H|T]` is syntactic sugar for `|(H,T)`.
`[]` is syntactic sugar for `nil`.

Lists: `append`

```
append([],X,X).
append([H|T],X,[H|T1]) :- append(T,X,T1).
```

Goal: `?- append([a,b,c],[d,e,f],A).`
Answer: `A=[a,b,c,d,e,f]`

Goal: `?- append([a,b,c],A,[a,b,c,d,e,f]).`
Answer: `A=[d,e,f]`

Goal: `?- append(A,B,[1,2,3]).`
Answer:
`A=[], B=[1,2,3]`
`A=[1], B=[2,3]`
`A=[1,2], B=[3]`
`A=[1,2,3], B=[]`

Lists: Sum of All Elements

```
sum([],0).
sum([H|T],X) :- sum(T,X1), X is X1+H.
```

Goals: `sum([1,2,3,4],X)`
Answer: `A=10`

`sum([1,2,3,4],10)`
Answer: `Yes`

`sum([1,2,3,4],11)`
Answer: `No`

`sum(A,10)`
Error!!!

Lists: member

```
member(H,[H|_]).
member(X,[H|T]) :- member(X,T).
```

Goals: `?- member(1,[1,2,3,4]).`
Answer: `Yes`

`?- member(10,[1,2,3,4]).`
Answer: `No`

`?- member(A,[1,2,3]).`
Answer: `A=1`
`A=2`
`A=3`

`?- member(1,A).`
Answer: `A=[1|_]`
`A=[_,1|_]`
`A=[_,_,1|_]`
Infinite list of bindings!!