Resolution and Logic Programming

★ Ground resolution
★ Unification and occur check
★ General Resolution
★ Logic Programming
★ SLD-resolution
★ The programming language Prolog
  ⇒ Syntax
  ⇒ Arithmetic
  ⇒ Lists
• We want to show $\Phi \models \Psi$, for two propositional formulas $\Phi$, $\Psi$.

• Assume $\Phi$ is $\Phi_1 \land \cdots \land \Phi_n$, in CNF, and $\Psi$ is $L_1 \land \cdots \land L_m$, 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 $\text{Res}(\chi_1, \chi_2)$ applied to two formulas, and returning a (simpler) formula $\chi$, such that, if $\{\chi_1, \chi_2, \chi\}$ is unsatisfiable, then so is $\{\chi_1, \chi_2\}$. 
• We hope to produce the iteration

\[
\begin{align*}
\{\Phi_1, \ldots, \Phi_n, \neg \Psi\} \\
\{\Phi_1, \ldots, \Phi_n, \neg \Psi, \text{Res}(\neg \Psi, \Phi_{k_1}) = \chi_1\} \\
\{\Phi_1, \ldots, \Phi_n, \neg \Psi, \chi_1, \text{Res}(\chi_1, \Phi_{k_2}) = \chi_2\} \\
\vdots \\
\{\Phi_1, \ldots, \Phi_n, \neg \Psi, \chi_1, \ldots \chi_{l-1}, \text{Res}(\chi_{l-1}, \Phi_{k_l}) = \bot\} \quad \text{—unsatisfiable}
\end{align*}
\]

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 \textit{refutation procedure}.

Motivation (2)
• Given the CNF propositional formula $\Phi \equiv \Phi_1 \land \Phi_n$, where $\Phi_i$ are disjuncts, $1 \leq i \leq n$

• For each $i$, $1 \leq i \leq n$, $\Phi_i \equiv \neg p_{i1} \lor \neg p_{i2} \lor \cdots \lor \neg p_{ik_i} \lor q_{i1} \lor \cdots \lor q_{il_i}$

• $\Phi_i$ is equivalent to $p_{i1} \land \cdots \land p_{ik_i} \rightarrow q_{i1} \lor \cdots \lor q_{il_i}$ which we call a clause.

• We represent the clause by $p_{i1}, \ldots, p_{ik_i} \rightarrow q_{i1}, \ldots, q_{il_i}$

• We represent $\Phi$ as the set of clauses

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

which we call the clausal form of $\Phi$. 
\neg(p_1 \land \cdots \land p_k) \text{ can be written as } p_1 \land \cdots \land p_k \rightarrow \top, \text{ or as } p_1, \ldots, p_k \rightarrow \top

q_1 \lor \cdots \lor q_l \text{ can be written as } \bot \rightarrow q_1, \ldots, q_l, \text{ or as } \rightarrow q_1, \ldots, q_l

\bot \text{ can be written as } \bot \rightarrow \top, \text{ and is denoted by } \Box \text{ (empty clause).}
Given two clauses

\[ \chi_1 : p_1, \ldots, p_k, \ldots p_m \rightarrow q_1, \ldots q_n \]
\[ \chi_2 : r_1, \ldots r_m \rightarrow s_1, \ldots s_l \ldots s_n \]

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

\[ p_1, \ldots, p_{k-1}, p_{k+1}, \ldots, p_m r_1, \ldots, r_m \rightarrow q_1, \ldots, q_n, s_1, \ldots, s_{l-1}, s_{l+1}, \ldots, s_n \]

This is similar to the following cancelling rule in arithmetic.

\[
\begin{align*}
    a + b & = c \\
    c & = d + e \\
    \hline
    a + b + \phi & = \phi + d + e
\end{align*}
\]
Ground Resolution Example

\( \Phi_1 \) is \( p \land q \rightarrow r \)
\( \Phi_2 \) is \( \rightarrow p \)
\( \Phi_3 \) is \( \rightarrow q \)
\( \Psi \) is \( r \rightarrow \)

\( \chi_1 = \text{Res}(\Phi_1, \Psi) \) is \( p, q \rightarrow \)
\( \chi_2 = \text{Res}(\chi_1, \Phi_2) \) is \( q \rightarrow \)
\( \chi_3 = \text{Res}(\chi_2, \Phi_3) \) is \( \Box \)

Alternatively

\( \chi_1 = \text{Res}(\Phi_1, \Phi_2) \) is \( q \rightarrow r \)
\( \chi_2 = \text{Res}(\chi_1, \Phi_3) \) is \( \rightarrow r \)
\( \chi_3 = \text{Res}(\chi_2, \Psi) \) is \( \Box \)
A predicate logic clause:

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

Meaning:

\[ \forall x \forall y \forall z \exists w \left( p(x, y) \land q(f(x), z) \rightarrow r(y, z, w) \lor s(g(z), w) \right) \]

- 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.
Consider the following first order clauses.

\[ \chi_1 : A_1, \ldots, A_k, \ldots, A_m \rightarrow B_1, \ldots, B_n \]
\[ \chi_2 : C_1, \ldots, C_m \rightarrow D_1, \ldots, D_l, \ldots, D_n \]

where the \( A \)s, \( B \)s, \( C \)s, and \( D \)s are first order atoms. Assume there exists a substitution \( \theta \) such that \( A_k\theta = D_l\theta \). We call \( \theta \) a unifier. Then \( \text{Res}(\chi_1\theta, \chi_2\theta) \) is

\[ A_1\theta, \ldots, A_{k-1}\theta, A_{k+1}\theta, \ldots, A_m\theta, C_1\theta, \ldots, C_n\theta \rightarrow B_1\theta, \ldots, B_{m_2}\theta, D_1\theta, \ldots, D_{l-1}\theta, D_{l+1}\theta, \ldots, D_{n_2}\theta \]

which is the same as

\[ (A_1, \ldots, A_{k-1}, A_{k+1}, \ldots, A_m, C_1, \ldots, C_n \rightarrow B_1, \ldots, B_{m_2}, D_1, \ldots, D_{l-1}, D_{l+1}, \ldots, D_{n_2})\theta \]
Non-Ground Resolution Example

\[ \chi_1 : \ p(x, y) \rightarrow q(y, z) \]
\[ \chi_2 : \ q(f(w), v) \rightarrow r(v) \]
\[ \theta : \ [f(w)/y, z/v] \]
\[ \chi_1\theta : \ p(x, f(w)) \rightarrow q(f(w), z) \]
\[ \chi_2\theta : \ q(f(w), z) \rightarrow r(z) \]
\[ \text{Res}(\chi_1\theta, \chi_2\theta) : \ p(x, f(w)) \rightarrow r(z) \]
Given two atoms, $A$, $B$, can we find a unifying substitution $\theta$, such that $A\theta = B\theta$? Answer: YES.

A *most general unifier (mgu)* is a unifying substitution $\theta$ such that for every other unifier $\theta'$, there exists a substitution $\sigma$ such that

$$A\theta' = (A\theta)\sigma$$
$$B\theta' = (A\theta)\sigma$$
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 $t'_1 = t''_1, \ldots, t'_k = t''_k$.
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.
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) \( x = g(y) \)
(2) \( f(x, h(x), y) = f(g(z), w, z) \)

Apply step 5 and replace (2) by

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

Apply step 4 and replace (4) by

(6) \( w = h(x) \)
Example (2)

Current set:

\[(1') \ x = g(y)\]
\[(2') \ x = g(z)\]
\[(3') \ w = h(x)\]
\[(4') \ y = z\]

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

\[(1'') \ x = g(y)\]
\[(2'') \ g(y) = g(z)\]
\[(3'') \ w = h(g(y))\]
\[(4'') \ y = z\]

Replace \((2'')\) by

\[y = z \leftarrow \text{already in the set}\]

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

\[x = g(z)\]
\[w = h(g(z))\]
\[y = z\]

Substitution:

\[\left[g(z)/x, h(g(z))/w, z/y\right]\]
Example (3)

\[ p(x, f(x, h(x), y))[g(z)/x, h(g(z))/w, z/y] \text{ is } \]
\[ p(g(z), f(g(z), h(g(z)), z)) \]

\[ p(g(y), f(g(z), w, z))[g(z)/x, h(g(z))/w, z/y] \text{ is } \]
\[ p(g(z), f(g(z), h(g(z)), z)) \]
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:

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

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.
Consider the following first order clauses.

\[ \chi_1 : A_1, \ldots, A_k, \ldots, A_m \rightarrow B_1, \ldots, B_n \]
\[ \chi_2 : C_1, \ldots, C_m \rightarrow D_1, \ldots, D_l, \ldots, D_n \]

where the \( A_s, B_s, C_s, \) and \( D_s \) are first order atoms. Denote by \( \theta \) the mgu of \( A_k \) and \( D_l \). Then \( \text{Res}(\chi_1, \chi_2) \) is

\[
(A_1, \ldots, A_{k-1}, A_{k+1}, \ldots, A_m, C_1, \ldots, C_n \rightarrow B_1, \ldots, B_m, D_1, \ldots, D_{l-1}, D_{l+1}, \ldots, D_n) \theta
\]

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 \( \chi_1, \chi_2 \in S_i \) such that \( \text{Res}(\chi_1, \chi_2) \) is defined. If \( \text{Res}(\chi_1, \chi_2) = \square \), the original set \( S \) is unsatisfiable. Otherwise, construct \( S_{i+1} = S_i \cup \text{Res}(\chi_1, \chi_2) \). If \( S_{i+1} = S_i \) for all possible pairs \( \chi_1 \) and \( \chi_2 \), then \( S \) is satisfiable.
Example of General Resolution

Original set:

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

Application of the resolution procedure:

8. \( q(a) \rightarrow [a/x] \quad 3,6 \)
9. \( \rightarrow q(a), s(f(a)) [a/x] \quad 2,4 \)
10. \( \rightarrow s(f(a)) \quad 8,9 \)
11. \( \rightarrow q(a), r(a, f(a)) [a/x] \quad 1,4 \)
12. \( \rightarrow r(a, f(a)) \quad 8,11 \)
13. \( \rightarrow t(f(a)) [f(a)/y] \quad 5,12 \)
14. \( s(f(a)) \rightarrow [f(a)/x] \quad 7,13 \)
15. \( \square \quad 10,14 \)
Soundness: If the unsatisfiable clause \( \square \) 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 \( \square \) can be derived by the resolution procedure.
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.
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 $\Gamma$. 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. \( \leftarrow p(d,a) \) \([d/x,a/y] \)
8. \( \leftarrow p(d,z), q(z,a) \) \([d/x,a/y] \)
9. \( \leftarrow q(b,a) \) \([b/z] \)
10. \( \leftarrow p(b,a) \) \([b/x,a/y] \)
11. □
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 applying 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

\[
\text{ancestor}(X, Y) :\quad \text{parent}(X, Y).
\]
\[
\text{ancestor}(X, Y) :\quad \text{parent}(X, Z), \text{ancestor}(Z, Y).
\]

\[
\begin{align*}
\text{parent}(\text{bob}, \text{allen}). \\
\text{parent}(\text{catherine}, \text{allen}). \\
\text{parent}(\text{dave}, \text{bob}). \\
\text{parent}(\text{ellen}, \text{bob}). \\
\text{parent}(\text{fred}, \text{dave}). \\
\text{parent}(\text{harry}, \text{george}). \\
\text{parent}(\text{ida}, \text{george}). \\
\text{parent}(\text{joe}, \text{harry}).
\end{align*}
\]

Goal: \text{ancestor}(\text{fred}, \text{bob})
Answer: Yes

Goal: \text{ancestor}(\text{fred}, A)
Answer: A=dave
A=bob
A=allen

Goal: \text{ancestor}(A, \text{allen})

Goal: \text{ancestor}(A, B)
ancestor(fred, A)
[fred/X,A/Y] [fred/X,A/Y]

parent(fred, A) parent(fred, Z), ancestor(Z, A)
[dave/A] [dave/Z]

ancestor(dave, A)
[A/Y] [dave/X,A/Y]

parent(dave, A) parent(dave, Z), ancestor(Z, A)
[bob/A] [bob/Z]

ancestor(bob, A)
[bob/X,A/Y] [bob/X,A/Y]

parent(bob, A) parent(bob, Z), ancestor(Z, A)
[allen/A] [allen/Z]

ancestor(allen, A)
[allen/X,A/Y] [allen/X,A/Y]

parent(allen, A) parent(allen, Z), ancestor(Z, A)
fail fail
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.
The predicate \textit{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 \textit{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 \textit{is}
Correct program:

\[
\text{factorial}(0,1).
\]
\[
\text{factorial}(N,X) \leftarrow
\quad N > 0, \ N1 \text{ is } N-1, \ \text{factorial}(N1,X1), \ X \text{ is } X1*N.
\]

Goal: \(?- \ \text{factorial}(5,X)\).
Answer: \textcolor{red}{X=120}

Wrong program:

\[
\text{factorial}(0,1).
\]
\[
\text{factorial}(N,X) \leftarrow
\quad N > 0, \ N1 \text{ is } N-1, \ X \text{ is } X1*N, \ \text{factorial}(N1,X1).
\]

Goal: \(?- \ \text{factorial}(5,X)\).
Error!!!
Examples of lists:

- \([1, 2, 3, 4]\)
- \([]\) — empty list.
- \([1|2, 3, 4]\) — same as \([1, 2, 3, 4]\),
  same as \(|(1, |(2, |(3, |(4, \text{nil}))))\)

?- \([\text{H}|\text{T}] = [1, 2, 3, 4].\)
Answer: \(\text{H}=1, \text{T}=[2, 3, 4]\)

?- \(\text{H}=a, \text{T}=[b, c, d], \text{X}=[\text{H}|\text{T}].\)
Answer: \(\text{H}=a, \text{T}=[b, c, d], \text{X}=[a, b, c, d]\)

Warning:

?- \(\text{H}=[a, b, c], \text{T}=[d, e, f], \text{X}=[\text{H}|\text{T}]\)
Answer: \(\text{X}=[[a, b, c], d, e, f]\)

\([\text{H}|\text{T}]\) is syntactic sugar for \(|(\text{H}, \text{T})\).
\([]\) is syntactic sugar for \(\text{nil}\).
Lists: append

\[
\begin{align*}
\text{append}([], X, X). \\
\text{append}([H|T], X, [H|T1]) & :\text{-} \text{append} (T, X, T1).
\end{align*}
\]

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

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

Goal: \(?- \text{append}(A, B, [1, 2, 3]).\)  
Answer: \(A= [], B= [1, 2, 3]\)  
\(A= [1], B= [2, 3]\)  
\(A= [1, 2], B= [3]\)  
\(A= [1, 2, 3], B= []\)
sum([],0).
sum([H|T],X) :- sum(T,X1), X is X1+H.

Goals:

\[ \text{sum}([1,2,3,4],X) \]
Answer: \textbf{A=10}

\[ \text{sum}([1,2,3,4],10) \]
Answer: \textbf{Yes}

\[ \text{sum}([1,2,3,4],11) \]
Answer: \textbf{No}

\[ \text{sum}(A,10) \]
Error!!!
Lists: \texttt{member}

\begin{verbatim}
member(H, [H|_]).
member(X, [H|T]) :- member(X, T).
\end{verbatim}

Goals:

?- member(1, [1,2,3,4]).
Answer: \texttt{Yes}

?- member(10, [1,2,3,4]).
Answer: \texttt{No}

?- member(A, [1,2,3]).
Answer: \texttt{A=1}
\hspace{1cm} \texttt{A=2}
\hspace{1cm} \texttt{A=3}

?- member(1, A).
Answer: \texttt{A=[1|_]}
\hspace{1cm} \texttt{A=[_,1|_]}
\hspace{1cm} \texttt{Infinite list of bindings!!}