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 \( \Phi, \Psi \).
- 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 \( \Phi \models \neg \Psi \) is equivalent with showing that the set of formulas \( \{ \Phi_1, \ldots, \Phi_n, \neg \Psi \} \) is unsatisfiable.
- \textbf{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 \( \{ \chi_1, \chi_2 \} \).

Motivation (2)

- We hope to produce the iteration
  \[
  \{ \Phi_1, \ldots, \Phi_n \rightarrow \top \},
  \{ \Phi_1, \ldots, \Phi_n \rightarrow \top, \text{Res}(\neg \Psi, \Phi_0) = \chi_1 \},
  \{ \Phi_1, \ldots, \Phi_n \rightarrow \top, \text{Res}(\chi_1, \Phi_0) = \chi_2 \}
  \]
  \[
  \vdots
  \]
  \[
  \{ \Phi_1, \ldots, \Phi_n \rightarrow \top, \chi_1, \ldots, \chi_{k-1}, \text{Res}(\chi_{k-1}, \Phi_0) = \bot \} \quad \text{unsatisfiable}
  \]
  where \( 1 \leq k \leq n, 1 \leq i \leq l. \)
- 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{resolution procedure}.

CNF and Clausal Form (1)

- Given the CNF propositional formula \( \Phi \equiv \Phi_1 \land \cdots \land \Phi_n \), where \( \Phi_i \) are disjuncts, \( 1 \leq i \leq n \).
- For each \( i, 1 \leq i \leq n, \Phi_i \equiv p_{i1} \lor \cdots \lor p_{ik} \lor q_{i1} \land \cdots \land q_{il} \).
- \( \Phi \) is equivalent to \( p_{11} \land \cdots \land p_{ik} \lor q_{11} \land \cdots \land q_{il} \) which we call a \textit{clause}.
- We represent the clause by \( p_{i1} \cdots p_{ik} \rightarrow q_{i1} \cdots q_{il} \).
- We represent \( \Phi \) as the set of clauses
  \[
  \{ (p_{i1} \cdots p_{ik} \rightarrow q_{i1} \cdots q_{il}), \cdots | 1 \leq i \leq n \}
  \]
  which we call the \textit{clausal form} of \( \Phi \).

CNF and Clausal Form (2)

\( \neg(p_1 \land \cdots \land p_k) \) can be written as \( p_1 \land \cdots \land p_k \rightarrow \top \), or as \( p_1 \cdots p_k \rightarrow \neg \).
\( q_1 \lor \cdots \lor q_l \) can be written as \( \bot \rightarrow q_1, \ldots, q_l \),
or as \( q_1, \ldots, q_l \rightarrow \bot \).
\( \bot \) can be written as \( \bot \rightarrow \top \), and is denoted by \( \square \) (empty clause).

Ground Resolution

Given two clauses
\[
\chi_1 : p_{i1} \cdots p_{ik} \rightarrow q_{i1} \cdots q_{il},
\chi_2 : q_{s1} \cdots q_{sl} \rightarrow s_{l1} \cdots s_{lk}.
\]
If \( p_i \) and \( q_i \) are the same propositional symbol, then \( \text{Res}(\chi_1, \chi_2) \) is
\[
p_{i1} \cdots p_{ik} \rightarrow q_{i1} \cdots q_{il} \land p_{s1} \cdots p_{sl} \rightarrow s_{l1} \cdots s_{lk}.
\]
This is similar to the following cancelling rule in arithmetic.
\[
a+b=c \quad \Rightarrow \quad a+c=d+e
\]
\[
a+b+d = d+c+e.
\]
Ground Resolution Example

ϕ₁ is \( p \land q \rightarrow r \)
ϕ₂ is \( p \rightarrow p \)
ϕ₃ is \( q \rightarrow q \)
ψ is \( r \rightarrow r \)

\( \chi₁ = \text{Res}(ϕ₁, ψ) \) is \( p \land q \rightarrow r \)
\( \chi₂ = \text{Res}(ϕ₂, ψ) \) is \( q \rightarrow p \)
\( \chi₃ = \text{Res}(ϕ₃, ψ) \) is \( q \rightarrow q \)

Alternatively
\( \chi₁ = \text{Res}(ϕ₁, ψ) \) is \( q \rightarrow r \)
\( \chi₂ = \text{Res}(ϕ₂, ψ) \) is \( r \)
\( \chi₃ = \text{Res}(ϕ₃, ψ) \) is \( \Box \)

Predicate Logic Clauses

A predicate logic clause:

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

Meaning:

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

- 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.

Non-Ground Resolution

Consider the following first order clauses.

\( \chi₁ : A₁ \land A₂ \land \ldots \land Aₙ \rightarrow B₁ \land B₂ \)
\( \chi₂ : C₁ \land C₂ \land \ldots \land Cₙ \rightarrow D₁ \land D₂ \)

where the \( Aᵢ, Bᵢ, Cᵢ, \) and \( Dᵢ \) are first order atoms. Assume there exists a substitution \( θ \) such that \( Aθ = Dθ \). We call \( θ \) a unifier. Then \( \text{Res}(χ₁, χ₂) \) is

\( Aθ \land Aθ \land \ldots \land Aθ \land B₁ \land B₂ \land C₁ \land C₂ \land \ldots \land Cₙ \land D₁ \land D₂ \)

which is the same as

\( (A₁ \land \ldots \land Aₙ \land B₁ \land \ldots \land Bₙ \land C₁ \land \ldots \land Cₙ \land D₁ \land \ldots \land Dₙ)θ \)

Unification, MGU

Given two atoms, \( A, B \), can we find a unifying substitution \( θ \) such that \( Aθ = Bθ \)? 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

\( Aθ' = (Aθ)σ \)
\( Bθ' = (Aθ)σ \)

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. Form \( p₁(x₁, \ldots, xₖ) = p₂(x₁', \ldots, xₖ') \) derive these of equations \( x₁ = x₁', \ldots, xₖ = xₖ' \).
3. Enum all equations of the form \( x = x' \), where \( x \) is a variable.
4. Transform all equations of the form \( t = s \), where \( t \) is not a variable, into \( s = t \).
5. Let \( f = f' \) be an equation where \( f \) and \( f' \) are not variables. If the function symbols of \( f \) and \( f' \) are not identical return "no solution." Otherwise, replace the equation \( f(x₁, \ldots, xₖ) = f'(x₁', \ldots, xₖ') \) by the equations \( x₁ = x₁', \ldots, xₖ = xₖ' \).
6. Let \( x = y \) 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 \( y \).

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 \( x = f \), where \( f \) does not contain \( x \). The mgu contains all the bindings \( f/x \) where \( x = f \) is an equation in our set.
Example of Applying the Unification Algorithm

Unify the atoms:
\[ p(x, f(x, h(x)), y) \] and \[ p(g(z), f(g(z)), w, z) \]

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. \[ y = z \]

Replace \((2'')\) by

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

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

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
\[ p(x_1, x_2, \ldots, x_n, y_1) \] and \[ p(f(x_1, x_2, \ldots, x_n), f(x_1, x_2, \ldots, x_n)) \]

This produces:

1. \[ x_1 = f(y_1, y_2) \]
2. \[ x_2 = f(f(x_1, x_2, \ldots, x_n)) \]
3. \[ x_3 = f(f(f(x_1, x_2, \ldots, x_n), f(f(x_1, x_2, \ldots, x_n)))) \]

.....

\[ x_n = \text{term with } 2^n \text{ occurrences of } y_1 \]

\[ x_0 = \text{term with } 2^{n-1} \text{ occurrences of } y_0 \]

Using step 6, we must return “no solution”; detecting the fact that \( y_0 \) occurs in the right hand side of last equation may require exponential time.

General Resolution

Consider the following first order clauses:

\[ X_1 : A_1 \rightarrow B_1 \]
\[ X_2 : C_1 \rightarrow B_1 \]

where the \( A_i, B_i, C_i \) and \( D_i \) are first order atoms. Denote by \( T \) the sup of all \( A_i \) and \( D_i \). Then \( \text{Res}(X_1, X_2) \) is

\[ \text{Res}(X_1, X_2) = \{ A_1 \rightarrow B_1; \quad \ldots; \quad A_n \rightarrow B_n; \quad C_1 \rightarrow B_1; \quad \ldots; \quad C_n \rightarrow B_1; \quad D_1 \rightarrow B_1; \quad \ldots; \quad D_n \rightarrow B_1 \} \]

If there exist no two unifiable atoms \( A_i \) and \( D_j \) then the resolution rule is undefined.

Resolution Procedure: Let \( S \) be a set of clauses and define \( S_0 = S \). Assume that \( S_n \) has been constructed. Choose two clauses \( X_i, X_j \in S_i \) such that \( \text{Res}(X_i, X_j) \) is defined. If \( \text{Res}(X_i, X_j) = \emptyset \), the original set \( S \) is unsatisfiable. Otherwise, construct \( S_{i+1} = S_i \cup \text{Res}(X_i, X_j) \). If \( S_{i+1} = S \) for all possible pairs \( j \) and \( j \) then \( S \) is satisfiable.

Example of General Resolution

Original set:

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

Application of the resolution procedure:

1. \[ p(x) \rightarrow q(x, f(x)) \rightarrow [a/x] \] 3, 6
2. \[ p(x) \rightarrow q(x, s(x)) \rightarrow [a/x] \] 2, 4
3. \[ r(a) \rightarrow r(f(a)) \rightarrow 8, 9
4. \[ r(a) \rightarrow r(f(a)) \rightarrow [a/x] \] 1, 4
5. \[ r(a) \rightarrow r(f(a)) \rightarrow 8, 11
6. \[ r(a, f(x)) \rightarrow r(x) \rightarrow 9, 12
7. \[ r(f(a), x) \rightarrow r(f(a)) \rightarrow 8, 13
8. \[ r(f(a)) \rightarrow r(f(a)) \rightarrow 9, 14
9. \[ r(f(a)) \rightarrow r(f(a)) \rightarrow 10, 15
10. \[ r(f(a)) \rightarrow r(f(a)) \rightarrow 11, 14
11. \[ r(f(a)) \rightarrow r(f(a)) \rightarrow 12, 15
12. \[ r(f(a)) \rightarrow r(f(a)) \rightarrow 13, 15
13. \[ r(f(a)) \rightarrow r(f(a)) \rightarrow 14, 15
14. \[ r(f(a)) \rightarrow r(f(a)) \rightarrow 15, 15
15. \[ □ \] 10, 14
**Soundness and Completeness of Resolution**

**Soundness:** If the unsatisfiable clause □ 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 □ can be derived by the resolution procedure.

---

**Logic Programming**

From now on, instead of writing clauses as

\[ A_1, \ldots, A_n \rightarrow B_1, \ldots, B_m \]

we shall prefer to write clauses as

\[ B_1, \ldots, B_m \leftarrow A_1, \ldots, A_n \]

For \( n = 1 \) we have Horn clauses, typically denoted as

\[ H \leftarrow A_1, \ldots, A_n \]

\( H \) — the head, \( A_1, \ldots, A_n \) — 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 \( \Gamma \).

Typical search rule: clauses are tried in the order in which they are written.

---

**Example of Resolution for Logic Programs**

<table>
<thead>
<tr>
<th>Logic program</th>
<th>Applying the resolution procedure, with computation and search rules</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. ( q(x,y) \leftarrow p(x,y) )</td>
<td>7. ( \leftarrow p(x,a) ) [( x/a \rightarrow y )] 6,1</td>
</tr>
<tr>
<td>2. ( q(x,y) \leftarrow p(x,z,q(x,z)) )</td>
<td>8. ( \leftarrow p(x,a) [( x/a \rightarrow y )] 7,2</td>
</tr>
<tr>
<td>3. ( p(x,a) \leftarrow )</td>
<td>9. ( \leftarrow q(x,y) [( y/a \rightarrow z )] 8,5</td>
</tr>
<tr>
<td>4. ( p(x,b) \leftarrow )</td>
<td>10. ( \leftarrow p(x,a) [( x/a \rightarrow y )] 9,1</td>
</tr>
<tr>
<td>5. ( p(x,b) \leftarrow )</td>
<td>11. □</td>
</tr>
<tr>
<td>6. Goal: ( \leftarrow q(x,a) )</td>
<td>10,3</td>
</tr>
</tbody>
</table>

---

**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 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 \( \leftarrow \) operator.
- The dot . acts as a clause separator.

---

**Prolog Example**

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

**parent**(bob, allen), **parent**(catherine, allen),

**parent**(dave, bob), **parent**(ellen, bob),

**parent**(harry, george), **parent**(idos, george),

**parent**(joe, harry).

**Goal:** ancestor(fred, bob)

**Answer:** Yes

**Goal:** ancestor(fred, a)

**Answer:** A:bob

**Goal:** ancestor(A, allen)

**Goal:** ancestor(A, Z)
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**:

- $\exists x \; x \equiv 2 \cdot 3$.
  - Answer: Yes

- $\exists x \; x \equiv 2 \cdot 3$.
  - Answer: Yes

- $\exists x \; x \equiv 2 \cdot 3$.
  - Answer: Yes

- $\exists x \; x \equiv 2 \cdot 3$.
  - Answer: Yes

- $\exists x \; x \equiv 2 \cdot 3$.
  - Answer: Yes

A Factorial Program

Correct program:

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

Goal: $\exists \; \text{factorial}(5, X)$.

Answer: $X = 120$

Wrong program:

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

Goal: $\exists \; \text{factorial}(5, X)$.

Error!!!

Lists (By Example)

Examples of lists:

- $[1, 2, 3, 4]$ is empty list.
- $[1, 2, 3, 4]$ is same as $[1, 2, 3, 4]$.
- $[1, 2, 3, 4]$ is same as $[1, 2, 3, 4]$.
- $[1, 2, 3, 4]$ is same as $[1, 2, 3, 4]$.

- $X = [1, 2, 3, 4]$.
  - Answer: Yes

- $X = [1, 2, 3, 4]$.
  - Answer: Yes

- $X = [1, 2, 3, 4]$.
  - Answer: Yes

Warning:

- $X = [a, b, c, d, e, f]$.
  - Answer: Yes

- $X = [a, b, c, d, e, f]$.
  - Answer: Yes

- $X = [a, b, c, d, e, f]$.
  - Answer: Yes

- $X = [a, b, c, d, e, f]$.
  - Answer: Yes

| [HT] is syntactic sugar for $[1, 2, 3]$.
| [H] is syntactic sugar for nil.

Lists:

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

Goal: $\exists \; \text{append}([a, b, c], [d, e, f], A)$.

Answer: $A = [a, b, c, d, e, f]$

Goal: $\exists \; \text{append}([a, b, c], A, [a, b, c, d, e, f])$.

Answer: $A = [a, b, c, d, e, f]$

Goal: $\exists \; \text{append}([a, b, c, d, e, f])$.

Answer: $A = [a, b, c, d, e, f]$

```

```
Lists: Sum of All Elements

\[
\begin{align*}
    \text{sum}([1,0]) &= \text{sum}(HT), X \text{ is } X+H. \\
    \text{sum}([H,T],X) &\leftarrow \text{sum}(T, X), X \text{ is } X+H.
\end{align*}
\]

<table>
<thead>
<tr>
<th>Goals</th>
<th>Answer</th>
</tr>
</thead>
<tbody>
<tr>
<td>\text{sum}([1,2,3,4],X)</td>
<td>A=10.0</td>
</tr>
<tr>
<td>\text{sum}([1,2,3,4],10)</td>
<td>Yes</td>
</tr>
<tr>
<td>\text{sum}([1,2,3,4],11)</td>
<td>No</td>
</tr>
<tr>
<td>\text{sum}(A,10)</td>
<td>Error!!!</td>
</tr>
</tbody>
</table>

Lists: member

\[
\begin{align*}
    \text{member}(L,[H,L]) &= \text{member}(X,[H,T]) \leftarrow \text{member}(T,X). \\
    \text{member}(L,[H,L]) &\leftarrow \text{member}(L,A).
\end{align*}
\]

<table>
<thead>
<tr>
<th>Goals</th>
<th>Answer</th>
</tr>
</thead>
<tbody>
<tr>
<td>?= \text{member}(1,[1,2,3,4])</td>
<td>A=[1,1]</td>
</tr>
<tr>
<td>?= \text{member}(10,[1,2,3,4])</td>
<td>A=[1,1]</td>
</tr>
<tr>
<td>?= \text{member}(A,[1,2,3])</td>
<td>Infinite list of bindings!!</td>
</tr>
<tr>
<td>?= \text{member}(A,[1,2,3])</td>
<td>A=1</td>
</tr>
<tr>
<td>?= \text{member}(A,[1,2,3])</td>
<td>A=2</td>
</tr>
<tr>
<td>?= \text{member}(A,[1,2,3])</td>
<td>A=3</td>
</tr>
</tbody>
</table>