AI Fundamentals: rule-based systems

Maria Simi
Rule-based systems: the basics

LESSON 1- SLD RESOLUTION
Rules are everywhere

1. Expert systems
2. Natural language processing, e.g. chatbots
3. Rule-based programming
4. Business logics in organizations
5. End user programming, e.g. in domotics
Resolution theorem proving

Given the fundamental problem KB ⊨ α an equivalent problem is “KB ∪ ¬α is unsatisfiable”

This can be solved by a deductive system showing that KB ∪ ¬α ⊨ { }, where { } is the empty clause meaning False.

**Resolution by refutation** is a method which is correct and complete:

1. transform the KB in clausal form (a conjunction of disjunction of literals)
2. add to KB the negation of the goal in clausal form
3. use the resolution rule as unique inference rule.

This strategy works for PROP and FOL with different complexity results:

1. It is decidable and NP-complete for PROP
2. It is semi-decidable for FOL.
Resolution rule

Clauses are set of literals, i.e. atomic formulas or their negation: \( \{p_1, p_2, \ldots, p_k\} \)

**Resolution rule for PROP** (\( c_1 \) and \( c_2 \) are clauses):

\[
\begin{array}{c}
\frac{c_1 \cup \{p\} \quad c_2 \cup \{\neg p\}}{c_1 \cup c_2 \quad \text{(the resolvent)}}
\end{array}
\]

In a resolution refutation we aim to deduce the empty clause:

\[
\begin{array}{c}
\frac{\{p\} \quad \{\neg p\}}{\{\}}
\end{array}
\]

**Resolution rule for FOL:**

\[
\begin{array}{c}
\frac{c_1 \cup \{p\} \quad c_2 \cup \{\neg q\}}{[c_1 \cup c_2] \gamma}
\end{array}
\]

and \( \text{MGU}(p, q) = \gamma \) and \( \gamma \) is not fail

a fundamental operation is **unification**
Resolution in the predicate calculus

**Unification:** is a process to determine whether two expressions can be made identical by a substitution of terms to variables.

The result is the unifying substitution, the **unifier**, or FAIL, if the expressions are not unifiable.

For example:

\[
\{ P(f(y), A), Q(B, C) \} \quad \{ \neg P(x, A), R(x, C), S(A, B) \} \quad x \text{ and } y \text{ are variables}
\]

With the substitution \{\( x/f(y) \)\} the literal \( \neg P(x, A) \) becomes \( \neg P(f(y), A) \), and it is possible to apply the resolution rule.

The resolvent is:

\[
\{ Q(B, C), R(f(y), C), S(A, B) \}
\]

Given two expressions there may be different substitutions that make them identical. We are interested in computing the **most general unifier (MGU)**, the one that does only the essential instantiations.
Unification algorithm [Martelli, Montanari, 1982]

- Computes the MGU by means of a rule-based equation-rewriting system
- Initially the working memory (WM) contains the equality of the two expressions to be unified
- The rules modify the equations in the WM
- The algorithm terminates with failure or when there are no applicable rules (success)
- At the end, if there is no failure, the WM contains the MGU.

*Note*: different from the AIMA unification algorithm but easier to understand.
The rules

1. \( f(s_1, \ldots, s_n) = f(t_1, \ldots, t_n) \rightarrow s_1 = t_1, \ldots, s_n = t_n \)
2. \( f(s_1, \ldots, s_n) = g(t_1, \ldots, t_m) \rightarrow \text{fail when } f \neq g \text{ or } n \neq m \)
3. \( x = x \rightarrow \text{remove equation} \)
4. \( t = x \rightarrow x = t \quad \text{bring variable to the left} \)
5. \( x = t, x \text{ does not occur in } t \rightarrow \text{apply } \{x/t\} \text{ to other equations} \)
6. \( x = t, t \text{ is not } x, x \text{ occur in } t \rightarrow \text{fail (occur check)} \)

Nota: when comparing two different constants, rule 2 applies, as a special case where \( n=m=0 \), and we fail.
Unification algorithm: example 1

Computing the MGU of $P(A, y, z)$ and $P(x, B, z)$

Step 3

$x = A$

$y = B$

MGU!
Unification algorithm: example 2

Computing the MGU of $P(f(x), x)$ and $P(z, z)$

Step 3

$z = f(x)$

$x = f(x)$ rule 6

FAIL!

(occurr check)
Reasoning with Horn clauses

By limiting expressivity to only a certain interesting subset of first-order logic, resolution procedures becomes much more manageable.

We limit the degree of uncertainty we can express by considering clauses that have at most one positive literal (Horn clauses). Three cases:

1. **Rules:**
   
   \[
   \{\neg \text{Child}, \neg \text{Male}, \text{Boy}\}
   \]
   
   is logically equivalent to
   
   \[
   \text{Child} \land \text{Male} \Rightarrow \text{Boy}
   \]
   
   which has a natural interpretation as a rule.

2. **Facts:** \{Child\}

3. **Goals or queries:** \{\neg \text{Boy}\} only negative literals (negative clauses)

Propositional Horn clauses have linear-time deduction algorithms.
Goal trees are and-or trees

1. Toddler
2. Baby ⇒ Child
3. Toddler ⇒ Child
4. Child, Male ⇒ Boy
5. Infant ⇒ Child
6. Child, Female ⇒ Girl
7. Female

GOAL: Girl
Example of SLD resolution

\{\text{Toddler}\}
\{\neg \text{Baby, Child}\}
\{\neg \text{Toddler, Child}\}
\{\neg \text{Child, } \neg \text{Male, Boy}\}
\{\neg \text{Infant, Child}\}
\{\neg \text{Child, } \neg \text{Female, Girl}\}
\{\text{Female}\}

GOAL: \{\neg \text{Girl}\}

A SLD derivation of a clause \( c \) from \( S \) is a sequence of clauses \( c_1, c_2, \ldots, c_n \), such that \( c_n = c \), \( c_1 \in S \), and \( c_{i+1} \) is a resolvent of \( c_i \) and some clause in \( S \).

\( S \vdash_{\text{SLD}} A \) iff \( S \vdash A \)
Logic programs

A logic program is a set of **definite** Horn clauses (facts and rules).

\[ A :- B_1, B_2, ..., B_n \] (A head, \( B_1, B_2, ..., B_n \) body)

**Declarative interpretation**

A is true.

A is true if \( B_1, B_2, ..., B_n \) are all true.

The goal (query) is a negative clause whose logical meaning is \( \neg (G_1 \land G_2 \land ... \land G_k) \)

written as \( ?- G_1, G_2, ..., G_k \)

**Procedural interpretation**

The head of a rule can be seen as a function call and the body as functions to be called in sequence. When they all return the main procedure returns.
Example of logic program

1. parent(X, Y) :- father(X, Y).
2. parent(X, Y) :- mother(X, Y).
3. ancestor(X, Y) :- parent(X, Y).
4. ancestor(X, Y) :- parent(X, Z), ancestor(Z, Y).
5. father(john, mark).
6. father(john, luc).
7. mother(lia, john).

8. ?- ancestor(lia, mark) the negation of the goal
SLD resolution

Given a logic program and a goal \( G_1, G_2, \ldots, G_k \) the SLD goal tree is constructed as follows. Each node of the tree corresponds to a conjunctive goal to be solved.

- The root node is \( ?- G_1, G_2, \ldots, G_k \)
- Let \( ?- G_1, G_2, \ldots, G_k \) a node in the tree; the node successors are obtained by considering the facts and rules in the program whose head is unifiable with \( G_1 \)
  - If \( A : - B_1, \ldots, B_m \) is a rule and \( \gamma = \text{MGU}(A, G_1) \), a descendent is the new goal \( ?- (B_1, \ldots, B_m, G_2, \ldots, G_k) \gamma \)
  - If \( A \) is a fact and \( \gamma = \text{MGU}(A, G_1) \), a descendent is the new goal \( ?- (G_2, \ldots, G_k) \gamma \)

**Note:** variables in rules are renamed before using them.

- Nodes that correspond to empty clauses are successes.
- Nodes without successors are failures.
SLD tree for goal \textit{ancestor}(lia, mark)

1. parent(X, Y) :- father(X, Y).
2. parent(X, Y) :- mother(X, Y).
3. ancestor(X, Y) :- parent(X, Y).
4. ancestor(X, Y) :- parent(X, Z), ancestor(Z, Y).
5. father(john, mark).
6. father(john, luc).
7. mother(lia, john).
Computed answers

In addition to ground goals such as `?-ancestor(lia, mark)` whose answer is YES or NO, we can also have goals with variables such as

`?- ancestor(X, mark).`

The logical meaning of the query is:

\[ KB \models \exists X \ ancestor(X, mark) \]  
Are there ancestors of Mark?

\[ KB \cup \neg \exists X \ ancestor(X, mark) \]  
unsatisfiable?

\[ KB \cup \{ \neg ancestor(X, mark) \} \vdash_{RES} \{ \} \]  
In this case the expected answer are the values that \( X \) is bound to during the resolution proof.

\( X=lia; X=john \)

Similarly: `?- ancestor(lia, Y)`

returns all the descendants of Lia (\( Y=\text{john}, Y=\text{mark}, Y=\text{luc} \)).
The SLD resolution strategy is complete for definite Horn clauses. This means that if $P \cup \{\neg G\}$ is unsatisfiable, then at least one of the leaves of the goal tree produces the empty clause (success). Moreover trying to satisfy the subgoals in the order they appear it is not restrictive, since in the end all of them must be satisfied. When there are variables in the goal, the substitution that we obtain is the computed answer. Completeness and efficiency are however influenced by:

- the order of expansion of the nodes (the visit strategy)
- the order in which we consider successor nodes at each level
- the order of literals in the body
Your turn

- Make sure you understand all these premises.
- Do something useful for the IIA students: implement the rule-based version of UNIFICATION to be used in connection with the AIMA code in file logic.py.
- Get familiar with the online version of SWISH Prolog (if you prefer, you can install it on your PC).

https://swish.swi-prolog.org/
References