Lecture 8: Prolog

2-AIN-108 Computational Logic

Martin Baláž, Martin Homola

Department of Applied Informatics
Faculty of Mathematics, Physics and Informatics
Comenius University in Bratislava

18 Nov 2014

Example

Logic Program:

$$
\begin{aligned}
father(abraham, isaac) &\leftarrow \\
mother(sarah, isaac) &\leftarrow \\
father(isaac, jacob) &\leftarrow \\
parent(X, Y) &\leftarrow father(X, Y) \\
parent(X, Y) &\leftarrow mother(X, Y) \\
ancestor(X, Y) &\leftarrow parent(X, Y) \\
ancestor(X, Z) &\leftarrow parent(X, Y), ancestor(Y, Z)
\end{aligned}
$$

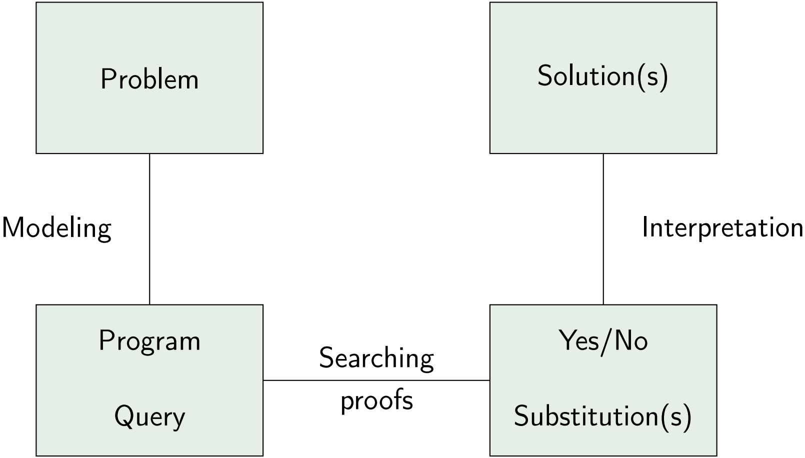
Query:

$$(\exists X)(\exists Y) ancestor(X, Y)?$$

Answer:

Yes for $X = abraham, Y = isaac; X = sarah, Y = isaac; X = abraham, Y = jacob.$

Programming with Prolog



First-Order Logic

$T \models (\exists X)(\exists Y) ancestor(X, Y)$ iff

for all interpretations I holds
$I \models T \Rightarrow I \models (\exists X)(\exists Y) ancestor(X, Y)$ iff

there does not exist an interpretation I such that $I \models T$ but
$I \not\models (\exists X)(\exists Y) ancestor(X, Y)$ iff

there does not exist an interpretation I such that $I \models T$ and
$I \models (\forall X)(\forall Y) \neg ancestor(X, Y)$ iff

there does not exist an interpretation I such that
$I \models T \cup \{(\forall X)(\forall Y) \neg ancestor(X, Y)\}$ iff

$T \cup \{(\forall X)(\forall Y) \neg ancestor(X, Y)\}$ is unsatisfiable

First-Order Logic

(a) $father(abraham, isaac)$
(b) $mother(sarah, isaac)$
(c) $father(isaac, jacob)$
(d) $\neg father(X, Y) \vee parent(X, Y)$
(e) $\neg mother(X, Y) \vee parent(X, Y)$
(f) $\neg parent(X, Y) \vee ancestor(X, Y)$
(g) $\neg parent(X, Y) \vee \neg ancestor(Y, Z) \vee ancestor(X, Z)$

(1) $\neg ancestor(X, Y)$ (Query)
(2) $\neg parent(X, Y)$ (Resolution of 1 and f using $\theta_1 = \{\}$)
(3) $\neg father(X, Y)$ (Resolution of 2 and d using $\theta_2 = \{\}$)
(4) $\perp$ (Resolution of 3 and a using $\theta_3 = \{X/abraham, Y/isaac\}$)

$P \models (\exists X)(\exists Y) ancestor(X, Y)$ iff

$P \cup \{(\forall X)(\forall Y) \neg ancestor(X, Y)\}$ is unsatisfiable iff

$P \cup \{\leftarrow ancestor(X, Y)\}$ is unsatisfiable

(a) $father(abraham, isaac) \leftarrow$

(b) $mother(sarah, isaac) \leftarrow$

(c) $father(isaac, jacob) \leftarrow$

(d) $parent(X, Y) \leftarrow father(X, Y)$

(e) $parent(X, Y) \leftarrow mother(X, Y)$

(f) $ancestor(X, Y) \leftarrow parent(X, Y)$

(g) $ancestor(X, Z) \leftarrow parent(X, Y), ancestor(Y, Z)$

(1) $\leftarrow ancestor(X, Y)$ (Query)

(2) $\leftarrow parent(X, Y)$ (Resolution of 1 and f using $\theta_1 = \{\}$)

(3) $\leftarrow father(X, Y)$ (Resolution of 2 and d using $\theta_2 = \{\}$)

(4) $\leftarrow$ (Resolution of 3 and a using $\theta_3 = \{X/abraham, Y/isaac\}$)

Resolution for Definite Logic Programs

SLD-resolution $\equiv$ Linear resolution with Selection function for Definite clauses.

Definition (Definite Goal)

A definite goal is a rule of the form

$$\leftarrow A_1, \ldots, A_n$$

where $0 \leq n$ and each A_i, $0 < i \leq n$, is an atom.

Definition (Resolvent)

Let G be a definite goal $\leftarrow A_1, \ldots, A_{k-1}, A_k, A_{k+1}, \ldots, A_m$, A_k be a selected atom, and r be a definite rule $B_0 \leftarrow B_1, \ldots, B_n$. We say that a goal G' is a resolvent derived from G and r using θ if θ is the most general unifier of A_k and B_0 and G' has the form $\leftarrow (A_1, \ldots, A_{k-1}, B_1, \ldots, B_n, A_{k+1}, \ldots, A_m)\theta$.

Definition (SLD-derivation)

Let P be a definite logic program and G be a definite goal. An SLD-derivation of $P \cup \{G\}$ is a (posibly infinite) sequence of goals $G = G_0, \ldots, G_i, \ldots,$ where each G_{i+1} is a resolvent obtained from G_i and a rule r_{i+1} from P using θ_{i+1}.

Definition (Successful, Failed, and Infinite Derivation)

A successful derivation ends in empty goal $\leftarrow$. A failed derivation ends in non-empty goal with the property that all atoms does not unify with the head of any rule. An infinite derivation is an infinite sequence of goals.

SLD-Tree

Definition (SLD-Tree)

Let P be a definite logic program and G be a definite goal.
An SLD-tree for $P \cup \{G\}$ is a minimal tree satisfying the following:

- Each node of the tree is a (possibly empty) definite goal
- The root is G
- If G' is a node of the tree and G'' is a resolvent derived from G', then G' has a child G''

Standard Prolog

- selects the first literal in the goal
- chooses rules for unification in order as they appear in the logic program
- uses depth-first search strategy

Answer

Definition (Correct Answer)

Let P be a definite logic program and G be a definite goal $\leftarrow A_1, \ldots, A_n$. An answer for $P \cup \{G\}$ is a substitution for variables in G. An answer θ for $P \cup \{G\}$ is correct iff $P \models (A_1, \ldots, A_n)\theta$.

Definition (Computed Answer)

Let $G_0, \ldots, G_n$ be a successful derivation using $\theta_1, \ldots, \theta_n$. Then $\theta_1 \ldots \theta_n$ restricted to the variables of G is the computed answer.

Soundness and Completeness

Theorem (Soundness)

Let P be a definite logic program and G be a definite goal. Every computed answer for $P \cup \{G\}$ is a correct answer for $P \cup \{G\}$.

Theorem (Completeness)

Let P be a definite logic program and G be a definite goal. For every correct answer θ for $P \cup \{G\}$ there exists a computed answer σ for $P \cup \{G\}$ and a substitution γ such that $\theta = \sigma\gamma$.

Fact (Termination)

SLD-resolution may not terminate.

SLDNF-Resolution

SLD-resolution augmented by the negation as failure rule.

Definition (Normal Goal)

A normal goal is a rule of the form

$$\leftarrow L_1, \dots, L_n$$

where $0 \leq n$ and each L_i, $0 < i \leq n$, is a literal.

Definition (Resolvent)

Let G be a normal goal $\leftarrow L_1, \dots, L_{k-1}, L_k, L_{k+1}, \dots, L_m$, L_k be a selected atom A, and r be a normal rule $B_0 \leftarrow M_1, \dots, M_n$. We say that a goal G' is a resolvent derived from G and r using θ if θ is the most general unifier of L_k and B_0 and G' has the form $\leftarrow (L_1, \dots, L_{k-1}, M_1, \dots, M_n, L_{k+1}, \dots, L_m)\theta$.

Negation as Failure

Definition (Negation as Failure Rule)

Let G be a normal goal $\leftarrow L_1, \ldots, L_{k-1}, L_k, L_{k+1}, \ldots, L_m$ and L_k be a selected negated atom $\sim A$. We say that a normal goal G' is obtained from G using negation as failure rule if $P \cup \{\leftarrow A\}$ has finitely failed SLDNF-tree and G' has the form $\leftarrow L_1, \ldots, L_{k-1}, L_{k+1}, \ldots, L_m$.

SLDNF-Derivation

Definition (SLDNF-Derivation)

Let P be a normal logic program and G be a normal goal.
An SLDNF-derivation of $P \cup \{G\}$ is a (possibly infinite) sequence of goals $G = G_0, \ldots, G_i, \ldots$ where each G_{i+1}

- is derived from G_i and a rule r_{i+1} from P using θ_{i+1}, or
- is obtained from G_i using negation as failure rule on selected literal $\sim A$. In such case, $r_{i+1} = \leftarrow A$ and θ_{i+1} is identity.

Definition (Successful, Failed, and Infinite Derivation)

A successful derivation ends in empty goal $\leftarrow$. A failed derivation ends in non-empty goal with the property that the selected literal is

- an atom which do not unify with the head of any rule, or
- a negated atom which do not have finitely failed SLDNF-tree.

An infinite derivation is an infinite sequence of goals.

SLDNF-Tree

Definition (SLDNF-Tree)

Let P be a normal logic program and G be a normal goal.
An SLDNF-tree for $P \cup \{G\}$ is a minimal tree satisfying the following:

- Each node of the tree is a (possibly empty) normal goal
- The root is G
- If G' is a node of the tree and G'' is a resolvent derived from G', then G' has a child G''
- If G' is a node of the tree and G'' is obtained from G' using negation as failure rule, then G' has a child G''

Definition (Finitely Failed SLDNF-Tree)

A finitely failed SLDNF-tree is finite and has only failed branches.

SLDNF-Derivation and SLDNF-Tree

Please note, that SLDNF-tree is defined in terms of SLDNF-derivation, and SLDNF-derivation is defined in terms of SLDNF-tree. Such cyclic definitions are not correct. Proper definitions are much more complex, although they capture the same idea. They can be found in:

Lloyd, J. W. (1987). Foundations of Logic Programming. Springer.

Answer

Definition (Correct Answer)

Let P be a normal logic program and G be a normal goal $\leftarrow L_1, \ldots, L_n$. An answer for $P \cup \{G\}$ is a substitution for variables in G. An answer θ for $P \cup \{G\}$ is correct iff $Comp(P) \models (L_1, \ldots, L_n)\theta$.

Definition (Computed Answer)

Let $G_0, \ldots, G_n$ be a successful derivation using $\theta_1, \ldots, \theta_n$. Then $\theta_1 \ldots \theta_n$ restricted to the variables of G is the computed answer.

Soundness and Completeness

Theorem (Soundness)

Let P be a normal logic program and G be a normal goal. Every computed answer for $P \cup \{G\}$ is a correct answer for $P \cup \{G\}$.

Fact (Termination)

SLDNF-resolution may not terminate.

Fact (Completeness)

SLDNF-resolution is not complete. Even if it terminates, it may not compute all answers (see floundering).

Floundering

```
man(dilbert). man(bill).
husband(bill).
single(X) :- man(X), \+ husband(X).

?- single(X).
X = dilbert; No

man(dilbert). man(bill).
husband(bill).
single(X) :- \+ husband(X), man(X).

?- single(X).
No
```

Floundering

What is the nature of floundering problem?

If we want to resolve $\leftarrow \sim husband(X)$, according to the "negation as failure" rule, we check whether $P \cup \{\leftarrow husband(X)\}$ has finitely failed SLDNF-tree.
Recall that $\leftarrow \sim husband(X)$ stands for $\sim(\exists X) \sim husband(X)$, and $\leftarrow husband(X)$ stands for $\sim(\exists X) husband(X)$. They are not complementary formulas!

On the other hand, if we want to resolve $\leftarrow \sim husband(dilbert)$, we check whether $P \cup \{\leftarrow husband(dilbert)\}$ has finitely failed SLDNF-tree. In this case, $\sim husband(dilbert)$ and $husband(dilbert)$ are complementary. Flounering problem can occur only when we resolve negated atom containing a variable.

Ordering of Rules Matters

```
on(a, b). on(b, c).
above(X, Y) :- on(X, Y).
above(X, Y) :- above(X, Z), on(Z, Y).

?- above(a, c).
Yes;
Error: Stack overflow.

on(a, b). on(b, c).
above(X, Y) :- above(X, Z), on(Z, Y).
above(X, Y) :- on(X, Y).

?- above(a, c).
Error: Stack overflow.
```

Ordering of Literals Matters

```
on(a, b). on(b, c).
above(X, Y) :- on(X, Y).
above(X, Y) :- above(X, Z), on(Z, Y).

?- above(a, c).
Yes;
Error: Stack overflow.

on(a, b). on(b, c).
above(X, Y) :- on(X, Y).
above(X, Y) :- on(Z, Y), above(X, Z).

?- above(a, c).
Yes;
No.
```