Computational Logic
Prolog

Martin Baláž

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

2011

Example

Logic Program:

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

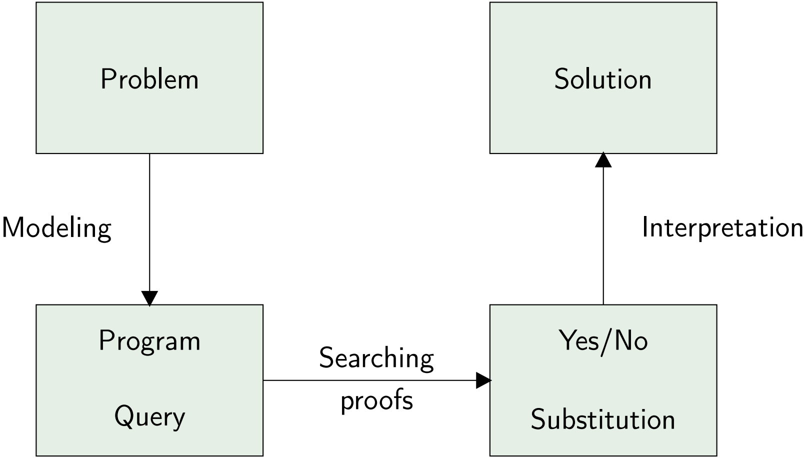
Query:

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

Answer:

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

Programming with Prolog



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

Let G be a goal $A_1 \wedge \cdots \wedge A_k \wedge \cdots \wedge A_m$, A_k be a selected atom, and r be a rule $B_0 \leftarrow B_1 \wedge \cdots \wedge B_n$. 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
$\leftarrow (A_1 \wedge \cdots \wedge A_{k-1} \wedge B_1 \wedge \cdots \wedge B_n \wedge A_{k+1} \wedge \cdots \wedge A_m)\theta$.

An *SLD-derivation* of $P \cup \{G\}$ is a (posibly infinite) sequence of goals $G_0, \ldots, G_i, \ldots$, where

- $G_0 = G$
- G_{i+1} is obtained from G_i and a rule r_{i+1} from P using θ_{i+1}

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.

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

Let P be a definite logic program and G be a definite goal G. 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*.

Properties

Let P be a definite logic program and G be a definite goal. Then every computed anwer for $P \cup \{G\}$ is a correct aswer for $P \cup \{G\}$.

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 $\theta = \sigma\gamma$.

Let P be a definite logic program and G be a definite goal. Then $P \cup \{G\}$ is unsatisfiable iff there exists a successful derivation of $P \cup \{G\}$.

Let M_P be the least model of a definite logic program P. Then $M_P = \{A \in \mathcal{B}_P \mid P \cup \{\leftarrow A\}$ has a successful derivation$\}$.

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 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''

A *computation rule* is a function from a set of definite goals to a set of atoms such that the value of the function for a goal is an atom, called the *selected atom*, in that goal.

A *search rule* is a strategy for searching SLD-trees to find success branches.

Example

Definite logic program P

$$\begin{array}{lcl} p(a,b) & \leftarrow & \\ p(c,b) & \leftarrow & \\ p(x,z) & \leftarrow & p(x,y), p(y,z) \\ p(x,y) & \leftarrow & p(y,x) \end{array}$$

Definite goal G

$$\leftarrow \quad p(a,c)$$

SLDNF-resolution $\equiv$ SLD-resolution augmented by the negation as failure rule.

A *negation as failure rule* states that $\sim A$ is true iff there exists a finite SLDNF-tree for A with only failed branches.

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

Ordering of rules matters.

Example:

```
reverse([X|Xs], Zs) :- reverse(Xs, Ys),
                       append(Ys, [X], Zs).
reverse([], []).

? reverse(Xs, [3,2,1]).
Xs = [1,2,3]
...
```

Ordering of literals matters.

Example:

```
reverse([], []).
reverse([X|Xs], Zs) :- append(Ys, [X], Zs),
                       reverse(Xs, Ys).
```

```
? reverse([1,2,3], Zs).
Zs = [3,2,1]
...
```

Floundering

Negation as failure

Example:

```
man(dilbert).
husband(bill).
single(X) :- man(X), not(husband(X)).

? single(X).
X = dilbert
```

Floundering

Negation as failure

Example:

```
man(dilbert).
husband(bill).
single(X) :- not(husband(X)), man(X).

? single(X).
No
```