Introduction to Logic in Computer Science: Autumn 2007

Ulle Endriss
Institute for Logic, Language and Computation
University of Amsterdam

Logic and Prolog

Today we are going to discuss the logical foundations of Prolog:

- Translating Prolog programs into Horn clauses
- Resolution as the reasoning engine underlying Prolog

Horn Clauses

In logic, a *clause* is a disjunction of literals. A *propositional Horn clause* is a clause with at most one positive literal. Observe that:

$$\neg A_1 \vee \cdots \vee \neg A_n \vee B \quad \equiv \quad A_1 \wedge \cdots \wedge A_n \rightarrow B$$

A *first-order Horn clause* is a formula of the form $(\forall x_1) \cdots (\forall x_n) A$, with A being a propositional Horn clause.

"Pure" Prolog programs (without cuts, negation, or any built-ins with side effects) can be translated into sets of Horn clauses:

- Commas separating subgoals become $\wedge$.
- `:-` becomes $\rightarrow$, with the order of head and body switched.
- All variables are universally quantified (scope: full formula).
- Queries are translated as negated formulas ($Q \rightarrow \bot$).

Example

The following Prolog program (with a query) ...

```
bigger(elephant, horse).
bigger(horse, donkey).
is_bigger(X, Y) :- bigger(X, Y).
is_bigger(X, Y) :- bigger(X, Z), is_bigger(Z, Y).
?- is_bigger(elephant, X), is_bigger(X, donkey).
```

... corresponds to the following set of FOL formulas:

$$\{ \quad bigger(elephant, horse),$$

$$bigger(horse, donkey),$$

$$\forall x.\forall y.(bigger(x, y) \rightarrow is\_bigger(x, y)),$$

$$\forall x.\forall y.\forall z.(bigger(x, z) \wedge is\_bigger(z, y) \rightarrow is\_bigger(x, y))$$

$$\forall x.(is\_bigger(elephant, x) \wedge is\_bigger(x, donkey) \rightarrow \bot) \quad \}$$

Alternative notation: set of sets of literals (implicit quantification)

Prolog and Resolution

When Prolog resolves a query, it tries to build a proof for that query from the premises given by the program (or equivalently: it tries to refute the union of the program and the negated query).

Therefore, at least for pure Prolog, query resolution can be explained in terms of deduction in FOL. In principle, any calculus could be used, but historically Prolog is based on *resolution*.

What next?

- Resolution for full FOL
- Resolution for Horn clauses (to get a feel for why Prolog "works", despite the undecidability of FOL)

Binary Resolution with Factoring

Aim: Show $\Delta \models \varphi$ (for a set of sentences Δ and a sentence φ).

Preparation: Compute Skolem Normal Form of formulas in Δ and of $\neg\varphi$ and write them as a set of clauses (variables named apart).

Input: Set of clauses (which we want to show to be unsatisfiable).

Algorithm: Apply the following two rules. The proof succeeds if the empty clause (usually written as $\square$) can be derived.

Binary Resolution Rule
$$\frac{\{L_1\} \cup C_1 \quad \{L_2^c\} \cup C_2}{\mu(C_1 \cup C_2)}$$
L_2^c is the complement of L_2 μ is an mgu of L_1 and L_2

Factoring
$$\frac{\{L_1, \ldots, L_n\} \cup C}{\sigma(\{L_1\} \cup C)}$$
σ unifies $\{L_1, \ldots, L_n\}$

Why Factoring?

Try to derive the empty clause from the following (obviously unsatisfiable) set of clauses *without using the factoring rule*.

$$\{\, \{P(x), P(y)\},\ \{\neg P(u), \neg P(v)\} \,\}$$

$\Rightarrow$ It's not possible!

This means that our *binary* resolution rule alone (without factoring) would not be a *complete* deduction system for FOL.

Remark: The *general resolution rule* allows us to resolve using subclauses (rather than just literals). In that case we can do without factoring.

SLD Resolution for Horn Clauses

SLD Resolution stands for **S**elective **L**inear Resolution for **D**efinite clauses, where:

- *linear* means we always use the latest resolvent in the next step;
- we have a *selection function* telling us which literal to use; and
- the input is restricted to Horn clauses, all but one of which have to be *definite clauses* (that's another word for Horn clauses with *exactly* one positive literal).

SLD Resolution is complete for the Horn fragment (proof omitted).

SLD Resolution in Logic Programming

Prolog implements SLD Resolution:

- *Linearity:* we start with the only negative clause (the negated query) and then always use the previous resolvent (new query).
- The *selection function* is very simple: it always chooses the first literal (in the current "query").
- The input is restricted to *one negative Horn clause* (negated query) and a number of *positive Horn clauses* (rules and facts).

In practice, one problem remains: if there is more than one way to resolve with the selected literal (i.e. more than one matching rule or fact) then we don't know which one will eventually lead to a successful refutation. In Prolog, always the first one is chosen and if this turns out not to be successful, *backtracking* is used to try another one.

Worked Example

Consider the following Prolog program:

```
parent(elisabeth, charles).
parent(charles, harry).
ancestor(X, Y) :- parent(X, Y).
ancestor(X, Z) :- parent(X, Y), ancestor(Y, Z).
```

What will happen if we submit the following query after the above program has been consulted by Prolog?

```
?- ancestor(elisabeth, harry).
```

Step 1: Translate into FOL

For the *program* we get the following formulas:

(1) $P(e, c)$

(2) $P(c, h)$

(3) $(\forall x)(\forall y)(P(x, y) \rightarrow A(x, y))$

(4) $(\forall x)(\forall y)(\forall z)(P(x, y) \wedge A(y, z) \rightarrow A(x, z))$

For the *negation of the query* we get:

(5) $\neg A(e, h)$

Step 2: Rewrite Formulas as Clauses

Formulas we get from translating a Prolog program already are in Prenex Normal Form and we don't need to Skolemise either (because there are no existential quantifiers).

We have to rewrite the implications as disjunctions. Here, we directly give the clauses (which correspond to disjunctions). Don't forget that variables have to be named apart.

(1) $\{P(e, c)\}$

(2) $\{P(c, h)\}$

(3) $\{\neg P(x_1, y_1),\ A(x_1, y_1)\}$

(4) $\{\neg P(x_2, y_2),\ \neg A(y_2, z_2),\ A(x_2, z_2)\}$

(5) $\{\neg A(e, h)\}$

Step 3: Apply SLD Resolution

(1)	$\{P(e, c)\}$	
(2)	$\{P(c, h)\}$	
(3)	$\{\neg P(x_1, y_1),\ A(x_1, y_1)\}$	
(4)	$\{\neg P(x_2, y_2),\ \neg A(y_2, z_2),\ A(x_2, z_2)\}$	
(5)	$\{\neg A(e, h)\}$	
(6)	$\{\neg P(e, y_3),\ \neg A(y_3, h)\}$	from (4,5) with $[e/x_2]$ and $[h/z_2]$
(7)	$\{\neg A(c, h)\}$	from (1,6) with $[c/y_3]$
(8)	$\{\neg P(c, h)\}$	from (3,7) with $[c/x_1]$ and $[h/y_1]$
(9)	$\square$	from (2,8)

Remark: If there had been variables in our query, then the substitutions made to them would have been part of the answer.

Summary

- Pure Prolog corresponds to sets of Horn clauses.
- The reasoning engine underlying Prolog can be explained in terms of SLD Resolution.