Slides to Accompany *Programming Languages and Methodologies*

R. J. Schalkoff

Chapter 3, Part 1: Prolog

Why Prolog?

Our interest in Prolog is on many levels. This includes:

- Exploring declarative programming
- Exploring Logic Programming
- Prolog syntax
- Prolog semantics
- The Prolog development cycle
- Prolog applications, including:
  - constraint satisfaction problems (CSPs)
  - `minic` grammar scanner parsers (employing the LGN)

Resources

The SWI-Prolog home page is:

`www.swi-prolog.org`

- Here you will find SWI Prolog in various forms, including a self-extracting executable for Windows and a rpm for linux, as well as a User's Guide (pdf).
- You will also find various additional tools (IDEs, editors, etc.) and Prolog links.

Alternatives to SWI-Prolog include the GNU Prolog compiler (`gprolog`), available at:

`http://www.gnu.org/software/prolog/`

The Prolog Incremental Development Cycle

Typically, development in Prolog proceeds iteratively as follows:

- creation of the Prolog database (using an editor) as a file or set of files
- interpretation of the source file(s) (via the Prolog predicate `consult` or `[]`)
- execution of the program (a query) with possible tracing
- source file modifications
- re-interpretation of the source file(s)
- further queries
- and so forth...

Warning

Although the declarative programming concept is straightforward, when actually employing Prolog, a number of issues complicate the picture, including:

- The goal-satisfaction or unification mechanism (including backtracking);
- The spawning of subgoals in the attempted solution;
- The computational requirements of depth-first search;
- Advanced Prolog constructs such as lists and the cut; and
- Recursion

Introductory Points

1. Prolog is a useful language when the solution to a problem involves satisfaction of a number of constraints relating problem variables.
2. In Prolog, the programmer does not concentrate on the specification of program execution sequence, but rather attempts to specify the problem (or situation) through development of a database consisting of clauses.
   - Clauses may be either facts or rules.
   - It is left for the Prolog unification mechanism to provide a solution, if one exists.

Hello World in Prolog

Prolog Source File:

```
%% hello world program
%% hello.pro

go :- nl, write('hello, ece352 world'), nl.
```

Using the Program (and invoking the goal):

```
?- consult('hello.pro').
hello.pro compiled, 0.00 sec, 532 bytes.

Yes
?- listing(go).

go :-
        nl,
        write('hello, ece352 world'),
        nl.

Yes
?- go.

hello, ece352 world

Yes
?- halt.
```

PROLOG 'help' Notation Example

`member(?Elem, ?List)`

followed by a description. In this commonly-used notation,

- `+Argument` means input (i.e., the argument is bound before invoking the predicate)
- `-Argument` means output (i.e., the argument is bound to a value, if possible, in the process of finding a solution)
- `?Argument` means either (works both ways)

Note that GUI-based help is also available in some distributions.

Identifying and Reporting Syntax Errors

Can you find the (single) syntax error?

```
/* smpl-unify1.pro */

goal1(X,Y) :- first(X), second(Y).

goal2(X) :- first(X), second(X).

first(1).
first(2)
first(3).
second(2).
second(4).
second(6).
```

```
?- consult('./smpl-unify1.pro').
ERROR: smpl-unify1.pro:9:
/* above indicates the error on line 9 */
Syntax error: Operator expected
% ./smpl-unify1.pro compiled 0.00 sec, 1,288 bytes

Yes
```

Line #9 in the file was:

```
first(2)
```

Predicates, Clauses, Facts, Rules and Goals

Predicate. A predicate consists of a name and an optional set of arguments. The number of arguments is the *arity* of the predicate.

Clause. Clauses are either rules or facts. Clauses are built using predicates and logical connectives and allowed to contain variables which are assumed universally quantified.

- A Prolog clause is comprised of a head and an (optional) body or tail.
- If the body is empty, the clause is a fact, which is interpreted to be true.
- The period (.) ends the clause.

Facts. As noted, a fact is a rule with an empty tail. Although this is acceptable notation, in practice facts have the typical form:

```
predicate_name(term1, term2,...,termn).
```

where the period (.) terminates the clause. The following is an example of a database of Prolog facts:

```
is_fact (arg1, arg2).
a_fact_too (Y).
equal (X, X).
another_fact.
wheel_is_round.
round (wheel).
wheel (round).
```

Rules. A rule is a clause with a non-empty head and a tail. Rules have the typical form (logical representation):

```
predicate_name1(term1, term2,...) :-
                     predicate_name2(term1, term2,...),
                                        ...
                    predicate_nameR(termR, termR,...).
```

The entity **`termi`** above may be a constant, a variable (capitalized), a list or a functor. The following example is a clause which is a rule:

```
has(X, door) :- is_house (X).
```

Conjunction and Disjunction.

The use of the comma (’,’) in Prolog to separate clauses indicates that the conjunction of these clauses (subgoals) must be satisfied, e.g.,

```
a,b
```

forces both subgoals a AND b to succeed.

Prolog systems also provide the disjunction (OR) operator ';', which allows the disjunction of two predicates in the form

```
        a;b.
```

This allows alternatives to be specified directly. Note however, that this may also be achieved in the database by rewriting. For example, given

```
goal :- a;b.
```

this may be rewritten as the logically equivalent pair of clauses:

```
goal :- a.
goal :- b.
```

Variables

- All variables in Prolog begin with a capital (uppercase) letter.
- The name of a predicate may not be a variable.

The Anonymous Variable. The anonymous variable is used to represent universal quantification; (when any instantiation of the variable will suffice). It is not necessary to give this variable a name, instead the underbar or '_' is used.

Example:

```
matchany(_).
```

Goals

A Prolog program is incomplete without specification of a goal. A goal, entered at the Prolog interpreter prompt (`?-`) , is a Prolog clause which may be comprised of subgoals, logical connectives and contain variables. The Prolog system attempts to unify this goal with the database. An example of a simple goal used to query the Prolog database might be:

```
?-has(What, door).
```

Unification = Solution = Search

An understanding of the operation of the Prolog unification mechanism is fundamental to success in developing practical Prolog programs.

The rules for unification are:

1. Clauses are tested in the order in which they appear in the database.
2. When a subgoal matches the left side (head) of a rule, the right side (tail) becomes a new set of subgoals to unify.
3. The unifier proceeds from left to right in attempting to unify the predicates in the tail. When a subgoal is spawned, the unification/search process described in 1. repeats.
4. A goal is satisfied when a matching fact is found in the database for all the leaves of the goal tree.
5. When two or more clauses in the database with the same predicate name are identified as possible matches, the first clause in the database is tested for attempted unification. The rest are marked as points for possible **backtracking**, and are investigated if a previous choice fails to unify.

Logging a Prolog Session

SWI-Prolog allows logging the interactive session .

```
protocol(+File)
     Start protocolling on file File. If there is already
a protocol file open then close it first.
If File exists it is truncated.

noprotocol
     Stop making a protocol of the user interaction.
Pending output is flushed on the file.
```

factorial description in Prolog

```
/* a simple fact (factorial(1)=1) */

factorial(1,1).

/* a rule to recursively define (or describe) 'factorial'  */

factorial(N,Result) :- Imin1 is N-1,
factorial(Imin1,Fmin1),
Result is N*Fmin1.
```

Prolog `factorial` Database Use

```
?- consult('fact.pro').
fact.pro compiled, 0.00 sec, 408 bytes.

Yes
?- listing.

factorial(1, 1).
factorial(A, B) :-
        C is A - 1,
        factorial(C, D),
        B is A * D.

Yes
?- factorial(4, What).

What = 24

Yes
```

Another Example

The Development Process. Suppose the following dialog occurs:

```
PSD: Dean, tell me how you decide who gets tenure.

DEAN: That's easy. I award tenure to my faculty who publish,
get research and teach well.

PSD: Am I correct in my understanding that they must do all three of these?

DEAN: That's right.

PSD: How does a faculty member "publish"?

DEAN: The faculty member conducts research and documents the research.

PSD: How does the faculty member get research?
```

DEAN: The faculty member writes research proposals which subsequently become funded.

PSD: What does it mean for a faculty member to teach well?

DEAN: That's easy. A good teacher prepares lectures, delivers the lecture well, and gets good evaluations from the students.

PSD: Is that all there is to it?

DEAN: That's right.

PSD: Thanks for your time and expertise, Dean.

Corresponding PROLOG Representation Strategy.

```
gets_tenure(Faculty) :- publishes(Faculty),
                             gets_research(Faculty),
                                  teaches_well(Faculty).

publishes(Professor) :- does_research(Professor),
                           documents_research(Professor).

gets_research(Researcher) :- writes_proposals(Researcher),
                               gets_funded(Researcher).

teaches_well(Educator) :- prepares_lectures(Educator),
                             lectures_well(Educator),
                               gets_good_evaluations(Educator).
```