PROLOG Programming

University of Wales, Cardiff

January 1997

Omer F. Rana
omer@cs.cf.ac.uk
Room S/2.03

1 What is Prolog and why should I bother ?

PROLOG : PRogramming in LOGic

- 5th generation computer language
- Used in programming for artificial intelligence
- based on a subset of predicate calculus (Horn clauses)
- The *Edinburgh* syntax

A **DIFFERENT** programming style from **PASCAL** or **C**

A language for :
ARTIFICIAL INTELLIGENCE,

with numerous application domains (such as) *DATABASES,*
NATURAL LANGUAGE PROCESSING,
IMAGE ANALYSIS,
DECISION SUPPORT SYSTEMS,
FINANCIAL MODELS, INTERNET APPLICATIONS and many others ...

Understanding the experts
Intelligent Reasoning

2 Books n' stuff

1. Prolog Programming for Artificial Intelligence *Bratko* Addison-Wesley, 90
2. Programming in Prolog, *Clocksin and Mellish* Berlin : Springer-Verlag 87
3. The art of Prolog : advanced programming techniques, *Sterling and Shapiro* MIT P ress, 94
4. Problem solving with Prolog, *Stobo* London : Pitman 89
5. LPA MacProlog32 (technical manuals) *Logic Programming Associates*

However, for an introductory course, any book on Prolog is fine.

3 Course Outline

- Declarative programming style
- Translating simple problems into **Prolog**
- Language Syntax
- Computing with **Prolog**
- Programming techniques and styles
- Understanding execution of **Prolog** programs
- User interfaces in **MacProlog**
- Stand alone applications and link with other programming languages
- Applications : **Databases** and the **Internet**

Lab Handout for *LPA MacProlog*

4 *LPA* MacProlog32

- Will be used for programming in the labs
- On a (surprise surprise) Macintosh
- See *User Guide* for details
- Mail questions to me at *omer@cs.cf.ac.uk*
- **Friday-hour** for support
- **Programming $\equiv$ Practise**
- Lots of chance to *play* with **Prolog**, also compiler available on the internet for PC.
- Emphasis on **Logic Programming**

5 Comparison between Pascal and Prolog

5.1 PASCAL

- is **compiled** to an executable file.
- is **imperative** - consists of a sequence of instructions to execute.
- program flow composed of **conditionals** and **loops**.
- numerous **built-in** functions (language grammar).
- an expression can evaluate to one of **many** values.

5.2 PROLOG

- is **interpreted**.
- is **declarative** - contains a series of **definitions** which are accessed as required (in any order) by the program during execution.
- the **HOW** of solving the problem is **NOT** defined, only what is **TRUE or NOT** of a certain state of affairs is described.
- a statement can either be **TRUE** or **FALSE**
- a few **built-in** rules, and a few **fundamental operations**.

PASCAL programs contain a series of **INSTRUCTIONS** given to the computer as **PROCEDURES** to perform. The order of doing things is **SPECIFIED** and a number of **ACTIONS** performed as a consequence.

PROLOG programs contain a set of **FACTS** and **RULES**. Aspects of the problem which are **TRUE** or **FALSE** are described (defined), rather than the method to achieve a **RESULT**.

For instance :

```
likes(omer, donuts)
likes(paul, neighbours)
```

6 Example

In predicate calculus

$$\forall x[green(x) \wedge dragon(x) \rightarrow fly(x)]$$

```
green(fonzie)
green(puff)
green(martian)

dragon(godzilla)
dragon(puff)
```

The corresponding translation in **Prolog** will be :

```
fly(X) :-
green(X),
dragon(X).
```

body $\equiv$ **antecedents** that must be true.

A Logical Deduction is called a **Query**

- **Asking for Confirmation**

  Q : fly(puff)
  A : yes

  Q : fly(fonzie)
  A : no

  Q : fly(kermit)
  A : no (*no* $\equiv$ *unproven, not false*

- **Asking for values**

  Q : fly(X)
  A : X = puff

  $$\exists x[fly(x)]$$

  Q : dragon(X)
  A : X = godzilla
  X = puff

Find a fact to match the *goal* : **Binding**

Some **Prolog Facts :**

Find a fact to match the *goal* : **Binding**

So in the previous example, for instance, **puff** was found to be a valid answer. If more than one possible answers existed to the problem, then the first *goal* variable is *unbound* (i.e. its value discarded) and the next matching value *bound* to it.

- **Matching** of goals to facts
- **Searching** for solutions

Fundamental processes in the understanding of how **Prolog** works.

7 Another Example

In predicate calculus :

$$\forall x[student(x) \wedge money - left(x) \rightarrow holiday(x)]$$

$$\forall x[student(x) \wedge free - time(x) \rightarrow party(x)]$$

now pick a few students :

```
student(patrick) student(jane)
student(gwyn) student(ahmed)
student(helen) student(vijay)
```

those with money left / free time :

```
money-left(gwyn) free-time(ahmed)
money-left(jane) free-time(jane)
money-left(helen) free-time(vijay)
```

Translation Into Prolog :

```
holiday(X) :-
student(X),
money_left(X).

party(X) :-
student(X),
free_time(X).

lucky(X) :-
student(X), money_left(X) ;
student(X), free_time(X).

student(patrick).
student(jane).
student(gwyn).
student(ahmed).
student(helen).
student(vijay).
money_left(gwyn).
free_time(ahmed).
money_left(jane).
free_time(jane).
money_left(helen).
free_time(vijay).
```

Things to note :

- **Layout** : Keep it readable.
- **Spaces, Hyphens and Underscores**.
- **Full stops and commas**.
- **The Universe of discourse** (limited to that defined by the program)

8 Prolog Syntax

- Follows the Edinburg syntax

  /* one line comment */

  /* Multi
  line comment */
- % the rest of this line is a comment
- **Terms** are the building blocks of a program
- for instance, **Numbers**

  Integers : 12 0 -24

  Floating Point : 2.5 -0.34 10e-4

A **query** can be used to determine or decide upon number syntax, so for instance :

Q : integer(12)
A : yes
Q : float(10.2)
A : yes
Q : integer(-7.43)
A : no
Q : number(-2.34)
A : yes
Q : number(5322)
A : yes

- **integer**, **float** and **number** are built in predicates.
- **dragon**, **green** and **fly** are user defined predicates.

Atoms

Constants that start with a lower case letter, followed by zero or more alphanumeric characters, for instance :

donut
can_dance
fortran90

symbolic (sequence of symbolic characters)

++
&&

and those with *special meaning* in Prolog :

! (exclamation mark)
, (coma)
; (semi-colon)

quotes (any sequence of characters in single quotes)

'Apple'
'Alia'
'&them'

again a query format can be used to deduce atoms

Q : atom(alia)
A : yes
Q : atom(123)
A : no
Q : atomic(123)
A : yes (*because is an integer*)
Q : atomic('123')
A : yes (*because is an atom*)

What are the distinctions ?

Variables

alphanumeric sequence of characters (include _), starting with a capital letter or underscore, such as

X
Apple
_name
Failed_student

So a query such as :

Q : var(Apple)
A : yes
Q : nonvar(_name)
A : no
Q : var(ola)
A : no
Q : var('Apple')
A : no (*because is an atom*)

Variables can be anonymous, denoted by the _ (underscore alone), and used when a return value is not needed

for instance,

born(heather,cardiff)
born(ben, prague)

Q : born(_,cardiff)
A : yes

The question being asked is :

Is anybody born in cardiff ?, but I **do not care** who !

Note : *Do not use _2 or _(number) as a variable name, as this is how MacProlog32 uses anonymous variables, and represents them internally.*

9 Scope of a variable

Within a clause, all occurrences of an identical variable refer to the same value.

For instance :

fly(X) :- green(X), dragon(X).

and **happy(X) :- pass_exam(X,Y).**

here, **X** in *fly(X)* and *happy(X)* are not related.

However, the occurrance of **X** in the definition of *fly(X)* refer to the same thing.

Also, note the change in *arity* of the clause *happy(X)*.

There are some *exceptions* however for such *anonymous* variables :

balanced_child(X) :-
brother(X,_),
sister(X,_).

Here, the variable represented by _ refers to different people

and would be represented *internally* as :

balanced_child(X) :-
brother(X,_483),
sister(X,_485).

The advanced **Trace** *mechanism can be used to show internal variable names*

Consider the program :

```
check_number(0).

check_number(N) :-
N < 0,
N1 is N+2,
check_number(N1).

check_number(N) :-
N > 0,
N1 is N-2,
check_number(N1).
```

What happens when we give the queries :

check_number(4)
check_number(100)
check_number(15)
check_number(125)

Query :

Q : check_number(4)
A : yes

The program goes into a loop, nothing appears to be happening on the screen. The loop will only stop until there is no more memory left for evaluation. Hence, the program will have to be interrupted.

What can we do to prevent this happening ?

check_number(1).

Q : check_number(15)
A : yes

the new information also has to be added at a specific point, because of the way evaluation is performed by Prolog

10 Compound Terms

Atom (functor) followed by k $\geq$ 1 terms, enclosed in brackets and seperated by commas

k denotes the **arity**

e.g. student(ursula)
composer(bach,baroque)
disk_jockey(X,party(X))

tuples : sequences of terms, enclosed in brackets and separated by commas

e.g. representing a date :
23, march, 97)
as, is an atom, the tuple can be seen as :
','(10,','(march, 97)) (internal handling)

Q : compound(foo(a,B,32))
A : yes
Q : compound('&Others'(Names))
A : yes
Q : compound(123)
A : no