1

User Interface - Running Prolog

```
% prolog
C-Prolog version 1.4
| ?- halt.            or    ^D
[ Prolog execution halted]
```

Prolog starts in "top-level" mode where the user can ask questions, or request that input be obtained from a file (or the terminal). The system prompt indicates the mode:

`| ?-`	Prolog expecting a question, or request for input.
`|`	Prolog expecting a continuation line.

Obtaining Input from a File:

`|?- [question4].`	The file is read in, clauses are read in and **added to existing clauses**, directives are followed.
`|?- [-question4].`	The file is read in, clauses are read in and **replace existing clauses**, directives are followed.

©1993 John Mylopoulos

2

Obtaining Input from the Terminal :

```
| ? [user].          The input file is the terminal.
| human(ellen).      Predicate human is added.
| human(john).       Predicate human is added.
|  ^D                END-OF-INPUT Character
| user consulted 100bytes 3.9736e-09 sec.
yes
```

Asking Questions:

Assume we have read in the predicate `human` as above. Then we may ask questions:

```
| ?- human(john).<Return>
                     Is john human?
yes                  Prolog resolves question

| ?- human(X).       Who are all humans ?
X = ellen  ;         User types ; for more answers
                     or <Return> to stop search
X = john  ;

no                   no more answers
```


©1993 John Mylopoulos

3

File Handling

`see(F)`	File `F` becomes the current input stream.
`seeing(F)`	`F` is unified with current input stream
`seen`	Current input stream is closed
`tell(F)`	File `F` becomes the current output stream.
`telling(F)`	`F` is unified with current output stream.
`told`	Current output stream is closed.

©1993 John Mylopoulos

4

Prolog Syntax (Refined Version)

```
<clause>      ::= <lhs> [ :- <rhs>] .
<lhs>         ::= <functor>
<rhs>         ::= <literal> {,  <literal>}
<literal>     ::-  <variable> | <functor>
<literal>     ::-  "(" <variable> | <functor> ")"
<functor>     ::= <atom> [  ( <arg list> ) ]
    Note: if <arg list> is present, then  there must be no
    space between the atom and the left parenthesis.
                |   <list>
                |   <infix expr>
                |   <prefix expr>
                |   <postfix expr>
<arg list>    ::= <arg> { , <arg>}
<arg>         ::-  <literal> | <number> | <string>
                |   "("<literal> | <number> | <string> ")"
<list>        ::-  "[]"
                |   '[" <arglist> [ "|" <arg> ]  "]"
<infix expr>  ::= <arg> <symbol> <arg>
<prefix expr> ::= <symbol> <arg>
<postfix expr>::= <arg> <symbol>
<variable>    ::- start with initial Capital letter or under-
score
<atom>        ::- any sequence of characters,
                  quoted if necessary to avoid confustion
```

©1993 John Mylopoulos

5

Comments

% This is a single line comment.

/* This is a multi
line comment */

Scoping of Variables

All variables on the lefthand side of a clause are universally quantified; variables appearing on the righthand side are assumed to be existentially quantified.

All variables have a single clause as scope. For example:

```
a(X, Y, Z) :-  g(X, Y, Z), h(Z).
e(X, Y, Q) :-  g(Y, X, Z), h(Z).
```

The variable `Z` in the second clause bears no relationship to the `Z` in the first clause.

©1993 John Mylopoulos

6

Syntactic Sugar : Operator Notation

Some binary functors can be written in infix notation:

```
X . Y       equivalent to .(X,Y)
X + Y     equivalent to +(X,Y)
```

C-Prolog permits the definition of new infix, prefix and postfix operators using the builtin predicate **op**:

```
?- op(precedence, type, name).
```

precedence -- a number 1-1200 used to disambiguate expressions that don't use brackets. Operators with a lower precedence bind more tightly; operators with a highter precedence is a principle functor.

type **xfx**, **yfx**, or **xfy** - infix operator
xf or **yf** - prefix operator
fx or **fy** - postfix operator

name -- operator name, or list of operator names

Note: **x** requires brackets for terms of same precedence, while **y** permits terms of same precedence

This notation does not change the operation of any other predicates, **except** input and output predicates.

©1993 John Mylopoulos

Syntactic Sugar : Conveniences

To make the passing of predicates as sub-goals more uniform the following conjunction and disjunction operators are builtin and defined as if:

```
|?- op( 1100,xfy,[;]).   /* disjunction -- or */
|?- op(1000,xfy,[',']).  /* conjunction  -- and */
```

```
X ; Y :- X.
X ; Y :- Y.

X , Y :- X,Y.      or ','(X,Y) :- X,Y.
```

To simply perform a unification the **=** operator is defined by:

```
|?- op( 700, xfx, [=]).

X = X.     or    =(X,X).
```


©1993 John Mylopoulos