Miscellaneous Notes

Marek Sergot
Department of Computing
Imperial College, London

Autumn 2013

1 'Operators': infix, prefix, postfix

You do **not** need to memorise any of this. You might find it useful (see Sicstus manual for further details) but it is really intended to explain why the syntax of Prolog programs and commands sometimes looks different from what you might expect.

```
1 + 2
```

is simply an alternative way of writing

```
+(1,2)
```

'+' is declared in Prolog to be an **infix** operator.

Similarly, `X is 2+3`, `X = a`, `b \= a` are alternative ways of writing `is(X, +(2,3))`, `=(X,a)`, `\=(b,a)` respectively. '`is`', '`=`', '`\=`' are also declared to be infix operators.

Negation-by-failure '`\+`' is declared to be a **prefix** operator.

```
\+ happy(jack)
```

is an alternative way of writing

```
\+(happy(jack))
```

Prolog allows you to define your own operators if you want to. `Name` is declared as an operator of type `Type` and precedence `Precedence` by the command

```
:- op(Precedence, Type, Name).
```


`Precedence` is a number from 1 to 1200. It is used to disambiguate expressions in which the structure of the term denoted is not made explicit through the use of parentheses.

`Type` denotes the type of the operator: infix, prefix, postfix. The possible types for an infix operator are `xfx`, `xfy`, `yfx`. Operators of type '`xfx`' are not associative. Operators of type '`xfy`' are right-associative. Operators of type '`yfx`' are left-associative.

By analogy `fx` and `fy` are the possible types for a prefix operator. `xf` and `yf` are the possible types for a prefix operator. For example, if `not` were declared as a prefix operator of type `fy`, then

```
not not P
```

would be a permissible way to write `not(not(P))`. If the type were `fx`, the preceding expression would not be legal, although

```
not P
```

would still be a permissible form for `not(P)`. If `rouge` were declared as a postfix operator of type `xf` then

```
moulin rouge
```

would be a permissible form for `rouge(moulin)`.

An operator declaration can be cancelled by redeclaring the name with the same type but precedence 0.

You can examine the operators currently in force by querying

```
current_op(Precedence, Type, Name)
```

2 Some utilities for analysing and constructing terms

The built-in predicate `functor/3`

- decomposes a given term into its name and arity, or
- given a name and arity, constructs the corresponding compound term creating new uninstantiated variables for its arguments.

```
?- functor(g(a,b,c), F, Arity).
   F = g, Arity = 3

?-functor(Term, p, 4).
  Term = p(_1,_2,_3,_4)
```

The built-in predicate `arg/3`:

```
?- arg(q(a,m(b),c), 2, X).
   X = m(b)
```

The built-in predicate `Term =.. List` (pronounced 'univ') breaks a term into a list consisting of the functor followed by list of the arguments of the term.

```
?- g(a,b,c) =.. List
   List = [g,a,b,c]

 ?- Term =.. [p,1,q(2),3]
    Term = p(1,q(2),3)
```

Example

```
cycle_args(Term, Cycled) :-
   Term =.. [F,Arg|Args],
   append(Args, [Arg], CycledArgs),
   Cycled =.. [F|CycledArgs].

?- cycle_args(f(a,b,c), X).
   X = f(b,c,a)
```

WARNING: Do **NOT** use these utilities in your programs. You do not need them. They are only used for analysing, transforming, constructing programs. You won't be doing that in this course.

3 List notation

	Haskell	Prolog
empty list	`[]`	`[]`
fixed length list	`[1,2,3]`	`[1,2,3]`
pattern	`(x : xs)`	`[X | Xs]`
	`(x : y : xs)`	`[ X, Y | Xs]`

(I am not sure if the last one is allowed in Haskell.)

The original Prolog notation

```
a.b.nil
```

(It is still there!)

4 Example (list)

Haskell

```
sumInts :: [Int] -> Int
sumInts []       = 0
sumInts (x : xs) = x + (sumInts xs)
```

Prolog

```
% no types in Prolog
sumInts([], 0).
sumInts([X|Xs], X + SumXs) :-
   sumInts(Xs, SumXs).
```

What do we get with the following query?

```
?- sumInts([10,20,30], Sum).
```

5 Indentation and white space

Indent your programs for readability. Prolog doesn't care, but don't write:

```
pairs([],[]).pairs([H|T],[(Less,More)|PairedTail]) :- Less is
H-1,More is H+1,pairs(T,PairedTail).
```

or

```
pairs([],[]).
pairs([H|T],[(Less,More)|PairedTail]) :- Less is H - 1, More is H + 1, pairs(T,Pai
```

or

```
pairs([],[]).
pairs([H|T],[(Less,More)|PairedTail]) :-
Less is H - 1,
More is H + 1,
pairs(T,PairedTail).
```

Indenting clause bodies by two or three spaces on a new line usually works best. (More indentation makes lines too long.)

```
pairs([], []).
pairs([H|T], [(Less,More)| PairedTail]) :-
  Less is H - 1, More is H + 1,  % these go well together
  pairs(T, PairedTail).
```

Some people like to separate clauses with a blank new line. (I don't.)

6 Commands/queries in source files

You can put Prolog commands/queries in source files. They are executed when the source file is consulted/compiled.

For example, if you want your program to consult/compile another program `auxiliary.pl` automatically add at the beginning of your source file:

```
:- compile(auxiliary).
```

or

```
:- compile('auxiliary.pl').
```

Alternatively (better) use `ensure_loaded` instead of `compile`.

```
:- ensure_loaded(auxiliary).
```

(It does what you would expect.)

You might want to declare an operator for readability:

```
:- op(1200, xfx, likes).
```

You can also add write statements, messages, etc, etc.