If-then-else

If $P$ then $Q$, else $R$ can be written as follows:

$$S := P \rightarrow Q ; R.$$ 

Here's an example:

$$\text{max}(X, Y, Z) :=
\begin{align*}
& (X =< Y \\
& \quad \rightarrow Z=Y \\
& \quad ; Z=X \\
& ).
\end{align*}$$

Interestingly, one common use of the cut predicate is to mimic the "if-then-else" construct found in imperative languages. Here's how we can define it:

$$S := P, !, Q.$$  
$$S := - R.$$  

If-then-else (cont)

Another example:

Write a predicate to add an element to a list with the restriction that no duplicates are added to the list. Define the predicate $\text{add}(X, L1, L2)$ to mean "the result of adding $X$ to $L1$ is $L2"."

Here's how to do it with cut:

$$\text{add}(X, L1, L2) :- \text{member}(X, L1), !, L2 = L1.$$
$$\text{add}(X, L1, L2) :- L2 = [X|L1].$$

Here's how to do it using if-then-else:

$$\text{add}(X, L1, L2) :- \text{member}(X, L1) \rightarrow L2 = L1$$
$$\quad ; L2 = [X|L1].$$

univ

The standard built-in predicate called 'univ' (=..) translates a predicate and its arguments into a list whose first element is the predicate name and whose remaining elements are the arguments. It works in reverse as well.

For example,

?- $\text{pred}(\text{arg1, arg2}) =.. \ X.$
$X = [\text{pred, arg1, arg2}]$

?- $\text{pred} =.. \ X.$
$X = [\text{pred}]$

?- $X =.. [\text{pred, arg1, arg1}].$
$X = \text{pred}(\text{arg1, arg2})$

?- $X =.. [\text{pred}].$
$X = \text{pred}$
Example using `univ`

Define polygons figures as follows:

```prolog
square(Side)
triangle(Side1,Side2,Side3)
circle(R)
...
```

We'd like to define a predicate that enlarges each of these figures.

```prolog
enlarge(Fig,Factor,Fig1).
```

Here's one way:

```prolog
enlarge(square(A),F,square(A1) :-
  A1 is F*A.
enlarge(circle(R),F,circle(R1) :-
  R1 is F*R1.
```

Using `univ`, we can do it much more elegantly:

```prolog
enlarge(Fig,F,Fig1) :-
  Fig=..[Type|Parameters],
multiplylist(Parameters,F,Parameters1),
Fig1=..[Type|Parameters1].
multiplylist([],_,[[]]).
multiplylist([X|L],F,[X|L1]) :-
  X is F*X, multiplylist(L,F,L1).
```

call, functor, arg

call allows you to call a predicate. E.g.,

```prolog
Goal=..[Functor | Arglist].
call(Goal).
```

Alternatively, you can do this with `functor` and `arg`.

```prolog
functor(Term,F,N)
```

functor is true if F is the principal functor of Term and
N is the arity of F.

```prolog
arg(N,Term,A)
```

arg is true if A is the Nth argument in Term, assuming
that arguments are numbered from left to right starting
with 1.

E.g.,

```prolog
?- functor(t(f(x),X,t),Fun,Arity).
Fun=
Arity=3
?- arg(2,f(X,t(a),t(b)),Y).
Y=t(a)
?- functor(D,examdate,3),
arg(1,D,22),
arg(2,D,april),
arg(3,D,2004).
D=examdate(22,april,2004)
```

assert/retract

Here is an example illustrating how clauses may be added
and deleted from the Prolog data base. The example
shows how to simulate an assignment statement by using
`assert` and `retract` to modify the association be-
tween a variable and a value.

```prolog
:- dynamic x/1 .
x(0). % provide an initial value
assign(X,V) :- Old =..[X,[]], retract(Old),
New =..[X,V], assert(New).
```

Here is an example using the assign predicate.

```prolog
?- x(N).
N = 0
Yes
?- assign(x,5).
Yes
?- x(N).
N = 5
```

Other Useful Syntax

Semi-colon for disjunction:

```prolog
happy(X) :- fed(X),wellslept(X),drydiaper(X)
;outside(X).
```