Updated summary of Chapter 2, *Prolog Programming in Depth*

Michael A. Covington 2010 September 29

Chapter 2 is basically about **predicates that have side effects**.

Important note

We are using only a small, old-fashioned part of the input-output system. The new ISO standard system is described in the appendix on pages 469-475.

Output predicates

Each output predicate succeeds and has no backtrack points.

write(*term*) Writes the term on standard output.
If *term* is an atom that requires quotes, the quotes are left out.
Thus **write('Hello')** writes **Hello** without quotes.

writeq(*term*) Same, but includes quotes if they are required.
Terms written with **writeq** can be read back in by **read**.

nl Starts a new line on standard output.

put(*number*) Writes, on standard output, the character whose ASCII code is the number. For example, **put(97)** writes **a**.

put_char(*atom*) (NEW) Writes out a one-character atom.
Like **write,** but presumably faster, and it can only output a single-character atom, not any other kind of term.

tell(*atom*) Redirects standard output to the file whose name is given in the atom, such as 'u:\\temp\\myfile.txt'. Write backslashes double.

told Ends redirection of standard output.

Input predicates

Each of these predicates reads something from standard input, then unifies (matches) its argument with what it read, and succeeds if the unification is successful. No backtrack point is set.

read(*X*) Reads a Prolog term followed by a period, and unifies it with *X*.
Gives the atom **end_of_file** if it attempts to read past end of file.
(Rarely used. This is what **consult** uses to read the clauses of your program.)

get0(*X*) Reads a character and unifies its numeric ASCII code with *X*.
For example, if the character is **a**, *X* will be matched to 97.
Gives -1 if it attempts to read past end of file.

get(*X*) Rarely used. Like **get0** but skips whitespace characters.

get_char(*X*) (NEW) Reads a character from standard input as a one-character atom and unifies it with *X*. Gives the atom **end_of_file** if it attempts to read past end of file.

peek_char(*X*) (NEW) Tells you the next character waiting to be read on standard input, but does not actually consume it. Gives the atom **end_of_file** if it attempts to look past end of file.

at_end_of_stream (NEW) True if standard input is at end of file.

see(*atom*) Redirects standard input to the file whose name is given in the atom (analogous to **tell**).

seen Ends redirection of standard input.

Dynamic declarations

A predicate is dynamic (changeable at runtime) if it was created by asserta or assertz or if it is declared dynamic. If you want to have clauses in your program that can be retracted at run time, you need a declaration such as:

:- dynamic mypred/2.

which says, "Predicate **mypred** with 2 arguments is dynamic."

Predicates for modifying the knowledge base

(modifying the program already loaded into memory at run time)

asserta(*clause*) Adds the clause to the knowledge base, putting it before other clauses for the same predicate, if any.

assertz(*clause*) Like **asserta** but puts the new clause after other clauses for the same predicate.

retract(*clause*) Removes the first clause that matches the argument (which can be only partly instantiated, such as **retract(father(X,Y))**). Can backtrack to alternatives.

retractall(*clause*) Removes all the clauses that match the argument. (Not in ISO standard.)

abolish(*name/arity*) Removes the predicate from the knowledge base completely, forgetting its dynamic declaration also.

listing Writes, on standard output, all the dynamic clauses that currently exist. This is a way of saving to a file something that you created with asserta and assertz.

In wI, it apparently writes all clauses