Tutorial 7

prepared by Anya Tafliovich[1]

July 16, 2003

[1]based on material provided by Diane Horton, Eric Joanis, Alfredo Gabaldon

1 Using XSB Prolog

To load (or "consult") a file called file.P:

```
consult('file.P').
consult(file).
['file.P'].
[file].
```

All of the above are equivalent.
Do not forget the dot at the end of the line.

To define some predicates without having to first save them to file,
consult "user":

```
[user].
```

It's a good idea, however, to get in the habit of using files most of the time, because what you type here will be lost as soon as XSB exits.

Many built-in predicates are available in the basics module.
To load it:

```
[basics].
```

To exit XSB:

```
halt.
Ctrl-D
```

2 Debugging in XSB Prolog

```
Traces: let you see every call one by one.

  Turning trace on:

    trace.

  Turning trace off:

    notrace.

Spy point: let you enter trace mode when a particular predicate is called.

  Adding a spy point for predicate pred with arity n
  (i.e., pred takes n arguments):

    spy(pred/n).

  Adding a spy point for predicate pred, tracing every instance of the
  predicate, regardless of arity:

    spy(pred).

  Removing a spy point:

    nospy(pred/n).
    nospy(pred).

Useful commands while in trace mode:

  <enter>: move one step forward
  l: leap causes the program to resume (i.e., to leave trace mode) until
     the next spy point, or until the program is done.
  s: skip the current subprogram:  the current subprogram is executed
     with trace off, and the trace resumes when that subprogram is done.
     Useful to quickly go through long subprograms that you've already
```

debugged and you don't need to go through the details.

Dealing with misbehaving code:

If your code gets into infinite recursion, you can hit Ctrl-C to stop it. The interpreter will put you in break mode, which you can leave by typing "abort.". Ctrl-D will also take you out of break mode, but it then resumes your program rather than aborting it. Unfortunately, the second time you use Ctrl-C, XSB will exit completely.

3 Example 1 - Family Relations

Suppose we have the following database in file ‘‘family.P’’.

```
male(edward).
female(alice).
...
parent(albert,edward).
parent(victoria,edward).
...
```

Then in XSB:

```
| ?- [family].     (loading the program)
yes

| ?- male(albert).
yes

| ?- male(victoria).
no
```

Aside:

In Prolog all variables begin with an Upper case letter;
all constants begin with a lower case letter;

```
| ?- female(Person).
Person = alice ;
Person = victoria ;
no
```

Aside:

To ask for more answers, type ; (a semicolon), then <enter>

```
| ?- parent(Person, edward).
Person = albert ;
Person = victoria ;
no

| ?- parent(Person, edward), female(Person).
Person = victoria ;
no
```

How does Prolog get these answers?
Draw a search tree.

Defining rules:

```
1) sibling (X, Y) :- parent (Z, X), parent (Z, Y).

2) brother (X, Y) :- male (X), sibling (X, Y).

3) niece (X, Y)   :- female (X), parent (Z, X), sibling (Z, Y).

4) cousin (X, Y)  :- parent (Z, X), sibling (Z, W), parent (W, Y).

5) grandparent (X, Y) :- parent (X, Z), parent (Z, Y).

6) second_cousin (X, Y) :- grandparent (Z, X), sibling(Z,V),
                           grandparent (V, Y).
```

4 Example 2 - Trip Planning

Suppose we have the following database loaded.

```
plane(to, ny).
plane(ny, london).
plane(london, bombay).
plane(london, oslo).
plane(bombay, katmandu).
boat(oslo, stockholm).
boat(stockholm, bombay).
boat(bombay, maldives).
...
```

a) cruise (X,Y) -- there is a possible boat journey from X to Y.

```
cruise (X, Y) :- boat (X, Y).
cruise (X, Y) :- boat (X, Z), cruise (Z, Y).
```

b) trip (X, Y) -- there is a possible journey (using plane or boat)
from X to Y.

```
leg (X, Y) :- plane (X, Y).
leg (X, Y) :- boat (X, Y).

trip (X, Y) :- leg (X, Y).
trip (X, Y) :- leg (X, Z), trip (Z, Y).
```

c) stopover (X, Y, S) -- there is a trip from X to Y with a stop in S.

1) Assume that neither X nor Y can equal S.

```
stopover (X, Y, S) :- trip (X, S), trip (S, Y).
```

```
  2) Assume S could be X or Y (or even both):

    hop (X, X).
    hop (X, Y) :- trip (X, Y).

    stopover (X, Y, S) :- hop (X, S), hop (S, Y).


d) plane_cruise (X, Y) -- there is a trip from X to Y that has at least
                          one plane leg, and at least one boat leg.

  plane_cruise (X,Y) :- plane (X,Z), boat (Z,Y).
  plane_cruise (X,Y) :- boat (X,Z), plane (Z,Y).

  plane_cruise (X,Y) :- leg (X,Z), plane_cruise (Z,Y).
  plane_cruise (X,Y) :- leg (Z,Y), plane_cruise (X,Z).


  Some intuition:

    to get a mixed trip: have (plane, boat) or (boat, plane) at some point,
    then add legs at either side of the trip.

  Why not:
     plane_cruise(X,Y) :- plane_cruise(X,Z), leg(Z,Y). -- ?

  Answer: infinite recursion!
```

e) cost (X, Y, C) -- there is a trip from X to Y that costs less than C.

Aside:

In Prolog:	+, −, <, >	are used as usual
	is	is used to test for equality, but the right side must be instantiated with a value

Add costs to "leg" and to "trip":

```
leg (X,Y,C) :- plane (X,Y,C).
leg (X,Y,C) :- boat (X,Y,C).

trip (X,Y,C) :- leg (X,Y,C).
trip (X,Y,C) :- leg (X,Z,C1), trip (Z,Y,C2), C is C1 + C2.
```

Now "cost" is simple, because "trip" is doing the addition for you:

```
cost (X, Y, C) :- trip (X, Y, C_trip), C_trip < C.
```