Programming Paradigms CSI2120 – Winter 2018

Jochen Lang

EECS, University of Ottawa

Canada

uOttawa

L'Université canadienne
Canada's university

Logic Programming in Prolog

- **Databases**
- **Managing the Prolog database of rules**
  - Dynamic rules
  - Adding rules
  - Removing rules
  - Inspecting rules

Organizing the Database: Composite Terms

- **Predicate student**
  - student(jim, white, 17, main, ottawa, ontario, 10, 12, 83, ...)
    - which says that there is a student named Jim White who lives at 17 Main in Ottawa, Ontario born on October 12th, 1983.
- **Better by a combination of terms**
  - student(*name*(john, white), *address*(17, main, ottawa, ontario), *born*(10, 12, 83), ...)
- **Flexible queries to the database**

```
?- student(name(john,_), X, born(_, _, Y)).
?- student(X, address(_, _, _, quebec), _).
?- student(X, address(_, _, ottawa, _), born(_, _, Y)),  Y>87.
```

Databases in Prolog

- **Prolog can be very efficient when it comes to managing a database**
  - Each of the entities are represented using facts
  - Using structures, lists.

Example: Library

- **A record of a book has a call number of the library. A book record stores the location, title and authors.**

```
book(callNum(qa76, '73P76C57', 2003),
     location(  mrt,  general ),
     ['Programming' , in,  'Prolog' ],
     [name( clocksin, [william, f] ),
      name( mellish, [christopher, s] )]).
```

Borrowers

- **A user record for the library contains the user's name, an identifier, an address and number of books currently taken out.**

```
reader(name(blake, [ann]), 33333,
       address([100, main], ottawa, k1a2b2),3).
reader(name(brady,[jim,b]), 12345,
       address([2, second], ottawa, k1n3m3),0).
reader(name(carp,[tony,a]), 765432,
       address([3, third], ottawa, k1k4p4),0).
```

Records of Books on Loan

```
loan(33333,
     callNum(qa76, '73P76C57', 2003),
     date(nov, 25, 2013)).
loan(765432,
     callNum(q336, 'B74', 2001),
     date(oct, 20, 2013)).
```

Query if a Book is Available

```
 ?- book(callNum(X, Y, 1994), Location, Title,
Authors),\+ loan(_,callNum(X, Y, 1994), _).
X = qa76,
Y = '73P76S74',
Location = location(mrt, general),
Title = ['The', art, of, 'Prolog', :, advanced,
programming, techniques],
Authors = [name(sterling, [leon]),
name(shapiro, [ehud, y])]
```

Relationships

- **Looking for books by a given author (by last name)**

```
wrote(Auth, CallN, Title) :-
    book(CallN, _, Title, Authors),
    member(name(Auth,_), Authors).
```

- **Seeing if a book is available**

```
borrowed(Name, Title) :-
    reader(Name, Id, _Addr, _),
    loan(Id, CallN, _DateDue),
    book(CallN, _, Title, _Auths).
```

Managing Loans – Dynamic Rules

- **Define a predicate as dynamic**

```
:- dynamic book/4, reader/4, loan/3.
```

  - New rules can be added to the Prolog database with assert
    - assertz adds the clause at the end of the predicate
    - asserta adds the clause at the beginning of the predicate

```
newLoan(Cn, Id, Due) :-
     assertz(loan(Id, Cn, Due)).
```

- Example: New loan adds a record

```
?- newLoan(callNum(qa76, '73P76S74', 1994),
           12345,
           date(nov, 15, 2013)).
```

Managing Loans – Dynamic Rules II

- **Rules can be removed from the Prolog database with retract**
- **Example: Returning a book**
  - retracting the loan
  - correcting the borrower record

```
returns(Id, Cn) :-
    retract(loan(Id, Cn, _Due)),
    retract(reader(Nm, Id, A, N)),
    N1 is N - 1, % loan was retracted
    assertz(reader(Nm, Id, A, N1)).
```

Inspecting the Database

- **Rules can be listed with `listing\0` and `listing\1`**

```
?- listing(loan).
:- dynamic loan/3.
loan(33333, callNum(qa76, '73P76C57', 2003), date(nov, 25, 2013)).
loan(765432, callNum(q336, 'B74', 2001), date(oct, 20, 2013)).
loan(12345, callNum(qa76, '73P76S74', 1994), date(nov, 15, 2013)).
true.

?- returns( 33333, X ).
X = callNum(qa76, '73P76C57', 2003).

?- listing(loan).
:- dynamic loan/3.
loan(765432, callNum(q336, 'B74', 2001), date(oct, 20, 2013)).
loan(12345, callNum(qa76, '73P76S74', 1994), date(nov, 15, 2013)).
true.
```

Storing Information in the Database

- **Dynamic informs the interpreter that the definition of a predicate may change during execution**
  - The predicate assert can be used to store solutions
  - Warning: assert can lead to strange effects
  - A false relationship can become true at a later time.
  - Example:

```
?- solve(problem, solution).
false.
?- assertz(solve(problem,solution)).
true.
?- solve(problem, solution).
true.
```

Another Example

```
:- dynamic letter/2.

alphabet([a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,s,t,u,v,w,x,y,z]).

% add a rule for each letter queried
letter(A, B):-
  alphabet(C),
  letter(A, C, B),
  asserta(letter(A,B)). % Add a new fact

letter(A, [A|_], 1). % boundary case, letter matches

letter(A, [B|C], D):-
  \+(A=B), % letter does not match, keep searching
  letter(A, C, E),
  D is E+1. % Count on the way out of the recursion

?- letter(h,X).
```

Generating Facts

- **Multiplication table**
  - add a product fact for the multiplication table to 9

```
maketable :- L=[0,1,2,3,4,5,6,7,8,9],
         member(X,L),
         member(Y,L),
         Z is X*Y,
         assertz(product(X,Y,Z)),
         fail.
```

State Machines – An Adventure

- **Keep track of the location with a dynamic fact**

```
 :-dynamic here/1.
here(kitchen).
```

- **List the rooms**

```
room(kitchen).
room(office).
…
```

- **List items in the rooms**

```
location(desk, office).
location(apple, kitchen).
…
```

- **List doors between the rooms**

```
door(office, hall).
door(kitchen, office).
…
```

Adventure Rules

```
connect(X,Y) :- door(X,Y).
connect(X,Y) :- door(Y,X).
```

- **Test if we can go to a room**

```
can_go(Place):-
  here(X),
  connect(X, Place).
```

- **List all places where we can go to from a room**

```
list_connections(Place) :-
  findall(X,connect(Place, X),B),
  tab(2),
  write(B),
  nl.
```

Adventure Rules (cont'd)

- **List all items in a room**

```
list_things(Place) :-
  findall(X,location(X, Place),B),
  tab(2),
  write(B),
  nl.
```

Player Actions

```
goto(Place):-
  can_go(Place),
  move(Place),
  look.

move(Place):-
  retract(here(X)),
  asserta(here(Place)).

look :-
  here(Place),
  write('You are in the '), write(Place), nl,
  write('You can see:'), nl, list_things(Place),
  write('You can go to:'), nl, list_connections(Place).
```

More on Dynamic Predicates

- **All rules of a dynamic predicate can be queried**

```
:- dynamic a/2.
a(1,2).
a(3,4).
a(X,Y):- b(X), b(Y).

?- clause(a(X,Y),B).
X = 1,
Y = 2,
B = true ;
X = 3,
Y = 4,
B = true ;
B = (b(X), b(Y)).
```

Dynamic Predicates - Removing Rules

```
?- listing(a).
:- dynamic a/2.
a(1, 2).
a(3, 4).
a(A, B) :- b(A), b(B).
true.

?- retract((a(A,B):-b(A),b(B))).
true.

?- listing(a).
:- dynamic a/2.
a(1, 2).
a(3, 4).
true.
```

Remove all rules of a predicate

```
 ?- retractall(a(X,Y)).
true.

?- listing(a).
:- dynamic a/2.
true.
```

Saving the Current Rules

- **Change the output stream to a file**

```
?- tell(‘cache.pl').
true.
```

- **Use listing to list all currently existing facts and rules**

```
?- listing.
true.
```

- **Back to the console (file is closed)**

```
?- told.
true.
```

Implementing `findall` Ourselves

```
find_all(X,Goal,Bag) :- post_it(X,Goal),
        gather([],Bag).
post_it(X,Goal) :- call(Goal), % try Goal
       asserta(data999(X)), % assert above others
       fail. % force backtracking
post_it(_,_). % Done, no more solutions
gather(B,Bag) :-
       data999(X), % next recorded solution
       retract(data999(X)), % erase posting
       gather([X|B],Bag), % continue
       !. % cut off tail end
gather(S,S). % Done
```

Summary

- **Databases**
- **Managing the Prolog database of rules**
  - Dynamic rules
  - Adding, removing, inspecting and saving rules
- **Detailed examples**
  - Library
  - Adventure
  - An implementation of findall