Lecture 11: Database Manipulation and Collecting Solutions

• Exercises
  – Solutions to exercises LPN chapter 10

• Theory
  – Discuss database manipulation in Prolog
  – Discuss built-in predicates that collect all solutions to a problem into a single list
Suppose we have the given database. Write all of Prolog's answers to the following queries.

\[\begin{align*}
\text{p(1).} \\
p(2) & : - !. \\
p(3).
\end{align*}\]

?- p(X).
\[
\begin{align*}
X & = 1 \\
X & = 2.
\end{align*}
\]

?- p(X),p(Y).
\[
\begin{align*}
X & = Y, \ Y = 1 \\
X & = 1, \ Y = 2 \\
X & = 2, \ Y = 1 \\
X & = Y, \ Y = 2.
\end{align*}
\]
Solution to Exercise 10.2

First, explain what the program above does.

*The predicate class is true if the second argument is positive if the first is a positive number, negative if the first is negative and zero if the first is 0.*

Second, improve it by adding green cuts.

class(Number, positive) :- Number > 0, !.
class(0, zero) :- !.
class(Number, negative) :- Number < 0.
Lecture 11: Database Manipulation and Collecting Solutions

• Theory
  – Discuss database manipulation in Prolog
  – Discuss built-in predicates that collect all solutions to a problem into a single list
Database Manipulation

• Prolog has five basic database manipulation commands:
  – assert/1
  – asserta/1
  – assertz/1
  – retract/1
  – retractall/1
Database Manipulation

• Prolog has five basic database manipulation commands:

- assert/1
- asserta/1
- assertz/1  \(\text{Adding information}\)
- retract/1
- retractall/1  \(\text{Removing information}\)
Start with an empty database
<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Start with an empty database</td>
<td>yes</td>
</tr>
<tr>
<td>?- listing.</td>
<td>yes</td>
</tr>
</tbody>
</table>
Using assert/1

?- assert(happy(mia)).
yes
Using assert/1

happy(mia).

?- assert(happy(mia)).
yes
?-
Using assert/1

happy(mia).

?- assert(happy(mia)).
   yes
?- listing.
   happy(mia).
?-
Using assert/1

happy(mia).

?- assert(happy(mia)).
yes
?- listing.
happy(mia).
?- assert(happy(vincent)),
    assert(happy(marsellus)),
    assert(happy(butch)), assert (happy(vincent)).
Using assert/1

happy(mia).
happy(vincent).
happy(marsellus).
happy(butch).
happy(vincent).

?- assert(happy(mia)).
yes
?- listing.
happy(mia).
?- assert(happy(vincent)),
    assert(happy(marsellus)),
    assert(happy(butch)), assert (happy(vincent)).
yes
?-
Changing meaning of predicates

- The database manipulations have changed the meaning of the predicate happy/1
- More generally:
  - database manipulation commands give us the ability to change the meaning of predicates during runtime
Dynamic and Static Predicates

• Predicates whose meaning changing during runtime are called **dynamic** predicates
  – happy/1 is a dynamic predicate
  – Some Prolog interpreters require a declaration of dynamic predicates

• Ordinary predicates are sometimes referred to as **static** predicates
Asserting rules

happy(mia).
happy(vincent).
happy(marsellus).
happy(butch).
happy(vincent).

?- assert( (naive(X):- happy(X)).
Asserting rules

happy(mia).
happy(vincent).
happy(marsellus).
happy(butch).
happy(vincent).

naive(A):- happy(A).

?- assert( (naive(X):- happy(X)).
yes
?-
Removing information

• Now we know how to add information to the Prolog database
  – We do this with the `assert/1` predicate

• How do we remove information?
  – We do this with the `retract/1` predicate, this will remove one clause
  – We can remove several clauses simultaneously with the `retractall/1` predicate
Using retract/1

happy(mia).
happy(vincent).
happy(marsellus).
happy(butch).
happy(vincent).

naive(A):- happy(A).

?- retract(happy(marsellus)).
Using retract/1

happy(mia).
happy(vincent).
happy(butch).
happy(vincent).

naive(A):- happy(A).

?- retract(happy(marsellus)).
   yes
?-
Using retract/1

happy(mia).
happy(vincent).
happy(butch).
happy(vincent).

naive(A):- happy(A).

?- retract(happy(marsellus)).
yes
?- retract(happy(vincent)).
Using retract/1

happy(mia).
happy(butch).
happy(vincent).

naive(A):- happy(A).

?- retract(happy(marsellus)).
yes
?- retract(happy(vincent)).
yes
Using retract/1

happy(mia).
happy(butch).
happy(vincent).

naive(A):- happy(A).

?- retract(happy(X)).
Using retract/1

naive(A):- happy(A).

?- retract(happy(X)).
  X=mia;
  X=butch;
  X=vincent;
  no
  ?-
Using asserta/1 and assertz/1

- If we want more control over where the asserted material is placed we can use the variants of assert/1:
  - **asserta/1**
    places asserted material at the **beginning** of the database
  - **assertz/1**
    places asserted material at the **end** of the database
Memoisation

- Database manipulation is a useful technique
- It is especially useful for storing the results to computations, in case we need to recalculate the same query
- This is often called memoisation or caching
Example of memoisation

:- dynamic lookup/3.

addAndSquare(X,Y,Res):-
    lookup(X,Y,Res), !.

addAndSquare(X,Y,Res):-
    Res is (X+Y) * (X+Y),
    assert(lookup(X,Y,Res)).
Example of memoisation

:- dynamic lookup/3.

addAndSquare(X,Y,Res):-
    lookup(X,Y,Res), !.

addAndSquare(X,Y,Res):-
    Res is (X+Y) * (X+Y),
    assert(lookup(X,Y,Res)).

?- addAndSquare(3,7,X).
Example of memoisation

:- dynamic lookup/3.

addAndSquare(X,Y,Res):-
    lookup(X,Y,Res), !.

addAndSquare(X,Y,Res):-
    Res is (X+Y) * (X+Y),
    assert(lookup(X,Y,Res)).

lookup(3,7,100).

?- addAndSquare(3,7,X).
X=100
yes
?-
Example of memoisation

:- dynamic lookup/3.

addAndSquare(X,Y,Res):-
    lookup(X,Y,Res), !.

addAndSquare(X,Y,Res):-
    Res is (X+Y) * (X+Y),
    assert(lookup(X,Y,Res)).

lookup(3,7,100).

?- addAndSquare(3,7,X).
   X=100
   yes
?- addAndSquare(3,4,X).
Example of memoisation

:- dynamic lookup/3.
addAndSquare(X,Y,Res):-
    lookup(X,Y,Res), !.
addAndSquare(X,Y,Res):-
    Res is (X+Y) * (X+Y),
    assert(lookup(X,Y,Res)).
lookup(3,7,100).
lookup(3,4,49).

?- addAndSquare(3,7,X).
   X=100
   yes
?- addAndSquare(3,4,X).
   X=49
   yes
Using retractall/1

```prolog
:- dynamic lookup/3.

addAndSquare(X,Y,Res):-
    lookup(X,Y,Res), !.

addAndSquare(X,Y,Res):-
    Res is (X+Y) * (X+Y),
    assert(lookup(X,Y,Res)).

lookup(3,7,100).
lookup(3,4,49).

?- retractall(lookup(_,_,_)).
```
Using retractall/1

:- dynamic lookup/3.

addAndSquare(X,Y,Res):-
    lookup(X,Y,Res), !.

addAndSquare(X,Y,Res):-
    Res is (X+Y) * (X+Y),
    assert(lookup(X,Y,Res)).

?- retractall(lookup(_, _, _)).
yes
?-

© Patrick Blackburn, Johan Bos & Kristina Striegnitz
Red and Green Cuts

Red cut

:- dynamic lookup/3.

addAndSquare(X,Y,Res):-
    lookup(X,Y,Res), !.

addAndSquare(X,Y,Res):-
    Res is (X+Y) * (X+Y),
    assert(lookup(X,Y,Res)).
Red and Green Cuts

**Red cut**

:- dynamic lookup/3.

addAndSquare(X,Y,Res):-
lookup(X,Y,Res), !.

addAndSquare(X,Y,Res):-
Res is (X+Y) * (X+Y),
assert(lookup(X,Y,Res)).

**Green cuts**

:- dynamic lookup/3.

addAndSquare(X,Y,Res):-
lookup(X,Y,Res), !.

addAndSquare(X,Y,Res):-
\+ lookup(X,Y,Res), !,
Res is (X+Y) * (X+Y),
assert(lookup(X,Y,Res)).
A word of warning...

• A word of warning on database manipulation:
  – Often is a useful technique
  – But can lead to dirty, hard to understand code
  – It is non declarative, non logical
  – So should be used cautiously

• Prolog interpreters also differ in the way `assert/1` and `retract/1` are implemented with respect to backtracking
  – Either the assert or retract operation is cancelled over backtracking, or not
Consider this database

child(martha, charlotte).
child(charlotte, caroline).
child(caroline, laura).
child(laura, rose).

descend(X, Y):=- child(X, Y).
descend(X, Y):=- child(X, Z),
                descend(Z, Y).

?- descend(martha, X).
   X=charlotte;
   X=caroline;
   X=laura;
   X=rose;
   no
Collecting solutions

- There may be many solutions to a Prolog query
- However, Prolog generates solutions one by one
- Sometimes we would like to have all the solutions to a query in one go
- Needless to say, it would be handy to have them in a neat, usable format
Collecting solutions

• Prolog has three built-in predicates that do this: **findall/3**, **bagof/3** and **setof/3**

• In essence, all these predicates collect all the solutions to a query and put them into a single list

• But there are important differences between them
findall/3

• The query

?- findall(O,G,L).

produces a list L of all the objects O that satisfy the goal G

- Always succeeds
- Unifies L with empty list if G cannot be satisfied
A findall/3 example

child(martha,charlotte).
child(charlotte,caroline).
child(caroline,laura).
child(laura,rose).

descend(X,Y):- child(X,Y).
descend(X,Y):- child(X,Z),
                descend(Z,Y).

?- findall(X,descend(martha,X),L).
L=[charlotte,caroline,laura,rose]
yes
Other findall/3 examples

child(martha,charlotte).
child(charlotte,caroline).
child(caroline,laura).
child(laura,rose).

descend(X,Y):- child(X,Y).
descend(X,Y):- child(X,Z),
              descend(Z,Y).

?- findall(f:X,descend(martha,X),L).
L=[f:charlotte,f:caroline,f:laura,f:rose]
yes
Other findall/3 examples

child(martha,charlotte).
child(charlotte,caroline).
child(caroline,laura).
child(laura,rose).

descend(X,Y):- child(X,Y).
descend(X,Y):- child(X,Z),
              descend(Z,Y).

?- findall(X,descend(rose,X),L).
L=[ ]
yes
Other findall/3 examples

child(martha,charlotte).
child(charlotte,caroline).
child(caroline,laura).
child(laura,rose).

descend(X,Y):- child(X,Y).
descend(X,Y):- child(X,Z),
             descend(Z,Y).

?- findall(d,descend(martha,X),L).
L=[d,d,d,d]
yes
findall/3 is sometimes rather crude

```
child(martha, charlotte).
child(charlotte, caroline).
child(caroline, laura).
child(laura, rose).

descend(X, Y):- child(X, Y).
descend(X, Y):- child(X, Z), descend(Z, Y).

?- findall(Chi, descend(Mot, Chi), L).
L=[charlotte, caroline, laura, rose, caroline, laura, rose, laura, rose, rose]
yes
```
The query

\[
?- \text{bagof}(O,G,L).
\]

produces a list \(L\) of all the objects \(O\) that satisfy the goal \(G\)

- Only succeeds if the goal \(G\) succeeds
- Binds free variables in \(G\)
Using bagof/3

\[
\begin{align*}
\text{child}(\text{martha}, \text{charlotte}) & . \\
\text{child}(\text{charlotte}, \text{caroline}) & . \\
\text{child}(\text{caroline}, \text{laura}) & . \\
\text{child}(\text{laura}, \text{rose}) & . \\
\end{align*}
\]

\[
\begin{align*}
\text{descend}(X,Y):- & \text{child}(X,Y) . \\
\text{descend}(X,Y):- & \text{child}(X,Z), \text{descend}(Z,Y) . \\
\end{align*}
\]

?- bagof(Chi, descend(Mot, Chi), L).
Mot=caroline
L=[laura, rose];
Mot=charlotte
L=[caroline, laura, rose];
Mot=laura
L=[rose];
Mot=martha
L=[charlotte, caroline, laura, rose];
no
Using bagof/3 with ^

child(martha, charlotte).
child(charlotte, caroline).
child(caroline, laura).
child(laura, rose).

descend(X, Y):-
    child(X, Y).
descend(X, Y):-
    child(X, Z),
    descend(Z, Y).

?- bagof(Chi, Mot^descend(Mot, Chi), L).
L = [charlotte, caroline, laura, rose, caroline, laura, rose, laura, rose, rose]
The query

?- setof(O,G,L).

produces a sorted list L of all the objects O that satisfy the goal G
- Only succeeds if the goal G succeeds
- Binds free variables in G
- Remove duplicates from L
- Sorts the answers in L
Using setof/3

child(martha,charlotte).
child(charlotte,caroline).
child(caroline,laura).
child(laura,rose).

descend(X,Y):-
     child(X,Y).
descend(X,Y):-
     child(X,Z),
     descend(Z,Y).

?- bagof(Chi,Mot^descend(Mot,Chi),L).
L=[charlotte, caroline, laura, rose, caroline, laura, rose, laura, rose, rose]
yes

?-
Using setof/3

child(martha,charlotte).
child(charlotte,caroline).
child(caroline,laura).
child(laura,rose).

descend(X,Y):-
    child(X,Y).
descend(X,Y):-
    child(X,Z),
descend(Z,Y).

?- bagof(Chi,Mot^descend(Mot,Chi),L).
L=[charlotte, caroline, laura, rose, caroline, laura, rose, laura, rose, rose]
yes

?- setof(Chi,Mot^descend(Mot,Chi),L).
L=[caroline, charlotte, laura, rose]
yes

?-
Solution to Exercise 11.1

```
q(foo,blug).
q(a,b).
q(1,2).

?- assert(q(a,b)), assertz(q(1,2)), asserta(q(foo,blug)).
```
Solution to Exercise 11.1

?- assert(q(a,b)), assertz(q(1,2)), asserta(q(foo,blug)).
?- retract(q(1,2)), assertz( (p(X) :- h(X)) ).
Solution to Exercise 11.1

q(foo,blug).
q(a,b).
p(X) :- h(X)

?- assert(q(a,b)), assertz(q(1,2)),
    asserta(q(foo,blug)).
?- retract(q(1,2)), assertz( (p(X) :- h(X)) ).
Solution to Exercise 11.1

```
q(foo,blug).
q(a,b).
p(X) :- h(X)

?- assert(q(a,b)), assertz(q(1,2)), asserta(q(foo,blug)).
?- retract(q(1,2)), assertz( (p(X) :- h(X)) ).
?- retract(q(_,_)), fail.
```
Solution to Exercise 11.1

Possibility 1

q(foo,blug).
q(a,b).
p(X) :- h(X)

Possibility 2 (SWI Prolog)

p(X) :- h(X)

?- assert(q(a,b)), assertz(q(1,2)), asserta(q(foo,blug)).
?- retract(q(1,2)), assertz( (p(X) :- h(X)) ).
?- retract(q(_,_)),fail.
Solution to Exercise 11.2

?- findall(X,q(blob,X),List).
List = [blob, blob, blob, blaf, dang, dang, flab].
Solution to Exercise 11.2

\( q(\text{blob}, \text{blug}) \).
\( q(\text{blob}, \text{blag}) \).
\( q(\text{blob}, \text{blig}) \).
\( q(\text{blaf}, \text{blag}) \).
\( q(\text{dang}, \text{dong}) \).
\( q(\text{dang}, \text{blug}) \).

?- bagof(X, q(X,Y), List).
  Y = blag,  
  List = [blob, blaf] ;
  Y = blig,  
  List = [blob] ;
  Y = blob,  
  List = [flab] ;
  Y = blug,  
  List = [blob, dang] ;
  Y = dong,  
  List = [dang].
Solution to Exercise 11.2

\[
\begin{align*}
q(blob, blug) &. & q(blaf, blag) &. & q(flab, blob) &. \\
q(blob, blag) &. & q(dang, dong) &. \\
q(blob, blig) &. & q(dang, blug) &. \\
\end{align*}
\]

?- setof(X, Y ^ q(X, Y), List).
List = [blaf, blob, dang, flab].
Datum: Dienstag, den 6. September 2011
Zeit: 10.30-12.00
Ort: Appelstraße 4 (3703), Multimedia Hörsaal (Raum 023)
Art: Schriftlich
Hilfsmittel: Kein erlaubt
Stoff: Beide KI Vorlesung und Einführung zu Prolog (50%-50%)

Sieh Beispiel Klausuren auf KI Webseite!

Viel Glück!