Constructive Logic (15-317), Spring 2022
Recitation 9: Prolog Part 2 (3-23-2022)
clogic staff

1 Warming up w/ Prolog!

In Prolog, lists are built in similarly to SML. The syntax for pattern matching on a list is `[Head | Tail]`. Using this we can implement a variety of programs for manipulating lists.

Task 1. Implement a prolog program, `mymerge/3` which merges two sorted lists.

```
mymerge(L, [], L).
mymerge([], L, L).
mymerge([H1 | T1], [H2 | T2], [H1 | Out]) :-
    H1 =< H2,
    mymerge(T1, [H2 | T2], Out).
mymerge([H1 | T1], [H2 | T2], [H2 | Out]) :-
    H1 > H2,
    mymerge([H1 | T1], T2, Out).
```

2 Getting hotter w/ Prolog!

In Prolog, lists are built in similarly to SML. The syntax for pattern matching on a list is `[Head | Tail]`. Using this we can implement a variety of programs for manipulating lists.

Task 2. Implement a merge sorting procedure for lists, `mysort/2`, with mode (+, -) that takes in a list and merge sorts it.

Hints:

- Use a helper procedure, `split/3`, that has mode (+, -, -), that takes in a list and splits it in half into two output lists.
- There are two base cases to consider....

```
split([], [], []).
split([X], [X], []).
split([H1 | [H2 | T]], [H1 | L1], [H2 | L2]) :-
    split(T, L1, L2).
mysort([], []).
mysort([X], [X]).
mysort([X1 | [X2 | L]], O) :-
    split([X1 | [X2 | L]], Left, Right),
    mysort(Left, SLeft),
    mysort(Right, SRight),
    mymerge(SLeft, SRight, O).
```

3 Now we're spicy w/ Prolog!

There are 4 men with last names Smith, Carpenter, Baker and Tailor. Very confusingly, their lastname does NOT correspond to their profession (either a tailor, baker, carpenter or smith). They each have a son. These sons have the same last name as their fathers, and even more confusingly, their professions do not correspond to their last names either. For example, Smith is not a smith, and SmithSon is also not a smith.

You also know:

1. No son has the same profession as his father.
2. Baker has the same profession as Carpenter's son.
3. Smith's son is a baker.

Task 3. Find the professions of the fathers and the sons using a Prolog program.

Hints:

- Use a variable for each profession you are trying to find.
- It might be useful to encode the professions in a list.
- List membership may also be useful. Recall that the following rules define list membership:

  ```
  member(X, [X|_]).
  member(X, [H|T]) :- member(X, T).
  ```

- You can say that A is not B.
  Example:

  ```
  A \= B
  ```

  Are there multiple solutions? If so, what constraints could you add to make it so there is only one solution?

```
profession(smith).
profession(baker).
profession(carpenter).
profession(tailor).

professions([(smith,Ps), (baker, Pb), (carpenter,Pc), (tailor, Pt)],
             [(son_of(smith),Pss), (son_of(baker),Pbs), (son_of(carpenter), Pcs), (son_of(taile
  profession(Ps), profession(Pb), profession(Pc), profession(Pt),
  profession(Pss), profession(Pbs), profession(Pcs), profession(Pts),
  % Profession is different from name
  Ps \= smith,
  Pb \= baker,
  Pc \= carpenter,
```

```
Pt \= tailor,
% Son's professions are different from name
Pss \= smith,
Pbs \= baker,
Pcs \= carpenter,
Pts \= tailor,
% No son has the same profession as his father
Ps \= Pss,
Pb \= Pbs,
Pc \= Pcs,
Pt \= Pts,
% Baker has the same profession as Carpenter's son
Pb = Pcs,
% Smith's son is a baker
Pss = baker,
% Professions don't repeat
Ps \= Pb, Ps \= Pc, Ps \= Pt, Pb \= Pc, Pb \= Pt, Pc \= Pt,
Pss \= Pbs, Pss \= Pcs, Pss \= Pts, Pbs \= Pcs, Pbs \= Pts, Pcs \= Pts.
```