Today

- Quiz 10
- Exam 3 overview
- Prolog lists

Assignments

- HW-12 due Thurs
- HW-13 out (due during Final presentations)

Announcements

- Exam 3 on Thursday
Exam 3 Overview

Basics ...

• 5 multipart questions
• 15% of final grade
• closed notes, book, etc.

Topics ...

Reading (1 question, 5 parts)

• Questions from R-6, R-7, R-9, plus general typing question

Lectures/Homework

• Two Haskell questions (≈ 1\(\frac{1}{3}\) on “new” material since last time)
• One Prolog question (unification, basic queries, proof trees)
• One review question on grammars
Lists in Prolog

Lists are special structures

\[ [1,2,3,4] \]
\[ [a,b,c,d] \]
\[ ['a',a,5,r(x,y)] \]

- Lists are “heterogeneous” (can contain values of different “types”) in prolog

Special syntax for accessing the head and tail of a list

\[ [H \mid T] \]

- H is the head and T is the tail

\[ [1\mid[2,3]] = [1,2,3] \]
\[ [1\mid[2\mid[3]]] = [1,2,3] \]
\[ [1,2\mid[3]] = [1,2,3] \]
\[ [1,2,3\mid[]} = [1,2,3] \]

\[ [X\midXs] = [1,2,3] \]
\[ X = 2, \]
\[ Xs = [2,3] \]

- Like Haskell’s (\( :) \)), the (\( \mid \)) operator is used in pattern matching
List predicates: Recursion Strikes Back

How would we write a list member predicate?

In Haskell:

\[
\begin{align*}
\text{member } \_ \ [\] & = \text{False} \\
\text{member } x \ (y:ys) & = \\
& \quad | x == y \quad = \text{True} \\
& \quad | \text{otherwise } = \text{member } x \ ys \\
\end{align*}
\]

In Prolog:

\[
\begin{align*}
\text{member}(X,[X|\_]). \\
\text{member}(X,[\_|T]) \ :- \ \text{member}(X,T).
\end{align*}
\]

This example works like pattern matching in Haskell ...

- try first rule ... if it doesn’t succeed, try next candidate
- unfold the rule
- and so on

Note that we can “reuse” X in the first pattern!

Q: What is the proof tree for member(2,[1,2])?
Q: What is the proof tree for member(2,[1,3])?

Note unlike Haskell we don’t need an explicit False case
How would we write a last predicate?

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

Q: What is the proof tree for last(2,[1,2])?

And more examples of list processing ...

- consider a prefix predicate
  
  ```
  prefix([1,2], [1,2,3,4])
  ```

- we can write this w/out reusing variables:
  
  ```
  prefix([], _).
  prefix([X|T1], [Y|T2]) :- X = Y, prefix(T1, T2).
  ```

- or using shared variables:
  
  ```
  prefix([], _).
  prefix([X|T1], [X|T2]) :- prefix(T1, T2).
  ```
Another example:

- the append predicate

\[
\text{append}([1,2],[3,4],[1,2,3,4])
\]

- and one way to write it:

\[
\begin{align*}
\text{append}([],L,L). \\
\text{append}([H|T],L,[H|\text{NewT}]) :& \quad \text{append}(T,L,\text{NewT}).
\end{align*}
\]

- note that we are “constructing” the list in the head!
  - if \(T\ append\ L\ is\ \text{NewT}\), then \([H|T]\ append\ L\ is\ [H|\text{NewT}]\)
We can use \texttt{append} to define a reverse predicate

\begin{verbatim}
reverse([],[]).
reverse([H|T],L) :- reverse(T,R), append(R,[H],L).
\end{verbatim}

Q: What is the proof tree for \texttt{reverse([1,2], A)}?
Q: What happens if the goal is \texttt{reverse(X, [1,2])}?
   – Hint: Draw the proof tree ...

- The “signature” for reverse is: \texttt{reverse(+List1, ?List2)}
- That is, \texttt{List1} must be an input
- \texttt{List2} can be an input or output

Calculating the length of a list

\begin{verbatim}
length([], 0).
length([_|T], N) :- length(T, N1), N is 1 + N1.
\end{verbatim}

Q: What is the proof tree for \texttt{length([1,2,3], N)}?
Watch out for backtracking ...

A relation to zero-out an unwanted value ...

- e.g., zero_out(3,[1,3,2,3,5],Xs) returns Xs = [1,0,2,0,5]

\[
\begin{align*}
\text{zero_out} & (\_,[\_],[\_]). & \quad \text{\% base case} \\
\text{zero_out} & (X,[X|Ys],[0|Zs]) :- \quad \text{\% found value} \\
& \quad \text{zero_out}(X,Ys,Zs). \\
\text{zero_out} & (X,[Y|Ys],[Y|Zs]) :- \quad \text{\% didn’t find val} \\
& \quad X \leftarrow Y, \text{zero_out}(X,Ys,Zs).
\end{align*}
\]

Q: What happens if we leave out the \( X \leftarrow Y \) relation?

... HINT: Create a proof tree to see!