Administrivia

• Reminder:
  – this Thursday
  – change of venue
  – instead of class here, please attend the Master’s Seminar at 5pm
  – presented by the Cognitive Science Program
  – Speech, Language and Hearing Sciences Building, Rm. 205
  – I’ll be giving a research talk (accessible to non-specialists) on treebanks and statistical natural language parsers
Today’s Topics

• Homework 1 review
• DCG rules and left vs. right recursion recap
• adding parse tree output to DCG rules
Homework 1 Review
Homework 1 Review

• Question 1:
  Using Perl and \texttt{wsj2000.txt}
  
  – What is the maximum number of consonants occurring in a row within a word?
  – How many words are there with that maximum number?
  – List those words
  – Give your Perl program
Homework 1 Review

• Question 2:
  Using your Perl program for Question 1
  – modify your Perl program to report the sentence number as well as the word encountered in Question 1
  – submit your modified program

• Example:
  – 676 Pennsylvania
  – means on line number 676 the word Pennsylvania occurs
Homework 1 Review

• Question 3:
  (optional 438/mandatory 538)

• using Perl and \texttt{wsj2000.txt}
  – find the words with the longest \texttt{palindrome} sequence of letters as a substring
  – give your Perl code

\textbf{Example:}
\textit{common} has a palindrome sequence of length 2: \textit{ommo}
Regular Grammars: recap

• **language**: *Sheeptalk*
  - baa!
  - baaa!
  - ba...a!

• **right recursive regular grammar**:
  - \( s \rightarrow [b], \) ays.
  - \( \text{ays} \rightarrow [a], \) a.
  - \( \text{a} \rightarrow [a], \) exclam.
  - \( \text{a} \rightarrow [a], \) a.
  - \( \text{exclam} \rightarrow [!] \).

• **left recursive regular grammar**:
  - \( s \rightarrow a, [!] \).
  - \( \text{a} \rightarrow \text{firsta}, [a] \).
  - \( \text{a} \rightarrow a, [a] \).
  - \( \text{firsta} \rightarrow b, [a] \).
  - \( b \rightarrow [b] \).

\( \text{?- } s([b,a,a,!] ,[[]]). \)  \( \text{Yes} \)
\( \text{?- } s([b,a,!] ,[[]]). \)  \( \text{Infinite loop} \)
\( \text{?- } s([b,a,a,!] ,[[]]). \)  \( \text{Yes} \)
\( \text{?- } s([b,a,!] ,[[]]). \)  \( \text{No} \)
Regular Grammars: recap

- **left recursive regular grammar:**
  
  \[
  \begin{align*}
  s &\rightarrow a, [!]. \\
  a &\rightarrow \text{firsta}, [a]. \\
  a &\rightarrow a, [a]. \\
  \text{firsta} &\rightarrow b, [a]. \\
  b &\rightarrow [b].
  \end{align*}
  \]

- **given Prolog left-to-right, depth-first evaluation strategy:**

  \[
  \text{?- } s([b,a,!],[]).
  \]

  enters an infinite loop

- **derivation tree:**
Regular Grammars: recap

- left recursive regular grammar:
  
  \[
  \begin{align*}
  s &\rightarrow a, [!]. \\
  a &\rightarrow \text{firsta}, [a]. \\
  a &\rightarrow a, [a]. \\
  \text{firsta} &\rightarrow b, [a]. \\
  b &\rightarrow [b].
  \end{align*}
  \]

- no re-ordering of the grammar rules will prevent Prolog from looping

- re-ordered:
  
  \[
  \begin{align*}
  s &\rightarrow a, [!]. \\
  a &\rightarrow a, [a]. \\
  a &\rightarrow \text{firsta}, [a]. \\
  \text{firsta} &\rightarrow b, [a]. \\
  b &\rightarrow [b].
  \end{align*}
  \]

- makes things worse!
- this DCG will make Prolog loop even on grammatical input
Extra Argument: Parse Tree

- Recovering a parse tree
  - when want Prolog to return more than just Yes/No answers
  - in case of Yes, we can compute a syntax tree representation of the parse
  - by adding an extra argument to nonterminals
  - applies to all grammar rules (not just regular grammars)

Example
- *sheeptalk again*
- **DCG:**
  
  \[
  \begin{align*}
  s & \rightarrow [b], [a], a, [!]. \\
  a & \rightarrow [a]. \quad \text{(base case)} \\
  a & \rightarrow [a], a. \quad \text{(recursive case)}
  \end{align*}
  \]

```
  s
     /\  
    /   
   b    a
     /\  
    /   
   a   a
    /   
   !   a
```

Extra Argument: Parse Tree

• Tree:

```
        s
       / \
      b   a
     /   /
    a   a
   /     /
 a     a
```

```
s(b,a,a(a,a(a)),!)
```

• Prolog data structure:
  
  – term
  – hierarchical
  – allows sequencing of arguments
  – functor(arg_1,..,arg_n)
  – each arg_i could be another term or simple atom
Extra Arguments: Parse Tree

• **DCG**
  - \( s \rightarrow [b],[a],a,(!] \).
  - \( a \rightarrow [a] \). (base case)
  - \( a \rightarrow [a],a \). (right recursive case)

• **base case**
  - \( a \rightarrow [a] \).
  - \( a(subtree) \rightarrow [a] \).
  - \( a(a(a)) \rightarrow [a] \).

• **recursive case**
  - \( a \rightarrow [a],a \).
  - \( a(subtree) \rightarrow [a],a(subtree) \).
  - \( a(a(A)) \rightarrow [a],a(A) \).

**Idea:** for each nonterminal, add an argument to store its subtree
Extra Arguments: Parse Tree

- **Prolog grammar**
  - $s \rightarrow [b], [a], a, ![].$
  - $a \rightarrow [a].$ (base case)
  - $a \rightarrow [a], a.$ (right recursive case)

- **base and recursive cases**
  - $a(a(a)) \rightarrow [a].$
  - $a(a(a,A)) \rightarrow [a], a(A).$

- **start symbol case**
  - $s \rightarrow [b], [a], a, ![].$
  - $s(tree) \rightarrow [b], a(subtree), ![].$
  - $s(s(b,a,A,)) \rightarrow [b], [a], a(A), ![].$
Extra Arguments: Parse Tree

• **Prolog grammar**
  
  - \( s \rightarrow \text{[}b\text{], } \text{[}a\text{], } a, [!] \).
  
  - \( a \rightarrow \text{[}a\text{]} \).  
    (base case)
  
  - \( a \rightarrow \text{[}a\text{], } a \).  
    (right recursive case)

• **Prolog grammar computing a parse**
  
  - \( s(s(b,a,A,!)) \rightarrow \text{[}b\text{], } \text{[}a\text{], } a(A), [!] \).
  
  - \( a(a(a)) \rightarrow \text{[}a\text{]} \).
  
  - \( a(a(a,A)) \rightarrow \text{[}a\text{], } a(A) \).
Class Exercise

- Augment the right recursive regular grammar for Sheeptalk to return a parse tree

- DCG:
  
  \[ s \rightarrow [b], \text{ays}. \]
  
  \[ \text{ays} \rightarrow [a], a. \]
  
  \[ a \rightarrow [a], \text{exclam}. \]
  
  \[ a \rightarrow [a], a. \]
  
  \[ \text{exclam} \rightarrow [!]. \]

- \( ?- s(\text{Tree}, [b, a, a, !], []). \)