Administrivia

• Last Thursday:
  – class replaced by
  – Cognitive Science Master’s Seminar Series talk on Statistical Natural Language Parsing and Treebanks
Administrivia

• **This Thursday**
  – class replaced by this Friday’s Linguistics Colloquium
Administrivia

• **This Friday:**
  – **Computational Linguistics Colloquium**
  – **Presenters:** Sandiway Fong, Mike Hammond and Ying Lin
  – **Time:** 3:00-4:30 PM
  – **Location:** Speech and Hearing 205H
  – **Abstract:**
    – In this talk, we will demonstrate the use of various software to collect and analyze linguistic data of various sorts.
    – Fong will give a tutorial on using his freely-available treebankviewer software to explore syntactic structures in treebanks.
    – Hammond will demonstrate the use of his "Finite State Playground", a suite of command-line tools that can be used to construct and manipulate finite-state language models.
    – Lin will discuss the use of speech technology and statistical tools for phonetic data analysis.
Today’s Topics

• DCG topics
  – language enumeration
  – ambiguity: multiple parses
  – stepping outside regular grammars just a little bit

• Homework 2
Regular Grammars: Language Enumeration

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

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

- this DCG always halts:
  \[
  \begin{align*}
  \text{?- } s & ([b, a, a, !], []). \text{ Yes} \\
  \text{?- } s & ([b, a, !], []). \text{ No}
  \end{align*}
  \]

  as a decider

- **But not in all modes of operation**
  - Prolog’s top-down, left-to-right, depth-first strategy also allows
    the grammar to be used an
    enumerator

  \[
  \begin{align*}
  \text{?- } s & (L, []). \\
  \text{- } L & \text{ is a variable, we’re not naming}
  \text{ the list. Let Prolog figure it out.}
  \end{align*}
  \]
Regular Grammars: Language Enumeration

• **Try**
  
  `- s(L, []).`
  
  – Prolog returns one answer and waits for action
  (type ; to get the next, <return> to finish)

• **Output:**

  `- s(L, []).`
  `L = [b,a,a,!] ? ;`
  `L = [b,a,a,a,!] ? ;`
  `L = [b,a,a,a,a,!] ? ;`
  `L = [b,a,a,a,a,a,!] ? ;`
  `L = [b,a,a,a,a,a,a,!] ? ;`
  `L = [b,a,a,a,a,a,a,a,!] ? ;`

Prolog is enumerating the strings of the language
Regular Grammars: Language Enumeration

- Let’s see why this happens
  \[ s(L,[]). \]

- **DCG:**
  1. \( s \rightarrow [b], \text{ays}. \)
  2. \( \text{ays} \rightarrow [a], a. \)
  3. \( a \rightarrow [a], \text{exclam}. \)
  4. \( a \rightarrow [a], a. \)
  5. \( \text{exclam} \rightarrow [!]. \)

- **derivation tree:**
Regular Grammars: Language Enumeration

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

- doesn’t halt when faced with a string not in the language

- **Example:**
  
  \[- s([b, a, !], []).\]

  enters an infinite loop

- **Question:**
  
  - how well does it do when enumerating?
  
  - i.e.
    
    \[- s(L, []).\]

  - *does this query halt?*
Regular Grammars: Language Enumeration

- **left recursive regular grammar:**
  1. \( s \rightarrow a, \text{[!].} \)
  2. \( a \rightarrow \text{firsta, [a].} \)
  3. \( a \rightarrow a, \text{[a].} \)
  4. \( \text{firsta} \rightarrow b, \text{[a].} \)
  5. \( b \rightarrow \text{[b].} \)

  Doesn’t halt when faced with a string not in the language

- Perhaps surprisingly:
  
  \[ \text{- } s(L, [\text{.}]) \]

  Enumerates just fine.

- **derivation tree:**

  \[
  \begin{array}{c}
  \text{s} \\
  \downarrow \\
  \text{a} \\
  \downarrow \\
  \text{firsta} \\
  \downarrow \\
  \text{b} \\
  \downarrow \\
  \text{b} \\
  \end{array}
  \quad \begin{array}{c}
  \text{s} \\
  \downarrow \\
  \text{a} \\
  \downarrow \\
  \text{firsta} \\
  \downarrow \\
  \text{b} \\
  \downarrow \\
  \text{b} \\
  \end{array}
  \]

  \text{and so on}
Regular Grammars: Language Enumeration

- However, this slightly re-ordered left recursive regular grammar:
  1. $s \rightarrow a, \text{[!]}$.
  2. $a \rightarrow a, \text{[a]}$.
  3. $a \rightarrow \text{firsta}, \text{[a]}$.
  4. $\text{firsta} \rightarrow b, \text{[a]}$.
  5. $b \rightarrow \text{[b]}$.

- won’t halt when enumerating

- Why?

\[
\begin{array}{c}
s \\
\downarrow \\
a \\
\downarrow \\
a \\
\downarrow \\
a \\
\downarrow \\
a \\
\downarrow \\
\vdots \\
\end{array}
\]

descends infinitely using rule 2
DCG: Ambiguity

• A context-free grammar that returns a parse:

avoiding some nasty recursion for infinite loops

\[ s(s(NP,VP)) \rightarrow np(NP), \]
\[ vp(\cdot). \]
\[ np(np(i)) \rightarrow [i]. \]
\[ np(np(D,N)) \rightarrow det(D), noun(N). \]
\[ np(np(D,N,PP)) \rightarrow det(D), noun(N), pp(PP). \]

\[ det(d(\text{the})) \rightarrow [\text{the}]. \]
\[ det(d(a)) \rightarrow [a]. \]
\[ noun(n(\text{boy})) \rightarrow [\text{boy}]. \]
\[ noun(n(\text{telescope})) \rightarrow [\text{telescope}]. \]
\[ vp(vp(V,NP)) \rightarrow verb(V), np(NP). \]
\[ vp(vp(V,NP,PP)) \rightarrow verb(V), np(NP), pp(PP). \]
\[ verb(v(saw)) \rightarrow [\text{saw}]. \]
\[ pp(pp(P,NP)) \rightarrow p(P), np(NP). \]
\[ p(p(\text{with})) \rightarrow [\text{with}]. \]
DCG: Ambiguity

• Parsing:

?- s(X,[i,saw,the,boy,with,a,telescope],[]).
X = s(np(i),vp(v(saw),np(d(the),n(boy),pp(p(with),np(d(a),n(telescope)))))) ;
X = s(np(i),vp(v(saw),np(d(the),n(boy)),pp(p(with),np(d(a),n(telescope)))))) ;
no

• Prolog finds both parses by backtracking.
• But why does Prolog return the parses in this order?

5min exercise: re-arrange grammar to produce the opposite order of parses
Regular Grammars: Stepping Outside

• **Beyond regular grammars**
  - $a^n b^n = \{ab, aabb, aaabbb, aaaabbbb, \ldots \}$ $n \geq 1$
    is not a regular language
    \textit{(accept fact for now)}

• A regular grammar extended to
  allow both left and right recursive rules can accept/generate it
  
  \[ a \rightarrow \{a\}, b. \]
  \[ b \rightarrow \{b\}. \]
  \[ b \rightarrow a, \{b\}. \]

• **Example:**
  this grammar accepts
  - aabb aaaabbbb
  and rejects
  - aab aaaabbbb

• **Intuition:**
  - grammar implements the stacking of partial trees balanced for a’s and b’s:
Homework 2

• Ground rules:
  – all in one single file please
  – no more multiple attachments
Homework 2

• Question 1
  – Consider a language $L_{3or5} = \{111, 11111, 111111, 1111111, 11111111, 111111111, 1111111111, \ldots\}$
  – each member of $L_{3or5}$ is a string containing only 1s
  – the number of 1s in each string is divisible by 3 or 5

• Give a regular grammar that generates language $L_{3or5}$

• Show your Prolog code works on strings in the language and rejects strings outside the language
• **Question 2:** Language \( L = \{a^{2n}b^{n+1} | n \geq 1\} \) is also non-regular

  • nearly twice as many a’s as in the b’s but there’s always one b too many for this to be true

  • but can be generated by a regular grammar extended to allow left and right recursive rules

• Given a Prolog grammar satisfying these rules for \( L \)

• Show it accepts and rejects correctly

• **Legit rules:**
  
  X → aY
  X → a
  X → Ya

  – where \( X, Y \) are nonterminals, \( a \) is some arbitrary terminal
Homework 2

• **Question 3 (Optional 438)**
  A right recursive regular grammar that generates a (rightmost) parse for the language \{a^n | n \geq 2\}:
  \[
  \begin{align*}
  &s(s(a,B)) \rightarrow [a], b(B). \\
  &b(b(a,B)) \rightarrow [a], b(B). \\
  &b(b(a)) \rightarrow [a].
  \end{align*}
  \]

• **Example**
  \[
  \begin{align*}
  &?- s(Tree,[a,a,a],[]). \\
  &\quad Tree = s(b(b(a),a),a)
  \end{align*}
  \]

• **Can you do this for any right recursive regular grammar?** e.g. the one for sheeptalk

• **A corresponding left recursive regular grammar will not halt in all cases when an input string is supplied**

• **Modify the right recursive grammar to produce a left recursive parse, e.g.**

  \[
  \begin{align*}
  &?- s(Tree,[a,a,a],[]). \\
  &\quad Tree = s(b(b(a),a),a)
  \end{align*}
  \]

• **Explain your answer**