*October 14, 2004*

*Chapter 10.1–10.2*

*CFGs, Parsing*

# *Overview*

- What is parsing?
- Inadequate grammars for human languages
    - lists,
    - regex,
    - CFG
- A simple top-down parser for CFGs

# *What is Parsing?*

- Assigning (syntactic) structure to input strings
- Here: Based on context-free grammars
- What other knowledge systems could a parser use?

## *Some inadequate grammars*

- Lists of sentences
- Lists of regular expressions of parts of speech
- Context free grammars

# *Some inadequate grammars*

- Lists of sentences: Why?
- Lists of regular expressions of parts of speech
- Context free grammars

## *Why are lists inadequate?*

- Learnability
- No predictions about what makes a possible human language
- No representation of sentence structure

## *Some inadequate grammars*

- ~~Lists of sentences~~
- Lists of regular expressions of parts of speech: Why?
- Context free grammars

## *Why are regular expressions inadequate?*

- Redundancy: There are many places where you can get nominal groups, and you can always get any kind of nominal group there.
- Structural ambiguity: no representation
- Dependencies (first step towards semantics): no representation

## *Some inadequate grammars*

- ~~Lists of sentences~~
- ~~Lists of regular expressions of parts of speech~~
- Context free grammars: What? (then Why?)

# *Context-Free Grammar*

- A CFG is a quadruple: $\langle C, \Sigma, P, S \rangle$:
  - $C$ is the set of *categories* (aka *non-terminals*, e.g., { S, NP, VP, V };
  - $\Sigma$ is the *vocabulary* (aka *terminals*), e.g., { Kim, snow, adores }
  - $P$ is a set of *rewrite rules* of the form $\alpha \longrightarrow \beta_1, \beta_2, \ldots, \beta_n$
  - $S \in C$ is the *start-symbol*
  - For each rule '$\alpha \longrightarrow \beta_1, \beta_2, \ldots, \beta_n' \in P : \alpha \in C$and$\beta_i \in C \cup \Sigma; 1 \leq i \leq n$.

# *Context-Free Languages*

- CFGs are more powerful than regular expressions/FSAs – that is, the former can generate languages that the latter can't.
- A case in point: $a^n b^n$
  - S $\longrightarrow$ a S b
  - S $\longrightarrow$ $\epsilon$

# *Context-Free Languages*

- There are languages CFGs can't generate (non-context-free languages), notably those that incorporate *cross-serial dependencies*, such as Swiss German.
- Perhaps more importantly, CFGs are cumbersome and inefficient for representing natural language syntax.
- Most (but not all) modern theories of syntax include a notion of phrase structure (CFG), and then extend it.

# *Swiss German example (Shieber 1985) (1/2)*

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| …mer | d'chind | em Hans | es | huus | lönd | hälfe | aastriiche |
| …we | the children-ACC | Hans-DAT | the | house-ACC | let | help | paint |

'…we let the children help Hans paint the house'

- Cross-serial dependency:
  - *let* governs the case on *children*
  - *help* governs the case on *Hans*
  - *paint* governs the case on *house*

## *Swiss German example (Shieber 1985) (2/2)*

- Define a new language $f$(Swiss German)=

| | | | | | |
|---|---|---|---|---|---|
| $f$(d'chind) | = | a | $f$(Jan säit das mer) | = | w |
| $f$(em Hans) | = | b | $f$(es huus) | = | x |
| $f$(lönde) | = | c | $f$(aastriiche) | = | y |
| $f$(hälfe) | = | d | $f$([other]) | = | z |

- Let $r$ be the regular language $wa^*b^*xc^*d^*y$.
- $f(\mathrm{S}wissGerman) \cap r = wa^m b^n xc^m d^n y$
- $wa^m b^n xc^m d^n y$ is not context-free
- Context free languages are closed under intersection
- $\therefore$ Swiss German is not context-free.

# *Strongly v. weakly context-free*

- A language is *weakly* context-free if the set of strings in the language can be generated by a CFG.
- A language is *strongly* context-free if it is weakly context free and the set of structures assigned to the strings by the CFG are the right ones.
- Shieber's proof shows that Swiss German is *weakly* not context-free and therefore *a fortiori strongly* not context-free.
- A prior paper by Bresnan et al had argued that Dutch was *strongly* not context-free, but the argument was dependent on linguistic analyses.

# *Toy CFG*

- $C$: { S, VP, PP, NP, V, P }
- $\Sigma$: { Kim, snow, Oslo, adores, in, snores }
- $P$: (see next slide)
- $S$: S

# *Toy CFG*

$S \longrightarrow NP\ VP$

$VP \longrightarrow V\ NP$

$VP \longrightarrow V$

$VP \longrightarrow VP\ PP$

$PP \longrightarrow P\ NP$

$NP \longrightarrow NOM\ PP$

$NP \longrightarrow NOM$

$NOM \longrightarrow Kim$

$NOM \longrightarrow snow$

$NOM \longrightarrow Oslo$

$V \longrightarrow adores$

$V \longrightarrow snores$

$P \longrightarrow in$

## *Human (deliberate) parsing*

- What tree does the toy CFG assign to this sentence:
  *Kim adores snow in Oslo*
- How did you determine that?

# *Four parameters of parsing algorithms*

- Top-down v. bottom-up
- Breadth-first v. depth-first
- Best-first v. exhaustive
- Uni- v. bi-directional

## *Outline of* Top-Down-Parse

- Initialize agenda with (S, first word)
- Pop that state off the agenda
- Loop:
  - Check if we're fi nished, if so return tree
  - Check if the node we're trying to expand is a POS
  - If so, check whether the current word of the input has the current node as a possible POS
  - If so, apply the lexical rules of the grammar to that node to build more trees, and add results to the agenda. (NB - Apply-Lexical-Rules will have to return (tree, word) pairs, where the word is the next word in the string.)

# *Outline of* TOP-DOWN-PARSE

- Loop:

  ...

  - If the current node wasn't a part of speech, apply the non-lexical rules of the grammar to that node, and add the resulting search states to the agenda.
  - If after doing all that, the agenda is empty, reject the sentence.
  - Otherwise, take the next search state of the top of the agenda, and do the loop again.

# *Corrected version of* TOP-DOWN-PARSE

```
function TOP-DOWN-PARSE(input,grammar) returns a parse tree
  agenda ← (Initial S tree, Beginning of input)
  css ← POP(agenda)
  loop
    if SUCCESSFUL-PARSE?(css) then
      return TREE(css)
    else
      if CAT(NODE-TO-EXPAND(css)) is a POS then
        if CAT(NODE-TO-EXPAND(node-to-expand)) ∈
           POS(CURRENT-INPUT(css)) then
            PUSH(APPLY-LEXICAL-RULE(css,grammar),agenda)
      else
        PUSH(APPLY-RULES(css,grammar),agenda)
      if agenda is empty then
        return reject
      else
        css ← POP(agenda)
  end
```

# *Four parameters and* TOP-DOWN-PARSE

- Top-down or bottom-up?
- Breadth-first or depth-first?
- Best-first or exhaustive?
- Uni- or bi-directional?

# *Problems with this algorithm*

- What happens when you parse *Kim adores snow in Oslo*?
- What happens when you add the rule NP $\longrightarrow$ NP PP and try to parse *Kim adores snow in Oslo*?
- Inefficient reparsing of subtrees
- ... solve all three with dynamic programming: chart parsing.

# *A note on the homework*

- Assignment 2 is due next Thursday.
- The parsing section of Assignment 2 has you examining the final state of the chart in a chart parser after it has parsed various sentences.
- The LKB chart parser is not an implementation of the Earley chart parser. It only stores completed (or 'passive') edges.

# *Overview*

- What is parsing?
- Inadequate grammars for human languages
    - lists,
    - regex,
    - CFG
- A simple top-down parser for CFGs

# *Next time*

- A better parser for CFGs
- Finite state parsing