

# Parsing context-free grammars

David Chiang

4 Mar 2008

The notation used in these lectures comes from: Stuart M. Shieber, Yves Schabes, and Fernando C. N. Pereira. Principles and implementation of deductive parsing. *Journal of Logic Programming*, 24(1-2):3-36, July–August 1995. <http://www.eecs.harvard.edu/~shieber/Biblio/Papers/infer.pdf>

## 1 Formal language theory basics

- Alphabet: a finite set of symbols
- String: a finite, possibly empty sequence of symbols drawn from an alphabet
  - Customarily, we use variables  $w, v, u$  to range over strings.
  - The empty string is denoted  $\epsilon$ . It is *not* a symbol of an alphabet, but a metasympol
  - We write  $v \cdot w$  or just  $vw$  for *concatenation*, and  $(w^n)$  for repeated concatenation
  - True or false:  $a \cdot b = b \cdot a$ ?  $a(bc) = (ab)c$ ?  $a \cdot \epsilon = a$ ?  $a \cdot \epsilon = \epsilon \cdot a$ ? If  $v = \epsilon$  and  $w = ab$ ,  $vw = w$ ?
  - What is  $\epsilon^5$ ?  $a^0$ ?
- Language: a possibly infinite, possibly empty set of strings over an alphabet
  - The empty language is  $\emptyset$
  - True or false:  $\emptyset = \{\epsilon\}$ ?  $\{a\} \cup \{\epsilon\} = \{a\}$ ?
  - What is an example of an infinite set of finite strings?
- Grammar: a finite description of a possibly infinite language
- Parsing: does a string  $w$  belong to the language of a grammar  $G$ , and if so, with what structure (tree)?

## 2 Review: shift-reduce parsing

- Parser state (or item):  $[\alpha \bullet v]$ 
  - $\alpha$  is the parse stack, a sequence of partial parse trees
  - $v$  is the input list, the suffix of the sentence not yet parsed
- Start state (axiom):  $[\bullet w]$
- Goal state:  $[S \bullet]$
- Operations (inference rules; item above line is the *antecedent* and item below line is the *consequent*)

$$\begin{array}{l} \text{Shift} \quad \frac{[\alpha \bullet av]}{[\alpha a \bullet v]} \\ \\ \text{Reduce} \quad \frac{[\alpha \beta \bullet v]}{[\alpha X \bullet v]} \quad (X \rightarrow \beta) \in G \end{array}$$

- Defines a search space, which we can think of as a directed graph, or as a proof system.
- How do you know which operation to use? Ulf's parser: decision tree
- Much slower but guaranteed-correct solution: backtracking
  1. Initialize the *chart* to  $\emptyset$  and the *agenda* to the start state
  2. Repeat:
    - (a) Take an item from the agenda (call it the *trigger*) and move it to the chart.
    - (b) Form all possible consequents of the trigger, and add them to the agenda (if not already in chart or agenda).until goal is in chart (return yes) or agenda is empty (return no)
- If the agenda is a stack, we get depth-first search; if it is a queue, we get breadth-first search.

Disadvantages of this algorithm:

- Exponential time
- Not very goal-driven

### Example

S  $\rightarrow$  NP VP  
S  $\rightarrow$  VP  
NP  $\rightarrow$  John  
NP  $\rightarrow$  Det N  
NP  $\rightarrow$  NP PP  
Det  $\rightarrow$  the  
N  $\rightarrow$  man | telescope  
VP  $\rightarrow$  V NP  
VP  $\rightarrow$  VP PP  
V  $\rightarrow$  saw | see  
P  $\rightarrow$  with

0 John 1 saw 2 the 3 man 4 with 5 the 6 telescope 7

[•John saw the man with the telescope]  
[John •saw the man with the telescope]  
[NP •saw the man with the telescope]  
[NP saw •the man with the telescope]  
[NP V •the man with the telescope]  
[NP V the •man with the telescope]  
[NP V Det •man with the telescope]  
[NP V Det man •with the telescope]  
[NP V Det N •with the telescope]  
[NP V NP •with the telescope]

### 3 Top-down parsing

Now we turn to an algorithm which is the opposite of shift-reduce parsing: instead of starting from the words and trying to build up to  $S$ , we start with  $S$  and try to build down to the words.

- Parser state:  $[u \bullet \alpha]$
- Start state (axiom):  $[\bullet S]$
- Goal state:  $[w \bullet]$
- Operations (inference rules)

$$\text{Predict } \frac{[u \bullet X\gamma]}{[u \bullet \beta\gamma]} \quad (X \rightarrow \beta) \in G$$

$$\text{Shift } \frac{[u \bullet a\gamma]}{[ua \bullet \gamma]}$$

Disadvantages of this algorithm:

- Exponential time
- Not very input-driven. In fact, with rules of the form  $X \rightarrow X\beta$ , it may not even terminate.

## 4 Earley parser

Bottom-up (shift-reduce) parsing is not very goal-driven and top-down parsing is not very input-driven. What happens if we try to combine the two algorithms? The states can have partial parses to the left *and* to the right of the dot, and now we also need to keep track of where we are in the input string.

- Parser state:  $[\alpha \bullet \beta, j]$ .  $j$  is the position of the dot.
- Start state (axiom):  $[\bullet S, 0]$
- Goal state:  $[S \bullet, n]$
- Operations (inference rules)

$$\begin{array}{l} \text{Predict} \quad \frac{[\alpha \bullet X \gamma, j]}{[\alpha \bullet \beta \gamma, j]} \quad (X \rightarrow \beta) \in G \\ \text{Shift} \quad \frac{[\alpha \bullet w_{j+1} \gamma, j]}{[\alpha w_{j+1} \bullet \gamma, j + 1]} \\ \text{Reduce} \quad \frac{[\alpha \beta \bullet \gamma, j]}{[\alpha X \bullet \gamma, j]} \quad (X \rightarrow \beta) \in G \end{array}$$

Unfortunately this algorithm has all the weaknesses of both algorithms combined!

### Example

$$\begin{array}{l} [\text{NP V NP} \bullet \text{PP}, 4] \qquad [\text{NP VP} \bullet \text{PP}, 4] \\ [\text{NP V NP} \bullet \text{P NP}, 4] \\ [\text{NP V NP} \bullet \text{with NP}, 4] \\ [\text{NP V NP with} \bullet \text{NP}, 5] \\ [\text{NP V NP P} \bullet \text{NP}, 5] \\ [\text{NP V NP P} \bullet \text{Det N}, 5] \\ [\text{NP V NP P} \bullet \text{the N}, 5] \\ [\text{NP V NP P the} \bullet \text{N}, 6] \\ [\text{NP V NP P Det} \bullet \text{N}, 6] \\ [\text{NP V NP P Det} \bullet \text{telescope}, 6] \\ [\text{NP V NP P Det telescope} \bullet, 7] \\ [\text{NP V NP P Det N} \bullet, 7] \\ [\text{NP V NP P NP} \bullet, 7] \\ [\text{NP V NP PP} \bullet, 7] \end{array}$$

The above is an example of redundant search paths, which we can write more abstractly as:

$$\frac{[\alpha \bullet X\gamma, j]}{[\alpha \bullet \beta\gamma, j]}$$

$$\vdots$$

$$\frac{[\alpha\beta \bullet \gamma, k]}{[\alpha X \bullet \gamma, k]}$$

The parsing of  $\beta$  will be repeated for many different values of  $\alpha$  and  $\gamma$ , and the only thing it depends on is  $j$ . But we can merge all these redundant paths by dropping  $\alpha$  and  $\gamma$  at the Predict step, and then recovering them at the Reduce step. To recover them correctly, it is enough for each state to remember the position of its left boundary.

$$\frac{[i, \alpha \bullet X\gamma, j]}{[j, \bullet\beta, j]}$$

$$\vdots$$

$$\frac{[i, \alpha \bullet X\gamma, j] \quad [j, \beta \bullet, k]}{[i, \alpha X \bullet \gamma, k]}$$

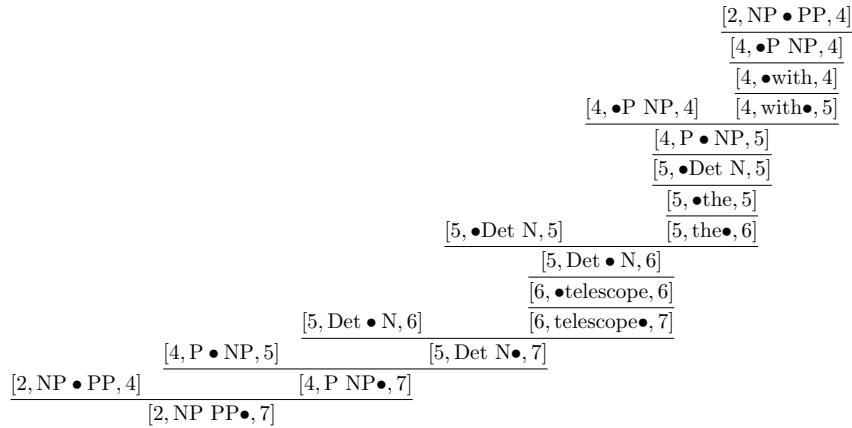
That means our algorithm will look like this:

- Parser state:  $[i, \alpha \bullet \beta, j]$ .  $i$  is the position of the left boundary,  $j$  is the position of the dot.
- Start state (axiom):  $[0, \bullet S, 0]$
- Goal state:  $[0, S \bullet, n]$
- Operations (inference rules)

$$\text{Predict} \quad \frac{[i, \alpha \bullet X\gamma, j]}{[j, \bullet\beta, j]} \quad (X \rightarrow \beta) \in G$$

$$\text{Shift} \quad \frac{[i, \alpha \bullet w_{j+1}\gamma, j]}{[i, \alpha w_{j+1} \bullet \gamma, j+1]}$$

$$\text{Complete} \quad \frac{[i, \alpha \bullet X\gamma, j] \quad [j, \beta \bullet, k]}{[i, \alpha X \bullet \gamma, k]} \quad (X \rightarrow \beta) \in G$$



Since the Complete rule (formerly known as Reduce) now has two antecedents, the search space now has the structure of a *hypergraph*. So we need a modified search algorithm.

1. Initialize the *chart* to  $\emptyset$  and the *agenda* to the start state
2. Repeat:
  - (a) Take an item from the agenda (call it the *trigger*) and move it to the chart.
  - (b) Form all possible consequents of the trigger *with other items from the chart*, and add them to the agenda (if not already in chart or agenda).

until goal is in chart (return yes) or agenda is empty (return no)

Remarks:

- The running time of this algorithm is  $\mathcal{O}(|G|^2 n^3)$ .
- The new part of the search algorithm, finding co-antecedents, needs to be done in constant time. Usually one or more indexes are needed for this.
- Building trees
  - Can't store the trees in the states and maintain polynomiality
  - So, whenever we create a new item, store a pointer to its antecedents
  - When we compare two items, we do *not* compare the pointers
  - Instead, when we create an item that already exists, add the pointer to the existing list of pointers
  - Then, after the search is done, we can follow the pointers back to build trees (there may be exponentially or infinitely many).