1 Motivation

- Parsing is used loosely to encompass two tasks:
  - Recognition: decide whether an input string belongs to a language
  - Parsing: calculate a possible structure or all possible structures of an input string, if any. Parsing is often statistical: choose the best structure or $k$ best structures based on some model

- An essential component of the ideal NLP pipeline

- Lots of different parsing algorithms
  - Bottom-up (shift-reduce)
  - Top-down
  - Left-corner
  - Earley, Graham-Harrison-Ruzzo
  - CKY (Cocke-Kasami-Younger): simple, efficient, has some nice properties for statistical parsing. Probably the most common in NLP literature today

2 Chomsky normal form

Definition We say that a CFG is in Chomsky normal form if each of its productions has one of the following forms:

- $X \rightarrow YZ$
- $X \rightarrow a$

Construction Any CFG $G$ that does not generate $\epsilon$ can be converted into a weakly equivalent CFG (i.e., one generating the same language but not necessarily the same trees) in Chomsky normal form.

1. $\epsilon$ removal

   (a) Compute the set of nullable symbols $\{X \mid X \Rightarrow^* \epsilon\}$ as follows:
i. If \((X \rightarrow \epsilon) \in G\), then \(X\) is nullable
ii. If \((X \rightarrow Y_1 \cdots Y_n) \in G\) and \(Y_1, \ldots, Y_n\) are all nullable, then \(X\) is nullable

(b) For each production \((X \rightarrow Y_1 \cdots Y_n) \in G\), add all productions formed by deleting one or more nullable \(Y_i\) (but don’t add \(X \rightarrow \epsilon\))

2. Unit removal

(a) Form a directed graph out of all unit productions \(X \rightarrow Y\). The transitive closure of this graph is the relation \(X \Rightarrow^* Y\)

(b) For each production \((X \rightarrow Y_1 \cdots Y_n) \in G\), add all productions formed by replacing one or more \(Y_i\) with a \(Z_i\) s.t. \(Y_i \Rightarrow^* Z_i\)

3. Binarization. For each production \((X \rightarrow Y_1 \cdots Y_n) \in G\), replace the production with the productions

\[
X \rightarrow Y_1 X Y_2 \cdots Y_n \\
X Y_2 \cdots Y_n \rightarrow Y_2 X Y_3 \cdots Y_n \\
\vdots \\
X Y_{n-1} Y_n \rightarrow Y_{n-1} Y_n
\]

3 The CKY algorithm

Overview

- **bottom-up**: starts with the input words and tries to build progressively larger constituents
- **dynamic programming**: memorizes smaller constituents so as not to keep rebuilding them
- \(O(n^3|G|)\) time
- independently discovered by at least three people

Pseudocode

1. Input: \(w_1 \cdots w_n\)

2. Data structure: a chart with cells indexed \([i, j]\). If \(X \in chart[i, j]\), then that means we have discovered that \(X \Rightarrow^* w_{i+1} \cdots w_j\)

3. For each input word \(w_i\) and production \(X \rightarrow w_i\), enter \(X\) into chart\([i-1, i]\)

4. For \(\ell\) from 2 to \(n\)

   - For \(i\) from 0 to \(n - \ell\)
     - \(j = i + \ell\)
     - For \(k\) from \(i + 1\) to \(j - 1\)
* For all productions $X \rightarrow YZ$: If $Y \in chart[i,k]$ and $Z \in chart[k,j]$, enter $X$ into $chart[i,j]$

5. If $S \in chart[0,n]$, return yes, else return no

Exercise: implement this algorithm.

**Alternative version**

Axiom $[X,i, i+1] \quad (X \rightarrow w_{i+1}) \in G$

Inference rule $[Y,i,k] \quad [Z,k,j] \quad [X,i,j] \quad (X \rightarrow YZ) \in G$

Goal $[S,0,n]$

Search algorithm:

1. Initialize the chart to $\emptyset$ and the agenda to contain all the axioms.
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 an item in 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)

**Generalization**  What if we don’t want to convert our grammar to Chomsky normal form? In this notation, we are free to add inference rules to handle more types of rules:

$[X,i,i] \quad (X \rightarrow \epsilon) \in G$

$[Y,i,j] \quad [X,i,j] \quad (X \rightarrow Y) \in G$

$[Y_1,i,k_1] \quad [Y_2,k_1,k_2] \quad [Y_3,k_2,j] \quad [X,i,j] \quad (X \rightarrow Y_1Y_2Y_3 \in G)$

and so on. But this last inference rule has a problem: there are $O(n^4)$ ways of instantiating it, which means our parser will run in $O(n^4)$ time.

So let’s do a kind of binarization on the fly: instead of grabbing all the children of a constituent at once, create a new kind of item for partial constituents: $[X \rightarrow \alpha \cdot \beta, i,j]$, which means that the
part of $X$ to the left of the dot has been recognized, spanning from $i$ to $j$. Then we can rewrite
our inference rules like so:

\[
\begin{align*}
\text{Axiom} & \quad [X \rightarrow \bullet \beta, i, i] \\
& \quad (X \rightarrow \beta) \in G \\
\text{Inference rules} & \quad [X \rightarrow \alpha \bullet Y \beta, i, k] \\
& \quad [Y, k, j] \\
& \quad [X \rightarrow \alpha Y \bullet \beta, i, j] \\
& \quad [X \rightarrow \alpha \bullet, i, j] \\
& \quad [X, i, j]
\end{align*}
\]

Goal \quad [S, 0, n]

This parser will work for an arbitrary CFG. Call the items of the form \([X \rightarrow \bullet \beta, i, i]\) zero-width
items. The axiom generates an awful lot of them, many of which lead to nowhere. It would be
better to generate them only when we have reason to believe that we will want an item \([X, i, j]\) in
the future. This is true in two cases: first, we always want the goal item \([S, 0, n]\), and second, if
we’ve generated \([X \rightarrow \alpha Y \bullet \beta, i, j]\), then we’ll be wanting \([Y, j, k]\) in the future.

If we rewrite our parser to generate zero-width items only on demand in this way, we get Earley’s
algorithm:

\[
\begin{align*}
\text{Axiom} & \quad [S \rightarrow \bullet \beta, 0, 0] \\
& \quad (S \rightarrow \beta) \in G \\
\text{Inference rules} & \quad [X \rightarrow \alpha \bullet Y \beta, i, j] \\
& \quad [Y \rightarrow \bullet \gamma, j, j] \\
& \quad [X \rightarrow \alpha Y \bullet \beta, i, k] \\
& \quad [Y, k, j] \\
& \quad [X \rightarrow \alpha \bullet, i, j] \\
& \quad [X, i, j]
\end{align*}
\]

Goal \quad [S, 0, n]