Part III: Syntax

Notes on Natural Language Processing

Chia-Ping Chen

Department of Computer Science and Engineering
National Sun Yat-Sen University
Kaohsiung, Taiwan ROC

Introduction

- A *sentence* is a string of words.
- However, not every word string is a sentence.
- Certain rules apply.
- The structural rules for sentences constitute *syntax*.
- Changing *sentence* to *statement*, the same principles apply to programming languages.

Formal Definition

- A context-free grammar $G = (V, T, S, P)$ consists of
  - V: a set of **variables** (a.k.a. **non-terminals**)
  - T: a set of **terminals**
  - S: the **start symbol**
  - P: a set of **production rules**
- A production rule has the following form

$$A \rightarrow \alpha$$

  - $A \in V$
  - $\alpha \in (V \cup T)^*$

An Example

- We use `NP` as the start symbol of a CFG G. The production P consists of the following rules.
  - `NP` $\longrightarrow$ `Det Nominal`
  - `NP` $\longrightarrow$ `ProperNoun`
  - `Nominal` $\longrightarrow$ `Noun` $|$ `Noun Nominal`
  - `Det` $\longrightarrow$ `a` $|$ `the`
  - `Noun` $\longrightarrow$ `flight` $|$ `...`
- Many noun phrases can be generated by this grammar.

The Language of a CFG

- We say that $\beta A \gamma$ derives $\beta \alpha \gamma$ in one step, denoted by

$$\beta A \gamma \ \Rightarrow \ \beta \alpha \gamma,$$

  if there is a production rule of $A \rightarrow \alpha$.

- If there exists a sequence of $i = 1, 2, \ldots, m-1$ such that $\alpha_i \Rightarrow \alpha_{i+1}$, then we write

$$\alpha_1 \ \stackrel{*}{\Rightarrow} \ \alpha_m.$$

- The *language* of a CFG G is the set of strings

$$L(G) = \{w \mid S \stackrel{*}{\Rightarrow} w, \ w \in T^*\}.$$

Sentence Generation with CFG

- Start with the symbol S
- Until all symbols are terminals
  - pick a variable from the **sentential form**, i.e., any $\alpha \in (V \cup T)*$ such that

$$S \stackrel{*}{\Rightarrow} \alpha.$$

  - apply a rule with that variable as the left side
- Note such a generation
  - often have multiple choices
  - may not terminate

Derivation

- The **derivation** of a string w is the sequence of rule applications resulting in a terminal string w.
- Can you find the derivation for

$$\texttt{NP} \stackrel{*}{\Rightarrow} \textit{a flight}\,?$$

- A more complex example of grammar is given in Figure 12.2, and 12.3. Figure 12.4 is the **parse tree** for a sentence.

Parse Tree

- Given a derivation one can create a **parse tree**.
  - rooted by S
  - a node and its children is the left and right sides of a production rule
- A parse tree can also be represented in the **bracket form**. See (12.2).
- The **parse problem** of a context-free grammar is to find an algorithm to decide the parse tree for all sentences in the language of the grammar.

CFG and Constituency

- Suppose a sentence w is in $L(G)$.
- In the parse tree of w, any non-leaf node, say N, is the root of a subtree. The leaves under N is a **constituent**.
- The substring can be replaced by any string derivable from N. The meaning will be different, but the grammar is correct.

CFG for English

- English can be modeled by CFG quite well.
- T is the lexicon.
- Each element in V represents a POS, a phrase, a clause, or a sentence.
- A rule in P specifies how a POS, a phrase, or a clause can be formed.

Sentence-Level Rules

- **declarative**: `S` $\longrightarrow$ `NP VP`
- **imperative**: `S` $\longrightarrow$ `VP`
- **yes-no-question**: `S` $\longrightarrow$ `Aux NP VP`
- **wh-subject-question**: `S` $\longrightarrow$ `Wh-NP VP`
- **wh-non-subject-question**: `S` $\longrightarrow$ `Wh-NP Aux NP VP`

Agreement

- **number**

  `He does.   They do.`

- **case**

  `He hit me.   I hit him back.`

- **gender** (e.g. German a female noun goes with “eine”)

Gender of Computer

- la computadora
  - Even the smallest mistakes are stored in long-term memory for possible later retrieval.
  - As soon as you commit to one, you find yourself spending half your paycheck on accessories for it.
- el computador
  - They are supposed to help you solve problems, but half the time they *are* the problems.
  - As soon as you commit to one, you realize that if you had waited a little longer, you could have gotten a better model.

Verb Subcategorization

- A verb phrase consists of a verb and its **complement**, such as

  `VP` $\longrightarrow$ `Verb` | `Verb NP` | `Verb NP PP` | `Verb PP`

  `VP` $\longrightarrow$ `Verb S`

- The complement is called the **subcategorization frame.**
- Different verbs have different compliments.
  - a **transitive** verb cannot use `VP` $\longrightarrow$ `Verb PP`
  - an **intransitive** verb cannot use `VP` $\longrightarrow$ `Verb NP`.

Coordination

- A **coordinate phrase** consists of two phrases of the same kind, separated by a conjunction.
  - coordination of noun phrases

$$\texttt{NP} \longrightarrow \texttt{NP}\ \textit{and}\ \texttt{NP}$$

  - also applies to other phrases and clauses

$$\texttt{VP} \longrightarrow \texttt{VP}\ \textit{and}\ \texttt{VP}$$

$$\texttt{S} \longrightarrow \texttt{S}\ \textit{and}\ \texttt{S}$$

Treebanks

- If each sentence of a corpus is annotated with its **parse tree**, the corpus is called a **treebank**.
- often created by using an automatic parser to parse the sentences in a corpus, followed by hand correction of parse results.
- A well-known treebank is the **Penn Treebank**.
  - includes treebanks from Brown, Switchboard, as well as Chinese and Arabic sentences

Equivalence and Normal Form

- Two grammars are said to be **equivalent** if they have the same languages.
- For any CFG, there is an equivalent CFG in the **Chomsky normal form (CNF)**.
- A grammar in the Chomsky normal form has rules of only two forms

$$A \rightarrow B\ C$$
$$A \rightarrow a$$

Dependency Grammars

- In a **dependency grammar**, constituents and phrase structure rules do not play fundamental roles as they do in a CFG.
- A sentence is described in terms of words and the **binary relations** between words.
  - These relations are can be syntactic or semantic.
- Figure 12.14 gives a parse using dependency grammar.
  - There are no non-terminal nodes.
  - Each link is between two lexical nodes (words), defining a grammatical relation.

Head

- Syntactic constituents can be associated with a **lexical head**.
- The **head** is the word in the constituent that is the most important grammatically.
  - For example, V is the lexical head of VP, N is the lexical head of NP.
- If each rule is designated a head (a rhs symbol), then a parse tree will be annotated with heads (called a **lexicalized tree**).
- If not, a rule-based system can be used to decide head.

Disfluency

- Written and spoken languages differ.
- Specifically for spoken language, we have **disfluency**.
- Disfluency includes **fillers, word repetition, word fragments**, and **restarts**.
- An example of **restart** is given in Figure 12.19.
  - reparandum (to be fixed)
  - repair
  - interruption point
  - editing phase

Syntactic Parsing

- Given a grammar G and a string w, we want to decide if $w \in L(G)$.
- parsing algorithm for G

$$\text{For any input } w\text{, is } w \in L(G)?$$

- general parsing algorithm

$$\text{For any input } w, G\text{, is } w \in L(G)?$$

- If $w \in L(G)$, there exists a parse tree for w.

Parsing Algorithms

- Top-down parsing
- Bottom-up parsing
- CYK algorithm
- Earley algorithm
- Chart parsing
- Partial parsing

Parsing As Search

- A parsing algorithm for G searches through *the space of all possible parse trees* to find a parse tree for a given input.
- Specifically, given an input string w, it finds a tree T such that
  - T is rooted by S;
  - The leaves of T are labeled w (from left to right);

Top-Down Parsing

- Initialization: starts with a single S as root
- Iteration: until all leaves are terminals
  - expand variables in parallel
  - remove hypothesis without prefix match
- Let’s look at Figure 13.3 for an example.

Bottom-Up Parsing

- Initialization: starts with the input string w
- Iteration: until a tree rooted by S is found,
  - looks for places in the parse-in-progress where the right-side of some rule can fit
- Let's look at Figure 14.4 for an example.

Pros and Cons

- Top-down parsing
  - pros: all hypotheses are rooted in S
  - cons: waste efforts on trees not consistent with w (the first four trees in the third ply)
- Bottom-up parsing
  - pros: all hypotheses are consistent with w
  - cons: waste significant resources on hypotheses that cannot be grammatical (the left branch)

Non-Parallel Search

- In our discussion of top-down parsing, candidate trees are explored in parallel.
- Alternatively, the candidate trees can be explored in a *non-parallel* way, in which
  - if there is no more candidate to be extended, return **not found**;
  - a candidate is extended; (this is where depth-first and breadth-first search methods differ)
  - if a candidate of w is created, return **found**;

Left Corner Table

- We say B is a **left-corner** of A if there is a derivation

$$A \stackrel{*}{\Rightarrow} B\alpha.$$

- Imagine the subtree rooted by A as a triangle, B is clearly the left corner.
- For each variable in the grammar, we build a **left-corner table** which lists all possible POS tags as its left corner.

Bottom-Up Filtering

- In a successful parse, the next input word, say v, must be the first word of the subtree rooted by the left-most leaf variable, say N.
- Therefore, the POS of v must be in the left-corner table entry for N.
- For all rules with N as the left-side, we only need to consider those rules whose leftmost variable on the right-side has the POS of v as a left corner.
- This is called bottom-up filtering, as we use information of input words (at the bottom) to discard impossible hypotheses.

Ambiguity

- When there are more than one parse trees for an input string, we have **ambiguity in parsing**.
- Different parses actually connote different meanings.
- In English, there are several kinds of ambiguity
  - attachment ambiguity

    `I shot an elephant in my pajamas.`

  - coordination ambiguity

    `old men and women`

Repeated Parsing of Subtrees

- The basic top-down parser can be very *inefficient* for an input sentence with ambiguity.
- Figure 13.7 gives such an example.

```
A flight from Indianapolis to Houston on TWA.
```

  - The depth-first, left-to-right scheme is used.
  - One can see parse trees discarded and rebuilt.

Dynamic Programming Parsing

- Efficient parsing algorithm does exist!
- The basic idea is **dynamic programming.**
  - all subtrees are implicitly stored in a table;
- We introduce some well-known parsing methods
  - CYK parsing
  - Earley algorithm

CYK Algorithm and Data Structure

- First of all, the grammar has to be in the Chomsky normal form. Convert if not.
- For an input string of length n, we create a matrix of size $(n+1) \times (n+1)$.
- Cell $[i, j]$ contains a set of non-terminals Y_{ij}

$$Y_{ij} = \{V \mid V \overset{*}{\Rightarrow} w_{i+1} \ldots w_j\}$$

- The index corresponds to gap between input words, as in

$$0 \texttt{ Book } 1 \texttt{ that } 2 \texttt{ flight } 3$$

- Cell $[0, n]$ represents the entire input sentence.

CYK Recognition

- Since the grammar is in CNF, for a nonterminal $A \in Y_{ij}$, there must exist nonterminals B, C and k such that

$$A \;\rightarrow\; BC,\; B \in Y_{ik},\; C \in Y_{kj}.$$

- The upper-diagonal part of this matrix will be filled in a left-to-right, bottom-to-top fashion.
- This is shown schematically in Figure 13.9. The pseudo code is shown in Figure 13.10.
- This scheme guarantees that we have the information we need when filling an entry.

CYK Parsing

- If $S \in Y_{0n}$, the input sentence w is in $L(G)$.
- This is a recognition of grammatical sentence.
- The algorithm can be augmented such that a parse for w can be extracted when $S \in Y_{0n}$.
  - For example, use pointers to keep track of B, C, k for given $A \in Y_{ij}$.
- There may be more than one parses. In stochastic CFG, the optimal parse can be found efficiently.

Earley Algorithm

- The CYK algorithm is an efficient bottom-up parsing.
- The Earley algorithm is an efficient top-down parsing.
- Suppose the length of an input string w is n.
- We introduce an array, called **chart**, of $n + 1$ entries, along with a left-to-right process for **chart**.
- The **chart** encodes all parses for the input sentence w.

State

- An entry of the chart is a list of **states**.
- A **state** contains the following data.
  - a subtree corresponding to a grammatical rule
  - the position of the subtree with respect to the input
  - the progress made in completing the subtree
- A state can be represented by a **dotted rule**

$$A \rightarrow \alpha \bullet \beta, \ [a, b]$$

  - a indicates where constituent A begins
  - b indicates where the dot is

Successful Parse

- Figure 13.13 gives the pseudo code for Earley algorithm.
  - initialization
  - process chart entry from left to right
    - predictor
    - scanner
    - completer
- The presence of

$$S \rightarrow \alpha \bullet, \ [0, n]$$

in **chart**$[n]$ indicates $w \in L(G)$.

Predictor

- At position i, check if the symbol after the dot in the dotted rule is a part-of-speech.
- If not, apply the predictor.
- For example, suppose we have

$$A \to \alpha \bullet V \beta, \; [j, i]$$

where V is not a POS.

- The predictor creates a new state in **chart**$[i]$ for each rule for V

$$V \to \bullet \delta, \; [i, i].$$

Scanner

- If the symbol after the dot is a POS, then the scanner is applied.
- For example, suppose we have

$$A \to \alpha \bullet V \beta, \; [j, i]$$

  where V is a POS.
- The scanner creates a new state in **chart**$[i+1]$ if the next input word v can have a POS of V,

$$V \to v \bullet, \; [i, i+1].$$

Completer

- The completer is applied if the dot is at the right end of the rule

$$A \rightarrow \gamma \bullet, \;\; [j, i].$$

- It finds all states created previously looking to expand A at position j

$$B \rightarrow \alpha \;\bullet A\, \beta, \;\; [k, j],$$

and add a new state to **chart**$[i]$

$$B \rightarrow \alpha\, A \bullet \;\beta, \;\; [k, i].$$

An Example

- Figure 13.14.

Retrieving Parse Trees

- To retrieve a parse tree from **chart**, information needs to be added in the chart.
- When the completer creates a new state, there is a unique previous state.
- That information can be stored in order to retrieve the parse.
- Figure 13.15 for the same example with augmented information.

Partial Parse

- Many language-processing tasks need not complete parse.
  - information extraction
  - information retrieval
- **Shallow parse** or **partial parse** are alternative parsing methods to complete parsing.
  - some words in the input may be ignored
  - some constraint on the grammar may be imposed

Chunking

- A **chunking** task aims to recognize certain types of basic phrases in the input sentence.
- Such phrases must be *non-recursive*. They correspond to *non-overlapping segments*.
- Each segment is called a **chunk**.
  - Non-recursiveness makes the parse *shallow*.
  - Recognizing only certain patterns makes the parse *partial*.
- Examples are given in (13.8)-(13.10).

Evaluation

- A chunking system, called a **chunker**, is evaluated by comparing chunker output and reference.
- There are two fundamental measures
  - **precision**: the ratio of correct chunks in the output

$$P = \frac{\text{number of correct chunks in the output}}{\text{number of chunks in the output}}$$

  - **recall**: the ratio of correct chunks recognized

$$R = \frac{\text{number of correct chunks in the output}}{\text{number of chunks in the text}}$$

Note there is a trade-off between precision and recall.

F-Measure

- The precision and recall can be combined in the F**-measure**, defined by

$$F_\beta = \frac{(\beta^2 + 1)PR}{\beta^2 P + R},$$

where β is a parameter to tune the relative importance of precision and recall.

- Note

$$\frac{1}{F_\beta} = \frac{\beta^2 P + R}{(\beta^2 + 1)PR} = \frac{\beta^2 * \frac{1}{R} + 1 * \frac{1}{P}}{(\beta^2 + 1)}.$$

Statistical Parsing

- When a sentence has more than one valid parses, we say that the parse is **ambiguous**.
- Introducing **probabilities** into the grammatical rules renders each parse different in its possibility.
- Note that disambiguation through statistical parsing is strictly **syntactic** in nature. *No semantic knowledge is required*.
- Parse with a probabilistic grammar is called **statistical parsing**.

PCFG

- A **probabilitic context-free grammar (PCFG)** (Booth, 1969) is a 4-tuple

$$G = (V, \Sigma, S, P).$$

- A production rule $A \rightarrow \beta$ is assigned a number

$$A \rightarrow \beta \quad [q],$$

where

$$q = Pr(\beta | A)$$

is the probability of re-writing A by β.

Consistent PCFG

- An example of PCFG is given in Figure 14.1.
- A PCFG is said to be **consistent** if

$$\sum_{w \in L(G)} p(w) = 1.$$

  - If we consider all production rules for a variable, the sum of probabilities of these rules ought to be 1.
- The example is consistent.

Parse Probability

- Given a PCFG, say G, the probability of a parse tree T is the product of the probabilities of rules (nodes) in T

$$Pr(T) = \prod_{n \in T} r(n).q$$

- By definition, the **yield** of a completed parse tree is the terminal string of the tree leaves.
- Suppose w is the yield of a parse tree T. The joint probability the joint probability of w and T satisfies

$$Pr(w, T) = Pr(T)Pr(w|T) = Pr(T),$$

as T uniquely determines w.

Disambiguation

- Given a sentence w, we want to find the completed parse tree T with the maximum posterior probability

$$\begin{aligned} T(w) &= \arg\max_{T\ s.t.\ Y(T)=w} Pr(T|w) \\ &= \arg\max_{T\ s.t.\ Y(T)=w} Pr(T,w) \\ &= \arg\max_{T\ s.t.\ Y(T)=w} Pr(T) \end{aligned}$$

where $Y(T)$ is the yield of T.

- An example of disambiguation is given in Figure 14.2.

Sentence Probability

- The probability of a sentence w is

$$Pr(w) = \sum_{T\ s.t.\ Y(T)=w} Pr(T, w)$$
$$= \sum_{T\ s.t.\ Y(T)=w} Pr(T)$$

- PCFG is a **language model** since it can be used to compute sentence probability.
- (14.13) illustrates why as a language model PCFG might be better than n-gram.

Probabilistic CYK Parsing

- By definition, the **parsing problem of PCFG** is to decide the most-likely parse T^* for a given grammar-sentence pair G, w.
- Specifically,
  - input
    - PCFG in CNF $G = (V, \Sigma, S, P)$
    - a sentence $w = w_1 \dots w_N \in \Sigma^*$
  - output
    - T^*
    - $Pr(T^*)$
- The parsing problem can be solved by extending CYK algorithm with probability.

Data Structure

- Refer to the CYK algorithm introduced earlier.
- Matrix Y is augmented. A variable A in Y_{ij} is attached a probability, representing the maximum probability for

$$A \stackrel{*}{\Rightarrow} w_{i+1} \ldots w_j.$$

- In fact, this can be implemented by a 3-D array

$$\pi[i, j, A] = Y_{ij}(A), \quad i, j = 0, \ldots, n, \quad A \in V.$$

- By definition, $\pi[0, n, S]$ is the maximum parse-tree probability for w.

Algorithm

- left-to-right, bottom-up
  - for $j = 1$ to n
    - $\pi[j-1, j, A] = p$ if

$$A \longrightarrow w_j, \quad [p].$$

    - for $i = j-2$ to 0,

$$\pi[i,j,A] = \max_{i+1 \leq k \leq j-1} p(A \longrightarrow BC) \times \pi[i,k,B] \times \pi[k,j,C]$$

Example

- Let's work out the example given in Figure 14.4.

Learning PCFG

- Using a **treebank**. With annotated parse trees

$$p(\alpha \rightarrow \beta | \alpha) = \frac{\text{Count}(\alpha \rightarrow \beta)}{\text{Count}(\alpha)}$$

- Without a treebank, we can use an automatic parser to generate parse trees.
- Gather counts on the parse trees.

Inside-Outside Algorithm

- Suppose there are two parses for a given sentence, how do we gather counts? Half-and-half?
- One uses the **probability count**. That is, assign as the count of a parse its probability (fractional).
- We have a **chicken-and-egg** situation
  - need probabilities to assign counts
  - need counts to decide probabilities
- Such a dilemma can often be resolved by an **iterative algorithm**: in this case, the **inside-outside algorithm**.

Inside/Outside Probabilities

- given a sentence $w = w_1 \dots w_n$
- **inside probability**: the sum of probabilities of all parse trees rooted by A with $w_j \dots w_k$ as leaves

$$I(j, A, k) = p(A \stackrel{*}{\Rightarrow} w_j \dots w_k)$$

- **outside probability**: the sum of probabilities of all parse trees rooted by S with $w_1 \dots w_{s-1}\ A\ w_{t+1} \dots w_n$ as leaves

$$O(s, A, t) = p(S \stackrel{*}{\Rightarrow} w_1 \dots w_{s-1}\ A\ w_{t+1} \dots w_n)$$

Recursion

- for the inside probability

$$I(j, A, k) = \sum_{B,C\in\Sigma} \sum_{l=j}^{k-1} p(A \rightarrow BC) I(j, B, l) I(l+1, C, k)$$

- for the outside probability

$$O(s, A, t) = \sum_{Z,Y\in\Sigma} \sum_{k=1}^{s-1} p(Y \rightarrow ZA) I(k, Z, s-1) O(k, Y, t)$$

$$+ \sum_{Z,Y\in\Sigma} \sum_{l=t+1}^{T} p(Y \rightarrow AZ) I(t+1, Z, l) O(s, Y, l)$$

$(k < s, l > t)$

Learning PCFG

- Define $\xi(A, B, C, s, t)$ as the posterior probability

$$\xi(A, B, C, s, t) = p(A \stackrel{*}{\Rightarrow} w_s \dots w_t, A \rightarrow BC \mid S \stackrel{*}{\Rightarrow} w)$$

- Using the inside/outside probabilities

$$\xi(A, B, C, s, t) = \frac{1}{p(S \stackrel{*}{\Rightarrow} w)} \sum_{k=s}^{t-1} p(A \rightarrow BC) I(s, B, k) I(k+1, C, t) O(s, A, t)$$

- The probability $p(A \rightarrow BC)$ can be updated by

$$p(A \rightarrow BC) = \frac{\sum_{s=1}^{n-1} \sum_{t=s+1}^{n} \xi(A, B, C, s, t)}{\sum_{B', C'} \sum_{s=1}^{n-1} \sum_{t=s+1}^{n} \xi(A, B', C', s, t)}$$

Computing Sentence Probability

- Note

$$I(s,A,t)O(s,A,t) = Pr(S \stackrel{*}{\Rightarrow} w_1 \ldots w_n, A \stackrel{*}{\Rightarrow} w_s \ldots w_t)$$

so

$$p(S \stackrel{*}{\Rightarrow} w_1 \ldots w_n) = \sum_A I(s,A,t)O(s,A,t), \text{ for any } s \leq t.$$

- Or, by definition,

$$p(S \stackrel{*}{\Rightarrow} w_1 \ldots w_n) = I(1,S,n)$$

Problems with PCFG

- **poor independence assumption**
  - Rule probabilities are assumed to be independent of the context.
- **lack of lexical conditioning**
  - Rule probabilities are assumed to be independent of words in the sentence.

Context-Dependent Example

- Consider the two rules

$$\begin{cases} \text{NP} \rightarrow \text{DT} \quad \text{NN} \\ \text{NP} \rightarrow \text{Pronoun} \end{cases}$$

- It turns out in English
  - when NP is the syntactic **subject**, the first rule has a high probability;
  - when NP is the syntactic **object**, the second rule has a high probability;
- With simple PCFG, we cannot distiniguish subject and object contexts for NP.

Lexical Conditioning Example

- Consider the two sentences

$$\begin{cases} \textit{workers dumped sacks into a bin} \\ \textit{fishermen caught tons of herring} \end{cases}$$

- Both sentences have a PP (*into a bin, of herring*), which is to be attached to VP (first sentence) or to NP (second).
- Using simple PCFG, **both** must be attached to VP or NP.
- Here the dilemma can be resolved if we recognize that *dump* and *into* go together, as well as *tons* and *of*.

Parser Evaluation

- **PARSEVAL**
- How many constituents in the hypothesis parse C_h agree with the reference C_r?
- A given contituent in C_h is labelled correct if there is a constituent in C_r with the same start point, end point, and non-terminal symbol.

Precision, Recall, F-Measure

- **precision**

$$P = \frac{\text{number of correct contituents in } C_h}{\text{number of contituents in } C_h}$$

- **recall**

$$R = \frac{\text{number of correct constituents in } C_h}{\text{number of contituents in } C_r}$$

- **F-measure**

$$F_\beta = \frac{(\beta^2 + 1)PR}{\beta^2 P + R},$$