Parsing
Introduction

Laura Kallmeyer

Heinrich-Heine-Universität Düsseldorf

Winter 2017/18
Table of contents

1 Introduction

2 Languages

3 Grammars

4 Grammar Formalisms

5 Parsing and Automata

6 The Chomsky Hierarchy
Parsing means performing an automatic syntactic analysis.

Two types of syntactic structures are used for natural languages:

1. Constituent structure
2. Dependency structure

See for instance the Stanford Parser, that gives both types of structures:
http://nlp.stanford.edu:8080/parser/index.jsp
Introduction

Constituent structure

- every word is a constituent
- several constituents can form a new constituent
- each constituent has a syntactic category
- the structure is usually tree-shaped
- oftentimes only continuous constituents

```
S
  NP
    DT NN VBD NP
      the man saw DT NN
          the girl
  VP
```
Introduction

Dependency structure

- every word is a node in the structure
- there is one additional node, *root*
- words are linked via directed labeled edges (dependencies)
- the structure is usually tree-shaped
- oftentimes only projective dependencies

Constituency parsing is mostly grammar-based while dependency parsing is mostly grammar-less.

This course is concerned with constituency parsing.
Examples of languages one might want to parse:

<table>
<thead>
<tr>
<th>Languages</th>
</tr>
</thead>
<tbody>
<tr>
<td>natural languages</td>
</tr>
<tr>
<td>programming languages</td>
</tr>
<tr>
<td>“biological” languages</td>
</tr>
<tr>
<td>formal languages</td>
</tr>
</tbody>
</table>

- **natural languages** such as, e.g., German, English, French, …
- **programming languages** such as, e.g., the set of all correct Java programs, …
- **“biological” languages** such as, e.g., the set of possible DNA sequences in a certain environment, …
- **formal languages** such as, e.g., the language containing all sequences \(ab, aabb, aaabbb, aaaaabbbb, \ldots\).
Languages (2)

Alphabet, languages

- An **alphabet** is a nonempty finite set $X$.

- A string $x_1 \ldots x_n$ with $n \geq 1$ and $x_i \in X$ for $1 \leq i \leq n$ is called a **nonempty word** on the alphabet $X$. $X^+$ is defined as the set of all nonempty words on $X$.

- A new element $\varepsilon \notin X^+$ is added: $X^* := X^+ \cup \{\varepsilon\}$. For each $w \in X^*$ concatenation of $w$ and $\varepsilon$ is defined as follows: $w\varepsilon := \varepsilon w := w$. $\varepsilon$ is called the **empty word**, and each $w \in X^*$ is called a word on $X$.

- A set $L$ is called a **language** iff there is an alphabet $X$ such that $L \subseteq X^*$.
Languages are described by grammars. We will concentrate on **generative grammars** (sometimes also called **rewriting grammars**).

Idea: you have

- a start symbol (often S)
- and productions (rewriting rules) that tell you how to replace symbols with other symbols. (e.g., $S \rightarrow NP \ VP$)
Grammars (2)

**Grammar** $G_{\text{telescope}}$

**Productions:**

\[
\begin{align*}
S & \rightarrow \text{NP} \ \text{VP} \\
\text{NP} & \rightarrow \text{D} \ \text{N} \\
\text{VP} & \rightarrow \text{VP} \ \text{PP} \\
\text{N} & \rightarrow \text{man} \\
\text{N} & \rightarrow \text{girl} \\
\text{N} & \rightarrow \text{telescope} \\
\text{D} & \rightarrow \text{the} \\
\text{NP} & \rightarrow \text{John} \\
\text{NP} & \rightarrow \text{Mary} \\
\text{P} & \rightarrow \text{with} \\
\text{V} & \rightarrow \text{saw}
\end{align*}
\]

In each derivation step $\alpha \Rightarrow \gamma$, the lefthand side symbol of a production is replaced with the righthand side.

**Derivation in** $G_{\text{telescope}}$

\[
\begin{align*}
S & \Rightarrow \text{NP} \ \text{VP} \Rightarrow \text{D} \ \text{N} \ \text{VP} \Rightarrow \text{the} \ \text{N} \ \text{VP} \Rightarrow \text{the girl} \ \text{VP} \Rightarrow \text{the girl} \ \text{V} \ \text{NP} \\
& \Rightarrow \text{the girl saw} \ \text{NP} \Rightarrow \text{the girl saw} \ \text{John}
\end{align*}
\]
Grammars (3)

The language generated by a grammar is the set of terminal strings one can derive from the start symbol.

**Language of** $G_{telescope}$

Sentences one can generate with $G_{telescope}$:

(1) John saw Mary
(2) John saw the girl
(3) the man with the telescope saw John
(4) John saw the girl with the telescope
...

A grammar formalism defines the form of rules and combination operations allowed in a grammar.

**Type 0 grammar**

A type 0 grammar (or unrestricted grammar) $G$ is a tuple $\langle N, T, P, S \rangle$ with

- $N$ and $T$ disjoint alphabets, the nonterminals and terminals,
- $S \in N$ the start symbol, and
- $P$ a set of productions of the form $\alpha \rightarrow \beta$ with $\alpha \in (N \cup T)^+, \beta \in (N \cup T)^*$. 
Grammar Formalisms (2)

Derivation

Let $G = \langle N, T, P, S \rangle$ be a type 0 grammar. The (string) language $L(G)$ of $G$ is the set $\{ w \in T^* \mid S \xRightarrow{*} w \}$ where

- for $w, w' \in (N \cup T)^*$: $w \Rightarrow w'$ iff there is a $\alpha \rightarrow \beta \in P$ and there are $v, u \in (N \cup T)^*$ such that $w = v\alpha u$ and $w' = v\beta u$.

- $\xRightarrow{*}$ is the reflexive transitive closure of $\Rightarrow$:
  - $w \xRightarrow{0} w$ for all $w \in (N \cup T)^*$, and
  - for all $w, w' \in (N \cup T)^*$: $w \xRightarrow{n} w'$ iff there is a $v$ such that $w \Rightarrow v$ and $v \xRightarrow{n-1} w'$.
  - for all $w, w' \in (N \cup T)^*$: $w \xRightarrow{*} w'$ iff there is a $i \in \mathbb{N}$ such that $w \xRightarrow{i} w'$.

A language is called a type 0 language iff it is generated by a type 0 grammar.
Grammar Formalisms (3)

**Type 1 grammar**

A type 0 grammar is called **context-sensitive** (or of **type 1**) if for all productions $\alpha \rightarrow \beta$, $|\alpha| \leq |\beta|$ holds. The only exception is $S \rightarrow \varepsilon$ which is allowed if $S$ does not appear in any righthand side.

**Example of a type 1 grammar**

$N = \{S, C\}$, $T = \{a, b, c\}$

Productions:
- $S \rightarrow abc$
- $S \rightarrow aabCbc$
- $abC \rightarrow aabCbC$
- $Cb \rightarrow bC$
- $Cc \rightarrow cc$

This grammar generates $\{a^n b^n c^n \mid n \geq 1\}$. 
A type 0 grammar is called **context-free** (or of **type 2**) if for all productions $\alpha \rightarrow \beta$, $\alpha \in N$.

Example of a type 2 grammar

$N = \{S, T\}$, $T = \{a, b, c, d\}$

Productions:

- $S \rightarrow aSb$
- $S \rightarrow aTb$
- $T \rightarrow ccTdd$
- $T \rightarrow \varepsilon$

This grammar generates the language $\{a^n c^{2m} d^{2m} b^n | n \geq 1, m \geq 0\}$. 
A type 0 grammar is called **regular** (or of **type 3**) if for all productions $\alpha \rightarrow \beta$, $\alpha \in N$ and $\beta \in T^*$ or $\beta = \beta'X$ with $\beta' \in T^*$, $X \in N$.

**Example of a type 3 grammar**

$N = \{S, A, B, C\}, \ T = \{a, b, c\}$

Productions:

$S \rightarrow aaS \quad S \rightarrow B \quad S \rightarrow C \quad B \rightarrow bB \quad B \rightarrow b \quad C \rightarrow cc$

This grammar generates the language denoted by $(aa)^*(b^+|cc)$.

The type 1/2/3 languages are the languages generated by the corresponding grammars.
A **parser** is a device that accepts a word $w$ and a grammar $G$ as input and that

1. decides whether $w$ is in the language generated by the grammar and
2. if so, it provides a syntactic analysis for $w$ or, if $w$ is ambiguous, a set of analyses, oftentimes represented in a compact way as a **derivation forest**.

A device that does only the first part of the task is called a **recognizer**.
Example for parsing:

Input: “the man saw the girl”.
Output:

```
S
   / \ 
  NP  VP
     /   /
    D    V
   /     /
  the  man saw
```

Input: “the man saw saw the girl”. Output: no.
A parser for grammars such as $G_{telescope}$ could for example work as follows:

1. Start from the terminal symbols.
2. Apply productions in reverse order thereby combining already recognized parts into new parts.
3. Success if an S can be found that spans the whole $w$. 
Automata are devices that accept a language. They are recognizers. An automaton has

- a set of states, containing an initial state and final states,
- a tape with the input string, and
- a finite control.

The automaton starts in the initial state. It reads the input string on the tape while changing states. If it ends up in a final state after having consumed the whole input, the word is accepted.

Oftentimes for a given grammar, an automaton can be constructed that accepts the string language of the grammar.
The hierarchy of the type 0, 1, 2 and 3 languages is called the **Chomsky Hierarchy**.

<table>
<thead>
<tr>
<th>Chomsky Hierarchy</th>
</tr>
</thead>
<tbody>
<tr>
<td>class</td>
</tr>
<tr>
<td>type 3</td>
</tr>
<tr>
<td>type 2</td>
</tr>
<tr>
<td>type 1</td>
</tr>
<tr>
<td>type 0</td>
</tr>
</tbody>
</table>

In this course, we are concerned with CFGs.
A textbook covering almost all the algorithms treated in this course.

Original edition of one of the best textbooks on formal language and automata theory.

Its German translation.

Chapter 3 introduces to parsing as deduction and discusses some properties of parsing algorithms.