Parsing

Introduction

Laura Kallmeyer

Heinrich-Heine-Universität Düsseldorf

Winter 2017/18

Table of contents

Introduction

Parsing means performing an **automatic syntactic analysis**.

Two types of syntactic structures are used for natural languages:

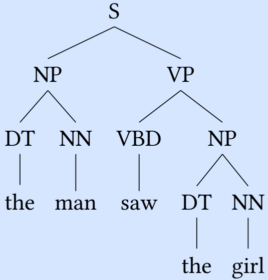
1. Constituent structure
2. Dependency structure

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

Introduction

Constituent structure

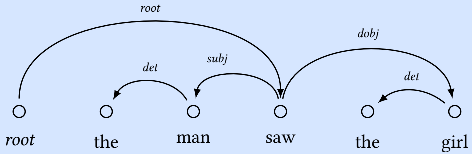
- every word is a constituent
- several constituents can form a new constituent
- each constituent has a syntactic category
- the structure is usually tree-shaped
- oftentimes only continuous constituents



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.

Languages (1)

Examples of languages one might want to parse:

Languages

- **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, $aabb$, $aaabbb$, $aaaabbbb$, $\ldots$.

Languages (2)

Alphabet, languages

- An **alphabet** is a nonempty finite set X.
- A string $x_1 \dots x_n$ 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 ε is defined as follows: $w\varepsilon := \varepsilon w := w$. ε 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 $L \subseteq X^*$.

Grammars (1)

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 $G_{telescope}$

Productions:

$S \rightarrow NP\ VP$	$NP \rightarrow D\ N$	$N \rightarrow N\ PP$	
$VP \rightarrow VP\ PP$	$VP \rightarrow V\ NP$	$PP \rightarrow P\ NP$	
$N \rightarrow$ man	$N \rightarrow$ girl	$N \rightarrow$ telescope	$P \rightarrow$ with
$D \rightarrow$ the	$NP \rightarrow$ John	$NP \rightarrow$ Mary	$V \rightarrow$ saw

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

Derivation in $G_{telescope}$

S $\Rightarrow$ NP VP $\Rightarrow$ D N VP $\Rightarrow$ the N VP $\Rightarrow$ the girl VP $\Rightarrow$ the girl V NP $\Rightarrow$ the girl saw NP $\Rightarrow$ the girl saw John

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

…

Grammar Formalisms (1)

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 $\{w \in T^* \mid S \overset{*}{\Rightarrow} w\}$ where

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

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 $\alpha \to \beta$, $|\alpha| \leq |\beta|$ holds. The only exception is $S \to \varepsilon$ which is allowed if S does not appear in any righthand side.

Example of a type 1 grammar

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

Productions:

$\text{S} \to \text{abc}$ $\quad \text{S} \to \text{aabCbc}$ $\quad \text{abC} \to \text{aabCbC}$

$\text{Cb} \to \text{bC}$ $\quad \text{Cc} \to \text{cc}$

This grammar generates $\{a^n b^n c^n \mid n \geq 1\}$.

Grammar Formalisms (4)

Type 2 grammar

A type 0 grammar is called **context-free** (or of **type 2**) if for all productions $\alpha \to \beta$, $\alpha \in N$.

Example of a type 2 grammar

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

Productions:

$\mathrm{S} \to \mathrm{aSb} \quad \mathrm{S} \to \mathrm{aTb} \quad \mathrm{T} \to \mathrm{ccTdd} \quad \mathrm{T} \to \varepsilon$

This grammar generates the language $\{a^n c^{2m} d^{2m} b^n \mid n \geq 1, m \geq 0\}$.

Grammar Formalisms (5)

Type 3 grammar

A type 0 grammar is called **regular** (or of **type 3**) if for all productions $\alpha \to \beta$, $\alpha \in N$ and $\beta \in T^*$ or $\beta = \beta' X$ with $\beta' \in T^*$, $X \in N$.

Example of a type 3 grammar

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

Productions:

S → aaS S → B S → C B → bB B → b C → cc

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

The type 1/2/3 languages are the languages generated by the corresponding grammars.

Parsing and Automata (1)

A **parser** is a device that accepts a word w and a grammar G as input and that

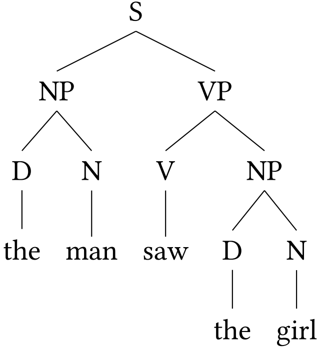
1. decides whether w is in the language generated by the grammar and
2. if so, it provides a syntactic analysis for w or, if w is ambiguous, a set of analyses, oftentimes represented in a compact way as a **derivation forest**.

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

Parsing and Automata (2)

Example for parsing:

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



Input: “the man saw saw the girl”. Output: no.

Parsing and Automata (3)

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.

Parsing and Automata (4)

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 Chomsky Hierarchy

The hierarchy of the type 0, 1, 2 and 3 languages is called the **Chomsky Hierarchy**.

Chomsky Hierarchy

class	grammar	automaton	others
type 3	regular grammar	FSA	reg. expr.
type 2	CFG	PDA	
type 1	CSG	LBA	
type 0	unrestricted grammars	Turing machine	

In this course, we are concerned with CFGs.

Grune, D. and Jacobs, C. (2008). *Parsing Techniques. A Practical Guide.* Monographs in Computer Science. Springer. Second Edition.
A textbook covering almost all the algorithms treated in this course.

Hopcroft, J. E. and Ullman, J. D. (1979). *Introduction to Automata Theory, Languages and Computation.* Addison Wesley.
Original edition of one of the best textbooks on formal language and automata theory.

Hopcroft, J. E. and Ullman, J. D. (1994). *Einführung in die Automatentheorie, Formale Sprachen und Komplexitätstheorie.* Addison Wesley, 3. edition.
Its German translation.

Kallmeyer, L. (2010). *Parsing Beyond Context-Free Grammars.* Cognitive Technologies. Springer, Heidelberg.
Chapter 3 introduces to parsing as deduction and discusses some properties of parsing algorithms.