

Formal Languages

Parsing
ISCL-BA-06

Çağrı Çöltekin
ccolt@infsys.uni-tuebingen.de

University of Tübingen
Seminar für Sprachwissenschaft

Winter Semester 2020/21

version: 2020-04-02 02:22:11.30

Introduction

Natural, artificial, formal languages

- Some languages in our list are natural languages
- In contrast some are designed, they are artificial
- Formal languages are those that we can study formally
 - we can *analyze* them in principled ways
 - provably answers the questions about them
- All languages in our list can be studied as formal languages (to some extent)

Ç. Çöltekin, INF | University of Tübingen

Winter Semester 2020/21 2 / 49

Introduction

Formal grammar

A formal *grammar* is a finite specification of a (formal) language.

- Since we consider languages as sets of strings, for a finite language, we can (conceivably) list all strings
- How to define an infinite language?
- Is the definition $\{ba, baa, baaa, baaaa, \dots\}$ ‘formal enough’?
- Using regular expressions, we can define it as *baa**
- But we will introduce a more general method for defining languages soon
- Are natural languages infinite?

Ç. Çöltekin, INF | University of Tübingen

Winter Semester 2020/21 3 / 49

Introduction

Operations on languages

Since we define languages as sets, all set operations are applicable to languages. If L_1 and L_2 are languages,

- Intersection: $L_1 \cap L_2$
- Union: $L_1 \cup L_2$
- Difference: $L_1 - L_2$
- Complement: $\Sigma^* - L_1$
- Concatenation: $L_1 L_2 = \{xy | x \in L_1 \text{ and } y \in L_2\}$

Ç. Çöltekin, INF | University of Tübingen

Winter Semester 2020/21 4 / 49

Introduction

Grammars: how to describe a language?

- In daily use, a ‘grammar’ is a book, it defines a language in detail
- But we are interested in more *formal* grammars
- The challenge is describing a possibly infinite set with a finite specification
- We already see that it was possible (e.g., regular expressions)
- Another possible way would be writing a computer program that determines if the given string is in the language
- However, we want more general descriptions: grammars that can describe any ‘describable’ language in a concise and easy to study formalism

Aside: can any language be described by a finite description?

Ç. Çöltekin, INF | University of Tübingen

Winter Semester 2020/21 5 / 49

Introduction

Phrase structure grammars

Some conventions

- We use uppercase letters (sometimes capitalized words) for non-terminal symbols: A, B, C, NP ; End
- We use lowercase letters (sometimes lowercase words) for terminals: a, b, c , cat, dog
- We use Greek letters letters for *sentential forms*, (sequences of terminal and non-terminal symbols): α, β, γ
- For sequences of terminal symbols (strings) we use lowercase letters from the end of the alphabet: u, v, w, x, y, z

Ç. Çöltekin, INF | University of Tübingen

Winter Semester 2020/21 12 / 49

Introduction

What is a language?

- English, German, Chinese
- Latin, Coptic, Sanskrit, Sumerian
- Proto-Germanic, Proto-Uralic, Proto-Dravidian
- Sign languages
- Esperanto
- Traffic signs, computer icons, emoticons
- Chemical formulas
- Arithmetic expressions
- Python, Java, C++
- XML, JSON, HTML, YAML
- HTTP, TCP, UDP,
- The set of strings $\{ba, baa, baaa, baaaa, \dots\}$

According to [Joshi and Martin (2009)], the best set of strings from the ‘library language’.

Ç. Çöltekin, INF | University of Tübingen

Winter Semester 2020/21 6 / 49

Introduction

Languages as sets of strings

We define a *formal language* as a set of finite-length string over an *alphabet*.

- The sheep language from the first slide was represented as a set: $\{ba, baa, baaa, baaaa, \dots\}$

The alphabet of a language is the set of “symbols” in the language, conventionally denoted as Σ .
- For the sheep language, $\Sigma = \{a, b\}$
- What is the alphabet for English syntax?

Ç. Çöltekin, INF | University of Tübingen

Winter Semester 2020/21 7 / 49

Introduction

Formal languages

Some definitions

Alphabet is the set of ‘atomic’ symbols in the language

String is a sequence of symbols from the alphabet. For example, 101100 is a string over alphabet $\Sigma = \{0, 1\}$

- Concatenation: if $x = 10$ and $y = 11000101$, their concatenation $xy = 1011000101$
- We represent the empty string with ϵ (some books use λ)
- The notation x^* indicates zero or more concatenation of string x with itself, e.g., $\epsilon, 01, 010101$ (the operation is called Kleene star)
- The notation x^+ is a shorthand for xx^*
- x^n means exactly n repetition of string x

Σ^* is all possible strings that can be defined over alphabet Σ

Sentence of a language is a string that is in the language (confusingly the term word is also common)

Ç. Çöltekin, INF | University of Tübingen

Winter Semester 2020/21 8 / 49

Introduction

Three different view on formal languages

- In formal language theory, a language is studied for itself. Languages are simply set of strings, we do not attach ‘meaning’ to them. The questions of interests are abstract. For example, ‘how to find the intersection of two languages for which we have grammars?’
- In computer science, we want to analyze the structure (of, e.g., a computer program) to get some information, or ‘meaning’. The most common area is compiler construction, but almost any syntactic analysis task is supported by formal definitions of the respective languages.
- In (computational) linguistics, the aim is to analyze sentences (syntax), and associate them with their meanings (semantics). Formal languages provide a way to study a seemingly chaotic object, natural language, in a principled way.

Ç. Çöltekin, INF | University of Tübingen

Winter Semester 2020/21 9 / 49

Introduction

Phrase structure grammars

- A phrase structure grammar is a generative device
- If a given string can be generated by the grammar, the string is in the language
- The grammar generates *all* and the *only* strings that are valid in the language
- A phrase structure grammar has the following components
 - Σ A set of terminal symbols
 - N A set of non-terminal symbols
 - $S \in N$ A special non-terminal, called the start symbol
 - R A set of *rewrite rules* or *production rules* of the form:

$$\alpha \rightarrow \beta$$

which means that the sequence α can be rewritten as β (both α and β are sequences of terminal and non-terminal symbols)

Ç. Çöltekin, INF | University of Tübingen

Winter Semester 2020/21 10 / 49

Introduction

Generating sentences from a PSG

1. Start with the symbol S as the first sentential form
 2. Pick a rule with matching the part of the current sentential form
 3. Apply the rewrite (production) rule
 4. Repeat 2 and 3, until there are no non-terminals left
- Exhaustively exploring all possible productions ‘enumerates’ all sentences of the language described by the grammar

Ç. Çöltekin, INF | University of Tübingen

Winter Semester 2020/21 11 / 49

Phrase structure grammars

A very simple example – the sheep language

Introduction

A grammar

1. $S \rightarrow BA$
2. $B \rightarrow b$
3. $A \rightarrow aA$
4. $A \rightarrow a$

Quick exercise: try to define a different grammar for the same language.

An example derivation

Sentential form	rule	notes
S		start symbol
BA	$S \rightarrow BA$	rule 1
bA	$B \rightarrow b$	rule 2
baA	$A \rightarrow aA$	rule 3
baaA	$A \rightarrow aA$	rule 3
baaa	$A \rightarrow a$	rule 4

Generation to parsing

Introduction

- The above procedure (generating all sentences from a generative grammar) gives us a possible way to do parsing:
 - Enumerate all sentences from the grammar
 - If the string we are interested comes out, it is in the language: parsing is successful
 - If it does not come out, it is not in the language: parsing failed (we'll get back to this point soon)
- We will also see later that this is in fact the idea behind top-down parsers

Phrase structure grammars

Another example: the goat language (a dialect of sheep language)¹

Introduction

The grammar

1. $S \rightarrow \text{Begin } B \text{ A End}$
2. $B \rightarrow b$
3. $A \rightarrow a$
4. $A \rightarrow aA$
5. $aA \text{ End} \rightarrow a'a$
6. $\text{Begin } b \text{ a} \rightarrow \text{Begin } b \text{ b a}$
7. $\text{Begin } b \text{ b} \rightarrow b \text{ b}$

A few exercises:

- Describe the language
- Derive the string $baaaa'a$
- Is the string $baa'a$ in the language?
- Can you write a simpler grammar for this language?

¹ Some claim that the grammar is just the same, but goats use the word *a* instead of the word *b*.

Phrase structure grammars

A few notes

Introduction

- The phrase structure grammars are not the only method for defining languages (sets)
- However, all known methods are either equivalent to, or less powerful than phrase structure grammars
- The formalism we sketched is general: any set (language) that can be generated by a computer program can be defined by a phrase structure grammar

Languages and Grammars

more definitions

Introduction

- The language that can be derived from a grammar G , is denoted by $L(G)$
- The notation $u \Rightarrow v$ is used to denote 'immediate derivation', e.g., $A \Rightarrow aA$
- If a sentential form β can be derived from another sentential form α with zero or more immediate derivations, we write $\alpha \Rightarrow^* \beta$
- If β can be derived from α with exactly n immediate derivations, we write $\alpha \Rightarrow^n \beta$
- Formally, $L(G) = \{w \in \Sigma^* \mid S \Rightarrow^* w\}$
- Two grammars G and G' are weakly equivalent if $L(G) = L(G')$

The Chomsky hierarchy of grammars

Introduction

- Type 0 Unrestricted phrase structure grammars
- Type 1 Context-sensitive or monotonic grammars
- Type 1.9 Mildly-context sensitive grammars
- Type 2 Context-free grammars
- Type 2.5 Linear grammars
- Type 3 Regular grammars
- Type 4 Finite (choice) grammars

Type 0: unrestricted PSG

Introduction

- As the names says - unrestricted, any form of the rewrite rules are allowed
- If a language can be generated at all, it can be defined/generated by a unrestricted PSG
- No general parsing algorithm exists, and in fact cannot exist
- In general, type 0 grammars are not interesting for practical applications
- The class of languages described by type 0 grammars is called *recursively enumerable* languages

Type 1: monotonic

Introduction

- We introduce one restriction to PSG: the right hand side (RHS) of a rule cannot be shorter than the left hand side (LHS)
- The rule applications cannot 'shrink' the sentential forms
- For example, our 'goat language grammar' is not monotonic, because of the rule $\text{Begin } b \text{ b} \rightarrow b \text{ b}$
- This also means no ϵ -rules
- Sometimes the language with only the empty string is allowed as an exception

Type 1: context sensitive

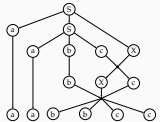
Introduction

- A context-sensitive grammar rewrites only one of its non-terminal on the LHS.
- Our 'goat language grammar' is not context-sensitive, because of the rule $aA \text{ End} \rightarrow a'a$
- Context-sensitive and monotonic grammars are equivalent
- Parsing is possible with Type 1 grammars, but inefficient
- In general, not much practical use

An example type 1 grammar: $a^nb^nc^n$

monotonic version

- $$\begin{aligned} S &\rightarrow abc \\ S &\rightarrow aSX \\ bXc &\rightarrow bXcc \\ cX &\rightarrow Xc \end{aligned}$$

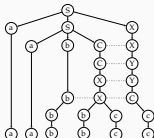


An example type 1 grammar: $a^nb^nc^n$

context-sensitive version

- $$\begin{aligned} S &\rightarrow abC \\ S &\rightarrow aSX \\ bXc &\rightarrow bXCC \\ CX &\rightarrow CY \\ CY &\rightarrow XY \\ XY &\rightarrow XC \\ C &\rightarrow c \end{aligned}$$

Exercise: try to write a (type 1) grammar for $a^nb^mcd^m$.



Type 2: context free

Introduction

- A context free language requires its LHS to have only a single non-terminal symbol. Rules are in the form

$$A \rightarrow \alpha$$

- This means the rewrite rules cannot be conditioned on context, they are independent of their environment
- It also means, each non-terminal defines its own language
- Context-free languages have efficient parsers, and used in practical applications
- All programming languages are (subclasses) of context free languages
- Most of natural language parsing is based on context-free parsing (more on this soon)

How do we know a language is regular?

regex examples

$bb(a|a^*a)$
 $bb(a^*a^*)?$

regular grammar

$S \rightarrow bB \quad B \rightarrow a \quad A \rightarrow aA \quad C \rightarrow^*E$
 $B \rightarrow bB \quad B \rightarrow aA \quad A \rightarrow aC \quad E \rightarrow a$

finite-state automaton

How do we know a language is not regular?

pumping lemma for regular languages

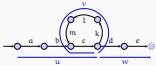
- What is the length of longest string generated by this FSA?
- Any FSA generating an infinite language has to have a loop (application of recursive rule(s) in the grammar)
- Part of every string longer than some number will include repetition of the same substring ('cklm' above)

Pumping lemma

definition

For every regular language L , there exist an integer p such that a string $x \in L$ can be factored as $x = uvw$,

- $uv^pw \in L, \forall i \geq 0$
- $v \neq \epsilon$
- $|uv| \leq p$



How to use pumping lemma

- We use pumping lemma to prove that a language is not regular
- Proof is by contradiction:
 - Assume the language is regular
 - Find a string x in the language, for all splits of $x = uvw$, at least one of the pumping lemma conditions does not hold
 - $uv^pw \in L (\forall i \geq 0)$
 - $v \neq \epsilon$
 - $|uv| \leq p$

Pumping lemma example

- prove $L = a^n b^n$ is not regular
- Assume L is regular: there must be a p such that, if uvw is in the language
 - $uv^pw \in L (\forall i \geq 0)$
 - $v \neq \epsilon$
 - $|uv| \leq p$
 - Pick the string $a^p b^p$
 - For the sake of example, assume $p = 5$, $x = aaaaaabbbb$
 - Three different ways to split



How do we know a language is context-free?

- Again, find a context-free grammar that generates the language
- Examples: $a^n b^n$

$S \rightarrow aSb$
 $S \rightarrow \epsilon$

This is for $n \geq 0$, to disallow $a^n b^n$, replace the second rule with $S \rightarrow ab$

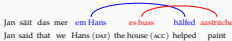
How do we know a language is not context-free?

pumping lemma for context-free languages

- The idea is similar to regular languages, but we can have 'embedded' structures as well as simple loops
- For any sufficiently long sentence $uvxyz$ in a context-free language
 - $uv^i xy^j z \in L (\forall i, j \geq 0)$
 - $|vxy| \geq 1$
 - $|vxy| \leq p$
- Again, the proof is by contradiction
 - Assume the language is context-free
 - Find a string $a = uvxyz$ and a number p in the language that does not satisfy the conditions above

Where do natural language syntax fit?

Cross-serial dependencies



- The above structure is not possible to parse using context-free languages
- Otherwise, experience so far indicates that a CF-based grammar can describe natural language syntax

Chomsky hierarchy: the picture



- Chomsky hierarchy of languages form a hierarchy (with some care about empty language)
- It is often claimed that mildly context sensitive grammars (dashed ellipse) are adequate for representing natural languages
- Note, however, not even every regular language is a potential natural language (e.g., $a^n b^{n^2}$). The possible natural languages probably cross-cut this hierarchy (shaded region)

Summary

- Phrase structure grammars are generative grammars that are finite specifications of (infinite) languages
- They form the basis of the theory of parsing
- More expressive grammar classes (type 0 and type 1) are not computationally attractive
- We will focus on more practical grammar classes, mainly context-free grammars, for the rest of the course
- Next: introduction to parsing
- Suggested reading: Grune and Jacobs (2007, chapter 2)

Example: deriving bbaaa'a

Sentential form	rule
S	(init)
$\text{Begin } B \text{ A End}$	$S \rightarrow \text{Begin } B \text{ A End}$
$\text{Begin } b \text{ A End}$	$B \rightarrow b$
$\text{Begin } b \text{ a A End}$	$A \rightarrow aA$
$\text{Begin } b \text{ a a A End}$	$A \rightarrow aA$
$\text{Begin } b \text{ a a a A End}$	$A \rightarrow aA$
$\text{Begin } b \text{ a a a a A End}$	$A \rightarrow aA$
$\text{Begin } b \text{ a a a a End}$	$A \rightarrow a$
$b \text{ a a a a End}$	$\text{Begin } b \text{ a} \rightarrow bba$
$b \text{ b a a a a}$	$a \text{ a End} \rightarrow a'a$

- The grammar
- $S \rightarrow \text{Begin } B \text{ A End}$
 - $B \rightarrow b$
 - $A \rightarrow a$
 - $A \rightarrow aA$
 - $a \text{ A End} \rightarrow a'a$
 - $\text{Begin } b \text{ a} \rightarrow bba$
 - $\text{Begin } b \text{ b} \rightarrow bb$

Example: deriving baa'a

Sentential form	rule
S	(init)
Begin B A End	S → Begin B A End
Begin b A End	B → b
Begin b a A End	A → a A
Begin b a a A End	A → a A
Begin b a a A End	A → a A
Begin b a a ' a	(none)

We are stuck with a sentential form with non-terminals.

The grammar
1. S → Begin B A End
2. B → b
3. A → a
4. A → a A
5. a A End → a ' a
6. Begin b a → b b a
7. Begin b b → b b

a ⁿ b ^m c ⁿ d ^m
1. S → X Y
2. X → aXC
3. X → aC
4. Y → BYd
5. Y → Bd
6. CB → BC
7. aB → ab
8. bB → bb
9. Cd → cd
10. Cc → cc

- Some explanation:
- Rule (1) generates a string with two parts X and Y
 - For X, we generate as many C's as a's (3), and for Y, we generate as many B's as d's (4)
 - We will eventually rewrite C's as c, and 'B's as 'b', but their order is not correct.
 - When recursions for X and Y terminate, we have equal number of a's and C, and equal number of B's d's, and a's are all at the beginning, and d's are all at the end
 - Rule (2) swaps B and C's
 - We allow rewriting B as b only after and a or b, and allow rewriting C as c only before d

Acknowledgments, references, additional reading material

- Please read Grune and Jacobs (2007) chapter 2, a large part of the lecture follows this chapter

 Grune, Dick and Conrad [Ed.] Jacobs (2007), Parsing Techniques: A Practical Guide, second, Monographs in Computer Science. The first edition is available at http://lba.riken.go.jp/Books/PTAPC_tec_2001/arc/BookIndex.pdf Springer, New York, isbn: 0387004900

