Introduction to theory of languages

Patryk Kiepas

MINES ParisTech & AGH

March 4, 2017

Course plan

1. Saturday, 25th of February 2017 – lecture
    - Languages
    - Grammars
2. Saturday, 4th of March 2017 – lecture
    - Parsing
    - ANTLR
3. Saturday, 11th of March 2017 – exercises
    - Grammars and languages
    - ANTLR
4. Saturday, 25th of March 2017 – exercises
    - ANTLR
5. Exam

Additional informations

Any questions?

Ask by mail: `kiepas@agh.edu.pl`

Course web-page

`http://home.agh.edu.pl/~kiepas` → **Teaching** → **Introduction to theory of languages (2017)**

Outline

Introduction

Linguistics

Scientific study of languages. Involves analysis of language:

- *form* – language evolution and task
- *context* – environment of language usage
- *semantics* – the meaning of the language

Some important aspects

- Phonetics
- Articulation
- Perception
- Acoustic features
- Morphology
- Syntax

Language types

1. Natural languages
   - *Ordinary* – evolves naturally in humans without planning
   - *Controlled* – a restricted subset of natural language in order reduce or eliminate ambiguity and complexity
2. **Artificial languages**
   - *Constructed* (planned *a priori* or *a posteriori*)
     - Engineered languages – experiments in *logic*, *philosophy*, *linguistics*
     - Auxiliary languages – international communication (e.g. Esperanto, Ido, Interlingua)
     - Artistic languages – aesthetic pleasure or humorous effect (e.g. Klingon)
   - **Formal**
     - Computer programming languages (e.g. Java, Haskell, C, C++, Ruby)
     - Files and formats descriptions (e.g. YAML, JSON, XML)

Description of natural languages

A really small bit of history

- In the late 1950's Noam Chomsky tried to describe natural languages
- Important paper: *"Three models for the description of language"*, Noam Chomsky (1956).
- In a result of his research two disciplines originated:
  1. **Theory of formal grammars**
  2. *Generative (transformational) grammars*

Figure 1: Professor of Linguistics (Emeritus) at MIT, Cambridge

Description of natural languages

What we know now?

- Description of natural languages is **hard**
- Description of any natural languages might be **impossible**

Why this is important?

- Better understanding of language creation processes
- More insights into functioning of our brain
- **Natural language processing (NLP)**
  - Translations (e.g. Google Translator)
  - Synthesis (e.g. speech generation)
  - Perceiving (e.g. robots, voice-control)

Description of formal languages

Result

Description of natural languages help us describe an artificial (formal) ones

Programming languages

- Protocol for communication with the computer
- Performing operations and computations
- Interpretation and execution
- Compilation
- Static code analysis

Data formats

- Structured data
- Interchangeable model for communication and data transmission

Alphabet

Alphabet

A set Σ of available symbols, the simplest elements in the language

Examples

- binary alphabet $\{0, 1\}$
- decimal numbers $\{0, 1, 2, 3, ..., 9\}$
- Latin alphabet $\{a, b, c, d, ..., z\}$
- Cyrillic

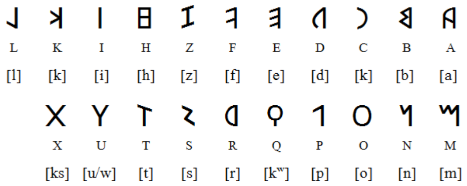
L K I H Z F E D C B A
[l] [k] [i] [h] [z] [f] [e] [d] [k] [b] [a]
X U T S R Q P O N M
[ks] [u/w] [t] [s] [r] [kʷ] [p] [o] [n] [m]

Figure 2: Ancient Latin alphabet

Word (I)

Word

Word w is a sequence of N symbols $w = x_1x_2...x_N$ where $x_i \in \Sigma$
(e.g. 010110, $ABCDAAE$)

Length

Length of the word w is a number of symbols it contains $|w| = N$
(e.g. $|010110| = 6$, $|ABCDAAE| = 7$)

Empty word

Special word ϵ with length $|\epsilon| = 0$

Word (II)

Words examples

- $w = 010110$ word over alphabet $\Sigma = \{0, 1\}$
- $w = abc13dj3$ word over alphabet $\Sigma = \{a, b, ...z, 0, 1, ...9\}$
- $w = ACGTCCGGTA$ word over alphabet $\Sigma = \{A, C, G, T\}$

Kleene star (closures)

- Σ^* – set of all words over Σ
- Σ^+ – set of all nonempty words $\Sigma^+ = \Sigma^* \backslash \{\epsilon\}$

Closures examples

- if $\Sigma = \{a\}$ then $\Sigma^* = \{\epsilon, a, aa, aaa, aaaa, aaaaa, aaaaaa, ...\}$
- if $\Sigma = \{a, b\}$ then $\Sigma^+ = \{a, b, aa, bb, ab, ba, aaa, bbb, ...\}$
- if $\Sigma = \{a, b, ..., z\}$ then $\Sigma^+ = \{cat, dog, a, aa, aaa, ...\}$

Language

Definition

Formal language $L \subseteq \Sigma^*$ is a subset of all words built over an alphabet Σ

Examples

- Language L_1 of palindromes in English
  $L_1 = \{mum, hannah, madam, ...\}$
- Morse code with alphabet $\Sigma = \{\cdot, -\}$, $L_2 = \{\cdot-, -\cdot\cdot..., --\cdot\cdot\}$
- Empty language
- English language
- Language L_3 with the set of words with fixed-size of N
- Language $L_4 = \{a^n b^n | n \geq 1\}$
- Language $L_5 = \{abc^n de | n \geq 0\}$
- Language $L_6 = \{a^m | m = 3n \land n \geq 1\}$

Grammar

Grammar

- Description of a language
- A recipe for composing elements into sentence
- Describes syntax of a language

Definition

Grammar is a system $G = (V_T, V_N, P, S)$ where:

- V_T – terminals (alphabet Σ)
- V_N – nonterminals
- P – production rules
- S – start symbol, $S \in V_N$

Grammar and languages

Definition

Grammar is a system $G = (V_T, V_N, P, S)$ where:

- V_N, V_T, P – are finite, nonempty sets
- $V_N \cap V_T = \emptyset$ – are disjoint
- $V = V_N \cup V_T$ – vocabulary (terminals and nonterminals)
- $P \subseteq V^+ \times V^*$

Derivation

Let $\alpha, \beta \in V$, then we say that:

- β **derives directly** from α (i.e. $\alpha \xRightarrow{p} \beta$) – if there exists production rule $p \in P$ that obtains β from α
- α_n **derives** from α_1 (i.e. $\alpha_1 \xRightarrow{*} \alpha_n$) – if there exists a sequence of direct derivations giving in the result α_n : $\alpha_1 \xRightarrow{p_1} \alpha_2 \xRightarrow{p_2} \alpha_3 \xRightarrow{p_3} \dots \xRightarrow{p_n} \alpha_n$, where $\{p_i : 0 \leq i \leq k \wedge p_i \in P\}$

Derivations

Grammars and languages

- Sentence generated by some G is every $w \in \Sigma^*$ that for each exists derivation from S
- Language $L(G)$ is generated by G and consists of the sentences derivate using grammar G
- Two grammars G_1 and G_2 have *(weak) equivalence* if $L(G_1) = L(G_2)$

Chomsky's hierarchy

Hierarchy

- Describe the grammar expressiveness
- Describe the grammar hardness
- Tells us what "mechanical procedure" we need to use in order to:
  - Accept language
  - Generate language

Grammar	**Language**	**Automaton**	**Production rules**
Type-0	Recursively enumerable	Turing machine	$\alpha \rightarrow \beta$
Type-1	Context-sensitive	Linear bounded ND TM	$\alpha A \beta \rightarrow \alpha \gamma \beta$
Type-2	Context-free	ND pushdown	$\alpha \rightarrow \gamma$
Type-3	Regular	Finite state	$A \rightarrow a$ and $A \rightarrow aB$

Limiting conditions

For all production rules $\forall(\alpha \to \beta) \in P$ it is true:

First condition

- $|\alpha| \leq |\beta|$ - they don't decrease length of a word

Second condition

- $\alpha \in V_N$ is a nonterminal
- $\beta \in V^+$ is not empty

Third condition

- $\alpha \in V_N$ is a nonterminal
- β has a form $\beta = a$ or $\beta = aB$ where $a \in V_T, B \in V_N$

Type-0 : Unrestricted rewriting system

Description

Type-0 grammar has no limitations (unrestricted)

Valid production rules

Production rules have form of $\alpha \rightarrow \beta$, where $\alpha, \beta \in V$

- $aaaA \rightarrow aBb$
- $LLQQ \rightarrow LQ$
- $S \rightarrow \epsilon$
- $C \rightarrow cC$
- $D \rightarrow E$
- $abcD \rightarrow abc$

Type-0 : Grammar example

Grammar

Let $G = (V_N, V_T, P, S)$, where

- $V_N = \{S, A, B, C, D\}$
- $V_T = \{a\}$
- $P = \{S \to ADBC, D \to DD, DB \to BEEE, ABE \to aAB, aABC \to a\}$

Derivations

$S \implies ADBC \implies ADBC \implies ABEEEC \implies aABEEC \implies aaABEC \implies aaaABC \implies aaa$

Language

$L(G) = \{a^m\}$, where $m = 3n \wedge n \geq 1$

Example sentences

$aaa, aaaaaa, aaaaaaaaa, aaaaaaaaaaaa, aaaaaaaaaaaaaaa, \ldots$

Type-1 : Context-sensitive grammar

Description

Productions rules of *type-1 grammar* don't decrease the length of the words (i.e. $|\alpha| \leq |\beta|$) during derivations

Valid rules

- $S \rightarrow \epsilon$
- $C \rightarrow cC$
- $D \rightarrow E$
- $aBc \rightarrow abBc$

Invalid rules

- $aaaA \rightarrow aBb$
- $LLQQ \rightarrow LQ$
- $abcD \rightarrow abc$

Type-1 : Grammar example

Grammar

Let $G = (V_N, V_T, P, S)$, where

- $V_N = \{S, A, B, C, D, E, F\}$
- $V_T = \{a, b, c\}$
- $P = \{S \to abC|Ac|Dbc|aEF|aB, A \to ab, Db \to ab, B \to bc, bC \to bc, F \to c, Ec \to bc\}$

Derivations

- $S \implies abC \implies abc$
- $S \implies Ac \implies abc$
- $S \implies aEF \implies aEc \implies abc$

Language and example sentence

$L(G) = \{abc\}$

Type-2 : Context-free grammar

Description

Rules $A \to \beta$ in context-free grammar have one variable (nonterminal) on the left hand side ($A \in V_N$) and they derive into any word ($\beta \in V^*$)

Valid rules

- $S \to \epsilon$
- $C \to cC$
- $D \to E$
- $F \to abcdef$

Invalid rules

- $aaaA \to aBb$
- $LLQQ \to LQ$
- $aBc \to abefBc$

Type-2 : Grammar example

Grammar

Let $G = (V_N, V_T, P, S)$, where

- $V_N = \{S, A, B\}$
- $V_T = \{a, b\}$
- $P = \{S \to aSB, S \to A, A \to ab, B \to b\}$

Derivations

$S \implies aSB \implies aaSBB \implies aaSBb \implies aaSbb \implies aaabbb$

Language

$L(G) = \{a^n b^n\}$, where $n \geq 1$

Example sentences

ab, *aabb*, *aaabbb*, *aaaabbbb*, *aaaaabbbbb*, *aaaaaabbbbbb*, ...

Type-3 : Regular grammar

Description

Rules in regular grammar have form of $A \rightarrow a$ and $A \rightarrow aB$ (right recursion) or $A \rightarrow Ba$ (left recursion), where $A, B \in V_N$ and $a \in V_T$

Valid rules

- $C \rightarrow cD$
- $D \rightarrow Dc$
- $S \rightarrow b$

Invalid rules

- $S \rightarrow \epsilon$
- $D \rightarrow E$
- $aBc \rightarrow abefBc$
- $F \rightarrow abcdef$

Grammar example – regular

Grammar

Let $G = (V_N, V_T, P, S)$, where

- $V_N = \{S, B\}$
- $V_T = \{a, b\}$
- $P = \{S \to aB, B \to bS, B \to b\}$

Derivation

$S \implies aB \implies abS \implies abaB \implies ababS \implies ababaB \implies \ldots$

Language

$L(G) = \{(ab)^n\}$, where $n \geq 1$.

Example sentences

$ab, abab, ababab, abababab, ababababab, abababababab, \ldots$

Normal forms

Derivation trees

Also : tree diagrams, phrase markers. For regular and context-free grammars.

Grammar examples

Grammar

Let $G = (V_N, V_T, P, S)$, where

- $V_N = \{S\}$
- $V_T = \{a, b\}$
- $P = \{S \to aS \vee S \to Sa, S \to b\}$

Derivations

$$S \implies aS \implies aaS \implies aaaS \implies aaaaS \implies aaaaaS \implies \ldots$$
$$S \implies Sa \implies Saa \implies Saaa \implies Saaaa \implies Saaaaa \implies \ldots$$

Language

$L(G) = \{a^n b\}$, where $n \geq 0$

Example sentences

$b, ab, aab, aaab, aaaab, aaaaab, aaaaaab, aaaaaaab, aaaaaaaab, \ldots$

Grammar example: *mirror language*

Grammar

Let $G = (V_N, V_T, P, S)$, where

- $V_N = \{S\}$
- $V_T = \{a, b\}$
- $P = \{S \rightarrow aSa, S \rightarrow bSb, S \rightarrow aa, S \rightarrow bb\}$

Derivations

$S \implies aSa \implies abSba \implies abbSbbs \implies abbaSabba \implies \dots$

Language

$L(G) = \{ww^R\}$, where w^R represents reflection of w, and $|w| \geq 1$. This language $L(G)$ is called a *mirror language*.

Example sentences

aa, *bb*, *aaaa*, *abba*, *baab*, *bbbb*, *abaaba*, *baaaab*, *abbbba*, *babbab*, *aaaaaa*...

Grammar example

Grammar

Let $G = (V_N, V_T, P, S)$, where

- $V_N = \{S, E, F\}$
- $V_T = \{a, b, c, d\}$
- $P = \{S \to ESF, S \to EF, E \to ab, F \to cd\}$

Derivations

$S \implies ESF \implies EESFF \implies EEESFFF \implies E^{n-1}SF^{n-1} \implies E^nF^n$

Language

$L(G) = \{(ab)^n(cd)^n\}$, where $n \geq 1$.

Example sentences

$abcd, ababcdcd, abababcdcdcd, ababababcdcdcdcd, \ldots$

Grammar example

Grammar

Let $G = (V_N, V_T, P, S)$, where

- $V_N = \{S, E, F\}$
- $V_T = \{a, b, c, d\}$
- $P = \{S \to ESF, S \to abcd, Ea \to aE, dF \to Fd, Eb \to abb, cF \to ccd\}$

Derivations

$S \Longrightarrow ESF \Longrightarrow EabcdF \Longrightarrow aEbcdF \Longrightarrow aEbcFd \Longrightarrow aabbcFd \Longrightarrow aabbccdd$

Language

$L(G) = \{a^n b^n c^n d^n\}$, where $n \geq 2$.

Sentences

$aabbccdd, aaabbbcccddd, aaaabbbbccccdddd, aaaaabbbbbcccccddddd, \ldots$

Language and grammar

Two common tasks:

- Check if language is legal (accepted by the grammar) – trace all the applicable rules (derive it language from the start symbol) or... use corresponding automaton!
- Generate language from grammar – start from *start symbol*, go through all applicable rules

Backus-Naur form (BNF)

Backus-Naur form (BNF)

Notation technique for *context-free grammars*. Frequently used to describe syntax of *programming languages*, *document formats* etc.

Syntax

```
<term> ::= __expression__
```

- `<term>` is a *nonterminal*
- `__expression__` is a sequence of one or more terminal and/or nonterminal symbols separated by vertical line `|`
- Terminal symbols: `a`, `b`, `c`, `A`, `0`, `1`, `2` etc.
- Nonterminal symbols: `<digit>`, `<postal-code>` etc.

Backus-Naur form (BNF)

Meta-symbols

- $::=$ – production rule definition
- $|$ – rule alternative
- $<>$ – nonterminals
- "" – literal
- $< EOL >$ – End Of Line

Examples

```
<digit> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
<postal-code> ::= <digit> <digit> <digit> <digit> <digit>
```

BNF example : Palindrome

Palindrome grammar

```
<letter>     ::= a | b | c | ... | y | z
<palindrome> ::= <letter> |
<palindrome> ::= a <palindrome> a | b <palindrome> b |
                 c <palindrome> c | d <palindrome> d |
                 e <palindrome> e | ...
                                  | z <palindrome> z
```

Results

```
a
bb
bab
pop
hannah
```

BNF example : Postal address

Postal address grammar

```
<postal-address> ::= <name-part> <street-address> <zip-part>
<name-part> ::= <first-name> <last-name> <EOL>
<street-address> ::= <number> <street-name> <apt-num> <EOL>
<zip-part> ::= <postal-code> <town-name> <EOL>
<apt-num> ::= <number> | ""
```

ANTLR v4

Parser generator

content...

ANTLR

A parser generator which allows to:

-

Usages

- Twitter search queries are parsed using ANTLR
- Lex Machina[a] extracts informations from legal texts using ANTLR

[a] lexmachina.com

ANTLR syntax (I)

Grammar structure

```
grammar ANY_NAME;
options {...}
import ... ;
tokens {...}
channels {...}
@actionName{...}
// lexer rules
LEXER_RULE1
LEXER_RULE2
// parser rules
parser_rule1
parser_rule2
```

Grammar properties

- Each section can be specified in any order
- Only one definition for sections: *options*, *imports*, *tokens*
- The header and at least one rule are mandatory

Reserved keywords

import, *fragment*, *lexer*, *parser*, *grammar*, *returns*, *locals*, *throws*, *catch*, *finally*, *mode*, *options*, *tokens*

Grammar file

The file name with grammar *ANY_NAME* must be called *ANY_NAME.g4*

Lexer vs parser

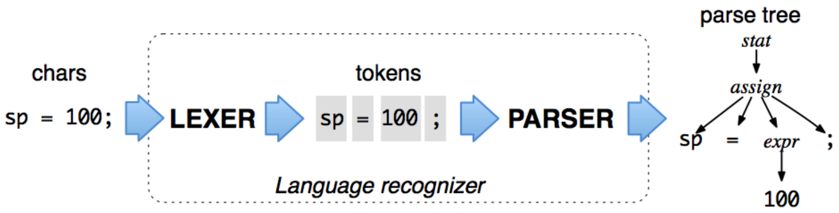

Figure 3: Caption goes here[1]

Lexer

Tokens (terminal symbols with semantics) are symbols for the parser.
Parser understands context-free grammar (Chomsky's level 2).

[1] From ANTLR4 on-line documentation

ANTLR syntax (II)

Syntax	**Description**
x	Match token, rule or subrule x
$xy....z$	Match a sequence of elements
(...|...|...)	Sub-rule with multiple alternatives
x?	Match x or skip it
$x*$	Match x zero or more times
$x+$	Match x one or more times
r : ...	Define rule r
r : (...|...|...)	Define rule r with multiple alternatives

ANTLR patterns

Pattern name	Examples
Sequence	`'[' INT+ ']'`
Sequence with terminator	`(statement ';')*`
Sequence with separator	`( expr (',' expr)*)?`
Choice	`type : 'int' | 'float'`
Token dependency	`ID '[' expr ']'`
Nested phrase	`expr : '(' expr ')' | ID`

Action and semantic predicate

First grammar

Simple grammar (Hello.g4)

```
// define a grammar called Hello
grammar Hello;
// match lower-case identifiers
ID : [a-z]+;
// skip spaces, tabs, newlines, \r (Windows)
WS : [ \t\r\n]+ -> skip;
// match keyword hello followed by an identifier
r : 'hello' ID;
```

Nested arrays

Nested arrays grammar (ArrayInit.g4)

```
grammar ArrayInit;
// matches at least one comma-separated value between {...}
init : '{' value (',' value)* '}';
// A value can be either a nested array or an integer (INT)
value : init | INT;
// define token INT as one or more digits
INT : [0-9]+;
WS : [ \t\r\n]+ -> skip;
```

// parser rules start with lowercase letters, lexer rules with uppercase

Parser tester

Parser

```
import org.antlr.v4.runtime.*;
import org.antlr.v4.runtime.tree.*;

public class Test {
    public static void main(String[] args) throws Exception {
        // create a CharStream that reads from standard input
        ANTLRInputStream input = new ANTLRInputStream(System.in);
        // create a lexer that feeds off of input
        CharStream ArrayInitLexer lexer = new ArrayInitLexer(input);
        // create a buffer of tokens pulled from the lexer
        CommonTokenStream tokens = new CommonTokenStream(lexer);
        // create a parser that feeds off the tokens buffer
        ArrayInitParser parser = new ArrayInitParser(tokens);
        ParseTree tree = parser.init();
        System.out.println(tree.toStringTree(parser));
    }
}
```

Calculator

```
grammar Expr;

prog: stat+;
stat: expr NEWLINE
    | ID '=' expr NEWLINE
    | NEWLINE;

expr: expr ('*'|'/') expr
    | expr ('+'|'-') expr
    | INT
    | ID
    | '(' expr ')';

ID : [a-zA-Z]+;
INT : [0-9]+;
// return newlines to parser (is end-statement signal)
NEWLINE:'\r'? '\n';
WS : [ \t]+ -> skip;
```

Generated parser

zawartość...

Generated lexer

zawartość...

Generated tree visitor

zawartość...

Generated tree listener

zawartość...

Importing grammars

show two files : lexer & grammar

ANTLR caveats

- Lexer/Parser rules order matter
-