

Introduction to theory of languages

Patryk Kiepas

MINES ParisTech & AGH

March 4, 2017

Course information

Plan

- 1 Saturday, 25th of February 2017 – lecture (introduction to grammars)
- 2 **Saturday, 4th of March 2017 – lecture (grammars and ANTLR)**
- 3 Saturday, 11th of March 2017 – exercises (grammars)
- 4 Saturday, 25th of March 2017 – exercises (ANTLR)
- 5 Exam

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)**

- 1 Course information
- 2 Theory
 - Languages
 - Grammars
 - Hierarchy of grammars
 - Examples of grammars
- 3 Practice
 - Notations of grammar
 - ANTLR
 - ANTLR examples

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

1 Natural languages

- *Ordinary* – evolves naturally in humans without planning
- *Controlled* – a restricted subset of natural language in order to 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 $\{0, 1, 2, 3, \dots, 9\}$
- Latin alphabet $\{a, b, c, d, \dots, z\}$
- Cyrillic

Ɑ	Ɱ	Ɐ	Ɒ	ⱱ	Ⱳ	ⱳ	ⱴ	Ⱶ	ⱶ	ⱷ
L	K	I	H	Z	F	E	D	C	B	A
[l]	[k]	[i]	[h]	[z]	[f]	[e]	[d]	[c]	[b]	[a]
ⱸ	ⱹ	ⱺ	ⱻ	ⱼ	ⱽ	Ȿ	Ɀ	Ⳁ	ⳁ	Ⳃ
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 $\Sigma = \{a, b, \dots, z, 0, 1, \dots, 9\}$
- $w = ACGTCCGGTA$ word over alphabet $\Sigma = \{A, C, G, T\}$

Kleene star (closures)

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

Closures examples

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

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
 $L_1 = \{mum, hannah, madam, \dots\}$
- Morse code with alphabet $\Sigma = \{., -\}$, $L_2 = \{.-, - \cdot \dots, - - \cdot\}$
- 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 $L_6 = \{a^m | m = 3n \wedge n \geq 1\}$

Grammar

- Description of a language and its syntax
- A recipe for composing elements of an alphabet into sentence

Definition

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

- V_T – **terminals (alphabet Σ)**
- V_N – **nonterminals**
- $V_N \cap V_T = \emptyset$
- $V = V_N \cup V_T$ – vocabulary
- P – **production rules** ($P \subseteq V^+ \times V^*$)
- V_N, V_T, P – are finite, nonempty sets
- S – **start symbol**, $S \in V_N$

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 $\{p_i : 0 \leq i \leq k \wedge p_i \in P\}$

Grammars and languages

- Sentence $w \in V_T^*$ is generated by grammar G if w **derives** from S ($S \xRightarrow{*} w$)
- We say grammar G **generate** language $L(G)$ if it contains all sentences derived from S , in a result $L(G) = \{w \in V_T^* : \exists(S \xRightarrow{*} w)\}$
- Two grammars G_1 and G_2 have (*weak*) *equivalence* if $L(G_1) = L(G_2)$

Grammar examples

Grammar

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

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

Derivations

$S \Rightarrow aS \Rightarrow aaS \Rightarrow aaaS \Rightarrow aaaaS \Rightarrow aaaaaS \Rightarrow \dots$
 $S \Rightarrow Sa \Rightarrow Saa \Rightarrow Saaa \Rightarrow Saaaa \Rightarrow Saaaaa \Rightarrow \dots$

Language

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

Example sentences

$b, ab, aab, aaab, aaaab, aaaaaab, aaaaaaab, aaaaaaaab, \dots$

Chomsky's hierarchy

Hierarchy

- Describe the grammar expressiveness
- **Observation:** regular language is also a context-free grammar etc. (Regular \subseteq Context-free \subseteq Context-sensitive \subseteq Unrestricted)
- The nested grammar is more restrictive than the enclosing one

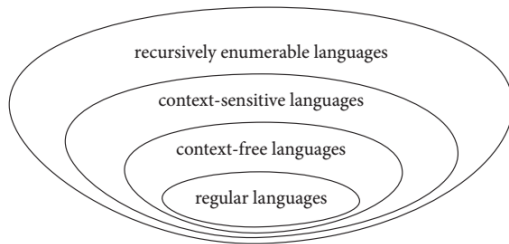


Figure 3: Chomsky's hierarchy

Limiting conditions

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

First condition

- $|\alpha| \leq |\beta|$ - they don't decrease length of a word, where $\alpha, \beta \in V^*$

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 : Recursively enumerable (unrestricted) grammar

Description

Type-0 grammar has no limitations (is 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

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

Derivations

$S \Rightarrow ADBC \Rightarrow ADBC \Rightarrow ABE EC \Rightarrow aABE EC \Rightarrow$
 $aaABE C \Rightarrow aa aABC \Rightarrow aaa$

Language

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

Example sentences

$aaa, aaaaaa, aaaaaaaaa, aaaaaaaaaaaaa, aaaaaaaaaaaaaaaaa, \dots$

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

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

Derivations

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

Language and example sentence

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

Type-2 : Context-free grammar

Description

Rules $A \rightarrow \beta$ 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

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

Invalid rules

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

Type-2 : Grammar example

Grammar

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

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

Derivations

$S \Rightarrow aSB \Rightarrow aaSBB \Rightarrow aaSbb \Rightarrow aaSbb \Rightarrow aaabbb$

Language

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

Example sentences

$ab, aabb, aaabbb, aaaabbbb, aaaaabbbbbb, aaaaaabbbbbbb, \dots$

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

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

Derivation

$S \Rightarrow aB \Rightarrow abS \Rightarrow abaB \Rightarrow ababS \Rightarrow ababaB \Rightarrow \dots$

Language

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

Example sentences

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

Normal forms

Let $A, B \in V_N$, $a \in V_T$ and $\beta \in V_N^*$, then:

The Chomsky normal-form

All productions of a grammar have the form of $A \rightarrow BC$ or $A \rightarrow a$

The Greibach normal-form

All productions of a grammar have the form $A \rightarrow a\beta$

Theorem

*Any context-free language can be generated by a grammar in **Chomsky normal-form***

Theorem

*Any context-free language can be generated by a grammar in **Greibach normal-form***

Normal forms

We saw that $L(G) = \{a^n b^n\}$, where $n \geq 1$ is a language created by a grammar G with production rules:

- $P = \{S \rightarrow aSB, S \rightarrow A, A \rightarrow ab, B \rightarrow b\}$

But there are other rules P' from grammar G' in a Chomsky normal-form that will give us the same language $L(G) = L(G')$:

- $P' = \{S \rightarrow CB, D \rightarrow SB, S \rightarrow CD, C \rightarrow a, B \rightarrow b\}$

Derivation a^3b^3

$S \Rightarrow CD \Rightarrow aD \Rightarrow aSB \Rightarrow aCDb \Rightarrow aaDb \Rightarrow aaSBb \Rightarrow aaSbb \Rightarrow aaCBbb \Rightarrow aaaBbb \Rightarrow aaabbb$

And rules P'' from grammar G'' in a Greibach normal form:

- $P'' = \{S \rightarrow aD, D \rightarrow aDB, D \rightarrow aBB, B \rightarrow b\}$

Derivation a^3b^3

$S \Rightarrow aD \Rightarrow aaDB \Rightarrow aaaBBB \Rightarrow aaabBB \Rightarrow aaabbB \Rightarrow aaabbb$

Derivation trees

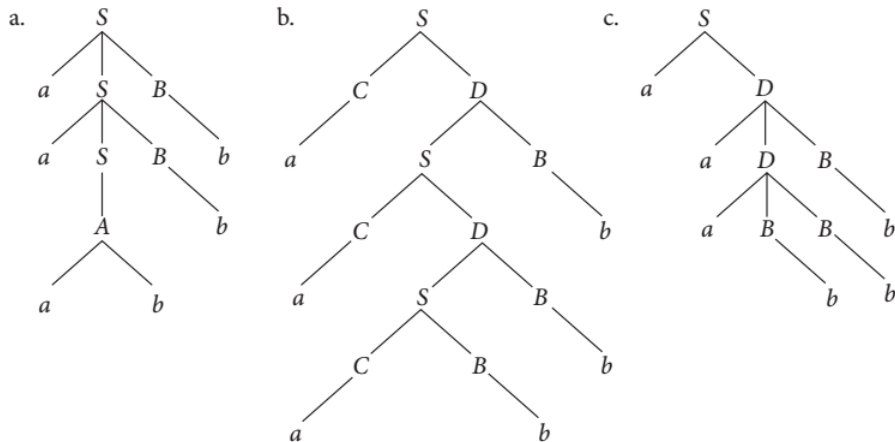


Figure 4: Derivations trees for word a^3b^3 . **a.** derivation in grammar G , **b.** derivation in grammar G' (Chomsky normal-form), **c.1** derivation in grammar G'' (Greibach normal-form)

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

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

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

$aa, bb, aaaa, abba, baab, bbbb, abaaba, baaaab, abbbba, babbab, aaaaaa, \dots$

Grammar example

Grammar

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

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

Derivations

$S \Rightarrow E\textcolor{red}{S}F \Rightarrow EE\textcolor{red}{S}FF \Rightarrow EEE\textcolor{red}{S}FFF \Rightarrow E^{n-1}\textcolor{red}{S}F^{n-1} \Rightarrow E^n F^n$

Language

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

Example sentences

$abcd, abab\textcolor{teal}{cdcd}, ababab\textcolor{teal}{cdcdcd}, abababab\textcolor{teal}{cdcdcdcd}, \dots$

Grammar example

Grammar

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

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

Derivations

$S \Rightarrow ESF \Rightarrow EabcdF \Rightarrow aEbcdF \Rightarrow aEbCdF \Rightarrow aabbcFd \Rightarrow aabbccdd$

Language

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

Sentences

$aabbccdd, aaabbbccdd, aaaabbbbccccdd, aaaaabbbbbccccccdd, \dots$

A few common tasks:

- Check language correctness
- Generate language from grammar
- Generate grammar for language
- Classify grammar (according to the Chomsky hierarchy)

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

$\langle \text{term} \rangle ::= _ _ \text{expression} _ _$

- $\langle \text{term} \rangle$ is a *nonterminal*
- $_ _ \text{expression} _ _$ 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: $\langle \text{digit} \rangle$, $\langle \text{postal-code} \rangle$ etc.

Backus-Naur form (BNF)

Meta-symbols

- $::=$ – production rule definition
- $|$ – rule alternative
- $\langle \rangle$ – nonterminals
- $''$ – literal
- $\langle EOL \rangle$ – End Of Line

Examples

$\langle \text{digit} \rangle ::= 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9$

$\langle \text{postal-code} \rangle ::= \langle \text{digit} \rangle \langle \text{digit} \rangle \langle \text{digit} \rangle \langle \text{digit} \rangle \langle \text{digit} \rangle$

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> | ""
```

Lexer vs parser

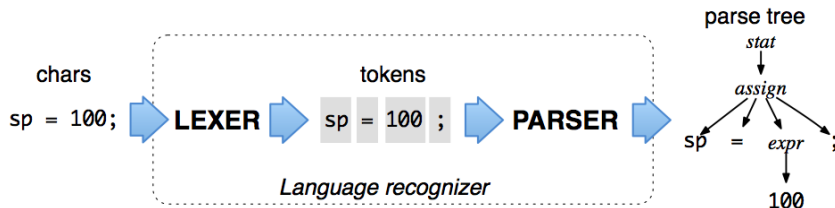


Figure 5: A scheme for lexical and semantic analysis¹

- Lexical analysis converts a sequences of characters into a sequence of tokens (strings with meanings)
- Semantic analysis matches a sequence of tokens into some language's construct

¹From ANTLR4 on-line documentation

Parser generator

Generate lexer and parser for given grammar.

ANTLR

A parser generator which allows to:

- Perform lexical analysis (lexer)
- Perform semantic analysis (parser)
- Code transformation
- Perform code evaluation

Usages

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

^alexmachina.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*

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

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 ;
```

ANTLR patterns

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

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;
```