SOM2IF15 – Compilation

Syntactic Analysis

Frédéric Loulergue

2022

Outline

Syntax: Context-Free Grammars

Definition (Context-Free Grammar)

A context-free grammar is a quadruple (NT, T, R, S) where:

1. NT is a finite set of symbols (**non-terminal** symbols, or syntactic categories)
2. T is a finite set of symbols (**terminal** symbols)
3. R is a finite set of **productions** (or rules), each of which is of the form: $V \rightarrow w$ where:
   - V (the **head**) is a single non-terminal symbol
   - w (the body) is a string composed of zero or more terminal or non-terminal symbols (w is a string over $T \cup NT$)
4. S is the initial symbol ($S \in NT$)

Syntax: Context-Free Grammar Example

Example: A Context-Free Grammar for SL

$$G = \quad (\quad \{X, D, N, A, E, I, P\}, \quad \{a, \ldots, z, 0, \ldots, 9, +, -, /, \times, =, (,), ;\}, \quad R, \ P \quad)$$

where R is the following sets of productions:

1. $D \to 0$

$\vdots$

9. $D \to 9$
10. $N \to D$
11. $N \to N\,D$
12. $X \to a$

$\vdots$

38. $X \to z$
39. $A \to X$
40. $A \to N$
41. $E \to A$
42. $E \to A + A$
43. $E \to A - A$
44. $E \to A/A$
45. $E \to A \times A$
46. $I \to \texttt{print}(X)$
47. $I \to \texttt{input}(X)$
48. $I \to X = E$
49. $I \to \texttt{ifz}\ X\ \texttt{then}\ X = E$
50. $I \to \texttt{ifdz}\ X\ \texttt{then}\ X = E$
51. $P \to \epsilon$
52. $P \to I; P$

Syntax: Context-Free Grammars

Definition (Derivation)

Let G be a grammar (NT, T, R, S), and two strings u and u' over $T \cup NT$.

- u' **immediately derived** from u (written $u \Rightarrow u'$) if u' is obtained by replacing in u a non-terminal symbol V by a sequence w with the rule $V \rightarrow w$.
- u' is **derived** from u (written $u \Rightarrow^* u'$) if there exists a finite sequence of immediate derivations from u to u'.

Definition (Generated Language)

The language generated by a grammar $G = (NT, T, R, S)$, is the set

$$\mathcal{L}(G) = \{ w \in T^* \mid S \Rightarrow^* w \}$$

That is every word (sequence of finite symbol) that can be derived from the initial element of the grammar.

Syntax: Context-Free Grammars – Derivation Example

P

Syntax: Context-Free Grammars – Derivation Example

$P \Rightarrow I ; P$ by rule 52: $P \rightarrow I ; P$

Syntax: Context-Free Grammars – Derivation Example

$$
\begin{array}{lcll}
P & \Rightarrow & I; P & \text{by rule 52: } P \rightarrow I; P \\
 & \Rightarrow & I; I; P & \text{by rule 52: } P \rightarrow I; P
\end{array}
$$

Syntax: Context-Free Grammars – Derivation Example

$$
\begin{array}{lcll}
P & \Rightarrow & I ; P & \text{by rule 52: } P \rightarrow I ; P \\
 & \Rightarrow & I ; I ; P & \text{by rule 52: } P \rightarrow I ; P \\
 & \Rightarrow & I ; I ; I ; P & \text{by rule 52: } P \rightarrow I ; P
\end{array}
$$

Syntax: Context-Free Grammars – Derivation Example

$$
\begin{array}{lcll}
P & \Rightarrow & I; P & \text{by rule 52: } P \rightarrow I; P \\
 & \Rightarrow & I; I; P & \text{by rule 52: } P \rightarrow I; P \\
 & \Rightarrow & I; I; I; P & \text{by rule 52: } P \rightarrow I; P \\
 & \Rightarrow & I; I; I; & \text{by rule 51: } P \rightarrow \epsilon
\end{array}
$$

Syntax: Context-Free Grammars – Derivation Example

$$
\begin{array}{lll}
P & \Rightarrow & I; P \qquad \text{by rule 52: } P \rightarrow I; P \\
 & \Rightarrow & I; I; P \qquad \text{by rule 52: } P \rightarrow I; P \\
 & \Rightarrow & I; I; I; P \qquad \text{by rule 52: } P \rightarrow I; P \\
 & \Rightarrow & I; I; I; \qquad \text{by rule 51: } P \rightarrow \epsilon \\
 & \Rightarrow & \texttt{input}(X); I; I; \qquad \text{by rule 47: } I \rightarrow \texttt{input}(X)
\end{array}
$$

Syntax: Context-Free Grammars – Derivation Example

$$
\begin{array}{lll}
P & \Rightarrow & I; P \qquad \text{by rule 52: } P \rightarrow I; P \\
 & \Rightarrow & I; I; P \qquad \text{by rule 52: } P \rightarrow I; P \\
 & \Rightarrow & I; I; I; P \qquad \text{by rule 52: } P \rightarrow I; P \\
 & \Rightarrow & I; I; I; \qquad \text{by rule 51: } P \rightarrow \epsilon \\
 & \Rightarrow & \texttt{input}(X); I; I; \qquad \text{by rule 47: } I \rightarrow \texttt{input}(X) \\
 & \Rightarrow & \texttt{input}(X); X = E; I; \qquad \text{by rule 48: } I \rightarrow X = E
\end{array}
$$

Syntax: Context-Free Grammars – Derivation Example

$$
\begin{array}{lll}
P & \Rightarrow & I; P \qquad \text{by rule 52: } P \rightarrow I; P \\
 & \Rightarrow & I; I; P \qquad \text{by rule 52: } P \rightarrow I; P \\
 & \Rightarrow & I; I; I; P \qquad \text{by rule 52: } P \rightarrow I; P \\
 & \Rightarrow & I; I; I; \qquad \text{by rule 51: } P \rightarrow \epsilon \\
 & \Rightarrow & \texttt{input}(X); I; I; \qquad \text{by rule 47: } I \rightarrow \texttt{input}(X) \\
 & \Rightarrow & \texttt{input}(X); X = E; I; \qquad \text{by rule 48: } I \rightarrow X = E \\
 & \Rightarrow & \texttt{input}(X); X = E; \texttt{print}(X); \qquad \text{by rule 46: } I \rightarrow \texttt{input}(X)
\end{array}
$$

Syntax: Context-Free Grammars – Derivation Example

$$
\begin{array}{lll}
P & \Rightarrow & I; P \qquad \text{by rule 52: } P \rightarrow I; P \\
 & \Rightarrow & I; I; P \qquad \text{by rule 52: } P \rightarrow I; P \\
 & \Rightarrow & I; I; I; P \qquad \text{by rule 52: } P \rightarrow I; P \\
 & \Rightarrow & I; I; I; \qquad \text{by rule 51: } P \rightarrow \epsilon \\
 & \Rightarrow & \texttt{input}(X); I; I; \qquad \text{by rule 47: } I \rightarrow \texttt{input}(X) \\
 & \Rightarrow & \texttt{input}(X); X = E; I; \qquad \text{by rule 48: } I \rightarrow X = E \\
 & \Rightarrow & \texttt{input}(X); X = E; \texttt{print}(X); \qquad \text{by rule 46: } I \rightarrow \texttt{input}(X) \\
 & \Rightarrow & \texttt{input}(X); X = A + A; \texttt{print}(X); \qquad \text{by rule 42: } E \rightarrow A + A
\end{array}
$$

Syntax: Context-Free Grammars – Derivation Example

$$
\begin{array}{lll}
P & \Rightarrow & I; P \qquad \text{by rule 52: } P \to I; P \\
 & \Rightarrow & I; I; P \qquad \text{by rule 52: } P \to I; P \\
 & \Rightarrow & I; I; I; P \qquad \text{by rule 52: } P \to I; P \\
 & \Rightarrow & I; I; I; \qquad \text{by rule 51: } P \to \epsilon \\
 & \Rightarrow & \texttt{input}(X); I; I; \qquad \text{by rule 47: } I \to \texttt{input}(X) \\
 & \Rightarrow & \texttt{input}(X); X = E; I; \qquad \text{by rule 48: } I \to X = E \\
 & \Rightarrow & \texttt{input}(X); X = E; \texttt{print}(X); \qquad \text{by rule 46: } I \to \texttt{input}(X) \\
 & \Rightarrow & \texttt{input}(X); X = A + A; \texttt{print}(X); \qquad \text{by rule 42: } E \to A + A \\
 & \Rightarrow & \texttt{input}(X); X = X + A; \texttt{print}(X); \qquad \text{by rule 39: } A \to X
\end{array}
$$

Syntax: Context-Free Grammars – Derivation Example

$$
\begin{array}{lll}
P & \Rightarrow & I; P \qquad \text{by rule 52: } P \to I; P \\
 & \Rightarrow & I; I; P \qquad \text{by rule 52: } P \to I; P \\
 & \Rightarrow & I; I; I; P \qquad \text{by rule 52: } P \to I; P \\
 & \Rightarrow & I; I; I; \qquad \text{by rule 51: } P \to \epsilon \\
 & \Rightarrow & \texttt{input}(X); I; I; \qquad \text{by rule 47: } I \to \texttt{input}(X) \\
 & \Rightarrow & \texttt{input}(X); X = E; I; \qquad \text{by rule 48: } I \to X = E \\
 & \Rightarrow & \texttt{input}(X); X = E; \texttt{print}(X); \qquad \text{by rule 46: } I \to \texttt{input}(X) \\
 & \Rightarrow & \texttt{input}(X); X = A + A; \texttt{print}(X); \qquad \text{by rule 42: } E \to A + A \\
 & \Rightarrow & \texttt{input}(X); X = X + A; \texttt{print}(X); \qquad \text{by rule 39: } A \to X \\
 & \Rightarrow & \texttt{input}(X); X = X + N; \texttt{print}(X); \qquad \text{by rule 40: } A \to N
\end{array}
$$

Syntax: Context-Free Grammars – Derivation Example

$$
\begin{array}{lll}
P & \Rightarrow & I; P \qquad \text{by rule 52: } P \to I; P \\
 & \Rightarrow & I; I; P \qquad \text{by rule 52: } P \to I; P \\
 & \Rightarrow & I; I; I; P \qquad \text{by rule 52: } P \to I; P \\
 & \Rightarrow & I; I; I; \qquad \text{by rule 51: } P \to \epsilon \\
 & \Rightarrow & \texttt{input}(X); I; I; \qquad \text{by rule 47: } I \to \texttt{input}(X) \\
 & \Rightarrow & \texttt{input}(X); X = E; I; \qquad \text{by rule 48: } I \to X = E \\
 & \Rightarrow & \texttt{input}(X); X = E; \texttt{print}(X); \qquad \text{by rule 46: } I \to \texttt{input}(X) \\
 & \Rightarrow & \texttt{input}(X); X = A + A; \texttt{print}(X); \qquad \text{by rule 42: } E \to A + A \\
 & \Rightarrow & \texttt{input}(X); X = X + A; \texttt{print}(X); \qquad \text{by rule 39: } A \to X \\
 & \Rightarrow & \texttt{input}(X); X = X + N; \texttt{print}(X); \qquad \text{by rule 40: } A \to N \\
 & \Rightarrow^4 & \texttt{input}(a); a = a + N; \texttt{print}(a); \qquad \text{by rule 12: } X \to a
\end{array}
$$

Syntax: Context-Free Grammars – Derivation Example

$$
\begin{array}{lll}
P & \Rightarrow & I; P \qquad \text{by rule 52: } P \to I; P \\
 & \Rightarrow & I; I; P \qquad \text{by rule 52: } P \to I; P \\
 & \Rightarrow & I; I; I; P \qquad \text{by rule 52: } P \to I; P \\
 & \Rightarrow & I; I; I; \qquad \text{by rule 51: } P \to \epsilon \\
 & \Rightarrow & \texttt{input}(X); I; I; \qquad \text{by rule 47: } I \to \texttt{input}(X) \\
 & \Rightarrow & \texttt{input}(X); X = E; I; \qquad \text{by rule 48: } I \to X = E \\
 & \Rightarrow & \texttt{input}(X); X = E; \texttt{print}(X); \qquad \text{by rule 46: } I \to \texttt{input}(X) \\
 & \Rightarrow & \texttt{input}(X); X = A + A; \texttt{print}(X); \qquad \text{by rule 42: } E \to A + A \\
 & \Rightarrow & \texttt{input}(X); X = X + A; \texttt{print}(X); \qquad \text{by rule 39: } A \to X \\
 & \Rightarrow & \texttt{input}(X); X = X + N; \texttt{print}(X); \qquad \text{by rule 40: } A \to N \\
 & \Rightarrow^4 & \texttt{input}(a); a = a + N; \texttt{print}(a); \qquad \text{by rule 12: } X \to a \\
 & \Rightarrow & \texttt{input}(a); a = a + D; \texttt{print}(a); \qquad \text{by rule 10: } N \to D
\end{array}
$$

Syntax: Context-Free Grammars – Derivation Example

$$
\begin{array}{lll}
P & \Rightarrow & I; P \qquad \text{by rule 52: } P \to I; P \\
 & \Rightarrow & I; I; P \qquad \text{by rule 52: } P \to I; P \\
 & \Rightarrow & I; I; I; P \qquad \text{by rule 52: } P \to I; P \\
 & \Rightarrow & I; I; I; \qquad \text{by rule 51: } P \to \epsilon \\
 & \Rightarrow & \texttt{input}(X); I; I; \qquad \text{by rule 47: } I \to \texttt{input}(X) \\
 & \Rightarrow & \texttt{input}(X); X = E; I; \qquad \text{by rule 48: } I \to X = E \\
 & \Rightarrow & \texttt{input}(X); X = E; \texttt{print}(X); \qquad \text{by rule 46: } I \to \texttt{input}(X) \\
 & \Rightarrow & \texttt{input}(X); X = A + A; \texttt{print}(X); \qquad \text{by rule 42: } E \to A + A \\
 & \Rightarrow & \texttt{input}(X); X = X + A; \texttt{print}(X); \qquad \text{by rule 39: } A \to X \\
 & \Rightarrow & \texttt{input}(X); X = X + N; \texttt{print}(X); \qquad \text{by rule 40: } A \to N \\
 & \Rightarrow^4 & \texttt{input}(a); a = a + N; \texttt{print}(a); \qquad \text{by rule 12: } X \to a \\
 & \Rightarrow & \texttt{input}(a); a = a + D; \texttt{print}(a); \qquad \text{by rule 10: } N \to D \\
 & \Rightarrow & \texttt{input}(a); a = a + 1; \texttt{print}(a); \qquad \text{by rule 2: } D \to 1
\end{array}
$$

Exercise

$G = (\{F, D, N\}, \{0, \ldots, 9, .\}, R, F)$
where R is the following sets of productions:

- $D \to 0$
- $\vdots$
- $D \to 9$
- $N \to ND$
- $N \to D$
- $F \to N.N$
- $F \to N$

Show that there exists a **derivation** such that: $F \Rightarrow^* 255.32$

Syntax: Context-Free Grammars

Definition (Derivation Tree)

Let G be a grammar (NT, T, R, S).
A **derivation tree** (or **parse tree**) is an ordered tree in which:

1. Each node is labelled with a symbol in $NT \cup T \cup \{\epsilon\}$
2. The root is labelled with S
3. Each interior node is labelled with a symbol in NT
4. If a certain node has the label $A \in NT$ and its children are $m_1, \ldots, m_k$ labelled respectively with $X_1, \ldots, X_k$ where $X_i \in NT \cup T$ for all $i \in [1, k]$, then $A \rightarrow X_1 \ldots X_k$ is a production of R
5. If a node has label ϵ, then that node is the unique child. If A is its parent, $A \rightarrow \epsilon$ is a production in R

Syntax: Context-Free Grammars

Example

$G = (\ \{X, D, N, A, E, I, P\},\ \{a, \ldots, z, 0, \ldots, 9, +, -, /, \times, =, (,), ;\},\ R,\ P)$

with R the following sets of productions:

1. $D \to 0$

⋮

9. $D \to 9$
10. $N \to D$
11. $N \to N\ D$
12. $X \to a$

⋮

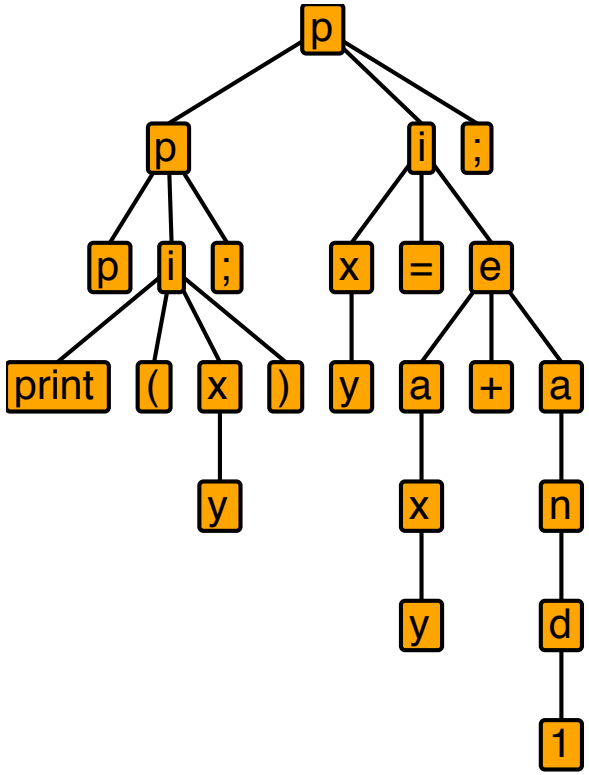
38. $X \to z$

39. $A \to X$
40. $A \to N$
41. $E \to A$
42. $E \to A + A$
43. $E \to A - A$
44. $E \to A / A$
45. $E \to A \times A$

46. $I \to \texttt{print}(X)$
47. $I \to \texttt{input}(X)$
48. $I \to X = E$
49. $I \to \texttt{ifz}\ X\ \texttt{then}\ X = E$
50. $I \to \texttt{ifdz}\ X\ \texttt{then}\ X = E$
51. $P \to \epsilon$
52. $P \to P\ I;$

Parse Tree

$\texttt{print}(y); y = y + 1;$



Syntax: Context-Free Grammars

Definition

A string of characters s admits a parse tree T, if s is the result of the left-to-right traversal of T

Definition (Ambiguity)

A grammar G is **ambiguous** if there exists at least one string of $\mathcal{L}(G)$ which admits more than one derivation tree

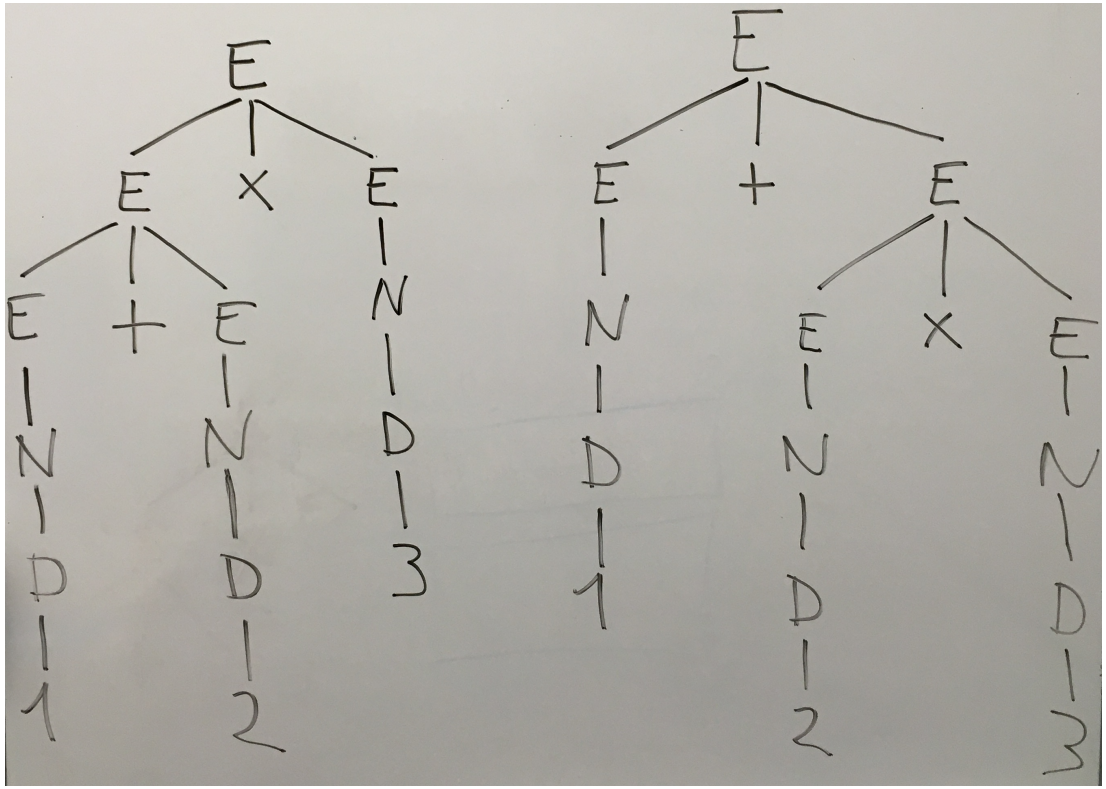
Syntax: Context-Free Grammars

Example

- $G = (\{ E, N, D \}, \{ 0, \ldots, 9, +, \times, \}, R, E)$

with R the following sets of productions:

- $D \to 0$
- ...
- $D \to 9$
- $N \to D$
- $N \to ND$
- $E \to N$
- $E \to E + E$
- $E \to E \times E$



- $1 + 2 \times 3$ admits two parse trees

Syntax: Context-Free Grammars

Exercise

- $G = (\{ B,\ E,\ S \}, \{ a, b, c, +, \neg, \vee, \wedge \},\ R,\ E)$
with R the following sets of productions:
  - $B \to a$
  - $B \to b$
  - $B \to c$
  - $S \to \vee$
  - $E \to B$
  - $E \to E\ S\ E$
  - $E \to \neg E$
  - $S \to \wedge$
- Find, at least, two parse trees for $a \vee \neg b \wedge c \vee b$.

Syntax: Context-Free Grammars

Backus-Naur Form (BNF)

- Usually $\rightarrow$ is replace by $::=$
- Usually non-terminal symbols are written between angle brackets
- Production with the same head are grouped together and separated using a vertical bar

There are variants of these conventions

Syntax: Context-Free Grammars

Backus-Naur Form: An Example

1. $\langle D \rangle ::= 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid\mid 9$
2. $\langle N \rangle ::= \langle D \rangle \mid \langle N \rangle \langle D \rangle$
3. $\langle X \rangle ::= a \mid b \mid c \mid d \mid e \mid f \mid g \mid h \mid i \mid j \mid k \mid l \mid m \mid n \mid o \mid p \mid q \mid r \mid s \mid t \mid u \mid v \mid w \mid x \mid y \mid z$
4. $\langle A \rangle ::= \langle X \rangle \mid \langle N \rangle$
5. $\langle E \rangle ::= \langle A \rangle \mid \langle A \rangle + \langle A \rangle \mid \langle A \rangle - \langle A \rangle \mid \langle A \rangle / \langle A \rangle \mid \langle A \rangle \times \langle A \rangle$
6. $\langle I \rangle ::= \texttt{print}(\langle X \rangle) \mid \texttt{input}(\langle X \rangle) \mid \langle X \rangle = \langle E \rangle \mid \texttt{ifz}\ \langle X \rangle\ \texttt{then}\ \langle X \rangle = \langle E \rangle$
7. $\langle P \rangle ::= \mid \langle I \rangle ; \langle P \rangle$

Syntax: Context-Free Grammars

Extended Backus-Naur Form (BNF)

BNF extended with regular expression notations

- Many variants
- Usually includes:
  - Grouping
  - Repetitions

Example

A program is a, possibly empty, sequence of instructions:

$$\langle P \rangle ::= \{ \langle I \rangle ; \}^*$$

How to Define the Syntax of a Programming Language?

BNF and Extended BNF

- There are many variants of these notations
- Extended BNF = BNF + grouping + repetitions

More Examples

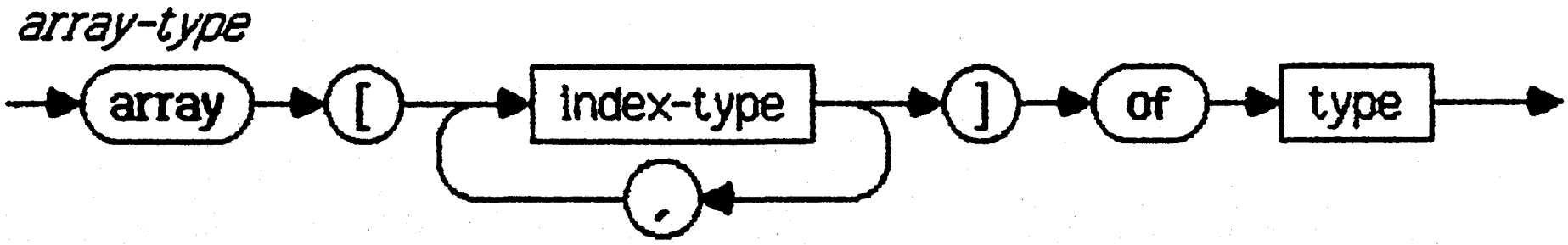
Java Reference Manual

- Terminal symbols are shown in `fixed` width font in the productions of the lexical and syntactic grammars [...]
- Nonterminal symbols are shown in *italic* type [...]
- The syntax $\{x\}$ on the right-hand side of a production denotes zero or more occurrences of x [...]
- The syntax $[x]$ on the right-hand side of a production denotes zero or one occurrences of x. That is, x is an optional symbol [...]

EnumDeclaration:
ClassModifier `enum` *TypeIdentifier [Superinterfaces] EnumBody*
EnumBody:
[EnumConstantList] [,] [EnumBodyDeclarations]
EnumConstantList:
EnumConstant , *EnumConstant*
EnumConstant:
EnumConstantModifier Identifier [([ArgumentList])] [ClassBody]
EnumConstantModifier:
Annotation

More Examples

Pascal



Outline

Syntactic Analysis

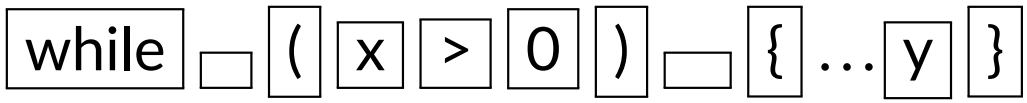
Two Phases

- Lexical analysis: source program $\longrightarrow$ sequence of words/tokens
- Parsing: sequence of tokens $\longrightarrow$ phrase structure (tree)

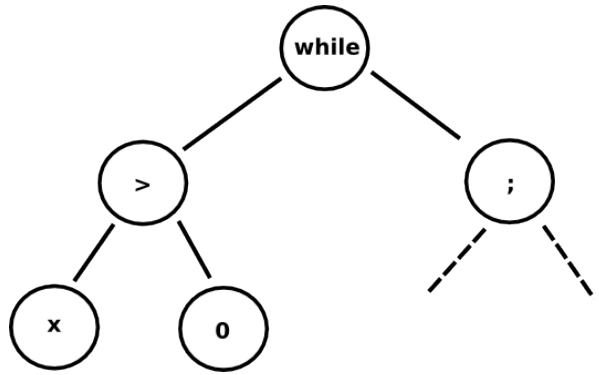
Example

```
while (x > 0) {
    x := x - 1;
    y := y + 1
}
```

$\Downarrow$ *lexical analysis*



$\Downarrow$ *parsing*



Lexical Analysis

Overview

- Tokens as regular expressions
- Regular expressions $\rightarrow$ automata (RE $\rightarrow$ NFA $\rightarrow$ DFA)
- Implementation of automata

References

- Automata Theory / “Théorie des langages”
- Appel Chapter 2

Tools

- JavaCC
- SableCC
- Lex
- OCamlLex
- …

Parsing

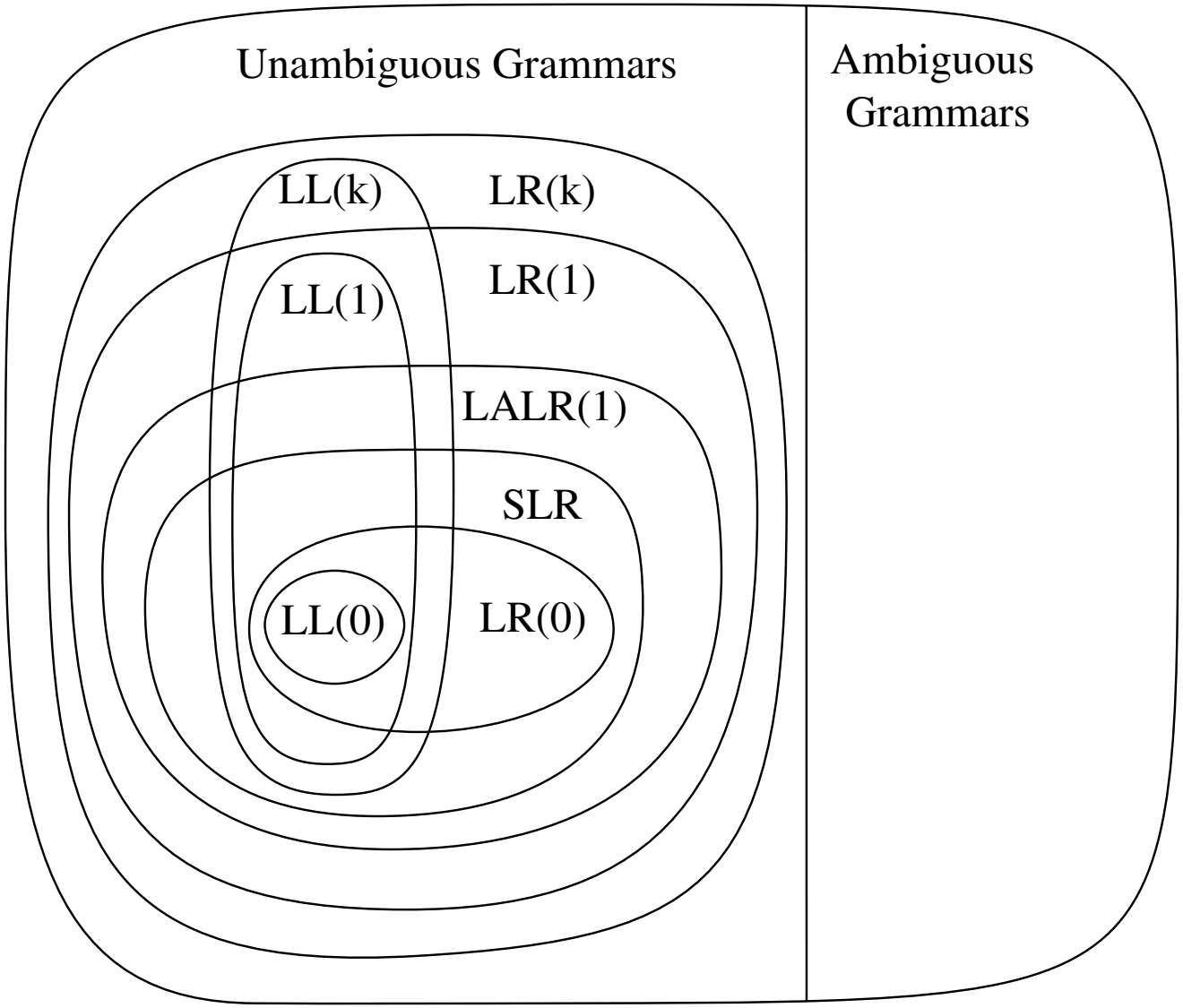
Expressivity of Formalisms

- A programming language cannot (in general) be described only by a regular expression
- More expressive formalism: context free grammars
- Notation: Backus Naur Form (BNF)

Algorithms for Parsing (Appel Chapter 3)

- LL(k): leftmost-derivation, k symbols lookahead
  - recursive-descent (predictive)
  - predictive parsing tables
- LR(k): rightmost-derivation, k token lookahead
  - stack + k tokens
  - 2 operations:
    - shift: input token $\rightarrow$ stack)
    - reduce: grammar rule applied to the stack
  - shift or reduce: automaton (parsing table)
- LALR(1): optimization of the parsing table

Parsing: Hierarchy of Grammar Classes



Appel page 66

Parsing: Parser Generators

Principle

- Input: grammar in BNF-like notation
- Output: programs for parsing
- Rely on lexers

Tools

- JavaCC: LL(k)
- SableCC: LALR(1)
- Yacc: LALR(1)
- Menhir: LR(1)
- ANTLR: ALL(*) (paper)

Outline

Lexical Analysis with ANTLR

Lexer Grammar

- Set of “lexer rules”
- Rule format:
  Non-terminal (uppercase) : regular expression

Example (file: `Example.g4`)

```
lexer grammar Example;

ID : [a-zA-Z]+ ;        // identifiers

INT: [0-9]+   ;         // integers

NEWLINE : '\r'? '\n' ; // line return

WS : [ \t] -> skip;     // skip spaces
```

Keywords and Identifiers

Non-disjoint Lexical Rules

- Assume `var` is a reserved word of the language
- Lexer grammar:

```
lexer grammar Example;
VAR : 'var' ;           // keyword var
ID  : [a-zA-Z]+ ;       // identifiers
```

- Priority: order of rules (`VAR` has priority over `ID`)
- `var` recognized as the `VAR` token
- `variable` recognized as an `ID` token

Parsing with ANTLR

Syntactic Rules

- BNF-like notation
- Rule format:
  Non-terminal (lowercase) `:` sequence of terminal/non-terminal
- Alternative: `|`
- End of rule: `;`
- Special terminal: `EOF` (end of file)

Example

```
grammar Pico;

// Grammar Rules:
prog: (instruction ';')* EOF                          #Program
    ;

instruction : 'VAR' Identifier ':=' expression  #Assign
    | 'WRITE' '(' expression ')'                      #Write
    ;

expression : expression op=('*'|'/') expression  #MulDiv
    | expression op=('+'|'-') expression          #AddSub
    | Identifier                                  #Idendifier
    | Constant                                    #Constant
    | '(' expression ')'                          #Parenthesis
    ;

// Lexer Rules:
Identifier : [a-z]+;

Constant : [0-9]+;

WS : [ \t\r\n]+ -> skip;
```

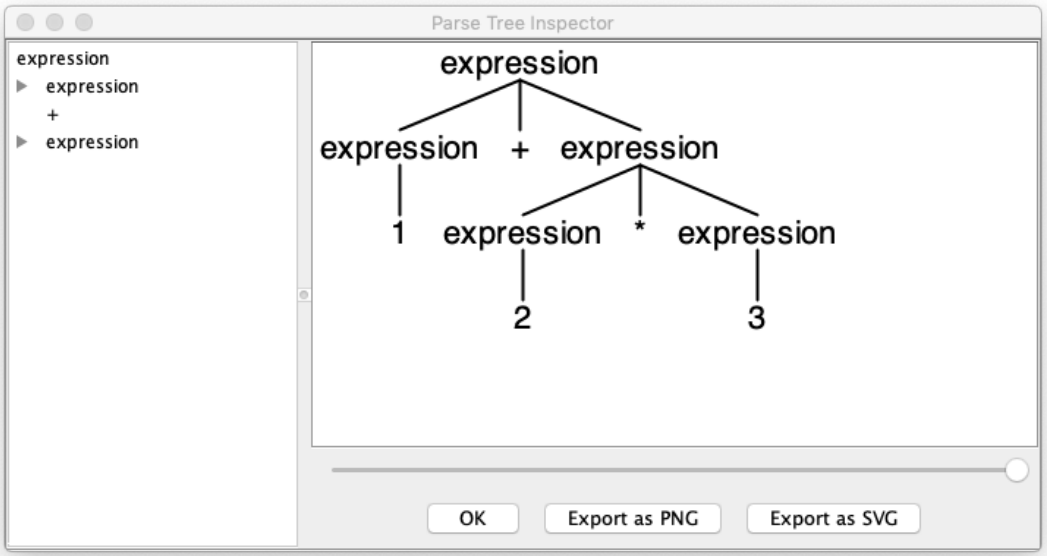
Usage

Code Generation

- `antlr4` *grammar file*
- `javac *.java`
- Test: `grun` *Grammar-name non-terminal* `-gui`

Example

```
grun Pico expression -gui
1+2*3
Control+D
```



Grammar Rule

A Grammar Rule in Details

```
expression : expression op=('*'|'/') expression #MulDiv
```

- ANTLR produces Java (or other language) code to represent **parse trees**
- Each node is an object of the class `Context`
- Code easier to use when each rule is labeled:
  each particular context will have its own "context" class
  Example: the label is `#MulDiv`
- Non-terminal symbols are translated as **access** methods
  Example:

  ```
  ExpressionContext expression(int i)
  ```

- Annotation `op=` allows to add a field `op` to the class `MulDivContext` that allows to get the recognized token (of type `Token`)