Theoretical Computer Science

Lab Session 12

April 22, 2021

Agenda

- Generative Grammars
- Chomsky Hierarchy.
- Context-Free Grammars: Backus-Naur Form

Models

- **Automata** (operational models):
  models suitable to recognize/accept, translate, compute language: receive an input string and process it.

- **Grammars** (generative models):
  Models suitable to describe how to generate a language: set of rules to build phrases of a language.

Grammars

A grammar is a set of rules to produce strings

Grammar: definition

A grammar is a tuple

$$\langle V_N, V_T, P, S \rangle$$

where

- V_N is the non-terminal alphabet;
- V_T is the terminal alphabet;

- Terminal symbols are elementary symbols - cannot be broken down into smaller units i.e. cannot be changed using the production rules of the grammar.

- Non-terminal symbols - can be replaced by groups of terminal and non-terminal symbols according to the production rules.

Grammar: definition

A grammar is a tuple

$$\langle V_N, V_T, P, S \rangle$$

where

- V_N is the non-terminal alphabet;
- V_T is the terminal alphabet;
- $V = V_N \cup V_T$ the alphabet;
- $P \subseteq (V^* \cdot V_N \cdot V^*) \times V^*$ is the (finite) set of rewriting rules of production;
- $S \in V_N$ is a particular element called axiom or initial symbol.

A grammar $\langle V_N, V_T, P, S \rangle$ generates a language on V_T.

Production Rule

Let $G = \langle V_N, V_T, P, S \rangle$ be a grammar.

A **production rule** $\alpha \to \beta$ is an element of P where

- $\alpha \in V^* \cdot V_N \cdot V^*$ is a sequence of symbols including at least one non-terminal symbol;
- $\beta \in V^*$ is a (potentially empty) sequence of (terminal or non-terminal) symbols.

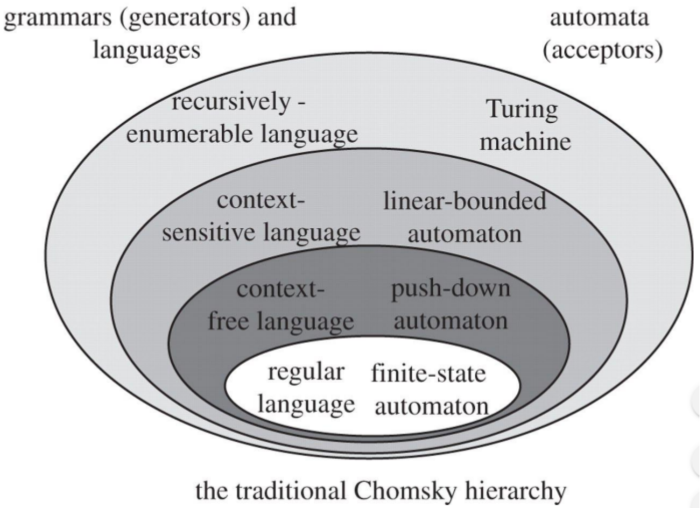
$V = V_N \cup V_T$

Chomsky Hierarchy

- Grammars are classified according to the form of their productions.
- Chomsky classified grammars in four types

  (type 3) Regular grammars
  (type 2) Context-Free grammars
  (type 1) Context-Sensitive grammars
  (type 0) Unrestricted grammars

Chomsky Hierarchy



Grammars, languages and automata

Chomsky hierarchy	Grammars	Languages	Minimal automaton
Type-0	Unrestricted	Recursively enumerable	Turing machine
Type-1	Context-sensitive	Context-sensitive	LBA
Type-2	Context-free	Context-free	NDPDA
Type-3	Regular	Regular	FSA

Strictly Regular grammars (type 3)

production rules restricted to a single non-terminal on the left-hand side and a right-hand side consisting of a single terminal, possibly

1. followed by a single non-terminal - right grammar
2. preceded by a single non-terminal - left grammar

but **NOT** both in the same grammar

Strictly Regular grammars (type 3)

production rules restricted to a single non-terminal on the left-hand side and a right-hand side consisting of a single terminal, possibly

1. followed by a single non-terminal - right grammar
2. preceded by a single non-terminal - left grammar

but **NOT** both in the same grammar

Example

Generate language with the strings of alternating a's and b's
$V_N = \{S, A, B\}; \;\; V_T = \Sigma_1 = \{\mathrm{a}, \mathrm{b}\}$

Set of Production rules P:

1. $S \rightarrow A$
2. $S \rightarrow B$
3. $A \rightarrow \mathrm{a}B$
4. $A \rightarrow \epsilon$
5. $B \rightarrow \mathrm{b}A$
6. $B \rightarrow \epsilon$

Strictly Regular grammars (type 3)

Strictly Right regular grammar

A right regular grammar is a formal grammar $\langle V_N, V_T, P, S\rangle$ such that all the production rules in P are of one of the following forms:

1. $A \rightarrow b$, where $A \in V_N$ and $b \in V_T$;
2. $A \rightarrow bB$, where $A, B \in V_N$ and $b \in V_T$;
3. $A \rightarrow \epsilon$, where $A \in V_N$ and ϵ denotes the empty string.

Strictly Left regular grammar

A left regular grammar is a formal grammar $\langle V_N, V_T, P, S\rangle$ such that all the production rules in P are of one of the following forms:

1. $A \rightarrow b$, where $A \in V_N$ and $b \in V_T$;
2. $A \rightarrow B$b, where $A, B \in V_N$ and $b \in V_T$;
3. $A \rightarrow \epsilon$, where $A \in V_N$ and ϵ denotes the empty string.

Extended regular grammars

Extended Right regular grammar

A left regular grammar is a formal grammar $\langle V_N, V_T, P, S\rangle$ such that all the production rules in P are of one of the following forms:

1. $A \rightarrow b$, where $A \in V_N$ and $b \in V_T$;
2. $A \rightarrow wB$, where $A, B \in V_N$ and $w \in V_T^*$;
3. $A \rightarrow \epsilon$, where $A \in V_N$ and ϵ denotes the empty string.

Extended Left regular grammar

A left regular grammar is a formal grammar $\langle V_N, V_T, P, S\rangle$ such that all the production rules in P are of one of the following forms:

1. $A \rightarrow b$, where $A \in V_N$ and $b \in V_T$;
2. $A \rightarrow Bw$, where $A, B \in V_N$ and $w \in V_T^*$;
3. $A \rightarrow \epsilon$, where $A \in V_N$ and ϵ denotes the empty string.

Exercises

Define Strictly Regular grammars that produce the following languages over the alphabet $\Sigma_1 = \{a, b\}$, $\Sigma_2 = \{0, 1\}$

1. $L_1 = \{0, 1\}^*$
2. $L_2 = \{(aab \mid bba)^*\}$

   Homework:
3. $L_3 = \{(aa \mid bb)^* aa\}$
4. $L_4 = \{(00^* 11^*)\}$

Solutions L_1

$L_1 = \{0, 1\}^*$

$V_N = \{S\}; \;\; V_T = \Sigma_2 = \{0,1\}$

Set of Production rules P:

1. S $\rightarrow \epsilon$
2. $S \rightarrow 0S$
3. $S \rightarrow 1S$

Solutions L_2

$L_2 = \{(aab \mid bba)^*\}$

$V_N = \{S, A, B, F, E\}$; $V_T = \Sigma_1 = \{\text{a, b}\}$

Set of Production rules P:

1. S $\rightarrow \epsilon$
2. $S \rightarrow aA$
3. $S \rightarrow bB$
4. $A \rightarrow aF$
5. $F \rightarrow bS$
6. $B \rightarrow bE$
7. $E \rightarrow aS$

Solutions

$L_3 = \{(aa \mid bb)^* aa\}$

$V_N = \{S, A, B, C\}; \;\; V_T = \Sigma_1 = \{\text{a, b}\}$

Set of Production rules P:

1. $S \rightarrow bC \mid aA$
2. $A \rightarrow aB \mid a$
3. $B \rightarrow bC \mid aA$
4. $C \rightarrow bS$

Context-Free grammars (type 2)

Defined by rules of the form $A \rightarrow \gamma$ where A is a non-terminal and γ is a string of terminals and non-terminals.

Example

Generate language with the $a^n b^n$ where $n > 0$

$V_N = \{S\}$; $V_T = \Sigma_1 = \{\text{a, b}\}$

Set of Production rules P $= \{S \rightarrow aSb \mid ab\}$

Exercises

Define context-free grammars that produce the following languages over the alphabet $\Sigma = \{a, b\}$:

1. Language of palindromes strings
   $L_1 = \{w \in \{a, b\}^* | w = w^R\}$
2. $L_2 = \{a^i b^j c^k |$ i, j, k$\geq$ 0 and i=j or i=k$\}$

   Homework:

3. L_3 Generate language with alternating a's and b's
4. $L_4 = \{a^n b^n c^m \mid n, m > 0\} \cup \{a^n b^m c^m \mid n, m > 0\}$

Solutions L_1

Language of palindromes strings;
$L_1 = \{w \in \{a, b\}^* | w = w^R\}$

$V_N = \{S, O, E\}$; $V_T = \Sigma = \{\text{a, b}\}$

Set of Production rules P:

1. $S \rightarrow O \mid E$
2. $E \rightarrow \epsilon \mid aEa \mid bEb$
3. $O \rightarrow a \mid b \mid aOa \mid bOb$

Solutions L_2

L_2=$\{a^i b^j c^k|$ i, j, k$\geq$ 0 and i=j or i=k$\}$

$V_N = \{S, X, Y, W, Z\}$; $V_T = \Sigma = \{$a, b$\}$

Set of Production rules P:

1. $S \rightarrow XY \mid W$
2. $X \rightarrow aXb \mid \epsilon$
3. $Y \rightarrow cY \mid \epsilon$
4. $W \rightarrow aWc \mid Z$
5. $Z \rightarrow bZ \mid \epsilon$

Context-Sensitive grammars (type 1)

The rules of the form $\alpha A \beta \rightarrow \alpha \gamma \beta$, where A is a non-terminal and α, β and γ are strings of terminals and non-terminals.

1. γ must be non-empty
2. The rule S$\rightarrow \epsilon$ is allowed if S does not appear on the right side of any rule

Context-Sensitive grammars (type 1)

The rules of the form $\alpha A\beta \rightarrow \alpha\gamma\beta$, where A is a non-terminal and α, β and γ are strings of terminals and non-terminals.

1. γ must be non-empty
2. The rule S$\rightarrow\epsilon$ is allowed if S does not appear on the right side of any rule

Example

Generate language $\{A^n B^n C^n | n > 0\}$

1. $S \rightarrow \mathrm{a}BC$
2. $S \rightarrow \mathrm{a}SBC$
3. $CB \rightarrow CZ$
4. $CZ \rightarrow BZ$
5. $BZ \rightarrow BC$
6. $\mathrm{a}B \rightarrow \mathrm{ab}$
7. $\mathrm{b}B \rightarrow \mathrm{bb}$
8. $\mathrm{b}C \rightarrow \mathrm{bc}$
9. $\mathrm{c}C \rightarrow \mathrm{cc}$

Exercises

Define context-sensitive grammars that produce the following languages:

1. $L_1 = \{WW \mid W \in \{a, b\}^*\}$
   Homework:
2. $L_2 = \{a^i b^j c^i d^j \mid i, j \geq 1\}$

3. $L_3 = \{W \in \{a, b, c\}^* \mid \#(a) = \#(b) = \#(c)$ and $\#(a) \geq 1\}$

Solutions L_2

$L_2 = \{a^i b^j c^i d^j \mid i, j \geq 1\}$

$V_N = \{S, A, B, C\}; \;\; V_T = \Sigma = \{\text{a, b, c}\}$

Set of Production rules P:

1. $S \rightarrow AB$
2. $A \rightarrow aAX \mid aX$
3. $B \rightarrow bBd \mid bYd$
4. $Xb \rightarrow bX$
5. $XY \rightarrow Yc$
6. $Y \rightarrow \epsilon$

Solutions L_3

$L_3 = \{W \in \{a, b, c\}^* |\ \#(a) = \#(b) = \#(c) \text{ and } \#(a) \geq 1\}$

$V_N = \{S, A, B, C\}$; $V_T = \Sigma = \{\text{a, b, c}\}$

Set of Production rules P:

1. $S \to ABC \ | \ ABCS$
2. $AB \to BA$
3. $AC \to CA$
4. $BC \to CB$
5. $BA \to AB$
6. $CA \to AC$
7. $CB \to BC$
8. $A \to a$
9. $B \to b$
10. $C \to c$

Unrestricted grammars (type 0)

The rules of the form $\alpha \to \beta$, where α and β are strings of non-terminals and terminals.

1. The grammars without any limitation on production rules.
2. α at least have one non-terminal
3. α cannot be an empty string

Unrestricted grammars (type 0)

The rules of the form $\alpha \rightarrow \beta$, where α and β are strings of non-terminals and terminals.

Example

Generate language $\{A^n B^n C^n | n > 0\}$

1. $S \rightarrow \mathrm{a}BC$
2. $S \rightarrow \mathrm{a}SBC$
3. $CB \rightarrow BC$
4. $\mathrm{a}B \rightarrow \mathrm{ab}$
5. $\mathrm{b}B \rightarrow \mathrm{bb}$
6. $\mathrm{b}C \rightarrow \mathrm{bc}$
7. $\mathrm{c}C \rightarrow \mathrm{cc}$

Exercises

Generate Unrestricted grammars for below languages:

1. $L_1 = \{W \mid W = a^i \text{ and } i = 2^k \text{ and } k > 0\}$
2. $L_2 = \{a^n b^m c^n d^m \mid n > 0, m > 0\}$

   Homework:

3. $L_3 = \{\, a^n b^{2n} c^{3n} \mid n \geq 1\}$

Solutions1 L_1

L_2=$\{W : W = a^i$ and $i = 2^k$ and $k > 0\}$

$V_N = \{S, A, B, C\}$; $V_T = \Sigma = \{$a, b, c$\}$

Set of Production rules P:

1. $S \rightarrow LAYR$
2. $ZA \rightarrow aAZ$
3. $Za \rightarrow aZ$
4. $ZR \rightarrow AAYR$
5. $aY \rightarrow Ya$
6. $AY \rightarrow YA$
7. $LY \rightarrow LZ$
8. $YR \rightarrow X$
9. $aX \rightarrow Xa$
10. $AX \rightarrow Xa$
11. $LX \rightarrow \epsilon$

Solutions L_2

L_2=$\{a^n b^m c^n d^m \mid n > 0, m > 0\}$

$V_N = \{S, A, B, C, X, Y\}$; $V_T = \Sigma = \{$a, b, c$\}$

Set of Production rules P:

1. $S \rightarrow XY$
2. $X \rightarrow aXC \mid aC$
3. $Y \rightarrow BYd \mid Bd$
4. $CB \rightarrow BC$
5. $aB \rightarrow ab$
6. $bB \rightarrow bb$
7. $Cd \rightarrow cd$
8. $Cc \rightarrow cc$

Context-Free grammars (type 2)

Defined by rules of the form $A \to \gamma$ where A is a non-terminal and γ is a string of terminals and non-terminals.

Backus Naur Form (BNF)

BNF (Backus Normal Form or Backus–Naur Form) is a notation technique for context-free grammars.

It is often used to describe the syntax of programming languages.

The BNF Notation

- Non-terminals are words in $\langle \ldots \rangle$
  Example: <*statement*>, and represents non-terminal symbols.
- Terminal symbols are grammar symbols enclosed in quotes (‘’) and often multicharacter strings indicated by single quotation marks. Example: ‘while’.
- Symbol $::=$ is often used for $\rightarrow$ (from the production rules).
- Symbol $|$ is used as a shorthand for a list of productions with the same left side.
  - Example: $\{S \rightarrow 0S1 \mid 01\}$ is shorthand for $\{S \rightarrow 0S1, S \rightarrow 01\}$.
- Symbol $[\ldots]$ is used to represent optional.
  - Example: a [b] can produce: *ab* or *a*.
- Symbol $\{\ldots\}$ is used to represent zero or more times.
  - Example: a{b} can produce: *ab* or *a* or *abbb*.

BNF: Example

$$\langle S \rangle ::= \langle X \rangle \text{`a'}\text{`a'} \langle X \rangle$$
$$\langle X \rangle ::= \text{`a'} \langle X \rangle \mid \text{`b'} \langle X \rangle \mid \epsilon$$

BNF: Example

$$\langle S \rangle ::= \langle X \rangle \text{`a'`a'} \langle X \rangle$$
$$\langle X \rangle ::= \text{`a'} \langle X \rangle \mid \text{`b'} \langle X \rangle \mid \epsilon$$

Can the string *bbaab* be produced by the Grammar?

BNF: Example

$$\langle S \rangle ::= \langle X \rangle \text{`a'`a'} \langle X \rangle$$
$$\langle X \rangle ::= \text{`a'} \langle X \rangle \mid \text{`b'} \langle X \rangle \mid \epsilon$$

Can the string *bbaab* be produced by the Grammar? We start by the initial symbol $\langle S \rangle$ and then replacing

$$\langle S \rangle = \langle X \rangle \text{`a'`a'} \langle X \rangle$$

BNF: Example

$$\langle S \rangle ::= \langle X \rangle \text{`a'`a'} \langle X \rangle$$
$$\langle X \rangle ::= \text{`a'} \langle X \rangle \mid \text{`b'} \langle X \rangle \mid \epsilon$$

Can the string *bbaab* be produced by the Grammar? We start by the initial symbol $\langle S \rangle$ and then replacing

$$\langle S \rangle = \underline{\langle X \rangle}\text{`a'`a'}\langle X \rangle$$

BNF: Example

$$\langle S \rangle ::= \langle X \rangle \text{'a''a'} \langle X \rangle$$
$$\langle X \rangle ::= \text{'a'} \langle X \rangle \mid \text{'b'} \langle X \rangle \mid \epsilon$$

Can the string *bbaab* be produced by the Grammar? We start by the initial symbol $\langle S \rangle$ and then replacing

$$\langle S \rangle = \text{'b'} \langle X \rangle \text{'a''a'} \langle X \rangle$$

BNF: Example

$$\langle S \rangle ::= \langle X \rangle \text{‘a’‘a’} \langle X \rangle$$
$$\langle X \rangle ::= \text{‘a’} \langle X \rangle \mid \text{‘b’} \langle X \rangle \mid \epsilon$$

Can the string *bbaab* be produced by the Grammar? We start by the initial symbol $\langle S \rangle$ and then replacing

$$\langle S \rangle = \text{‘b’} \underline{\langle X \rangle} \text{‘a’‘a’} \langle X \rangle$$

BNF: Example

$$\langle S \rangle ::= \langle X \rangle \text{`a'`a'} \langle X \rangle$$
$$\langle X \rangle ::= \text{`a'} \langle X \rangle \mid \text{`b'} \langle X \rangle \mid \epsilon$$

Can the string *bbaab* be produced by the Grammar? We start by the initial symbol $\langle S \rangle$ and then replacing

$$\langle S \rangle = \text{`b'`b'} \langle X \rangle \text{`a'`a'} \langle X \rangle$$

BNF: Example

$$\langle S \rangle ::= \langle X \rangle \text{`a'`a'} \langle X \rangle$$
$$\langle X \rangle ::= \text{`a'} \langle X \rangle \mid \text{`b'} \langle X \rangle \mid \epsilon$$

Can the string *bbaab* be produced by the Grammar? We start by the initial symbol $\langle S \rangle$ and then replacing

$$\langle S \rangle = \text{`b'`b'} \underline{\langle X \rangle} \text{`a'`a'} \langle X \rangle$$

BNF: Example

$$\langle S \rangle ::= \langle X \rangle \text{'a''a'} \langle X \rangle$$
$$\langle X \rangle ::= \text{'a'} \langle X \rangle \mid \text{'b'} \langle X \rangle \mid \epsilon$$

Can the string *bbaab* be produced by the Grammar? We start by the initial symbol $\langle S \rangle$ and then replacing

$$\langle S \rangle = \text{'b''b''a''a'} \langle X \rangle$$

BNF: Example

$$\langle S \rangle ::= \langle X \rangle \text{`a'`a'} \langle X \rangle$$
$$\langle X \rangle ::= \text{`a'} \langle X \rangle \mid \text{`b'} \langle X \rangle \mid \epsilon$$

Can the string *bbaab* be produced by the Grammar? We start by the initial symbol $\langle S \rangle$ and then replacing

$$\langle S \rangle = \text{`b'`b'`a'`a'} \underline{\langle X \rangle}$$

BNF: Example

$$\langle S \rangle ::= \langle X \rangle \text{`a'`a'} \langle X \rangle$$
$$\langle X \rangle ::= \text{`a'} \langle X \rangle \mid \text{`b'} \langle X \rangle \mid \epsilon$$

Can the string *bbaab* be produced by the Grammar? We start by the initial symbol $\langle S \rangle$ and then replacing

$$\langle S \rangle = \text{`b'`b'`a'`a'`b'} \langle X \rangle$$

BNF: Example

$$\langle S \rangle ::= \langle X \rangle \text{`a'`a'} \langle X \rangle$$
$$\langle X \rangle ::= \text{`a'} \langle X \rangle \mid \text{`b'} \langle X \rangle \mid \epsilon$$

Can the string *bbaab* be produced by the Grammar? We start by the initial symbol $\langle S \rangle$ and then replacing

$$\langle S \rangle = \text{`b'`b'`a'`a'`b'} \underline{\langle X \rangle}$$

BNF: Example

$$\langle S \rangle ::= \langle X \rangle \text{`a'`a'} \langle X \rangle$$
$$\langle X \rangle ::= \text{`a'} \langle X \rangle \mid \text{`b'} \langle X \rangle \mid \epsilon$$

Can the string *bbaab* be produced by the Grammar? We start by the initial symbol $\langle S \rangle$ and then replacing

$$\langle S \rangle = \text{`b'`b'`a'`a'`b'}$$

Exercises

Define BNF grammars for the following languages:

1. A simple list of the form A1,B2,A4,C3.
2. Simple expressions limited to the variable identifiers x, y, and z, that contain the binary operations of addition ($+$) and subtraction (-), and parentheses, e.g. $x + (z - y)$, $(x - x) + (z + y)$

Exercise

Define a BNF grammar for the language of Pascal variable declarations without defining user-defined types. e.g.

$$
\begin{aligned}
&var\ \ i : integer;\\
&var\ \ b : boolean;\\
&var\ \ my\_float : real;\\
&\qquad mychar : char;\\
&\qquad x, y, z : integer;
\end{aligned}
$$

Treat last declaration as a single line.

Solution

Define a BNF grammar for the language of Pascal variable declarations without defining user-defined types:
Grammar

$$
\begin{aligned}
<\!\mathit{decl}\!> &::= \text{`var'}\ <\!\mathit{decllist}\!>\text{`;'} \\
<\!\mathit{decllist}\!> &::= <\!\mathit{varandtype}\!>\{\text{`;'}<\!\mathit{varandtype}\!>\} \\
<\!\mathit{varandtype}\!> &::= <\!\mathit{ident}\!>\{\text{`,'}<\!\mathit{ident}\!>\}\text{`:'}<\!\mathit{type}\!> \\
<\!\mathit{ident}\!> &::= <\!\mathit{letter}\!>\{<\!\mathit{idchar}\!>\} \\
<\!\mathit{idchar}\!> &::= <\!\mathit{letter}\!> \mid <\!\mathit{digit}\!> \mid \text{`\_'} \\
<\!\mathit{type}\!> &::= \text{`integer'} \mid \text{`boolean'} \mid \text{`real'} \mid \text{`char'} \\
<\!\mathit{digit}\!> &::= \text{`0'} \ldots \text{`9'} \\
<\!\mathit{letter}\!> &::= \text{`a'} \ldots \text{`z'} \mid \text{`A'} \ldots \text{`Z'}
\end{aligned}
$$