3 - Context Free Languages and Push Down Automata

1. Context Free Grammars: introduction
2. Context Free Grammars: definitions
3. Context Free Grammars: normalisation
4. Regular Grammars
5. Push Down Automata

3.1 – Introduction

- The **expressivity** of regular languages is too limited for the representation of programming languages. Moreover, regular expressions and automata are not well suited to a readable description of languages.

- A new paradigm was introduced by Chomsky (1956), Schützenberger, Backus (1959) and Naur (1960) : whereas regular languages are built recursively from elementary languages by means of three operations, a **context-free language** is derived from a set of **rewriting rules** which constitute its grammar; these rules must respect some syntactic constraints to be a **Context-Free Grammar** (CFG).

3.2 – Definitions

- A Context-Free Grammar is a 4-uple (N, T, S, R) such that :
  - ✓ N is a finite alphabet of non terminal symbols.
  - ✓ T is a finite alphabet of terminal symbols.
  - ✓ S is a particular element of N, the start symbol.
  - ✓ R is a finite set of production rules in the form $A \rightarrow \alpha$, where A is a non terminal and α is a word from $\rightarrow (N \cup T)^*$
- The **derivation relation** $\Rightarrow$ between words from $(N \cup T)^*$ is defined as follows : $\alpha_1 A \alpha_2 \Rightarrow \alpha_1 \alpha \alpha_2$ if $A \rightarrow \alpha \in R$. Its reflexive and transitive closure is written : $\Rightarrow^*$
- The language generated by the grammar is the set of words α from T^* such that : $S \Rightarrow^* \alpha$. It is a **context-free language**.

3.2 – Definitions

- The grammar G is defined by the following rules and its start symbol is P:

$P \rightarrow P\ ;\ P$

$P \rightarrow V = E$

$NP \rightarrow NP\ \ PP$

$E \rightarrow E + E$

$E \rightarrow V$

$E \rightarrow C$

$V \rightarrow x$

$V \rightarrow y$

$V \rightarrow z$

$C \rightarrow 0$

$C \rightarrow 1$

3.2 – Definitions

- A **derivation** of a word α of the language is a sequence $S \Rightarrow \alpha_1 \Rightarrow \alpha_2 \Rightarrow \ldots \Rightarrow \alpha$.

  It is represented in a compact way by a **parse tree**. A parse tree is an ordered tree labelled with symbols from $N \cup T$. The root is labelled with S and the leaves with terminal symbols. Every node that is not a leaf is labelled with a non terminal symbol A and its ordered daughters are labelled with symbols constituting a word α such that $A \rightarrow \alpha$ is a production rule of the grammar.

- Two grammars are **equivalent** if they generate the same language.

- A grammar is **ambiguous** if there exists a word from its language that corresponds to two different parse trees at least. In natural languages, ambiguity has an important place whereas in programming languages ambiguity is rejected.

3.2 – Definitions

1. From the grammar to the language

Determine the languages generated by the following grammars (S is the start symbol). Are theses grammars ambiguous ? If yes, define a non ambiguous grammar which is equivalent to that one of the question.

a) $S \rightarrow \varepsilon \mid aaaS$

b) $S \rightarrow ab \mid aSb$

c) $S \rightarrow SS \mid \varepsilon \mid (S)$

d) $S \rightarrow \varepsilon \mid a_i S a_i$ *for any i such that* $1 \leq i \leq n$

2. From the language to the grammar

Determine CFGs generating the following languages.

a) $L_1 = \{ a^n b^p \mid 0 < p < n \}$

b) $L_2 = \{ a^n b^n c^m d^m \mid n, m \in \mathbb{N} \}$

c) $L_3 = \{ a^n b^m c^p \mid n = m \text{ or } p = m \}$

d) $L_4 = a(ab^*)^*$

3.2 – Definitions

3. Let G be the grammar : $E \rightarrow E + E \mid E - E \mid (E) \mid id$
E is the start symbol. Give all parse trees of the word *id - id - id.* Determine an equivalent non ambiguous grammar, by introducing a new non terminal respresenting expressions resulting from only substractions.

4. If-then-else grammars

Show that the following grammar is ambiguous (S is the start symbol) :

$S \rightarrow$ *if B then S* | *if B then S else S* | *s*

$B \rightarrow b$

3.3 – Normalisation

- **Theorem (elimination of empty production rules)**: every CFG is equivalent to a CFG in which the only possible empty production rule is $S \rightarrow \varepsilon$, where S is the start symbol of the grammar, and the other production rules do not include S in their right hand side.

- **Theorem (elimination of unary rules)**: every CFG is equivalent to a CFG without unary rules (a unary rule is in the form $A \rightarrow B$).

- **Theorem (elimination of useless non terminal symbols)**: every CFG is equivalent to a CFG in which all non terminal symbols derive a string from T^* and all non terminal symbols are in a derivation coming from the start symbol.

3.3 – Normalisation

- A CFG is **recursive** if it admits derivation relation in the form $A \Rightarrow^{+} \alpha A \beta$. It is **left-recursive** if $\alpha = \varepsilon$ and it is **right-recursive** if $\beta = \varepsilon$.

- A CFG is **immediately recursive** if it admits a production rule in the form $A \rightarrow \alpha A \beta$. It is **immediately left-recursive** if $\alpha = \varepsilon$ and it is **immediately right-recursive** if $\beta = \varepsilon$.

- **Elimination of the immediate left-recursivity :**

Replace the rules in the form $A \rightarrow A\,\alpha_1 \mid \ldots \mid A\,\alpha_m \mid \beta_1 \mid \ldots \mid \beta_p$ *where A is not the first symbol of* $\beta_1 \ldots \beta_p$ *with*

$$A \rightarrow \beta_1 A' \mid \ldots \mid \beta_p A' \mid \beta_1 \mid \ldots \mid \beta_p$$

$$A' \rightarrow \alpha_1 A' \mid \ldots \mid \alpha_m A' \mid \alpha_1 \mid \ldots \mid \alpha_m$$

where A' is a new non terminal

3.3 – Normalisation

- **Algorithm for the elimination of left-recursivity:**
  - *Order all non terminals of the grammar according a total order $A_1 < A_2 < \ldots < A_n$*
  - *While there is a non examined non terminal*
    - *Consider the first non examined non terminal A_i according to the total order*
    - *While there is a rule from the grammar in the form $A_i \rightarrow A_j \alpha$ where $j < i$*

      Remove the rule $A_i \rightarrow A_j \alpha$ from the grammar

      For every rule in the form $A_j \rightarrow \beta$

      Add the rule $A_i \rightarrow \beta \alpha$ to the grammar
    - *If there is a rule from the grammar in the form $A_i \rightarrow A_i \alpha$*

      Eliminate the immediate left recursivity corresponding to A_i

      Add the new non terminal A'_i to the end of the ordered list of non terminals

3.3 – Normalisation

- **Theorem** (elimination of left-recursivity): every CFG without empty production rule is equivalent to a CFG which is not left-recursive (right-recursive).

- **Theorem** (Greibach's normal form): every CFG without empty production rule is equivalent to a CFG in which the production rules have the following form : $A \rightarrow a\alpha$, where A is a non terminal symbol, a is a terminal symbol and α is a (possibly empty) sequence of non terminal symbols.

- **Theorem** (Chomsky's normal form): every CFG without empty production rule is equivalent to a CFG in which the production rules have one of the following forms : $A \rightarrow B\ C$, $A \rightarrow a$, where A, B, C are non terminal symbols and a is a terminal symbol

3.3 – Normalization

1. Clean the following grammar from the empty productions :

   $S \rightarrow S\ (\ S\ )\ |\ \varepsilon$

2. Clean the following grammar from the unary rules, the useless non terminal symbols and the empty productions (S is the start symbol):

   $S \rightarrow X\ |\ f\ X\ |\ Y$

   $X \rightarrow a\ b\ c\ Y\ |\ Y\ |\ \varepsilon$

   $Y \rightarrow a\ T\ |\ d\ Z$

   $Z \rightarrow e\ S\ |\ W\ |\ T\ K\ |\ e\ f$

   $K \rightarrow c\ V$

   $W \rightarrow U\ |\ a\ Y\ |\ b$

   $U \rightarrow b\ d\ X\ |\ Z$

   $T \rightarrow a\ T$

   $V \rightarrow a\ f$

3. Eliminate left recursivity from the following grammar :

   $E \rightarrow E + T\ |\ T$

   $T \rightarrow T \times F\ |\ F$

   $F \rightarrow (\ F\ )\ |\ i$

3.3 – Normalization

4. Eliminate empty production rules and left recursivity from the following grammar (S is the start symbol) :

   $S \rightarrow S\ a \mid T\ S\ c \mid d$

   $T \rightarrow T\ b\ T \mid \varepsilon$

5. Put the following grammar in Chomsky's normal form :

   $S \rightarrow a\ B \mid b\ A$

   $A \rightarrow a \mid a\ S \mid b\ A\ A$

   $B \rightarrow b \mid b\ S \mid a\ B\ B$

6. Put the following grammar in Greibach's normal form :

   $A \rightarrow B\ C$

   $B \rightarrow C\ A \mid b$

   $C \rightarrow A\ B \mid a$

3.4 - Regular Grammars

- **Definition**: A **right linear regular grammar** is a CFG that has production rules in the form : $A \rightarrow \alpha B$ or $A \rightarrow \alpha$, where A and B are a non terminal symbols and α is a (possibly empty) sequence of terminal symbols.

- **Theorem**: A language is regular iff it is generated by a regular grammar.

- The proof of this equivalence is based on the identification between non terminal symbols, terminal symbols, production rules of a CFG on the one hand and states, input symbols, transitions of an automaton on the other hand.

3.5 - Push Down Automata

- A **Push Down Automaton** is a FSA which is associated with a **stack** and the transitions of which depend on the symbol at the top of the stack, which can be modified at the same time.

- **Definition :** a Push Down Automaton is a 7-uple $(Q, \Sigma_i, \Sigma_s, \perp, q0, F, \tau)$ such that :
  - ✓ Q is a finite set of states.
  - ✓ Σ_i is a finite input tape alphabet of symbols.
  - ✓ Σ_s is a finite stack alphabet of symbols.
  - ✓ $\perp$ is a particular element of Σ_s, marking the bottom of the stack.
  - ✓ q_0 is a particular element of Q, the start state of the automaton.
  - ✓ F is a subset of Q, the accepting states of the automaton.
  - ✓ τ is a transition relation which associates a source state q_1 from Q, an input tape symbol *a* from $\Sigma_i \cup \{\varepsilon\}$, a stack top symbol *b* from Σ_s, with a target state q_2 from Q and a push word α of

3.5 - Push Down Automata

- The **principle of a Push Down Automaton** :
  - ✓ **Initialization**: Initially, a word *s* from $\sum_i^*$ is written on the tape. The other positions are filled with the blank symbol $\perp_i$. A pointer indicates the beginning of the tape.

    The control unit is in the state *q0* and the stack is empty (it includes only the bottom symbol $\perp$).
  - ✓ **Transition step**: A **transition** can occur if the control unit is in a state q_1, if the pointer indicates the symbol *a* on the tape, if there is the symbol *b* at the top of the stack and if the PDA has a transition $(q_1, a, b, q2, \alpha)$ in its transition relation. In this case, the pointer of the tape moves to the next position. The symbol *b* at the top of the stack is replaced by the string α. Finally, the control unit moves to state q_2.

    Instead of *a,* if the transition includes the empty word ε, the symbol of the tape indicated by the pointer does not matter and the pointer does not move.
  - ✓ **Termination**: If in a configuration of the PDA, no transition is possible, the PDA stops. At this moment, if the unit control is in an accepting state, if the stack is empty and the input word totally read, this one is said to be **accepted** by the PDA.

3.5 - Push Down Automata

- A **Deterministic Push Down Automaton** (DPDA) is a PDA with the following property : from any state q_1, any input tape symbol *a* and any stack top symbol *b,* there is one possible transition at most in the transition relation of the automaton.
  A PDA that is not a DPDA is a **Non Deterministic Push Down Automaton** (NPDA).

- There are languages that are recognized by NPDA and not by DPDA.

- **Theorem** : a language is context-free iff it is recognized by a PDA.

3.5 - Push Down Automata

1. Let A = $(\{q_0, q_1\}, \{0, 1\}, \{\perp, X\}, \perp, q_0, \{q_0, q_1\}, \tau)$ be an automaton such that $\tau = \{ (q_0, 1, \perp, q_0, \perp X), (q_0, 1, X, q_0, X X), (q_0, 0, X, q_1, X), (q_1, 1, X , q_1, \varepsilon) , (q_1, 0, \perp, q_0, \perp,) \}$

   a) Perform the computations on the following input words: 11011 and 10101101

   b) Describe the language recognized by this PDA and give a grammar that generates this language.

2. Build PDA recognizing the following languages (DPDA if it is possible)

   a) $\{a^n b^n \mid n \geq 1\}$

   b) $\{a^n b^n \mid n \geq 0\}$

   c) $\{a^p b^{p+q} c^q \mid p \geq 0, q \geq 1\}$

   d) The language over the alphabet $\{a, b\}$ that contains as many *a* as *b*.

   e) The language of palindromes.