# 3 - Context Free Grammars

# 3.1 – Introduction

- The **expressivity** of regular languages is too limited for the representation of natural languages (centre-embedded recursion).

- Moreover, regular expressions and automata are not well suited to the description of properties of natural languages (constituency).

- 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 build 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 $\alpha$ is a string 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 $\alpha$ 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 S:

| | |
|---|---|
| S → NP Vintr | NP → jean |
| S → NP Vtr NP | Det → le |
| S → NP Vc Compl | N → bébé |
| S → S PP | N → berceau |
| NP → Det N | Vintr → dort |
| NP → NP PP | Vtr → porte |
| Compl → Conj S | Vc → pense |
| PP → Prep NP | Prep → dans |
| | Conj → que |

# 3.2 – Definitions

- A **derivation** of a string $\alpha$ 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 string $\alpha$ 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. The start symbol is S. If the grammars are ambiguous, show it with an example.

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

   b) $S \rightarrow \varepsilon \mid aA \mid Bb \qquad A \rightarrow Sb \qquad B \rightarrow aS$

   c) $S \rightarrow S\ 0\ S\ 0\ S \mid 1$

   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.

   *Hints : for the three first cases, start from grammars generating the language* $\{a^n\ b^n\}$

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

   1\. $L_4 = a(ab^*)^*$

# 3.2 – Definitions

## 3. Ambiguous grammars

Show that the following grammars are ambiguous. Their start symbol is *S*.

- $S \rightarrow if\ B\ then\ S \mid if\ B\ then\ S\ else\ S \mid s$
  $B \rightarrow b$

- $S \rightarrow S\ PP \mid NP\ VP$
  $PP \rightarrow with\ NP$

  $VP \rightarrow V\ NP$

  $V \rightarrow meets$

  $NP \rightarrow NP\ PP \mid mary \mid john \mid peter$

# 3.3 - 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 $\alpha$ 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.

# 3.4 - Expressivity and the Chomsky hierarchy

- By relaxing the form of the rules in a CFG, we obtain more expressive classes of grammars.

- It is possible to construct a hierarchy of four classes, numbered from 0 to 3: the **Chomsky Hierarchy**.

- Every class of the hierarchy strictly includes the classes with a greater number : the expressivity  decreases with the numbering.

- At the same time, the complexity of the machines dedicated to  language recognition for every class decreases too with the numbering.

# 3.4 - Expressivity and the Chomsky hierarchy

| Type | Name | Rule Skeleton | Recognition complexity |
| --- | --- | --- | --- |
| 0 | Turing equivalent | $\alpha \rightarrow \gamma$ such that : $\alpha \neq \varepsilon$ | = Languages recursively enumerable by Turing machines |
| 1 | Context Sensitive | $\alpha A \beta \rightarrow \alpha \gamma \beta$ such that : $\gamma \neq \varepsilon$ | $\subset$ Recursive languages recognized by Turing machines |
| 2 | Context Free | $A \rightarrow \gamma$ | = Languages recognized by Push Down Automata |
| 3 | Regular | $A \rightarrow \gamma B$ or $A \rightarrow \gamma$ with $\gamma \in T^*$ | = Languages recognized by Finite State Automata |

# 3.4 - Expressivity and the Chomsky hierarchy

- The class of **Regular Languages** is not expressive enough for representing some natural languages.

- The class of **Context Free Languages** is not expressive enough for representing some natural languages (the syntax of Swiss-German, the morphology of Bambara).

- The adequacy of linguistic formalisms should not be only evaluated by considering the generated languages but also by examining their ability to express linguistic generalities, especially linguistic structures.

# 3.5 – Parsing with tabulation: introduction

- Parsing a sentence with a CFG consists in building all its derivations trees for this CFG.
- There are two fundamental methods of parsing corresponding to two directions of construction for the derivation tree: from the root to the leaves (**top-down**) or from the leaves to the root (**bottom-up**).
- The weakness of the top-down method: a lot of derivation trees are generated before examining the agreement with the input sentence; most of them do not agree.
- The weakness of the bottom-up method: even if the number of generated derivation trees is more restricted, because guided by the input sentence, some partial trees are useless because they cannot enter a tree rooted at the sentence category.

# 3.5 – Parsing with tabulation: introduction

- Since natural languages are highly **ambiguous**, sentences generally have several derivation trees.

- To deal with ambiguity, parsing algorithms resort to a specific form of **dynamic programming** called **tabulation**: intermediate results are stored in a table so that they can be re-used if necessary.

# 3.5 – Parsing with tabulation: the CKY algorithm

- The CKY algorithm is a **bottom-up** algorithm : it builds partial derivation trees from their leaves.

- The CKY algorithm uses **tabulation**. Intermediate parsing states are stored in a **chart** composed of items.

  If the sentence to parse is $w_1$ $w_2$ … $w_n$ , any item has the form <w, i, i+1> or < A , i , j> with the following meaning:

  - in the first case, the word w is the word $w_{i+1}$ of the sentence;
  - in the second case, there is a derivation tree with A labelling the root and the words $w_{i+1}$ … $w_j$ is of the sentence labelling its leaves.

# 3.5 – Parsing with tabulation: the CKY algorithm

- The CKY algorithm consists in filling the chart with items by application of the following derivation rules :

$$\frac{}{< w_{i+1}\ , i, i+1 >}\ \text{Init}$$

$$\frac{< \alpha_1, i_1, i_2 > < \alpha_2, i_2, i_3 > \ldots < \alpha_p, i_p, i_{p+1} >}{< A, i_1, i_{p+1} >}\ \text{Complete with } A \longrightarrow \alpha_1 \ldots \alpha_p \in G$$

- The process of parsing ends when no new item can be produced and it succeeds if the item < S, 0, n > is present in the chart.

- Different orders of rule application define different strategies of parsing. Relevant strategies are those which are complete: any derivable item is produced by application of the strategy.

# 3.5 – Parsing with tabulation: the CKY algorithm

- The CKY algorithm in the previous form is a **recognition** algorithm. To transform it into a **parsing** algorithm, every item must be augmented with a list of pointers to items that have contributed to its completion.

- The worst case running time of CKY is $O(n^3.|G|)$ , where *n* is the length of the parsed string and $|G|$ is the size of the grammar *G*. For this, G must be rendered into **Chomsky normal form** : all production rules have the form $A \longrightarrow B\ C$ or $A \longrightarrow a$.

# 3.5 – Parsing with tabulation: the Earley algorithm

- The Earley algorithm is a mixed algorithm : it makes **predictions** top-down; the predictions are confirmed by **scanning** the input sentence and then **completions** are performed bottom-up.

- The Earley algorithm uses **tabulation**. Intermediate parsing states are stored in a **chart** composed of items.

  If the sentence to parse is $w_1\ w_2 \ldots w_n$ , any item has the form $< A \longrightarrow \alpha \bullet \beta , i , j>$ with the following meaning: $w_{i+1} \ldots w_j$ is a segment of the sentence that has already been recognized as the sequence $\alpha$; a consecutive segment is expected to be recognized as the sequence $\beta$, so that the concatenation of the two segments will be recognized with the category A by means of the rule $A \longrightarrow \alpha \bullet \beta$ of the grammar.

  The dotted rule $A \longrightarrow \alpha \bullet \beta$ expresses the progress in the use of the grammar rule.

# 3.5 – Parsing with tabulation: the Earley algorithm

- The Earley algorithm consists in filling the chart with items by application of the following derivation rules :

$$\frac{}{< S \longrightarrow \bullet\, \alpha \;, 0, 0 >} \quad \text{Init with } S \longrightarrow \alpha \in G$$

$$\frac{< A \longrightarrow \alpha \;\bullet B\, \beta \;, i, j >}{< B \longrightarrow \bullet\; \gamma \;, j, j>} \quad \text{Predict with } B \longrightarrow \gamma \in G$$

$$\frac{< A \longrightarrow \alpha \;\bullet w_j\, \beta \;, i, j >}{< A \longrightarrow \alpha\, w_j \bullet \beta \;, i, j+1 >} \quad \text{Scan}$$

$$\frac{< A \longrightarrow \alpha \;\bullet B\, \beta \;, i, j > \quad < B \longrightarrow \gamma \bullet \;, j, k >}{< A \longrightarrow \alpha\, B \bullet \beta \;, i, k >} \quad \text{Complete}$$

# 3.5 – Parsing with tabulation: the Earley algorithm

- The process of parsing ends when no new item can be produced and it succeeds if the item $< S \longrightarrow \alpha \bullet, 0, n >$ is present in the chart.

- Different orders of rule application define different strategies of parsing. Relevant strategies are those which are complete: any derivable item is produced by application of the strategy.

- The Earley algorithm in the previous form is a **recognition** algorithm. To transform it into a **parsing** algorithm, every item must be augmented with a list of pointers to items that have contributed to its completion.

- The worst case running time of Earley is $O(n^3.|G|^2)$ , where *n* is the length of the parsed string and $|G|$ is the size of the grammar *G*.

# 3.5 – Parsing with tabulation: the Earley algorithm

- Example of a complete Earley strategy:

**function** EARLEY-PARSE (*sentence, grammar*)

**for** each rule $S \longrightarrow \alpha \in grammar$
ENQUEUE ($S \longrightarrow \bullet\, \alpha$ , *chart [0]* )
**for** i **from** 0 **to** LENGTH (*sentence*)
**for each** *item* $\in$ *chart[i]*
**if** INCOMPLETE (*item*)
**if** NEXT-SYMBOL (*item*) is a non terminal
PREDICT(*item, grammar, chart*)
**else**
SCAN (*item, sentence, chart*)
**else**
COMPLETE(*item, grammar, chart*)
**return** *chart*

# 3.5 – Parsing with tabulation: the Earley algorithm

1. Consider the following CFG :

   $S \longrightarrow NP\ \ VP$

   $NP \longrightarrow Det\ N \mid NP\ that\ VP$

   $VP \longrightarrow V\ NP$

   $Det \longrightarrow the \mid that \mid no$

   $N \longrightarrow agencies \mid book \mid flight \mid chance$

   $V \longrightarrow book \mid flight \mid have$

   For the sentence *"the agencies that book that flight have no chance."*, determine if the following items are derivable from the Earley algorithm.

a) $< S \rightarrow NP \bullet VP, 0, 2 >$

b) $< NP \rightarrow Det\ N \bullet, 2, 4 >$

c) $< VP \rightarrow V \bullet NP, 3, 4 >$

d) $< NP \rightarrow NP\ that\ VP \bullet , 0, 6 >$

# 3.5 – Parsing with tabulation: the Earley algorithm

2. Consider the following CFG :

$S \longrightarrow NP \ \ VP$

$NP \longrightarrow NP \ VP \mid fish$

$VP \longrightarrow V \ NP$

$V \longrightarrow fish$

Parse the following sentences with this grammar using the CKY and Earley algorithms.

a) *fish fish fish*

b) *fish fish fish fish*

c) *fish fish fish fish fish*