Grammaires, automates et langages formels

Ce chapitre aborde quelques concepts de base sur les grammaires, les automates et les langages formels [1] qui sont en lien avec les concepts et représentations des grammaires locales et des grammaires locales étendues étudiées à partir des chapitres 3 et 4 de la thèse. En plus des notions préliminaires disponibles dans l'annexe A, nous introduisons succinctement les concepts des grammaires formelles, des langages réguliers et des automates finis, des langages non contextuels et des automates à pile, des langages contextuels ainsi que des langages sans restriction et des Machines de Turing.

2.1 Grammaires formelles

Une **grammaire formelle** est un *formalisme* pour décrire un langage (cf. définition A.34). Cette description permet à la fois de définir, générer et reconnaître ce langage. Nous donnons une définition plus formelle ci-après.

Définition 2.1 (Grammaire formelle)**.** Une **grammaire formelle** est un quadruplet $\mathcal{G} = (V, T, P, S)$, où :

1. Ces concepts sont largement étudiés dans Baudot (1987), Perrin (1990), Davis et al. (1994), Hopcroft et al. (2000), Sakarovitch (2009), Sipser (2012), Berstel (2013), Linz (2016)

1. V est un ensemble non vide et fini de **variables**, aussi appelées symboles *non-terminaux*,
2. T est un ensemble non vide et fini de symboles **terminaux**, tel que $V \amalg T$,
3. P est un ensemble fini de **règles de réécriture**, et
4. $S \in V$ est un symbole initial nommé l'**axiome de la grammaire**

Une grammaire formelle $\mathcal{G}$, dorénavant *grammaire* ou $\mathcal{G}$, est alors un système fini composé d'un ensemble *disjoint* de symboles de V et de T, ou $V \cup T$ constitue le **vocabulaire de la grammaire**, noté Σ, d'un ensemble fini P de règles de réécriture, ou simplement *règles*, et d'un seul symbole initial $S \in V$. Les règles de P constituent l'élément central de la définition de toute grammaire et sont de la forme :

$$l \to r \qquad \text{ou :}$$

- l, la *partie gauche* de la règle, est une suite de mots contenant au moins un symbole de V ou de T, soit $l \in (V \cup T)^+$, autrement dit, $l \in \Sigma^+$, et
- r, la *partie droite* de la règle, est une suite de mots contenant des symboles de V ou de T, soit $r \in (V \cup T)^*$, autrement dit, $l \in \Sigma^*$.

La règle $l \to r$ se lit « l se *réécrit* r » et signifie que l peut engendrer r. L'ensemble de règles $P = \{(l, r) \mid l \in \Sigma^+, r \in \Sigma^*\}$ permet ainsi, par leur application successive, de réécrire des suites de mots à partir d'autres suites. Selon la façon dont r peut se dériver de l, soit sous forme directe, ou bien sous forme indirecte, on parle d'une *dérivation directe* ou simplement d'une *dérivation*.

Définition 2.2 (Dérivation directe)**.** Si en appliquant, en *une seule* étape, la règle $l \to r$ à un mot w de la forme $u\boldsymbol{l}v$ avec $u, v \in \Sigma^*$, il est possible d'obtenir un mot z de la forme $u\boldsymbol{r}v$. Dans ce cas, on dit que w « *engendre directement* » z, ou que z « ***dérive directement de*** » w et on note :

$$w \implies z$$

Définition 2.3 (Dérivation)**.** Si en appliquant, en *zéro ou plus* étapes, un ensemble de règles P à un mot w, il est possible d'obtenir une suite de mots $x_0, x_1, x_2, \ldots, x_n$ pour $n \geq 0$, tel que :

$$\begin{gathered} w = x_0, \\ x_0 \implies x_1 \implies x_2 \implies \cdots \implies x_n, \text{ et} \\ z = x_n \end{gathered}$$

On dit que w « *engendre* » z, ou que z « ***dérive de*** » w et on note :

$$w \overset{*}{\Longrightarrow} z$$

En particulier, n est appelé la *longueur de la dérivation*. En outre, si $n = 0$ alors $w \overset{*}{\Longrightarrow} w$.

Définition 2.4 (Forme sentencielle)**.** Un mot $w \in \Sigma^*$ est appelé **forme sentencielle** de $\mathcal{G}$ s'il peut être engendré par une séquence de dérivations de la forme $S \overset{*}{\Longrightarrow} w$, autrement dit, elle se définit comme tout mot w qui se dérive de l'axiome de la grammaire S, le mot en entrée étant aussi une forme sentencielle, la première.

Définition 2.5 (Phrase)**.** Un mot w est appelé une **phrase** de $\mathcal{G}$ s'il est une forme sentencielle de $\mathcal{G}$ constituée uniquement de symboles terminaux (zéro ou plus), autrement dit, elle se définit comme tout mot $w \in T^*$ qui se dérive de l'axiome de la grammaire S.

Définition 2.6 (Langage d'une grammaire)**.** Si $\mathcal{G} = (V, T, P, S)$ est une grammaire, alors le langage défini par $\mathcal{G}$, noté $L(\mathcal{G})$, est l'ensemble de phrases ($w \in T^*$) que la grammaire engendre. Plus formellement :

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

En particulier, deux grammaires $\mathcal{G}_1$ et $\mathcal{G}_2$ sont dites *faiblement équivalentes*, noté $L(\mathcal{G}_1) = L(\mathcal{G}_2)$, si elles définissent le même langage. En outre, elles sont appelées **fortement équivalentes**, si, et seulement si, elles sont faiblement équivalentes et utilisent les mêmes dérivations pour engendrer la même phrase.

Définition 2.7 (Grammaire de génération)**.** Si $L(\mathcal{G}_e)$ est le langage défini par $\mathcal{G}$ tel que $L(\mathcal{G}) = \{w \in T^* \mid S \overset{*}{\Longrightarrow} w\}$, alors $\mathcal{G}$ est appelée **grammaire de génération**.

Définition 2.8 (Grammaire de reconnaissance)**.** Si $L(\mathcal{G}_r)$ est le langage défini par $\mathcal{G}$ tel que $L(\mathcal{G}) = \{w \in T^* \mid w \overset{*}{\Longrightarrow} S\}$, alors $\mathcal{G}$ est appelée *grammaire d'analyse* ou **grammaire de reconnaissance**.

Observons que dans une **grammaire de génération** les phrases ($w \in T^*$) sont engendrées en substituant l'axiome de la grammaire S par des symboles à droite et en répétant ce processus jusqu'à avoir uniquement des symboles terminaux ($S \overset{*}{\Longrightarrow} w$). En revanche, dans une **grammaire de reconnaissance** les règles sont inversées et de la forme $r \to l$, de sorte qu'une phrase est *analysée* en effectuant la substitution par les symboles qui se trouvaient initialement à gauche et en répétant ce processus jusqu'à atteindre l'axiome de la grammaire ($w \overset{*}{\Longrightarrow} S$). Bien que l'ensemble des règles d'une grammaire puisse avoir la forme $r \to l$, dite de *composition*, ou $l \to r$, dite de *décomposition*, il est utile de remarquer que la grammaire définit toujours le même langage.

Nous pouvons à présent résumer les trois aspects caractéristiques d'une grammaire : définitionnel, génératif et analytique :

1. **Définitionnel** : $\mathcal{G}$ *définit* L, autrement dit, $\mathcal{G}$ définit les phrases qui appartiennent à L ainsi que leurs structures.
2. **Génératif** : $\mathcal{G}$ peut *engendrer* L, autrement dit, G permet d'engendrer des phrases de L.
3. **Analytique** : $\mathcal{G}$ peut *reconnaître* L, autrement dit, G permet de reconnaître si une phrase quelconque appartient ou non à L.

Avant de donner quelques exemples de grammaires formelles, nous précisons les conventions utilisées pour représenter ses éléments.

Notation 2.9 (Éléments d'une grammaire)**.** Pour représenter une grammaire, nous utilisons les conventions résumées dans le tableau 2.1 :

N°	Élement	Convention
1	V	$A, B, C, \ldots$
2	V	⟨ABC⟩ est une variable
3	T	$a, b, c, \ldots$
4	Σ	$\alpha, \beta, \gamma, \ldots$
5	S	à gauche de la première règle
6	P	règles sans ordre imposé
7	P	$\{A \to aB, A \to \varepsilon\} = A \to aB\|\varepsilon$

Tableau 2.1 – Conventions pour répresenter une grammaire formelle

1. Les symboles de V (*variables*) sont notés par des lettres latines majuscules du début de l'alphabet : $A, B, C, \ldots$
2. Les symboles de V entourés par des chevrons sont considérés comme uniques, par exemple ⟨ABC⟩ est traité de la même manière que la n'importe quelle autre variable comme A
3. Les symboles de T (*terminaux*) sont notés par des lettres latines minuscules du début de l'alphabet : $a, b, c, \ldots$
4. Les symboles de Σ ($V \cup T$) sont notés par des lettres grecques minuscules : $\alpha, \beta, \gamma, \ldots$
5. L'axiome de la grammaire (S) est placé à gauche de la première règle
6. L'ensemble de règles (P) n'est pas ordonné, aucun ordre de définition ne s'impose
7. Les règles qui partagent la même partie gauche peuvent s'écrire sous forme abrégée sur une seule ligne. Pour cela, tous les éléments de la partie droite sont regroupés, puis séparés par des traits verticaux |, où chaque trait vertical peut se lire « *ou bien* ». Plus formellement :

   Soit $N = \{(A, r_1), (A, r_2), \ldots, (A, r_n)\}$, tel que $N \subseteq P$, alors N peut s'abréger :
   $A \to r_1|r_2|\ldots|r_n$

   Par exemple, les deux règles : $A \to aB$ et $A \to \varepsilon$, sont équivalents à $A \to aB|\varepsilon$, c'est-à-dire, « *A se réécrit aB, ou bien, ε* ».

Considérons à présent les exemples suivants.

Exemple 2.10.

$$\mathcal{G}_1 = (V, T, P, S),$$

1. $V : \{S, A, B\}$
2. $T : \{a, b\}$
3. P :

S	→	A
A	→	aB | ε
B	→	bA

Par l'application de P, il est possible d'engendrer, par exemple, la phrase *abab* :

$$S \Longrightarrow A \Longrightarrow aB \Longrightarrow \boldsymbol{abA} \Longrightarrow abaB \Longrightarrow ababA \Longrightarrow \boldsymbol{abab}$$

Ce qui équivaut à écrire :

$$S \overset{*}{\Longrightarrow} abab$$

où :

- abA est une *forme sentencielle* ($w \in \Sigma^*$)
- *abab* est une *phrase* ($w \in T^*$)
- $S \overset{*}{\Longrightarrow} abab$ a une longueur de 6, c'est-à-dire, *abab* est le résultat de 6 dérivations directes

Finalement, observons que les phrases que $\mathcal{G}_1$ peut engendrer ont toutes la forme :

ε
ab
abab
ababab
abababab
ababababab
abababababab

Nous pouvons conclure que $\mathcal{G}_1$ définit le langage :

$$L(\mathcal{G}_1) = \{(ab)^n \mid n \geq 0\}$$

Définition 2.11 (Variable annulable et variable nulle)**.** Une variable $A \in V$ est dite **annulable** s'il existe une dérivation $A \overset{*}{\Longrightarrow} \varepsilon$. De plus, A est dite **nulle** si toutes ses dérivations engendrent le mot vide.

Notation 2.12 (Arbre de dérivation)**.** Étant donnée une grammaire quelconque, le processus de dérivation d'une phrase peut être visualisé plus facilement à l'aide d'une notation arborescente. Il s'agit de construire un arbre où la racine coïncide avec l'axiome de la grammaire (S), les nœuds avec les variables (V) et les feuilles avec les symboles de la phrase (T). Ce type de représentation est connu sous le nom d'**arbre de dérivation** ou *arbre syntaxique* du fait qu'il représente la structure syntaxique d'une phrase engendrée par une grammaire.

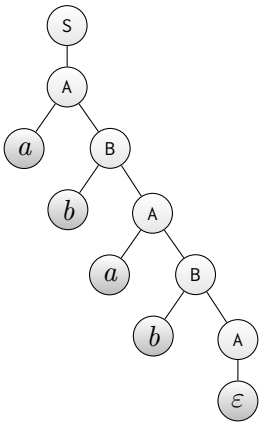

Figure 2.1 – Arbre de dérivation de la phrase *abab* engendré par la grammaire $\mathcal{G}_1$

Définition 2.13 (Grammaire ambiguë)**.** Une grammaire est dite **ambiguë** lorsqu'il existe plus d'un arbre de dérivation pour une phrase du langage.

Définition 2.14 (Langage ambigu)**.** Un **langage ambigu** est un langage défini par une grammaire ambiguë.

Définition 2.15 (Types de grammaires)**.** Dans les années 1956, Noam Chomsky a proposé une classification des grammaires en 4 catégories, connue désormais comme La **Hiérarchie de Chomsky**, ou chaque catégorie a des règles plus restrictives à mesure que le type augmente. La **Hiérarchie de Chomsky** est présenté dans le tableau 2.2

Type	Grammaire	Machine
3	Régulière	Automate fini
2	Non contextuelle	Automate à pile
1	Contextuelle	Automate linéairement borné
0	Contextuelle avec effacement	Machine de Turing

Tableau 2.2 – Hiérarchie de Chomsky

Même si la Hiérarchie de Chomsky est largement utilisée, il est utile de rappeler que mises à part les grammaires régulières (aussi appelées rationnelles), non contextuelles, contextuelles et contextuelles avec effacement (aussi nommées générales ou sans

restriction), il existe d'autres formalismes linguistiques pour décrire de grammaires formelles, citons par exemple :

- La description syntaxique systématique des lexique-grammaires (Gross, 1975)
- Les grammaires lexicales-fonctionnelles (*Lexical functional grammar*, Kaplan et Bresnan, 1982)
- Les grammaires d'arbres adjoints (*Tree-adjoining grammars*, Joshi et Schabes, 1997)
- Les grammaires syntagmatiques guidées par les têtes (*Head-driven phrase structure grammars*, Pollard et Sag, 1994)
- Les grammaires relationnelles (*Relational grammars*, Perlmutter, 1980)
- Les grammaires orientées relationnelles (*Arc pair grammar*, Johnson et Postal, 1981)
- Les grammaires catégorielles avec unification (*categorial unification grammars*, Uszkoreit, 1986)

En outre, en ce qui concerne les automates finis qui reconnaissent chaque type de langage, il est important de dire qu'il est possible de parvenir à reconnaître des langages moins restrictifs (type 0 et 1) à l'aide de langages plus restrictifs (type 2 et 3) et d'opérations qui se trouvent hors du cadre des automates. Par exemple, comme nous les étudions dans le chapitre 3 dédié aux grammaires locales, le fait que *tout langage est inclus dans un langage rationnel* (cf. section 3.5.8) permet à une grammaire locale de mettre en place une approche pour reconnaître des langages contextuels (type 1 ou 0) en reconnaissant d'abord un langage régulier ou non contextuel (type 3 ou 2), et en appliquant par la suite une opération pour exclure les séquences qui n'appartiennent au langage contextuel.

Dans les sections qui suivent nous présentons d'abord les automates finis, ensuite nous étudions les types de langages associés à l'Hiérarchie de Chomsky.

2.2 Automates finis

Les automates finis sont des machines abstraites qui vérifient sous forme séquentielle si un mot, passé en entrée, appartient ou non à un langage donné. Ces automates sont dits finis car restreints à un ensemble fini d'états qui mémorisent des informations. Les automates finis ont trouvé des applications directes dans plusieurs domaines qui vont de l'analyse lexicale de langues artificielles et naturelles, en passant par la compression de l'information, la recherche de motifs dans un texte, ou la vérification de systèmes, tel que des circuits électroniques, dotés d'un nombre fini d'états.

2.2.1 Automates finis déterministes

Les automates finis déterministes sont rapides et efficaces pour traiter des problèmes d'analyse des langages tel que la génération automatique de phrases (cf. définition 2.5) ainsi que leur recherche dans des données textuelles. Considérons les phrases :

(1) *Leonardo was an Italian mathematician*

(2) * *mathematician an Leonardo Italian was*

Un automate fini $\mathcal{A}$ qui reconnaît un langage quelconque L, est capable alors de dire si les phrases (1) et (2) appartiennent à L (propriété analytique). Dans la suite, nous présentons formellement ces automates ainsi que leur version non-déterministe.

Définition 2.16 (Automate fini déterministe)**.** Un **automate fini déterministe** est un quintuplet $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$, où :

1. Q est un ensemble fini d'**états**,
2. Σ est un **alphabet**,
3. $\delta : Q \times \Sigma \longrightarrow Q$ est la **fonction de transition**,
4. $q_0 \in Q$ est l'**état initial**, et
5. $F \subseteq Q$ est l'ensemble des **états finaux**.

Un automate fini $\mathcal{A}$ correspond à un **graphe orienté** (cf. A.16) formé par un ensemble fini d'**états** Q (*les nœuds du graphe*) et de **transitions** (*les arcs du graphe*) qui les relient. Un état est désigné comme **initial** ($q_0 \in Q$) et certains autres sont **finaux** ($F \subseteq Q$). Un automate est à tout instant dans un état unique, la **fonction de transition** δ décrit alors comment l'automate passe à un autre état en lisant un symbole de l'**alphabet**.

Chaque transition est définie comme un triplet (p, σ, q) de $Q \times \Sigma \times Q$ (l'**ensemble de transitions**) où : p est l'**état de départ**, σ est un symbole de l'alphabet, et q est l'**état d'arrivée**. Si $\mathcal{A}$ a une **transition étiquetée** par un symbole σ qui relie p à q, ceci indique que lorsque l'automate est dans l'état p et *lit* σ, il passe à l'état q. Cette **règle de transition** est notée : $\delta(p, \sigma) = q$. Considérons l'exemple 2.17 :

Exemple 2.17.

$$\mathcal{A}_1 = (Q, \Sigma, \delta, q_0, F),$$

1. $Q : \{q_0, q_1, q_2, q_3\}$
2. $\Sigma : \{0, 1\}$

3. δ : est décrite par la **table de transition d'état** suivante : [1]

q	$\delta(q,0)$	$\delta(q,1)$
q_0.	q_0	q_1
q_1	q_2	q_1
q_2,	q_3	q_3
q_3	q_2	q_2

4. q_0
5. $F : \{q_2\}$

Un automate fini peut être représenté graphiquement par un diagramme **d'états-transitions** comme celui de la figure 2.2. L'état initial, conventionnellement le premier à gauche, est distingué par un arc orienté vers un nœud aux contours en gras. Les états finaux sont indiqués par des nœuds avec double cercle. Les transitions sont des arcs orientés allant d'un nœud de départ vers un nœud d'arrivée. Les étiquettes des transitions se placent sur les arcs. Lorsque un automate a plusieurs transitions [2] reliant deux états, alors il est possible de placer sur un seul arc l'ensemble des étiquettes sous forme la d'une liste. Pour l'automate $\mathcal{A}_1$ de l'exemple 2.17, nous avons deux transitions $(q_3, 0, q_2)$ et $(q_3, 1, q_2)$ reliant q_3 à q_2 qui sont représentées par un seul arc étiqueté par $0, 1$.

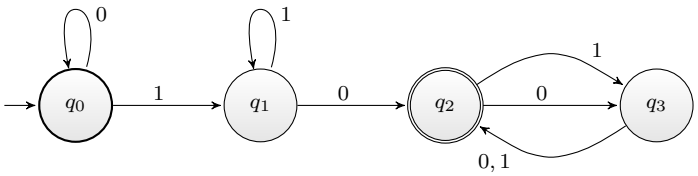

Figure 2.2 – Diagramme d'états-transitions d'un automate fini

Définition 2.18 (Chemin sur un automate fini)**.** Un **chemin** sur un automate fini est une suite de transitions menant d'un état p à un état r dont la concaténation des étiquettes successives (l'**étiquette du chemin**) forment un mot $w_i, w_{i+1}, \ldots, w_{j-1}, w_j$. Nous notons ceci comme $p \xrightarrow{w_{i,j}} r$. Un **chemin réussi** est nommé ainsi lorsque $p = q_0$ et $r \in F$, c'est-à-dire quand il existe un chemin allant de l'état initial à l'un des états finaux.

Définition 2.19 (Mot accepté par un automate fini)**.** Un mot w de Σ^* est **accepté** par un automate fini $\mathcal{A}$ s'il existe un chemin réussi qui est étiqueté par w, soit $q_0 \xrightarrow{w} r$, tel que $r \in F$.

1. Les tables de transition d'états sont une alternative pour décrire les automates finis. Les lignes représentent les états. L'état initial est marqué par un point après son nom et les états finaux avec une virgule. Les colonnes représentent l'alphabet. Les cellules, c'est-à-dire les intersections ligne/colonne, codent les transitions.
2. Soit un sous-ensemble $T \subseteq Q \times \Sigma \times Q$, tel que $T = \{(p, \sigma, q) \mid p \in Q, \sigma \in \Sigma, q \in Q, \delta(p, \sigma) = q\}$

Définition 2.20 (Langage reconnu par un automate fini)**.** Le **langage reconnu** par un automate fini $\mathcal{A}$, noté $L(\mathcal{A})$, est égal à l'ensemble des mots acceptés par $\mathcal{A}$. En particulier, deux automates $\mathcal{A}_1$ et $\mathcal{A}_2$ sont dits **équivalents** s'ils reconnaissent le même langage, soit $L(\mathcal{A}_1) = L(\mathcal{A}_2)$. En outre, remarquons qu'un automate fini peut accepter plusieurs mots, mais il reconnaît toujours un seul langage. L'inverse n'est pas vrai, pour un langage donné, il peut exister plusieurs automates qui le reconnaissent.

Considérons l'automate $\mathcal{A}_2$ de l'exemple 2.21.

Exemple 2.21.

$$\mathcal{A}_2 = (Q, \Sigma, \delta, q_0, F),$$

1. $Q : \{q_n \mid n \leq 15\}$
2. $\Sigma : \{a, c, d, e, i, l, p, r, s\}$
3. δ :

$$\begin{array}{cccc} \delta(q_0, p) = q_1 & \delta(q_1, a) = q_2 & \delta(q_2, r) = q_3 & \delta(q_3, a) = q_4 \\ \delta(q_4, d) = q_5 & \delta(q_5, i) = q_6 & \delta(q_6, s) = q_7 & \delta(q_7, e) = q_8 \\ \delta(q_8, s) = q_{11} & \delta(q_7, i) = q_9 & \delta(q_9, c) = q_{13} & \delta(q_{13}, a) = q_{15} \\ \delta(q_{15}, l) = q_{11} & \delta(q_9, a) = q_{14} & \delta(q_{14}, l) = q_{11} & \delta(q_{14}, c) = q_{11} \\ \delta(q_7, a) = q_{10} & \delta(q_{10}, l) = q_{11} & \delta(q_{10}, i) = q_{12} & \delta(q_{12}, c) = q_{11} \end{array}$$

4. q_0
5. $F : \{q_8, q_{11}\}$

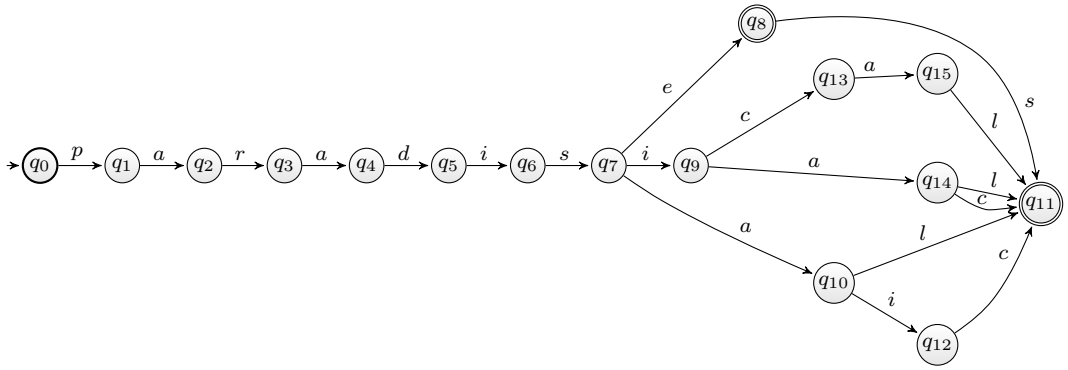

Figure 2.3 – Automate fini qui reconnaît le langage $L(\mathcal{A}_2) = \{paradise, paradises, paradisical, paradisial, paradisiac\ paradisal, paradisaic\}$

Quelques exemples de chemins sur l'automate $\mathcal{A}_2$ sont :

1. $q_0 \xrightarrow{parad} q_5$
2. $q_2 \xrightarrow{radise} q_8$

3. $q_0 \xrightarrow{paradise} q_8$

(1) n'est pas un chemin réussi puisque $q_5 \notin F$, (2) ne l'est pas non-plus car $q_2 \neq q_0$, (3) est un chemin réussi, il part de l'état initial (q_0) et arrive dans un des états finaux $q_8 \in F$. Étant donnée que $q_0 \xrightarrow{paradise} q_8$ est un chemin réussi, *paradise* est un mot accepté par $\mathcal{A}_2$. Les autres mots acceptés sont : *paradises*, *paradisical*, *paradisial*,..., l'ensemble de mots acceptés par $\mathcal{A}_2$, autrement dit, le langage reconnu par $\mathcal{A}_2$, est :

$$L(\mathcal{A}_2) = \{paradise, paradises, paradisical, paradisial, paradisiac, paradisal, paradisaic\}$$

Définition 2.22 (État accessible, co-accessible)**.** Un état $q \in Q$ est dit **accessible** s'il existe un chemin qui part de q_0 et mène à q ; il est dit **co-accessible** s'il existe un chemin qui part de q et mène à r tel que $r \in F$.

Définition 2.23 (Automate émondé)**.** Un automate $\mathcal{A}$ est **émondé** si tous ses états sont accessibles et co-accessibles.

Définition 2.24 (Déterminisme d'un automate)**.** Un automate $\mathcal{A}$ sur un alphabet Σ est un automate fini déterministe (DFA), si et seulement si les deux conditions suivantes sont satisfaites : 1. Il n'existe pas de transitions étiquetées par le mot vide (ε) ; et 2. Pour tout état p, il part vers q au plus une transition étiquetée par chaque symbole $\sigma \in \Sigma$. Les automates des exemples 2.17 et 2.21 sont des DFAs.

Définition 2.25 (Automate fini déterministe complet)**.** Un automate fini déterministe sur un alphabet Σ est un **automate fini déterministe complet** si pour tout état p part une et une seule transition étiquetée de σ vers q. Par exemple, l'automate de l'exemple 2.17 est un DFA complet, celui de l'exemple 2.21 ne l'est pas.

2.2.2 Automates finis non-déterministes

Rappelons qu'un automate fini déterministe est à tout instant dans un état unique et que lorsque il lit un symbole en entrée, il est possible de déterminer avec certitude quel sera l'état suivant. En effet, la fonction de transition d'un DFA est égal à $\delta : Q \times \Sigma \longrightarrow Q$, c'est-à-dire, qu'à partir d'un état p et d'un symbole σ dans $Q \times \Sigma$, elle a toujours un seul état q d'arrivée dans Q, soit $\delta(p, \sigma) = q$.

Cependant, lorsqu'à partir d'un symbole donné et d'un état de départ il est possible d'avoir comme destination plusieurs états d'arrivée, ou lorsqu'il est admis de changer d'état sans lire un symbole, ou lorsqu'on a la faculté de choisir quel sera le prochain état, ou de passer dans un autre d'état spontanément, est appelé automate fini non-déterministe (NFA). Au sens informatique, le non-déterminisme peut se comparer à une sorte de parallélisme où pour chaque transition qui comporte plus d'un état d'arrivée, l'automate fini a la capacité de créer simultanément plusieurs copies de lui-même et, pour chaque copie, d'explorer chacun des états d'arrivée possibles.

En outre, pour chaque automate fini non-déterministe il y a un automate fini déterministe équivalent. Autrement dit, tout automate fini non-déterministe peut être converti en un automate fini déterministe équivalent. Ceci est d'un grand intérêt puisque, même si les automates finis non-déterministes sont plus coûteux à utiliser, ils ont un nombre d'états et de transitions réduits, ils sont plus faciles à comprendre et à construire. Dans la suite, nous donnons une définition formelle d'un automate fini non-déterministe.

Définition 2.26 (Automate fini non-déterministe)**.** Un **automate fini non-déterministe** est un quintuplet $\mathcal{N} = (Q, \Sigma, \delta, q_0, F)$, où :

1. Q est un ensemble fini d'**états**,
2. Σ est un **alphabet**,
3. $\delta : Q \times \Sigma_\varepsilon \longrightarrow \wp(Q)$ est la **fonction de transition**,
4. $q_0 \in Q$ est l'**état initial**, et
5. $F \subseteq Q$ est l'ensemble des **états finaux**.

Les automates finis déterministes et non-déterministes sont similaires à l'exception du type de fonction de transition δ utilisée. Observons que cette fonction de transition se différencie par : 1. l'alphabet d'entrée de la fonction qui n'est pas simplement Σ mais l'union entre cet alphabet et un ensemble ne contentant que le mot vide $\{\varepsilon\}$, noté Σ_ε [1] ; 2. la sortie de la fonction qui n'est plus constituée par un seul état, mais par un ensemble des parties de Q, noté $\wp(Q)$, contenant zéro, un état ou plus. Cela est illustré par l'exemple 2.27.

Exemple 2.27.

$$\mathcal{N}_1 = (Q, \Sigma, \delta, q_0, F),$$

1. $Q : \{q_0, q_1, q_2, q_3\}$
2. $\Sigma : \{0, 1\}$
3. δ : est décrite par la table de transition d'état suivante :

q	$\delta(q, 0)$	$\delta(q, 1)$
$q_0 \cdot,$	$\varnothing$	$\{q_1, q_2\}$
$q_1,$	$\varnothing$	$\{q_1\}$
q_2	$\{q_3\}$	$\varnothing$
$q_3,$	$\varnothing$	$\{q_2\}$

4. q_0
5. $F : \{q_0, q_1, q_3\}$

1. Pour éviter toute confusion, il est requis que ε ne soit pas membre de Σ.

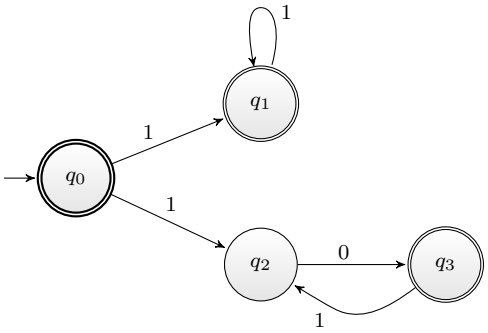

FIGURE 2.4 – Diagramme d'états-transitions d'un NFA

Définition 2.28 (ε–transitions)**.** Les transitions qui font passer un automate fini non-déterministe d'un état dans un autre sans qu'aucune opération de lecture d'un symbole ne soit effectuée, sont nommées des transitions spontanées ou **ε–transitions**. De même, un automate fini non-déterministe avec ε–transitions est abrégé en ε–NFA. Remarquons que ce type de transitions est possible car la fonction de transition accepte $\varepsilon \in \Sigma_\varepsilon$ en entrée.

Exemple 2.29. Un automate équivalent à $\mathcal{N}_1$ (exemple 2.27) qui utilise des transitions spontanées est donné à figure 2.5.

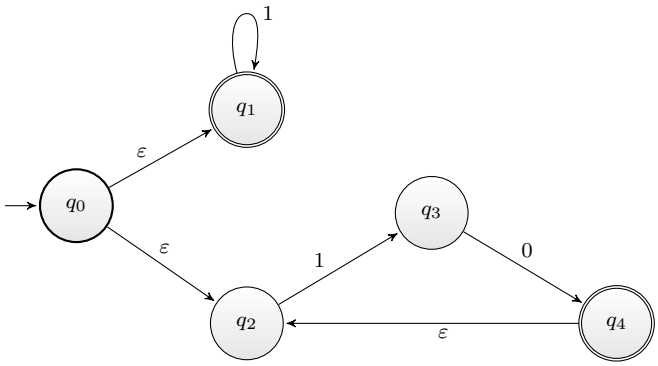

FIGURE 2.5 – Diagramme d'états-transitions d'un ε–NFA

Enfin, il est important de mentionner que tout automate fini non-déterministe a un automate fini déterministe équivalent. La démonstration de ceci est due à Rabin et Scott (1959) et consiste à construire à partir d'un automate fini *non-déterministe* $\mathcal{N} = (Q, \Sigma, \delta, q_0, F)$ qui reconnaît un langage L, un automate fini *déterministe* $\mathcal{A} = (Q', \Sigma, \delta', q_0', F')$ qui reconnaît aussi L.

2.3 Langages réguliers

Les **langages réguliers**, aussi appelés *langages rationnels*, sont les langages les plus restrictifs, mais à la fois les plus faciles à manipuler. Ces langages peuvent être reconnus

par les automates finis (cf. section 2.2) et les expressions régulières (cf. section 2.3.2), tous les deux équivalents quant à leur puissance descriptive [1]. Nous pouvons définir un langage régulier sous la forme suivante :

Définition 2.30 (Langage régulier)**.** Étant donné un langage L sur Σ, L est un **langage régulier** si, et seulement si, il existe un automate $\mathcal{A}$ qui reconnaît ce langage.

Nous avons déjà présenté les automates finis. Considérons à présent les grammaires régulières et les expressions régulières.

2.3.1 Grammaires régulières

Une **grammaire régulière**, à droite ou à gauche, est une grammaire formelle qui décrit un langage régulier [2]. Les grammaires régulières sont explicitées par la définition ci-après :

Définition 2.31 (Grammaire régulière à droite)**.** Une **grammaire régulière à droite** est un quadruplet $\mathcal{G} = (V, T, P, S)$, où (1) V et T sont des ensembles non vides et finis de symboles, (2) $V \amalg T$, $\Sigma = V \cup T$, (3) P est un ensemble fini de règles de réécriture et (4) $S \in V$ est l'axiome de la grammaire.

L'ensemble fini P des règles constitue l'élément distinctif d'une grammaire régulière à droite par rapport aux autres grammaires formelles. Les règles de P ont l'une des formes suivantes :

$$A \to aB$$
$$A \to a$$

où $A, B \in V$ sont des variables et $a \in T$ est un symbole terminal. Les grammaires régulières à droite permettent alors de remplacer une variable soit par un symbole terminal, soit par un symbole terminal suivi d'une variable. Si à la place de la règle $A \to aB$ on utilise la règle $A \to Ba$, c'est-à-dire, que B est placé à gauche de a, la grammaire est alors dénommée **grammaire régulière à gauche**.

Définition 2.32 (Grammaire régulière à gauche)**.** Une **grammaire régulière à gauche** est une grammaire dans laquelle toutes les règles ont la forme :

1. Toute expression régulière R peut être convertie en un automate fini $\mathcal{A}$ et vice versa. Voir a ce sujet (Hopcroft et al., 2000, p. 66)
2. Remarquons que même si une grammaire régulière est limitée à décrire un langage régulier, ce dernier peut aussi être défini par une grammaire non-régulière.

$$A \to Ba$$
$$A \to a$$

où $A, B \in V$ et $a \in T$.

N.B. 2.33. Il est important de remarquer qu'une grammaire régulière est soit régulière à droite, soit régulière à gauche, mais pas les deux à la fois. Dorénavant, sans autre précision, nous utiliserons le terme **grammaire régulière** pour designer une *grammaire régulière à droite*.

À présent, considérons la grammaire régulière de l'exemple 2.34.

Exemple 2.34.

$$\mathcal{G}_2 = (V, T, P, S),$$

1. $V : \{S, A\}$
2. $T : \{a, b, c\}$
3. P :

S	→	aA | c
A	→	bS

Par l'application de P, il est possible d'engendrer, les phrases :

```
c
abc
ababc
abababc
ababababc
abababababc
```

Soit le langage :

$$L(\mathcal{G}_2) = \{(ab)^n c \mid n \geq 0\}$$

En ce qui concerne la représentation d'une phrase engendrée par une grammaire régulière sous la forme d'un arbre de dérivation, il est utile de constater que le fait de limiter le remplacement d'une variable par une autre au plus, génère des arbres où, tout fils gauche existant est une feuille. La figure 2.6 illustre l'arbre de la phrase abc de $L(\mathcal{G}_2)$ engendrée par la dérivation :

$$S \Longrightarrow aA \Longrightarrow abS \Longrightarrow abc, \text{ soit } S \overset{*}{\Longrightarrow} abc$$

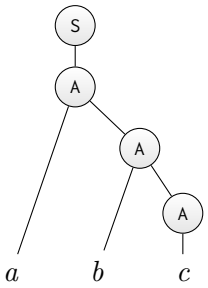

Figure 2.6 – Arbre de dérivation de la phrase *abc* engendrée par $\mathcal{G}_2$

Mis à part les automates finis et les grammaires régulières [1], les langages réguliers peuvent être définis par des expressions régulières. Nous donnons ensuite leur définition.

2.3.2 Expressions régulières

Une **expression régulière**, aussi appelée expression rationnelle, est un formalisme qui définit selon une notation précise un langage régulier. Les expressions régulières peuvent être définies de façon inductive comme suit :

Définition 2.35 (Expression régulière)**.** Une **expression régulière** R sur un alphabet Σ est soit :

1. Une expression régulière a pour tout $a \in \Sigma$, $\varnothing$ ou ε,
2. Une expression régulière formée à partir d'autres expressions régulières R_1 et R_2 par les opérations d'union $(R_1 \cup R_2)$, de concaténation $(R_1 \circ R_2)$ ou d'étoile de Kleene (R_1^*).

Dans (1), l'expression régulière a représente le langage $\{a\}$, l'expression régulière $\varnothing$ indique le langage vide $\{\}$, autrement dit, le langage qui ne contient pas de mots, et l'expression régulière ε est le langage $\{\varepsilon\}$, soit le langage qui contient juste le mot vide.

Dans (2), l'expression régulière $(R_1 \cup R_2)$ indique l'union des langages $L(R_1) \cup L(R_2)$, l'expression régulière $(R_1 \circ R_2)$ est égale à la concaténation des langages $L(R_1) \circ L(R_2)$, et l'expression régulière (R_1^*) est l'étoile de Kleene du langage $L(R_1)^*$.

À partir de l'application d'un nombre fini d'opérations données en (2), il est possible de construire des expressions régulières plus complexes que celles décrites en (1). Il est aussi utile de définir R^+ [2] comme la forme abrégée de RR^*. En outre, il est propice de

1. Ou non-régulières, remarquons de nouveau que même si une grammaire régulière est limitée à décrire un langage régulier, ce dernier peut aussi être défini par une grammaire non-régulière.

2. Soit l'ensemble de toutes les concaténations non-vides sur les membres d'un langage L. Notons que L^+ contient ε si, et seulement si, L contient ε et L^+ contient un mot non-vide w si, et seulement si, $w \in L^*$

définir un ordre de priorité pour les opérateurs : d'abord l'étoile de Kleene, ensuite la concaténation et finalement l'union. Nous donnons ci-après quelques exemples d'expressions régulières.

Exemple 2.36. Soit R une expression régulière sur un alphabet $\Sigma = \{0, 1\}$, nous pouvons définir R comme :

1. $0 \cup 1(0 \cup 1)^* = \{w \mid w \text{ est un numéro binaire}\}$
2. $(01)^*$, dont la grammaire est similaire à celle de l'exemple 2.10
3. $1^* \cup (10)^*$, dont le NFA équivalent correspond à celui de l'exemple 2.27

2.4 Langages non contextuels

Les expressions régulières et les automates finis sont équivalents quant à leur puissance descriptive. Cependant, il existe certains langages qui ne peuvent pas être décrits via ces deux formalismes. Un exemple est le langage sur $\Sigma = \{0, 1\}$ défini par $\{0^n 1^n \mid n \geq 0\}$, qui est le langage contenant des mots formés par une suite de plusieurs 0s suivi immédiatement du même nombre de 1s.

Lorsque un langage a une nature récursive il est alors utile d'avoir des méthodes formelles pour définir, engendrer et reconnaître le dit langage. Les **langages non-contextuels (CFLs)** sont une classe plus large que les langages réguliers. Ils permettent de formaliser des récursions et des structures imbriquées propres à certains langages. De plus, ils contiennent les langages réguliers, dans ce sens, tout langage régulier est aussi un langage non contextuel. Bien qu'il existe des langages non contextuel qui ne sont pas réguliers.

Un langage non contextuel peut être défini par une **grammaire non contextuelle** et, à l'instar des grammaires régulières, ces grammaires correspondent à une famille d'automates connue sous le nom des **automates à pile**. Dans la suite, nous explicitons ces deux notions.

2.4.1 Grammaires non contextuelles

Une grammaire non-contextuelle (CFG), aussi appelée grammaire algébrique ou grammaire indépendante du contexte, est une grammaire dotée de règles de réécriture plus expressives que celles d'une grammaire régulière. En effet, les CFGs peuvent définir certaines caractéristiques des langages qui ont une nature récursive, les rendant utiles, par exemple, pour étudier un langage naturel. C'est dans ce cadre que les grammaires non contextuelles ont trouvé une première utilité comme moyen pour comprendre et organiser les relations entre les parties du discours telles que les noms, verbes, pronoms, prépositions. Considérons à présent une définition plus formelle.

Définition 2.37 (Grammaire non contextuelle)**.** Une **grammaire non contextuelle** est une grammaire formelle, c'est-à-dire un quadruplet $\mathcal{G} = (V, T, P, S)$, où (1) V et T

sont des ensembles non vides et finis de symboles, (2) $V \amalg T$, $\Sigma = V \cup T$, (3) P est un ensemble fini de règles de réécriture et (4) $S \in V$ est l'axiome de la grammaire.

L'ensemble fini P des règles constitue l'élément distinctif d'une grammaire non contextuelle par rapport aux autres grammaires formelles. Il est défini comme [1] :

$$P = \{(A, \omega) \mid A \in V, \omega \in \Sigma^+\}$$

Les règles sont alors des paires ordonnées (A, ω) de la forme :

$$A \to \omega$$

où A est une variable et ω une suite de mots contenant n variables et/ou symboles terminaux ($n > 0$). Remarquons que la différence entre une grammaire qui décrit un langage régulier et celle qui décrit un langage non contextuel se situe dans la partie droite de la règle. En effet, tandis que dans une grammaire régulière il est possible de remplacer une variable par un symbole terminal, suivi ou non d'une variable, dans une grammaire non contextuelle, il est permis de remplacer une variable par n'importe quelle suite non vide d'éléments du vocabulaire de la grammaire.

Étudions l'exemple 2.38 qui décrit de façon simplifiée quelques relations entre les parties du discours de la langue anglaise.

Exemple 2.38.

$$\mathcal{H}_1 = (V^{\,2}, T, P, S),$$

1. V : $\{\langle \text{S} \rangle, \langle \text{NP} \rangle, \langle \text{VP} \rangle, \langle \text{NNP} \rangle, \langle \text{VBD} \rangle, \langle \text{DT} \rangle, \langle \text{JJ} \rangle, \langle \text{NN} \rangle\}$ [3]
2. T : $\{\texttt{a}\ , \ldots, \texttt{z}, \texttt{␣}\}$
3. P :

⟨S⟩	→	⟨NP⟩ ⟨VP⟩
⟨NP⟩	→	⟨NNP⟩ | ⟨DT⟩ ⟨JJ⟩ ⟨NN⟩
⟨VP⟩	→	⟨VBD⟩ ⟨NP⟩
⟨NNP⟩	→	`Maurice | Johannes | Niels`
⟨VBD⟩	→	`was | wrote | saw`
⟨DT⟩	→	`a | the`
⟨JJ⟩	→	`French | German | Danish`
⟨NN⟩	→	`linguist | book | garden`

1. Nous utilisons la définition de Baudot (1987, pp.44–45) qui restreint l'utilisation de variables nulles (cf. définition 2.5). Ceci dit, il est utile de remarquer que si l'on a une CFG avec ε-règles, il est parfois possible de la convertir dans une CFG où on a enlevé le mot vide.

2. Rappelons que les symboles V entourés par des chevrons sont considérés comme des symboles uniques.

3. Nous utilisons pour cet exemple un sous-ensemble du jeu d'étiquettes du Penn Treebank (Marcus et al., 1993). ⟨S⟩ : phrase ; ⟨NP⟩ : syntagme nominal ; ⟨VP⟩ : syntagme verbal ; ⟨NNP⟩ : nom propre, singulier ; ⟨VBD⟩ : verbe au passé ; ⟨DT⟩ : déterminant ; ⟨JJ⟩ : adjective ; ⟨NN⟩ : nom singulier.

La grammaire $\mathcal{H}_1$ possède : 8 variables ($|V| = 8$) ; 27 symboles terminaux correspondant aux lettres de l'alphabet anglais plus l'espace ($|T| = 27$) et 18 règles de réécriture ($|P| = 18$). À partir de cette grammaire il est possible d'engendrer, par exemple, les phrases suivantes :

(3) *Maurice was a French linguist*

(4) *Johannes wrote a Danish book*

(5) *Niels saw the German garden*

Notons que la grammaire $\mathcal{H}_1$ décrit les phrases de $L(\mathcal{H}_1)$ au niveau syntaxique, représentant la façon dont leurs parties sont agencées et hiérarchisées. Ceci est plus facile à constater à l'aide d'un arbre de dérivation. La figure 2.7 illustre celui de la phrase (3).

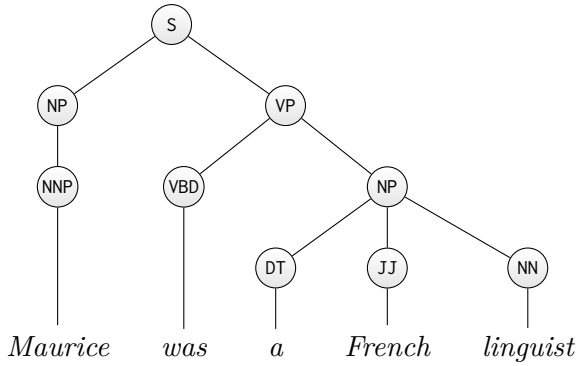

Figure 2.7 – Arbre de dérivation de la phrase « *Maurice was a French linguist* » engendrée par la grammaire $\mathcal{H}_1$

Observons qu'à la différence des arbres de dérivation issus des grammaires régulières, où tout fils gauche est nécessairement une feuille, un arbre de dérivation issu d'une grammaire non contextuelle comprend pour chaque fils, soit un symbole terminal, soit une variable. Ceci de telle sorte que si un nœud est défini par une variable, il constitue à la fois un sous-arbre, lui aussi contenant éventuellement d'autres sous-arbres, où chaque sous-arbre définit une partie de la phrase. Dans la figure 2.7, les sous-arbres de racine ⟨NP⟩ définissent les parties de la phrase « *Maurice* » et « *a French linguist* ».

Finalement, il est important de rappeler que la description réalisée par une grammaire non contextuelle est limitée au plan syntaxique. Considérons les phrases :

(6) * *Johannes was a French garden*

(7) * *the Danish book saw Maurice*

les phrases (6) et (7) ont été aussi engendrées par $\mathcal{H}_1$ et par conséquent elles ont une structure syntaxique identique à celle des autres phrases de $L(\mathcal{H}_1)$. Cependant, dans

la phrase (6) le verbe *was* produit une relation grammaticale entre *Johannes* et *a French garden* inattendu (un homme peut-il être un jardin ?), il en est de même pour la phrase (7) où le nom inanimé *the Danish book* est le sujet du verbe de perception *saw* (un livre peut-il regarder un homme ?).

2.4.2 Automates à pile

Un automate à pile (PDA) est une machine abstraite équivalente à un automate fini non-déterministe [1] muni d'une pile infinie. Il peut alors écrire des symboles dans la pile et les récupérer plus tard, ce qui lui confère une mémoire supplémentaire au-delà de la quantité finie disponible dans les NFAs. Ainsi, nous définissons formellement le concept d'**automate à pile non-déterministe (NPDA)** de la façon suivante :

Définition 2.39 (Automate à pile non-déterministe)**.** Un **automate à pile non-déterministe** est un sextuplet $\mathcal{S} = (Q, \Sigma, \Gamma, \delta, q_0, F)$, où :

1. Q est un ensemble fini d'**états**,
2. Σ est l'**alphabet d'entrée**,
3. Γ est l'**alphabet de la pile**,
4. $\delta : Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon \longrightarrow \wp(Q \times \Gamma_\varepsilon)$ est la **fonction de transition**,
5. $q_0 \in Q$ est l'**état initial**, et
6. $F \subseteq Q$ est l'ensemble des **états finaux**.

Notons que la fonction de transition δ d'un automate à pile définit en entrée un état de départ ($q_i \in Q$), un symbole d'entrée ($a \in \Sigma_\varepsilon$) et un symbole au sommet de pile ($x \in \Gamma_\varepsilon$), soit un triplet de la forme (q_i, a, x).

De même, observons que la sortie de la fonction de transition est un ensemble de paires ordonnées de la forme (q_j, y) de $\wp(Q \times \Gamma_\varepsilon)$. Ceci indique qu'étant donnée une entrée (q_i, a, x), la fonction de transition peut définir en sortie : 1. plusieurs états d'arrivée parmi les états de Q, 2. pour tout état d'arrivée $q_j \in Q$, un symbole $y \in \Gamma_\varepsilon$ à mettre à jour au sommet de pile lorsque le changement d'état est réalisé.

Par exemple, la règle de transition $\delta(q_i, a, x) = (q_j, y)$, précise que lorsque l'automate se trouve à l'état q_i et lit a sur l'entrée, il peut remplacer le symbole x au sommet de pile par y et passer dans l'état q_j. L'étiquette d'une telle transition est notée $a, x \to y$ est illustrée à la figure 2.8.

1. Rappelons que tout automate fini déterministe est aussi un automate fini non-déterministe

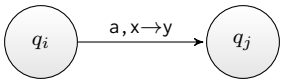

Figure 2.8 – Étiquette de transition d'un automate à pile

Définition 2.40 (Transition d'un automate à pile non-déterministe)**.** Si $\delta(q_i, a, x) = (q_j, y)$ est une règle de transition de $\mathcal{S}$, alors n'importe quel terme a, x ou y peut être égal à ε :

1. Si a est ε, soit $\varepsilon, x \to y$: aucune opération de *lecture* d'un symbole en entrée est effectuée.
2. Si x est ε, soit $a, \varepsilon \to y$: aucune opération d'*empilement et dépilement* d'un symbole dans la pile est effectuée.
3. Si y est ε, soit $a, x \to \varepsilon$: aucune opération d'*empilement* d'un symbole dans la pile est effectuée.

De plus, si à partir d'un état de départ q_i, en lisant un symbole a, l'automate a plusieurs états d'arrivée, par exemple dans une règle de transition de la forme $\delta(q_i, a, x) = \{(q_j, y), (q_k, z)\}$, cela indique que deux choses peuvent arriver, soit l'automate remplace le symbole x en haut de la pile par y et passe à l'état q_j, soit il remplace le symbole x en haut de la pile par z et passe à l'état q_k. Une telle configuration est illustrée à la figure 2.9.

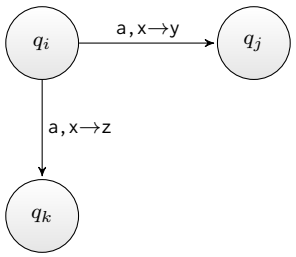

Figure 2.9 – Transition non-déterministe d'un automate à pile

Exemple 2.41.

$$\mathcal{S}_1 = (Q, \Sigma, \Gamma, \delta, q_0, F),$$

1. $Q : \{q_0, q_1, q_2, q_3\}$
2. $\Sigma : \{0, 1\}$
3. $\Gamma : \{0, 1\}$
4. δ : est décrite par la table de transition d'état suivante :

5. q_0
6. $F : \{q_0, q_3\}$

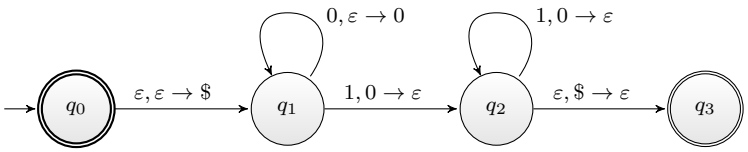

Figure 2.10 – Diagramme d'états-transitions d'un automate à pile qui reconnaît le langage $L = \{0^n 1^n \mid n \geq 0\}$

Définition 2.42 (Automate à pile déterministe)**.** Un automate à pile $\mathcal{A}$ est un **automate à pile déterministe** si la fonction de transition est définie par $\delta : Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon \longrightarrow (Q \times \Gamma_\varepsilon) \cup \{\varnothing\}$ [1] et elle satisfait la condition suivante : Pour chaque $q \in Q$, $a \in \Sigma$ et $x \in \Gamma$, une et une seule des sorties parmi les configurations suivantes n'est pas égal à $\varnothing$: $\delta(q, a, x)$, $\delta(q, a, \varepsilon)$, $\delta(q, \varepsilon, x)$ et $\delta(q, \varepsilon, \varepsilon)$

Définition 2.43 (Langage reconnu par un automate à pile)**.** Le **langage reconnu** par un automate à pile $\mathcal{A}$, noté $L(\mathcal{A})$, est égal à l'ensemble des mots acceptés par $\mathcal{A}$.

2.5 Langages contextuels

Les langages contextuels sont les langages engendrés par les grammaires contextuelles et reconnus par les automates linéairement bornés. Ces derniers sont équivalents à de machines de Turing (cf. sous-section 2.6.2) non déterministes. Nous définissons ci-dessous les grammaires contextuelles.

2.5.1 Grammaires contextuelles

Une **grammaire contextuelle**, aussi appelée grammaire dépendante du contexte, est une grammaire qui définit un langage contextuel. Considérons à présent une définition plus formelle.

Définition 2.44 (Grammaire contextuelle)**.** Une **grammaire contextuelle** est une grammaire formelle, c'est-à-dire un quadruplet $\mathcal{G} = (V, T, P, S)$, où (1) V et T sont des ensembles non vides et finis de symboles, (2) $V \amalg T$, $\Sigma = V \cup T$, (3) P est un ensemble fini de règles de réécriture et (4) $S \in V$ est l'axiome de la grammaire.

Comme pour le cas des grammaires régulières et non contextuelles, l'ensemble fini P des règles constitue l'élément central d'une grammaire contextuelle. Il est défini comme :

$$P = \{(\varphi A \psi, \varphi \omega \psi) \mid A \in V\varphi, \psi \in \Sigma^*, \omega \in \Sigma^+\}$$

1. Ceci indique que l'automate peut soit réaliser un seul mouvement ou ne pas réaliser d' actions en renvoyant $\varnothing$.

Les règles sont alors des paires ordonnées $(\varphi A\psi, \varphi\omega\psi)$ de la forme :

$$\varphi A\psi \to \varphi\omega\psi$$

où A est une variable, φ, ψ sont des suites de mots contenant n variables et/ou symboles terminaux ($n \geq 0$) et ω est une suite de mots contenant au moins une variable ou un symbole terminal. Remarquons que comme dans le cas d'une CFG, dans une grammaire contextuelle il est permis de remplacer une variable A par n'importe quelle suite non vide d'éléments ω du vocabulaire de la grammaire Σ. Cependant, pour que la réécriture soit acceptée, ω doit apparaître entre deux suites de mots, éventuellement vides, φ et ψ. Autrement dit, ω doit se trouver dans le *contexte* défini par $\varphi \dots \psi$.

2.6 Langages récursivement énumérables

Les langages récursivement énumérables, aussi dénommés généraux ou sans restriction, sont les langages engendrés par les grammaires contextuelles avec effacement et reconnus par les machines de Turing. Tous les types de langages présentés ci-dessus : réguliers, non contextuels, contextuels sont récursivement énumérables. Bien qu'il existe des langages récursivement énumérables qui ne sont d'aucun des types $(1, 2$ et $3)$. Cela étant dit, il n'existe pas d'algorithme pour dire si une chaîne w appartient à un langage défini pour une grammaire contextuelle avec effacement. Nous définissons ci-dessous les grammaires contextuelles avec effacement.

2.6.1 Grammaires contextuelles avec effacement

Une **grammaire contextuelle avec effacement**, aussi appelée grammaire sans restriction, est une grammaire qui définit un langage récursivement énumérable. Considérons à présent une définition plus formelle.

Définition 2.45 (Grammaire contextuelle avec effacement)**.** Une **grammaire contextuelle avec effacement** est une grammaire formelle, c'est-à-dire un quadruplet $\mathcal{G} = (V, T, P, S)$, où (1) V et T sont des ensembles non vides et finis de symboles, (2) $V \amalg T$, $\Sigma = V \cup T$, (3) P est un ensemble fini de règles de réécriture et (4) $S \in V$ est l'axiome de la grammaire.

À la différence de toutes les autres grammaires vues jusqu'à présent : régulières, non contextuelles et contextuelles, les grammaires contextuelles avec effacement n'ont pas d'autre restriction que d'avoir au moins une variable à gauche. L'ensemble fini P des règles est défini comme :

$$P = \{(\varphi A\psi, \omega) \mid A \in V\varphi, \psi, \omega \in \Sigma^*\}$$

Les règles sont alors des paires ordonnées $(\varphi A\psi, \omega)$ de la forme :

$$\varphi A\psi \to \omega$$

où A est une variable, et φ, ψ, ω sont des suites de mots contenant n variables et/ou symboles terminaux ($n \geq 0$). Remarquons que comme dans le cas d'une grammaire contextuelle, il est permis d'avoir n'importe quel élément à gauche de la règle. Cependant, il n'existe pas de restriction de contexte $\varphi \dots \psi$ dans l'élément à droite.

2.6.2 Machines de Turing

Une Machine de Turing (TM) est une machine abstraite qui est modélisée comme une extension d'automates à pile. Nous présentons brièvement ci-dessous leur définition [1].

Définition 2.46 (Machine de Turing)**.** Une **Machine de Turing** est un septuplet $\mathcal{M} = (Q, \Sigma, \Gamma, \delta, q_0, \mathfrak{b}, F)$, où :

1. Q est l'ensemble d'**états**,
2. $\Sigma \subseteq \Gamma$ est l'**alphabet d'entrée** qui ne contient pas $\mathfrak{b}$ [2],
3. Γ est l'alphabet des symboles du **ruban de lecture/écriture**,
4. $\delta : Q \times \Gamma \longrightarrow Q \times \Gamma \times \{L, R\}$ est la **fonction de transition**,
5. $q_0 \in Q$ est l'**état initial**,
6. $\mathfrak{b} \in \Gamma$ est un symbole spécial nommé **symbole blanc**, et
7. $F \subseteq Q$ est l'ensemble des **états finaux**.

Chaque transition de $\mathcal{M}$ est définie comme une paire ordonnée (q_i, γ_i) de $Q \times \Gamma$ où : q_i est l'état de départ, et γ_i est le symbole lu sur le ruban. La sortie de la fonction de transition $\delta(q_i, \gamma_i)$ est égale à un triplet (q_j, γ_j, D) de $Q \times \Gamma \times \{L, R\}$, dont le premier terme est l'état d'arrivée $q_j \in Q$, le deuxième est un symbole $\gamma_j \in \Gamma$ qui remplace γ_i et le troisième est un symbole de direction, L ou R, afin d'indiquer vers quelle position, gauche (L) ou droite (R), doit se déplacer la tête de lecture/écriture. Ceci est illustré dans la figure 2.11.

1. Nous invitons le lecteur à consulter Linz (2016, pp. 232–242) pour plus d'information et d'exemples détaillés concernant les machines de Turing.
2. Soit $\Sigma \subseteq \Gamma - \{\mathfrak{b}\}$

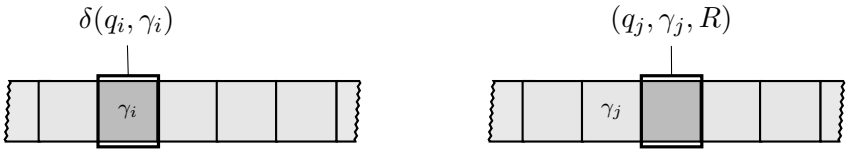

Figure 2.11 – Représentation d'un ruban de lecture/écriture d'une Machine de Turing qui est dans un état de départ q_i et qui lit le symbole γ_i pour passer dans un état d'arrivée q_j, en remplaçant γ_i par γ_j et en déplacent la tête de lecture vers la droite