Théorie des langages et des automates

Edition : 2006-2007

Auteur : Anna Desilles , Yannick Le Nir, Rachid Chelouah et Hervé de Milleville

Résumé : *Ce cours est une introduction à l'un des fondements de la science informatique : la théorie des langages*

- *Les concepts abordés sont les grammaires, la dérivation, les automates déterministes et non déterministes, les automates simples ou à pile, les langages réguliers et hors contexte.*
- *Nous ferons une brève introduction des langages contextuels et la machine de Turing*
- *Les processus algorithmiques d'analyse lexicale et d'analyse syntaxique sont traités.*

A la fin de ce cours, les étudiants ont un bagage suffisant pour aborder le cours de compilation, mais aussi toute sorte de problèmes pouvant se modéliser à l'aide d'automates

Présupposés : Cours d'algorithmique et notions de programmation.

EISTI : Département Informatique
Théorie des langages et des automates

TABLE DES MATIERES

1. Introduction à la théorie des langages

Une formation ingénieur en informatique débute généralement par des cours d'algorithmique et de programmation. Pour les aspects de programmation, il n'apparaît pas toujours rentable à l'étudiant de s'encombrer d'un cours de théorie du langage. C'est une double erreur :

- La qualité des œuvres d'un écrivain est fortement liée à la parfaite maîtrise de la langue. Cette maîtrise passe par la compréhension des rouages qui régissent cette langue. Il est en de même pour la conception de sources dans un langage de programmation.
- Plus généralement, l'ingénieur doit avoir une démarche scientifique dans la plupart des tâches qu'il aborde. C'est une garantie indispensable pour pouvoir s'approprier toutes les évolutions qu'il sera amené à rencontrer et donc assimiler rapidement. La seule démarche technique n'est pas suffisante.

La théorie des langages est née d'une tentative de modélisation des langues naturelles.

Même si aujourd'hui cette tentative n'a pas complètement aboutie, il est évident que l'une des raisons principales de l'expansion de cette théorie est sa parfaite adéquation avec la description des langages de programmation. Cette simple remarque suffit à expliquer la présence de ce cours dans une école d'ingénieur en informatique.

Il faut savoir que la recherche française dans ce domaine est mondialement reconnue (on parle de la **french school**). Nous devons citer ici les deux acteurs principaux : **M. Schützenberg et M. Nivat**.

Un programmeur parle à une machine à l'aide de langages de programmation. Comme pour les langues naturelles, les langages ont un alphabet, des mots et une grammaire.
Un source bien écrit doit donc être fait de phrases formées de mots du langages dont l'assemblage doit respecter des règles de grammaire.

La théorie du langage s'attache à définir les concepts et à établir les théorèmes concernant tous les problèmes de sources bien écrits dans un langage donné.

On trouvera donc cette théorie:

- Les concepts de langage et de grammaire,
- La catégorisation des grammaires et par la même des langages,
- Les propriétés des différentes catégories de langages et de grammaires,
- Les concepts algorithmiques permettant de chercher et/ou valider les mots et les phrases d'un langage : les différents types d'automates.

Nous aborderons aussi le processus d'analyse de sources d'un langage de programmation et nous situerons la théorie des langages dans ce processus.

- Analyse lexicale
- Analyse syntaxique
- Analyse sémantique

Nous donnerons enfin une démarche algorithmique complète de la partie du processus concernant l'analyse lexicale et l'analyse syntaxique.

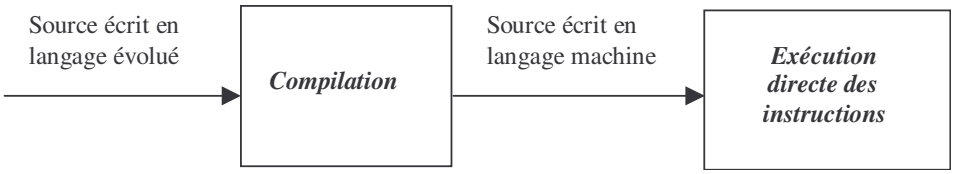
2. Le processus d'analyse de source

2.1. Description

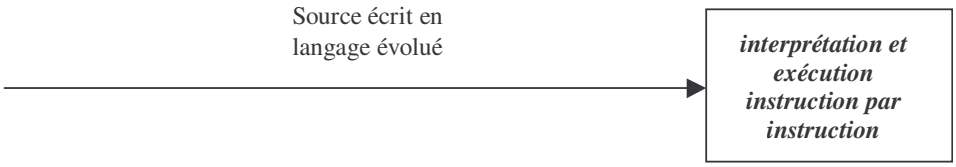
Une machine exécute des programmes écrits dans son propre langage : Langage machine. Les programmeurs conçoivent des logiciels en utilisant des langages plus évolués (plus proche de notre mode de pensée). Il existe donc nécessairement un processus de transformation pour que la machine puisse exécuter le programme.

On distingue deux types de processus :

Le processus 1 : Compilation puis exécution



Le processus 2 : Interprétation lors de l'exécution



Dans les deux processus, il y a toujours une étape d'analyse de source et par là même analyse d'instruction. Pour pouvoir traduire une instruction en langage machine, il faut que l'instruction en langage évolué soit comprise. On doit donc analyser cette instruction. Cette analyse se décompose en trois étapes :

- **L'analyse lexicale** : L'instruction est décomposée en mots du langage de programmation. Chaque mot est appelé lexème (en français) ou token (en anglais). En plus de la constitution des mots, on associe à chaque mot valide une catégorie dite syntaxique qui servira dans l'analyse lexicale. Lors de cette étape, on ne s'intéresse qu'à la constitution des mots et de leur validité.
- **L'analyse syntaxique** : L'analyse lexicale (si elle est valide) aboutit à une liste de lexèmes. L'analyse syntaxique vérifie que de la succession des catégories des lexèmes est valide
  Les deux phrases « Le garçon regarde la voiture » et « La voiture regarde le garçon » sont syntaxiquement valide car dans les deux cas on a la succession de catégories syntaxiques suivante : « .Déterminant Nom Verbe Déterminant Nom ».
- **L'analyse sémantique** : Une phrase syntaxiquement valide peut ne pas avoir de signification. Il s'agit ici de s'intéresser aux sens des lexèmes, de leur association et donc à ce que la phrase véhicule en terme de signification.
  La phrase « Le garçon regarde la voiture » est sémantiquement valide.
  La phrase « La voiture regarde le garçon » est sémantiquement invalide.
  L'instruction du langage de programmation Pascal « A := B AND C ; » est syntaxiquement valide mais n'est pas sémantiquement valide si l'une des variables A, B ou C n'est pas de type BOOLEAN.

2.2. Lien entre les processus d'analyse lexicale et analyse syntaxique

Une analyse lexicale reçoit une liste de symboles et après une analyse symbole par symbole produit une liste de mots avec leur catégories syntaxiques.
Une analyse syntaxique reçoit une liste de catégories syntaxiques et après une analyse catégorie par catégorie valide ou invalide syntaxiquement l'instruction.

Exemple n°1 : Etudions ces deux processus dans le cas d'une affectation numérique.

Nous définissons une affectation numérique avec les règles suivantes :

- R_1 : Les caractères autorisés sont les lettres, les chiffres, les caractères +, -, *, /, =, l'espace, le point,
- R_2 : Un identifiant est formé d'une lettre suivi de 0 à plusieurs chiffres ou lettres,
- R_3 : Les opérateurs sont les mots « + », « - » , « * » et « / »,
- R_4 : L'affectation est le mot « = »,
- R_5 : Un nombre peut être de la forme :
  - une suite de chiffres
  - une suite non vide de chiffres suivi d'un point suivi d'une suite non vide de chiffres
- R_6 : Un opérande est soit un opérateur soit un nombre,
- R_7 : Une affectation numérique peut être de la forme :
  - un identifiant suivi d'une affectation suivi d'un opérande,
  - un identifiant suivi d'une affectation suivi d'un opérande suivi d'un certain nombre de couples opérateur, opérande

Nous pouvons remarquer que les règles R_1, R_2, R_3, R_4 et R_5 servent à l'analyse lexicale et que les règles R_6 et R_7 concernent l'analyse syntaxique.
Par ailleurs les règles de R_2 à R_7 servent à définir des entités :

- Pour l'analyse lexicale, les entités sont un identifiant, un nombre, un opérateur ou une affectation. Chaque entité est une suite de caractères qui sont des lettres, des chiffres, +, -, *,/, =, l'espace et le point.
- Pour l'analyse syntaxique, les entités sont un opérande ou une affectation numérique. Chaque entité est une suite des catégories syntaxiques identifiant, nombre, opérateur ou affectation.

L'analyse lexicale et syntaxique ont donc en commun la définition d'entités qui sont des suites de symboles. Cette définition d'entités se fait à l'aide :

- d'un ensemble pour recenser les symboles autorisés
- d'un ensemble de règles de production pour définir les entités valides. Une entité est une suite de symboles.

Dans notre exemple :

- Les symboles sont :
  - Pour l'analyse lexicale, les lettres, les chiffres, les caractères +, -, *, /, =, l'espace, le point.
  - Pour l'analyse syntaxique, identifiant, affectation, nombre et opérateur.
- Les règles de production
  - R_2 à R_5 pour l'analyse lexicale
  - R_6 et R_7 pour l'analyse syntaxique

Nous venons de constater les points communs entre l'analyse lexicale et l'analyse syntaxique.
Cette partie commune aux deux processus est l'essence même de la **théorie des langages**. Le chapitre suivant aborde cette théorie.

Par contre l'analyse lexicale a pour but de décomposer une suite de caractères en une suite de lexèmes, tandis l'analyse syntaxique doit vérifier la validité syntaxique de cette suite de lexèmes.

3. Langages et grammaires

3.1. Les concepts de langage

Pour définir un langage, on a plusieurs possibilités :

- Après avoir défini son alphabet, on énumère les différents mots de ce langage. L'étude de ce cas n'a aucun intérêt .
- En plus de définir son alphabet, on énonce des règles de production pour définir les mots valides de ce langage. On parle alors de **grammaire** et de langage décidable.

La théorie des langages porte :

- Sur la définition de grammaires permettant d'engendrer des langage.
- Sur le classement de ces grammaires par catégories et l'étude des propriétés de ces catégories.
- Sur l'étude des automates qui sont les supports algorithmiques des grammaires pour vérifier qu'un mot donné est un mot - du langage : automate déterministe et indéterministe, automate simple automate à pile et machine de Turing.
- Sur l'indécidabilité de certains langages : ie aucune grammaire ne peut les engendrer.

Définition i) Alphabet et vocabulaire
Un alphabet A est un ensemble de symboles. Si les symboles sont déjà des suites de caractères, on peut alors parler aussi de vocabulaire

Définition ii) Mot et vocabulaire
Un mot associé à un alphabet A est une suite éventuellement vide de symboles de cet alphabet. Le mot vide sera noté ε. Dans le cas où A est un vocabulaire, on ne parlera pas de mots mais de phrase.

Définition iii) Longueur d'un mot
Soient A un alphabet et w un mot associé à cet alphabet : On définit la longueur de w comme le nombre de symboles contenus dans w. On note cette longueur $| w |$.

Définition iv) Les ensembles A^+ et A^*
Soit A un alphabet : On définit A^+ comme l'ensemble des mots associés à A tels que $| w | > 0$ et A^* comme l'ensemble des mots associés à A tels que $| w | \geq 0$

Définition v) Langage
Soit A un alphabet : un langage est un sous ensemble de A^* .

3.2. Les concepts de grammaire

3.2.1 Présentation du concept

Une grammaire permet de générer les mots d'un langage à l'aide de règles de production. Nous donnons ci-dessous quelques notations pour définir plus tard une grammaire de façon formelle.

Notations :
- Quand on voudra dans une expression séparer deux alternatives d'un choix, **on utilisera le symbole « | » .**
- Si E est un ensemble alors E^+ est l'ensemble des suites de 1 à plusieurs éléments de E.
- De même E^* est l'ensemble des suites de 0 à plusieurs éléments de E.

Définition vi) Grammaire
Une grammaire est un quadruplet G = <T, N, S, P> tel que
-T est un ensemble dit de terminaux
- N est un ensemble dit de non terminaux,
- S élément de N dit axiome de la grammaire
- P est un ensemble de règles de production. Une règle de production est de la forme :
$X \rightarrow Y$ où $X \in (T \cup N)^+$ et $Y \in (T \cup N)^*$

Le langage déduit de la grammaire :
Comme on va le voir plus loin, nous allons par combinaison successives des règles dites de production fabriquer les mots du langage. Le rôle de chaque élément d'une grammaire, dans le langage généré, est :

- L'ensemble T représente l'alphabet de notre langage.
- Les éléments de N sont appelés non terminaux car ils représentent des mots intermédiaires qui ne font pas partie des mots du langage mais qui servent à leur construction par combinaisons successives des règles de production.
- Dans chaque combinaison de règles pour former un mot, S représente le membre gauche de la première règle de la combinaison.

Nous allons illustrer ce concept de grammaire avec un exemple.
Exemple n° 2 : G = <T, N, S, P>
T = {identifiant, opérateur, nombre}
N = {opérande, expression}
S = expression
P = { opérande → identifiant | nombre, expression → opérande | expression opérateur expression }

La suite « identifiant opérateur nombre » est un mot (plutôt une phrase du langage) défini par cette grammaire car :

1) « expression » est l'axiome de la grammaire
2) on part de la règle « expression → expression opérateur expression »
3) on applique deux fois la règle « expression → opérande » et on en déduit :
« expression opérateur expression → opérande opérateur opérande »
5) on applique la règle « opérande → nombre » et on en déduit :
« opérande opérateur expression → **opérande opérateur nombre** »

La phrase « **opérande opérateur nombre »** n'est formée que d'éléments terminaux. Cette phrase est donc une phrase du langage.
Nous allons dans ce qui suit généraliser ce processus de génération de mots (ou de phrases).

3.2.2 Génération de mots par dérivation

Définition vii) Règle terminale

Soit G = <T, N, S, P> une grammaire et R ∈ P. R est une règle terminale si son membre à droite est un élément de T^*.

Dans l'exemple n° 2, la règle « opérande → identifiant » est une règle terminale

Définition viii) Dérivation directe

Soit G = <T, N, S, P> une grammaire. u ∈ $(T \cup N)^*$ se dérive directement en v ∈ $(T \cup N)^*$ selon G, si et seulement si $u = u_1 M u_2$, $v = u_1 N u_2$ et $M \rightarrow N$ est une règle de production.
On notera la dérivation directe de u en v par l'expression $u \rightarrow v$.

Dans l'exemple n° 2, si on prend « u = expression opérateur expression », « u_1 = expression opérateur », « $u_2 = \varepsilon$ », « *M = expression* » et « *N = opérande* » alors « v = expression opérateur opérande » dérive directement de u.
C'est la règle « expression → opérande » qui permet de dériver.

Définition ix) Dérivation

Soit G = <T, N, S, P> une grammaire. u ∈ $(T \cup N)^*$ se dérive en v ∈ $(T \cup N)^*$ selon G, si et seulement $\exists k \in N\ k > 0$ et $\exists u = u_1, u_2, ..., u_k = v$ et $\forall i < k\ u_i \rightarrow u_{i+1}$.

- On notera la dérivation de u en v par l'expression $u \xrightarrow{*} v$.

Dans l'exemple n° 2 :
si on prend « u = expression opérateur expression » et « v = opérande opérateur opérande » alors $u \xrightarrow{*} v$.
En effet, en appliquant deux fois la règle « expression → opérande », on obtient :
expression opérateur expression → opérande opérateur expression → opérande opérateur opérande

Définition x) Langage engendré par une grammaire

Soit G = <T, N, S, P> une grammaire. On appelle langage engendré par G en partant de S, le langage défini par $L_G(S) = \{ u \in T^* / S \xrightarrow{*} u \}$

Dans l'exemple n° 2 $L_G(S) = \{ u \in T^* / u =$(identifiant | nombre) (opérateur (identifiant | nombre)* }

Si on reprend l'analyse lexicale de l'exemple n° 1, on avait les règles suivantes :

- R_2 : Un identifiant est formé d'une lettre suivi de 0 à plusieurs chiffres ou lettres,
- R_3 : Les opérateurs sont les mots « + », « - » , « * » et « / »,
- R_4 : L'affectation est le mot « = »,
- R_5 : Un nombre peut être de la forme :
  - - une suite de chiffres
  - - une suite non vide de chiffres suivi d'un point suivi d'une suite non vide de chiffres

Une grammaire G = <T, N, S, P> de ce langage peut être définie par :

- T = lettre ∪ chiffre ∪ {+, -, *, /, = }
- N = { mot, nombre, opérateur, nombre, identifiant }
- S = mot
- P = {
- mot → opérateur**,** opérateur → **+ | - | * | /** ,=
- mot → nombre, nombre → (chiffre$^+$) | (chiffre$^+$).(chiffre$^+$),
- mot → identifiant, identifiant → lettre| identifiant (lettre|chiffre),
- }

Dans cet exemple $L_G(S)$ = { +, -, *, /, =, chiffre$^+$, chiffre$^+$.chiffre$^+$, lettre(lettre|chiffre)* }

-

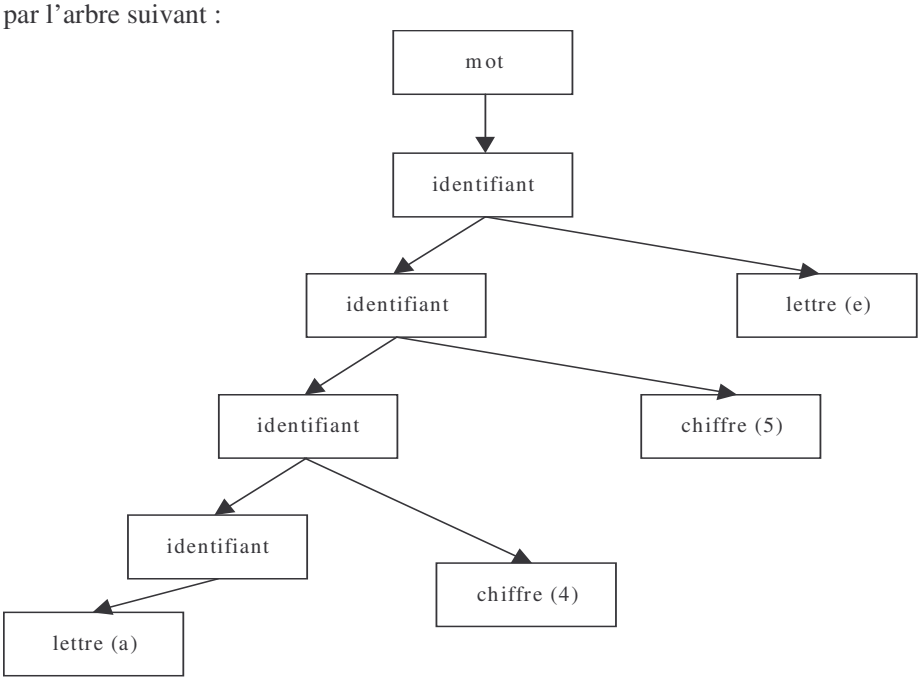
3.2.3 Génération de mots par arbre syntaxique

Une dérivation de règles de production pour générer un mot peut être représentée par un arbre.

Reprenons la grammaire relative à l'analyse lexicale d'une affectation numérique :
Une grammaire G = <T, N, S, P> de ce langage peut être définie par :

- T = lettre ∪ chiffre ∪ {+, -, *, /, ., = }
- N = { mot, nombre, opérateur, nombre, identifiant }
- S = mot
- P = {
- mot → opérateur**,** opérateur → **+ | - | * | /** ,=
- mot → nombre, nombre → (chiffre$^+$) | (chiffre$^+$).(chiffre$^+$),
- mot → identifiant, identifiant → lettre| identifiant (lettre|chiffre),
- }

« a45e » est un mot du langage généré par cette grammaire. En effet, on peut représenter la fabrication de cet identifiant par l'arbre suivant :



Chaque nœud avec ses fils directs représentent l'application d'un règle de production. Pour reconstituer le mot, il faut lire les feuilles de l'arbre de gauche à droite.

Définition xi) Arbre syntaxique

Un arbre syntaxique associé à une grammaire est un arbre dont :

- La racine contient l'axiome de la grammaire

- Chaque nœud qui n'est pas une feuille est le membre gauche d'une règle de production, ses fils sont les différents membres droits de cette même règle et ceux ci apparaissent dans le même ordre que dans la règle

- Chaque nœud feuille est un élément terminal de la grammaire.

Théorème I) Théorème : Un mot d'un langage ⇔ un arbre syntaxique
Soit G = <T, N, S, P> une grammaire et Ar un arbre syntaxique déduit de cette grammaire. Le mot obtenu en prenant de gauche à droite les différentes feuilles de cet arbre Ar est un mot du langage associé à la grammaire G.

4. Grammaires et automates d'états finis

4.1. La classification de Chomsky

Définition xii) Langage décidable
Un langage est décidable s'il peut être engendré par une grammaire.

Les langages décidables (ie : qui peuvent être engendrés par une grammaire) ont été classifiés par M. Chomsky en quatre types.

- Les langages de type 3 : rationnels ou réguliers
- Les langages de type 2 : algébriques ou hors contexte
- Les langages de type 1 : sensibles au contexte
- Les langages de type 0 : tous les autres décidables

Définition xiii) Grammaire de type 3 : rationnelle ou régulière
Une grammaire G = <T, N, S, P> est de type 3 si les règles de production sont de la forme :

$A \rightarrow \alpha$ **où** $A \in N$ **et** $\alpha \in T^*$
$A \rightarrow B\,\alpha$ **(ou.** $\alpha\,B$**)** **où** $A,B \in N$ **et** $\alpha \in T^*$

Définition xiv) Langage de type 3
Un langage est de type 3 s'il peut être engendré par une grammaire de type 3.

Si on reprend l'analyse lexicale de l'exemple n° 1, la grammaire G = <T, N, S, P> de ce langage est définie par :

- T = lettre ∪ chiffre ∪ {+, -, *, /, ., = }
- N = { mot, nombre, opérateur, nombre, identifiant }
- S = mot
- P = {
- mot → opérateur**,** opérateur → **+ | - | * | /** ,
- mot → nombre, nombre → (chiffre$^+$) | (chiffre$^+$).(chiffre$^+$),
- mot → identifiant, identifiant → lettre| identifiant (lettre|chiffre),
- }

Cette grammaire est clairement de type 3

Définition xv) Grammaire de type 2 : algébrique ou hors contexte
Une grammaire G = <T, N, S, P> est de type 2 sont des grammaires dont les règles de production sont de la forme :

$A \rightarrow \alpha$ **où** $A \in N$ **et** $\alpha \in (N \cup T)^*$

Définition xvi) Langage de type 2
Un langage est de type 2 s'il peut être engendré par une grammaire de type 2.

Si on reprend l'exemple n° 2 sans l'affectation, la grammaire G = <T, N, S, P> de ce langage est définie par :

- T = {identifiant, opérateur, nombre}
- N = {opérande, expression}
- S = expression
- P = { opérande → identifiant | nombre, expression → opérande | expression opérateur expression }
-

Cette grammaire est de type 2 à cause de la règle « expression → expression opérateur expression ».
Par contre si on remplace cette règle par les deux règles suivantes :

- morceau → opérateur opérande
- expression → expression morceau

On peut constater que la nouvelle grammaire est de type 3 et que cette grammaire engendre le même langage.

Théorème : Une grammaire n'engendre qu'un seul langage. La réciproque est fausse.

Définition xvii) Grammaire de type 1 : sensible au contexte
Une grammaire G = <T, N, S, P> est de type 1 sont des grammaires dont les règles de production sont de la forme : u A v → u w v où A ∈ N, u,v ∈ T^* et w ∈ $(N \cup T)^*$

Définition xviii) Langage de type 1
Un langage est de type 1 s'il peut être engendré par une grammaire de type 1.

Définition xix) Langage de type 0
Les langages de type 0 sont les langages qui peuvent être engendrés par une grammaire où les règles de production sont quelconques.

Théorème : Les différentes familles de langages sont incluses les unes dans les autres.
$L_3 \subset L_2 \subset L_1 \subset L_0$

4.2. La reconnaissance de mots par les automates d'états finis

4.2.1 Le concept d'automate

Tout système observé dans la nature peut être modélisé par :

- - Un ensemble d'états : un état permet de décrire un système à un instant donné.
- - Un ensemble d'événements : le système communique avec l'extérieur en recevant des messages ou des événements.
- - Un ensemble de transitions : la réception d'un événement et le traitement de ce dernier fait évoluer le système. On est donc amené à recalculer l'état du système. Ce nouvel état peut être le même. Une transition est un triplet (et_i, evt, et_f) où est l'état et_i du système avant l'événement evt se réalise et est l'état et_f du système juste après.

Définition xx) Automate d'états finis
Un automate d'états finis est un triplet <E, ξ, ψ> où E est un ensemble fini dit d'états, ξ un ensemble fini dit d'événements et ψ : <E, ξ> → E une relation définissant les transitions.

Le concept d'automate peut être utilisé pour toute sorte de problèmes. La notion d'états et d'événements ne sont pas obligatoirement liés avec l'axe temps.

Définition xxi) transition d'un non événement

Dans certain cas, l'ensemble ξ contiendra un élément particulier dit non-événement et noté ε. Toute transition associé à ce non-événement sera appelé ε-transition.

Ce non événement permet, entre autre, de modéliser la situation relativement fréquente suivante :

Un système envoie un message à l'extérieur, attend une réponse et cette réponse n'arrive pas.

On s'en servira d'une autre façon dans le chapitre sur « **les langages hors-contexte** »

Définition xxii) Automate déterministe et indéterministe

Un automate d'états finis est déterministe si $\forall$ $(et_i,evt) \in <E, \xi>$, il existe au plus un élément $et_f \in E$ tel que $\psi(et_i,evt) = et_f$. Un automate qui n'est pas déterministe est dit indéterministe.

4.2.2 Reconnaissance de motifs

L'une des utilisations les plus connues des automates est la reconnaissance de motifs.

Définition xxiii) Ensemble de motifs

Soit G un ensemble et C une condition portant sur les éléments de cet ensemble.

On définit $M_C(G) = \{ g \in G / C(g)$ est vraie $\}$. Cet ensemble est appelé ensemble de motifs de G associés à C.

Pour certaines conditions C, on peut reconnaître les éléments $M_C(G)$ en utilisant des automates spécialisés dits à reconnaissance de motifs.

Illustrons notre propos sur un exemple.

On considère l'ensemble G des suites finies de nombres binaires (0 ou 1) et C la condition définie par C(g) : g contient un nombre pair de 0.

Si on considère l'automate à reconnaissance de motifs défini par :

- E = {Début, Pair, Impair }
- F = { Pair }
- ξ = { 0, 1 }
- ψ est définie par le tableau qui suit :

Evénement / Etat	0	1
Début	Impair	Pair
Impair	Pair	Impair
Pair	Impair	Pair

Pour reconnaître qu'une suite finie de nombres binaires a un nombre pair de 0, on peut initialiser l'automate à l'état « Début », puis lui soumettre successivement les différents éléments de la suite pour faire évoluer son état. Après ces différentes opérations, si l'état de l'automate est un élément de F alors la suite a un nombre pair de 0 et un nombre impair sinon.

Définition xxiv) Automate à reconnaissance de motifs
Un automate d'états finis à reconnaissance de motifs est un quintuplet <E, F, ξ, ψ, q_0> tel que :

- **<E, ξ, ψ> est un automate d'états finis**
- **F ⊆ E est un ensemble non vide dit d'états finaux**
- **q_0 ∈ E état dit initial**

Théorème II) Equivalence entre automates déterministes et indéterministes
Pour tout automate indéterministe à reconnaissance de motifs AIM = <E, F, ξ, ψ, q_0>, il existe un automate déterministe à reconnaissance équivalent.

Preuve : Nous construisons l'automate équivalent par récurrence. On notera ADM = <E', F', ξ', ψ', q'_0> ce nouvel automate.

Les événements sont le mêmes soit ξ' = ξ
Un élément de E' est un sous ensemble de E soit E' ⊆ P(E)

Construction par récurrence :

Etape 0 : on définit $G_0 = \{ \{ q_0 \} \}$ et on pose E' = ∅ et $q'_0 = \{ q_0 \}$.
Etape n (n ≥1) :

- ***On initialise G_n à l'ensemble ∅***
- ***∀ e ∈ G_{n-1} et ∀ evt∈ ξ, on pose e' = { e' ∈ E / ∃ e ∈ E tel que e' = ψ(e, evt) }.***
  - ***Si e' ≠ ∅ alors***
    - ***E' = E' ∪ { e' }.***
    - ***G_n = G_n ∪ { e' }.***
    - ***si e' ∩ F ≠ ∅ alors F' = F' ∪ { e' }.***
    - ***on ajoute une nouvelle transition e = ψ' (e', evt)***
- ***Si G_n ≠ ∅ alors on passe à l'étape n+1 sinon on s'arrête et l'automate ADM est terminé***

Le lecteur pourra vérifier que les automates ADM et AIM sont équivalents.

Remarque : L'automate ADM peut avoir jusqu'à $2^{card(E)}$ états. Il peut donc prendre bien plus de place en mémoire que son équivalent AIM. L'automate est bien sûr plus rapide pour détecter les motifs.

Définition xxv) Motif reconnaissable par un automate à reconnaissance de motifs
Soit $M_C(G)$ un ensemble de motifs et A = <E, F, ξ, ψ, q_0> un automate à reconnaissance de motifs.
L'automate A permet de reconnaître les motifs de $M_C(G)$ si et seulement si :

1) G ⊆ ξ*
2) (u_1 ... u_n) ∈ $M_C(G)$ ⇔ Il existe au moins une configuration où l'automate après avoir été initialisé à l'état q_0 puis soumis aux événements u_1, ..., u_n se trouve dans un état e tel que e ∈ F.

Rappel : Dans le cas d'un automate indéterministe, il peut exister plusieurs configurations possibles pour une même série d'événements.

Nous nous rappelons qu'un langage est un sous ensemble de mots. Dans la plupart des langages, ce sous ensemble est défini à l'aide d'une condition que doivent vérifier les mots. En d'autres termes si on note G = A^* où A est l'alphabet du

langage et C la condition à vérifier alors L = $M_C(G)$. Il apparaît clairement que **théorie des langages** et **théorie des automates** sont intimement liées. Les chapitres qui suivent vont parfaitement illustrer ce propos.

4.3. Les langages réguliers ou rationnels et automates d'états finis simples

4.3.1 Résultat fondamental

Un langage est un sous ensemble de mots associés à un alphabet A. Si ce sous ensemble peut être défini par une grammaire, il paraît légitime d'imaginer que le langage L puisse être défini par L = $M_C(A^*)$ où C est une condition définie sur A^* à l'aide des règles de la grammaire.

Théorème III) Reconnaissance d'un langage de type 3 par un automate simple
Les mots d'un langage de type 3 (régulier ou rationnel) sont reconnaissables par un automate d'états finis à reconnaissance de motifs. Un automate d'états finis à reconnaissance de motifs définit de façon unique un langage de type 3.

4.3.2 Application à l'analyse lexicale d'une affectation numérique

Reprenons la grammaire relative à l'analyse lexicale d'une affectation numérique :
Une grammaire G = <T, N, S, P> de ce langage peut être définie par :

- T = lettre ∪ chiffre ∪ {+, -, *, /, ., = }
- N = { mot, nombre, opérateur, identifiant }
- S = mot
- P = {
- mot → opérateur**,** opérateur → **+ | - | * | /** ,=
- mot → nombre, nombre → (chiffre$^+$) | (chiffre$^+$).(chiffre$^+$),
- mot → identifiant, identifiant → lettre| identifiant (lettre|chiffre),
- }

Pour reconnaître les mots de ce langage, on peut utiliser l'automate d'états finis suivant :

- E = {début, identifiant, entier, réel, opérateur, affectation, nombre, point }
- F = { opérande, nombre, opérateur, affectation }
- ξ = lettre ∪ chiffre ∪ {+, -, *, / , = , . , }
- ψ est définie par le tableau qui suit :

Evénement **Etat avant**	**a..z A..Z**	**0..9**	**+ - * /**	**=**	**.**	
Début	identifiant	entier	opérateur	affectation		Début
identifiant	identifiant	identifiant				identifiant
entier		entier			point	nombre
point		réel				point
réel		réel				nombre
opérateur						opérateur
affectation						affectation

Le symbole sera ajouté à la fin de chaque mot à tester afin de traiter les états intermédiaires « réel » et « entier » pour les transformer en « nombre »

Les cases vides indiquent une transition impossible. Dans ce cas, l'automate est définitivement dans un état dit « poubelle ».

Attention : cet automate permet de valider les mots du langage :

$L = \{lettre|(lettre|chiffre)^*\} \cup \{ chiffre^+.chiiffre^+\} \cup \{ +, -, *, /, = \}$

Mais en aucune manière, il ne permet de décomposer en mots une phrase composée de ces mots. Le problème d'analyse lexicale n'est donc pas entièrement résolu. Il faut se référer au chapitre « Processus d'analyse lexical ».

4.3.3 Application à l'analyse syntaxique d'une expression numérique

Si on reprend l'exemple n° 2 arrangé (sans l'affectation), la grammaire G = <T, N, S, P> de ce langage est définie par :

- T = {identifiant, opérateur, nombre}
- N = {opérande, morceau, expression}
- S = expression
- P = { opérande $\rightarrow$ identifiant | nombre, morceau $\rightarrow$ opérateur opérande
- expression $\rightarrow$ expression morceau, expression $\rightarrow$ opérande }
-

Cette grammaire de type 3 permet d'engendrer le langage L :

$L_S(G) = \{ (identifiant | nombre) (opérateur (identifiant | nombre))^* \}$

Pour reconnaître les mots de ce langage, on peut utiliser l'automate d'états finis suivant :

- E = {début, débutMorceau, Expression }
- F = { Expression }
- ξ = {identifiant, opérateur, nombre}
- ψ est définie par le tableau qui suit :

Evénement **Etat avant**	**identifiant**	**opérateur**	**nombre**
Début	Expression		Expression
DébutMorceau	Expression		Expression
Expression		débutMorceau	

Les cases vides indiquent une transition impossible. Dans ce cas, l'automate est définitivement dans un état dit « poubelle ».

4.4. Les langages hors contexte ou algébriques et automates d'états finis à pile

4.4.1 L'automate à pile

Pour reconnaître des mots d'un langage engendré par une grammaire de type 2, un automate simple n'est pas suffisant. En particulier, pour le langage $L = \{ a^n b^n / n \in N \}$, il faudrait que l'automate garde dans une mémoire le nombre de « a » de la partie du mot déjà testé pour pouvoir constater qu'il y a autant de b.
Nous allons donc définir un nouvel automate : L'automate à pile. C'est un automate d'états finis à reconnaissance de motifs auquel on a ajouté une pile pour mémoriser certaines informations intermédiaires lors du parcours du mot testé.

Définition xxvi) Automate à pile

Un automate à pile est défini par <E, ξ, ξ_p, ψ> où :

- **E représente les états de l'automate.**
- **F $\subseteq$ E les états finaux de l'automate.**
- **ξ représente les événements de l'automate.**
- **ξ_p représente les symboles que l'on peut stocker dans la pile.**
- **ψ une relation définie par : <E, ξ, ξ_p> $\rightarrow$ <E, $(\xi_p)^*$> pour définir les transitions.**
- **$e_0 \in$ E est l'état initial de l'automate lors de l'analyse d'un mot..**
- **$\perp \in \xi_p$ est le symbole initial placé dans la pile lors de l'analyse d'un mot.**
- **est un élément de ξ dit symbole de fin de mot**

Une transition est donc un quintuplet (et_i, κ, κ_p, et_f, w).

Une telle transition signifie que si l'automate est dans l'état et_i, que le haut de la pile est κ_p et que l'on reçoit l'événement κ alors :

- - On dépile κ_p et on empile w.
- - L'automate passe dans l'état et_f.

Tester l'appartenance d'un mot « m » à un langage en utilisant un automate à pile consiste à :

- Initialiser la pile avec le symbole $\perp$.
- Initialiser l'automate à l'état e_0.
- Soumettre à l'automate, soit des non événements soit chaque symbole du mot comme un événement.
- Tester l'état de l'automate et l'état de la pile. On peut avoir deux sortes d'acceptation
  - La pile est vide : acceptation sur pile vide
  - L'état de l'automate a la caractéristique « final » : acceptation sur état final

Les symboles $\perp$ et

Pour certains automates à pile, on aura besoin de différencier le cas où on constate que la pile est vide pour la première fois des autres cas où la pile est vide. En introduisant à l'initialisation de la pile, le symbole $\perp$, on résout ce problème car dans les deux cas, la pile a un contenu différent :

- 1^{er} cas : la pile contient le symbole $\perp$ et on dépile $\perp$
- $2^{ème}$ cas : la pile est vide et on ne dépile rien.

Nous verrons dans le paragraphe « **Cas particuliers de transitions** » que l'initialisation de la pile avec le symbole $\perp$ nous imposera d'ajouter à la fin de chaque mot à tester un symbole dit de fin de chaîne noté .

Cas particuliers de transitions

Pour résoudre certains cas, on supposera qu'il existe dans ξ_p le symbole ε pour traduire le néant.

- (et_i i, κ, ε, et_f, w) : on ne dépile rien et on empile w
- (et_i, κ, κ_p, et_f, ε) : on dépile et on n'empile rien

Dans le cas d'acceptation sur pile vide, l'automate à pile contient les transitions suivantes :

- (et, , $\perp$, et, ε). $\forall$ et $\in$ E

Cela signifie qu'une fois tous les symboles du mot à tester ont été soumis à l'automate, alors pour terminer on soumet le symbole de fin de mot et on dépile $\perp$ si le sommet de la pile est $\perp$.

Théorème IV) Reconnaissance d'un langage de type 2 par un automate à pile

Une grammaire est hors contexte si et seulement si les mots du langage engendré peuvent être validés par un automate à pile.

Preuve : Nous ne donnerons ici qu'un automate à pile qu'on peut définir à l'aide d'une grammaire hors contexte G = <T, N, S, P>

L'automate à pile est défini par :

- ***E = {p, q, r}***
- ***F = {r}***
- ***ξ = T ∪ { }***
- ***ξp = T ∪ N ∪ { ⊥ }***
- ***e0 = p***
- ***Les transitions sont définies par :***
  - ***(p, ε, ε, q, S) : on démarre avec l'axiome***
  - ***∀(A → ϕ) ∈ P (q, ε, A, q, ϕ) : Pour chaque règle de production, on définit une ε-transition qui consiste à ne pas changer d'état, de dépiler le membre gauche et d'empiler de droite à gauche les éléments du membre droit.***
  - ***∀x ∈ T (q, x, x, q, ε) : Pour chaque élément terminal, si on reçoit cet élément à traiter et que le haut de la pile est égal à cet élément, on le dépile sans changer d'état***
  - ***(q, , ⊥, r, ε) : Pour le symbole de fin de mot , si ⊥ se trouve au sommet de la pile alors on le dépile et on place l'automate dans l'état final r.***

Remarque : L'automate construit dans ce théorème ne doit servir qu'à la démonstration du théorème car il est non déterministe et non optimisé. On pourra comparer ses performances à d'autres automates dans les paragraphes suivants.

4.4.2 Cas du langage L = { $a^n b^n$ / n ∈ N }

Version n° 1 : version déduite du langage

Un automate à pile pour ce langage peut être défini par :

- E = {Début, A, B }
- F = ∅
- ξ = { a, b, }
- ξp = { a, ⊥ }
- e_0 = Début
- ψ est définie par le tableau qui suit :

Etat avant	événement	dépiler
Début	a	ε
A	a	ε
A	b	a
B	b	a
Début		⊥
B		⊥

Etat après	Empiler
A	a
A	a
B	ε
B	ε
Début	ε
B	ε

L'acceptation de cet automate à pile se fait sur pile vide.

Appliquons cet automate au mot « aaabbb »

	Etat actuel	Evénement	dépiler	Nouvel état	empiler	Contenu de la pile	Mot restant à tester
Etape 0				Début	⊥	⊥	a a a b b b
Etape 1	Début	a	ε	A	a	a ⊥	a a b b b
Etape 2	A	a	ε	A	a	a a ⊥	a b b b
Etape 3	A	a	ε	A	a	a a a ⊥	b b b
Etape 4	A	b	a	B	ε	a a ⊥	b b

Etape 5	B	b	a	B	ε	a ⊥	b
Etape 6	B	b	a	B	ε	⊥	
Etape 7	B		⊥	B	ε	∅	

Version n° 2 : Directement déduite de la grammaire

La grammaire de ce langage est définie par G = <T, N, S, P>

- T = {a, b}
- N = { R1 }
- S = R1
- P = { R1 → ε, R1 → a R1 b}

Un automate à pile déduit de cette grammaire peut être défini par :

- E = {p, q, r }
- F = {r}
- ξ = { a, b, }
- ξp = { a ,b, S, ⊥ }
- e_0 = Début
- ψ est définie par le tableau qui suit :

Etat avant	événement	dépiler
p	ε	ε
q	ε	R1
q	ε	R1
q	a	a
q	b	b
q		⊥

Etat après	Empiler
q	R1
q	a R1 b
q	ε
q	ε
q	ε
r	ε

Appliquons cet automate au mot « aaabbb »

	Etat actuel	Evénement	dépiler	Nouvel état	empiler	Contenu de la pile	Mot restant à tester
Etape 0				p	⊥	⊥	a a a b b b
Etape 1	p	ε	ε	q	R1	R1 ⊥	a a a b b b
Etape 2	q	ε	R1	q	a R1 b	a R1 b ⊥	a a a b b b
Etape 3	q	a	a	q	ε	R1 b ⊥	a a b b b
Etape 4	q	ε	R1	q	a R1 b	a R1 b b ⊥	a a b b b
Etape 5	q	a	a	q	ε	R1 b b ⊥	a b b b
Etape 6	q	ε	R1	q	a R1 b	a R1 b b b ⊥	a b b b
Etape 7	q	a	a	q	ε	R1 b b b ⊥	b b b
Etape 8	q	ε	R1	q	ε	b b b ⊥	b b b
Etape 9	q	b	b	q	ε	b b ⊥	b b
Etape 9	q	b	b	q	ε	b ⊥	b
Etape 10	q	b	b	q	ε	⊥	
Etape 11	q		⊥	r	ε	∅	

Remarque :

Nous pouvons constater que l'automate de la version n° 1 est plus performant que celui de la version n° 2.

4.4.3 Cas du langage L = { $a^n b^p$ / n,p ∈ N et n ≠ p }

Version n° 1

L'automate à pile pour ce langage est défini par :

- E = {Début, A+,B-, AB, B+ }
- F = {A+, B-, B+ }
- ξ = { a, b, }
- ξp = { a, ⊥ }
- e_0 = Début
  - ψ est définie par le tableau qui suit :

Etat avant	événement	dépiler	Etat après	Empiler
Début	a	ε	A+	ε
Début	b	ε	B+	ε
A+	a	ε	A+	a
A+	b	⊥	AB	ε
A+	b	a	B-	ε
B-	b	a	B-	ε
B-	b	⊥	AB	ε
AB	b	ε	B+	ε
B+	b	ε	B+	ε

- A+ : la partie du mot testé ne comporte que les lettres « a ».
- B- : la partie du mot testé commence par 1 ou plusieurs « a », se continue par 1 ou plusieurs « b ». Dans cette partie le nombre de « b » est strictement inférieur au nombre de « a ».
- AB : la partie du mot testé commence par 1 ou plusieurs « a », se continue par 1 ou plusieurs « b ». Dans cette partie le nombre de « b » est égal au nombre de « a ».
- B+ : la partie du mot testé commence par 0 ou plusieurs « a », se continue par 1 ou plusieurs « b ». Dans cette partie le nombre de « b » est strictement supérieur au nombre de « a ».

L'acceptation de cet automate à pile se fait sur état final.

Appliquons cet automate au mot « aabbb »

	Etat actuel	Evénement	dépiler	Nouvel état	empiler	Contenu de la pile
Etape 0				Début	⊥	⊥
Etape 1	Début	a	ε	A+	ε	⊥
Etape 2	A+	a	ε	A+	a	a ⊥
Etape 3	A+	b	a	B-	ε	⊥
Etape 4	B-	b	⊥	AB	ε	∅
Etape 5	AB	b	ε	B+	ε	∅
Etape 6	B+		ε	B+	ε	∅

Version n° 2 : Version directement déduite de la grammaire

La grammaire de ce langage est définie par G = <T, N, S, P>

- T = {a, b}
- N = { R1 }
- S = R
- P = { R→ R1, R1 → a, R1 → a R1 b, R → R2, R2 → b, R2 → a R2 b }

L'automate à pile pour ce langage est défini par :

- E = {p, q, r }
- F = {r}
- ξ = { a, b, }
- ξp = { a, b, R, R1, R2, ⊥ }
- e_0 = p
  - ψ est définie par le tableau qui suit :

Etat avant	événement	dépiler	Etat après	Empiler
p	ε	ε	q	R
q	ε	R	q	R1
q	ε	R	q	R2
q	ε	R1	q	a R1 b
q	ε	R1	q	a
q	ε	R2	q	a R2 b
q	ε	R2	q	b
q	a	a	q	ε
q	b	b	q	ε
q		⊥	r	ε

L'acceptation de cet automate à pile se fait sur état final.

Appliquons cet automate au mot « aabbb »

	Etat actuel	Evénement	dépiler	Nouvel état	empiler	Contenu de la pile	Mot restant à tester
Etape 0				p	⊥	⊥	a a b b b
Etape 1	p	ε	ε	q	R	R ⊥	a a b b b
Etape 2	q	ε	R	q	R2	R2 ⊥	a a b b b
Etape 3	q	ε	R2	q	a R2 b	a R2 b ⊥	a a b b b
Etape 4	q	a	a	q	ε	R2 b ⊥	a b b b
Etape 5	q	ε	R2	q	a R2 b	a R2 b b ⊥	a b b b
Etape 6	q	a	a	q	ε	R2 b b ⊥	b b b
Etape 7	q	ε	R2	q	b	b b b ⊥	b b b
Etape 8	q	b	b	q	ε	b b ⊥	b b
Etape 9	q	b	b	q	ε	b ⊥	b
Etape 10	q	b	b	q	ε	⊥	
Etape 11	q		⊥	q	ε	∅	∅

Remarque :
Nous pouvons encore constater pour les mêmes remarques que l'automate de la version n° 1 est plus performant que celui de la version n° 2.

4.4.4 Cas du langage L = { $a^n b^p$ / n,p ∈ N et n ≤ p ≤ 2n }

Version directement déduite de la grammaire
La grammaire de ce langage est définie par G = <T, N, S, P>

EISTI : Département Informatique
Théorie des langages et des automates

- T = {a, b}
- N = { R }
- S = R
- P = { R→ ε, R → ab, R → abb, R → a R b }

L'automate à pile déduit directement de cette grammaire est défini par :

- E = {p, q, r }
- F = {r}
- ξ = { a, b, }
- ξp = { a, b, R, ⊥ }
- e_0 = p
  - ψ est définie par le tableau qui suit :

Etat avant	événement	dépiler	Etat après	Empiler
p	ε	ε	q	R
q	ε	R	q	ε
q	ε	R	q	a b
q	ε	R	q	a b b
q	ε	R	q	a R b
q	a	a	q	ε
q	b	b	q	ε
q		⊥	r	ε

Appliquons cet automate au mot « aabbb »

	Etat actuel	Evénement	dépiler	Nouvel état	empiler	Contenu de la pile	Mot restant à tester
Etape 0				p	⊥	⊥	a a b b b
Etape 1	p	ε	ε	q	R	R ⊥	a a b b b
Etape 2	q	ε	R	q	a b b	a b b ⊥	a a b b b
Etape 3	q	a	a	q	ε	b b ⊥	a b b b
Etape 4	q	ε	R	q	a R b	a R b b b ⊥	a b b b
Etape 5	q	a	a	q	ε	R b b b ⊥	b b b
Etape 6	q	ε	R	q	ε	b b b ⊥	b b b
Etape 7	q	b	b	q	ε	b b ⊥	b b
Etape 8	q	b	b	q	ε	b ⊥	b
Etape 9	q	b	b	q	ε	⊥	
Etape 10	q		⊥	q	ε	⊥	