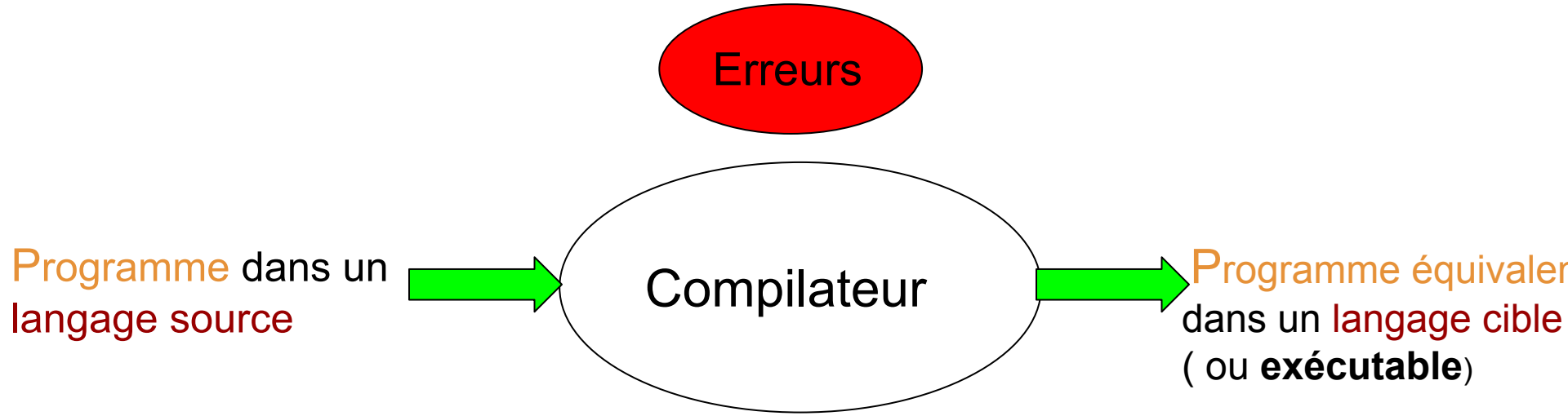
COMPILATION

Mohamed BOUHDADI & Faissal OUARDI

Théorie des langages et
COMPILATION
des langages de programmations impératifs

Syntaxe &
Sémantique de traduction
des langages de programmation impératifs

Fonction et Spécification d'un compilateur



1- Vérifier **si** le programme source ∈ au langage source : ANALYSE - ANALYSEUR

2- Traduire un programme bien formé en son programme équivalent exécutable :

SYNTHÈSE ou TRADUCTION ; SYNTHÉTISEUR ou TRADUCTEUR

Spécification d'un compilateur :

1- Comment spécifier le langage source ? Théorie des langages

2- Comment spécifier le langage cible ?

3- Comment vérifier l'appartenance d'une chaîne de caractères au langage source ? Théorie des automates

4- Comment traduire ou synthétiser une chaîne de caractères

(un programme) bien formée en son équivalent (unique) dans le langage cible

SYNTAXE et SÉMANTIQUE

La syntaxe d'un langage de programmation détermine ce qui constitue un programme valide du point de vue de la forme du texte.

Quelles séquences de caractères constituent des programmes ?

Théorie des langages et des automates : Classification de Chomsky
LA FORME DES CONCATÉNATIONS

La sémantique définit la signification d'un programme, c'est-`a-dire ce qu'il calcule, comment il le calcule.

Dans le cas de la compilation il s'agit d'une sémantique de traduction :
translational semantics

Sémantiques formelles:

La sémantique formelle est l'étude de la signification des programmes informatiques vue en tant qu'objets mathématiques

Les objets mathématiques dépendent des propriétés à connaître du programme.

L'objectif d'une sémantique formelle est de prouver qu'un programme est correct

Sémantiques usuelles d'un langage de programmation

Sémantique opérationnelle : la signification d'un programme est la suite des états de la machine qui exécute le programme.

Un programme est considéré comme la description d'un système de transition d'états.

Sémantique dénotationnelle :

Une fonction mathématique appelée *dénotation* est associée à chaque programme, et représente en quelque sorte son effet, sa signification.

Cette fonction prend par exemple pour argument l'état de la mémoire **avant** exécution, et a pour résultat l'état **après** exécution

Sémantique axiomatique :

Le programme n'est plus qu'un transformateur de propriétés logiques sur l'état de la mémoire.

si on a *p* vrai avant exécution, alors on a *q* vrai après. On ne s'intéresse plus à l'état précis de la mémoire tant que l'on sait dire si la propriété tient.

Sémantique algébrique

est une forme de sémantique axiomatique qui se base sur les lois algébriques pour la description et le raisonnement liés au sens d'un programme de façon formelle

Vers un cadre unique pour les différentes sémantiques :
UTP : Unifying Theories of Programming

Langages de Spécification formels

Un **langage de spécification** est un langage **forme**l utilisé pour décrire un système à un niveau beaucoup plus élevé qu'un langage de programmation, qui est utilisé pour produire un code exécutable pour un système.

Domaines d'utilisations :

- Ingénierie des protocoles de communications : Vérification et validation
- Conception électronique
- Les systèmes qualifiés de « critiques » :

  Une panne peut avoir des conséquences dramatiques :
  des morts ou des blessés graves, des dégâts matériels importants, ou des conséquences graves pour l'environnement

- les systèmes de transport : pilotage des avions, des trains, logiciels embarqués embarqués automobiles
- la production d'énergie : contrôle des centrales nucléaires;
- la santé: chaînes de production de médicaments, appareil médicaux (à rayonnement ou contrôle de dosages)
- le système financier : paiement électronique ;
- les applications militaires.

Exemples de langages de spécification formels :

Z, Object-Z, CSP (Calculus of Sequential Processes), T-CSP (Timed-CSP),

TCOZ (T-CSP Object-Z)

Calculus of Communicating Systems (CCS)

Language Of Temporal Ordering Specification (LOTOS)

SDL (Specification and Description Language), Estelle, Esterel

B, Event-B

Paradigmes et Langages de programmation

Un **langage de programmation** est une notation conventionnelle destinée à formuler des algorithmes et produire des programmes informatiques qui les appliquent.

Un langage de programmation est composé d'un alphabet, d'un vocabulaire, de règles de grammaire et de significations

Un **paradigme de programmation** est une façon d'approcher la programmation informatique et de traiter les solutions aux problèmes et leur formulation dans un langage de programmation approprié.

Un paradigme de programmation fournit (et détermine) la vue qu'a le développeur de l'exécution de son programme

Programmation impérative

La **programmation impérative** est un paradigme de programmation qui décrit les opérations en séquences d'instructions exécutées par l'ordinateur pour modifier l'état du programme.

L'état du programme à un instant donné est défini par le contenu de la mémoire centrale à cet instant.

La programmation impérative s'appuie sur le modèle des machines à états avec une mémoire centrale et des instructions qui modifient son état grâce à des affectations successives. (Machine de Turing et Architecture de von Neumann),

Exemples : le langage machine, Fortran, Algol, Cobol, Basic, Pascal, C, ADA, Smalltalk, C++, Objective C, Perl, Python, Php, Java, Javascript

Programmation fonctionnelle

La **programmation fonctionnelle** est un paradigme de type déclaratif qui considère le calcul en tant qu'évaluation de fonctions mathématiques.

Le paradigme fonctionnel n'utilise pas de machines à états pour décrire un programme, mais un emboîtement de fonctions qui agissent comme des « boîtes noires » que l'on peut imbriquer les unes dans les autres.

Exemples : Lisp, ML, Haskell, OCaml,

Programmation logique

La **programmation logique** définit les applications à l'aide d'un ensemble de faits élémentaires les concernant et de règles de logique leur associant des conséquences plus ou moins directes.

Ces faits et ces règles sont exploités par un démonstrateur de théorème ou moteur d'inférence, en réaction à une question ou requête.

Exemples : Prolog, Oz, Python : PyPy

Et bien d’autres paradigmes

Programmation concurrente,

Programmation par contraintes,

Programmation distribuée,

Programmation génétique,

métaprogrammation, etc

SYNTAXE : Théorie des langages et des automates
La hiérarchie ou classification de Noam Chomsky

TYPE 3 : Langages rationnels (réguliers)

TYPE 2 : Langages algébriques (indépendant du contexte)
Contexte-free

TYPE 1 : Langages contextuels (Contexte-sensitive)

TYPE 0 : Langages récursivement énumérables

$$\text{TYPE 3} \subsetneq \text{TYPE 2} \subsetneq \text{TYPE 1} \subsetneq \text{TYPE 0}$$

QUESTION : **Les langages de programmation sont de quel type ?**

REPONSE : Ils sont contextuels mais

ILs contiennent des sous langages qui sont indépendants du contexte et des sous langages qui sont réguliers

Langages impératifs : Instructions

Une **Instruction** est soit :

1- une **entrée** ou une **sortie** (input/output)

2- une séquence d'instructions : **instruction ; instruction**

3- une instruction d'assignement (l'affectation) :

Identificateur ← **Expression**

4 - une instruction d'alternative (choix conditionnel) :

si (Expression) alors Instruction sinon Instruction ;

5 - une instruction de répétition (bouclage ou itération) :

Tant que (Expression) faire Instruction ;

6- Appel de fonction (déclaration et définition) :

Identificateur (liste de paramètres effectifs) ;

La séquence, l'affectation, le choix et la répétition sont suffisantes pour implémenter toutes les solutions informatiques possibles

Ces 4 instructions constituent un système Turing Complet

Information - donnée : types

Types de base ou types primitifs :

alphabet, entiers, réels, booléens

Constructeurs de types :

1- Structure

2- Tableaux

3- Adresses :

LANGAGE DES IDENTIFICATEURS

Alphabet : ensemble fini de symboles ou lettres ou caractères, ou **TERMINAUX**
exemples : ASCII, {0,1}, {a, b, c }, { bleu, rouge, vert } sont des alphabets.

La seule information dont on dispose est l'ordre de chaque terminal

Opérations sur un alphabet la concaténation notée . , l'union notée + ou | et la répétition *

(a | b)*, a.(a | b).(a+b)* , (a+b).(a+b)*.b sont des langages

LANGAGE DES IDENTIFICATEURS

Question : Comment peut on définir un exemple de langage pour les identificateurs ?

Exemple : un identificateur est chaine de caractères qui commence par **une** lettre suivie de **zéro** ou **plusieurs** lettres ou chiffres et terminée par le symbole **$**

On introduire de nouveaux symboles NON-TERMINAUX pour simplifier la définition du langage des identificateurs ID :

lettre $\longrightarrow$ {a,b, ..z, A, B, ..Z } // $\Leftrightarrow$ [a..z,A..Z]

Chiffre $\longrightarrow$ {0,1,..,9} // $\Leftrightarrow$ [0..9]

ID → lettre.(lettre | chiffre)*.$ // * signifie zéro ou plusieurs : n ≥ 0

Une substitution des nouveaux symboles par ce qu'ils désignent donne :

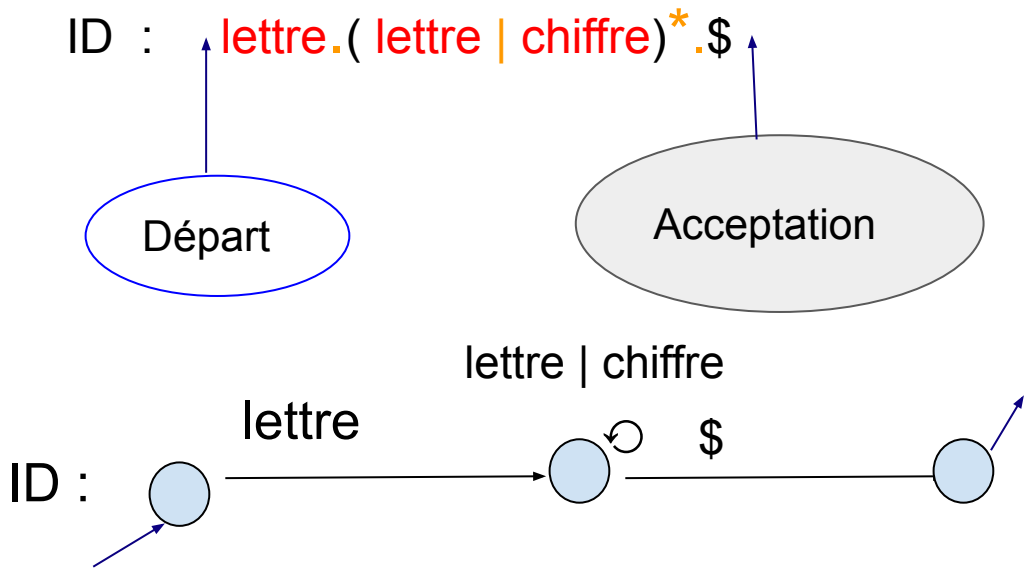
ID → {a,b, ..z, A, B, ..Z }. ({a,b, ..z, A, B, ..Z } | {0,1,..,9})*$

Les identificateurs sont définis indépendamment les uns des autres.

ID est défini en utilisant uniquement les symboles de l'alphabet (**les terminaux**).

Ce type de langage est dit **régulier ou rationnel**

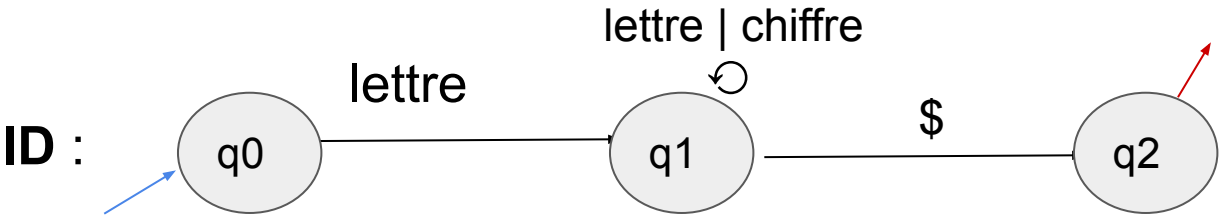
Le langage qui permet de décrire les langages réguliers : **les grammaires régulières**



Les mots a$, a1A$, Ba1?ad, 12a sont-ils des identificateurs ?

On gère l'avancée dans le mot en entrée et l'avancée dans le modèle des mots (le diagramme décrivant l'expression du langage)

L'action consiste à comparer le symbole courant qu'on lit en entrée avec un symbole courant dans le modèle.(qui permet de changer de configuration).



a$ q0,a,q1,$,q2 a$ ∈ ID

a1A$ q0,a,q1,1,q1,A,q1,$,q2 a1A$ ∈ ID

Ba1?ad q0 ,B,q1,a,q1,1,q1, ?, ∅ Ba1?ad ∉ ID

Analyse linéaire : structure d'automates à états finis

Le langage des expressions arithmétiques sur N:

E : ensemble des expressions arithmétiques

$\forall$ nb $\in$ N : nb $\in$ E

$\forall$ e1,e2 $\in$ E : (e1 + e2) $\in$ E et (e1 * e2) $\in$ E

NOTATION :

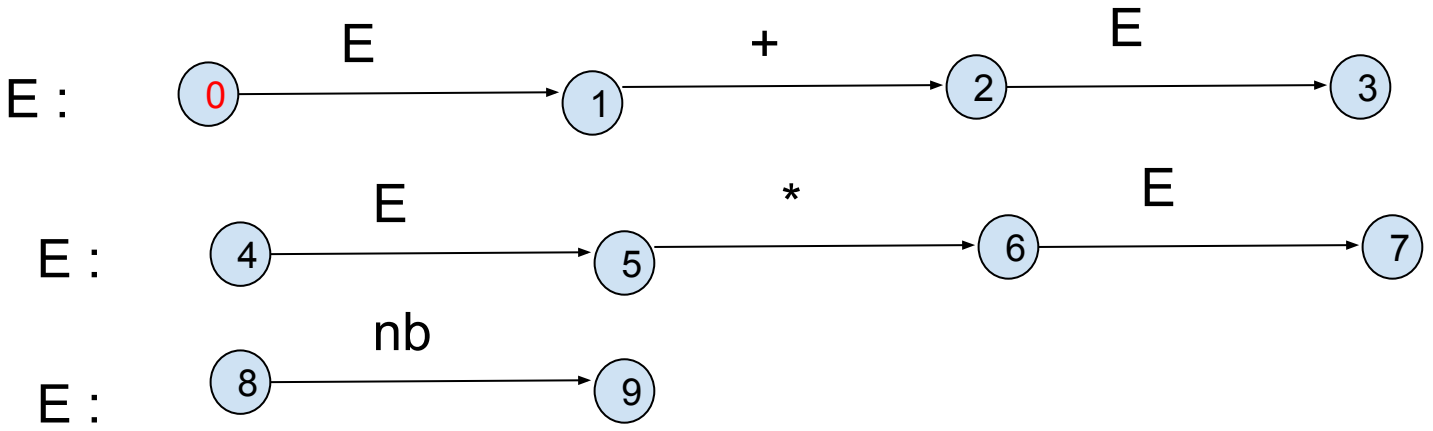
E ne peut être défini que de façon récursive

E $\longrightarrow$ nb | E$\longrightarrow$ E + E | E$\longrightarrow$ E * E

L'**alphabet =** {nb, +, *} , **E : non terminal** et désigne un langage,

Les parties droites désignent les concaténations de **terminaux** et **non terminaux** définissant la forme des expressions. Elles sont dites parties droites des **Productions** E

Formalisme : **< Terminaux, Non Terminaux, Non-Terminal Principal, Productions>**



mot = nb + nb * nb ∈ E ?

?

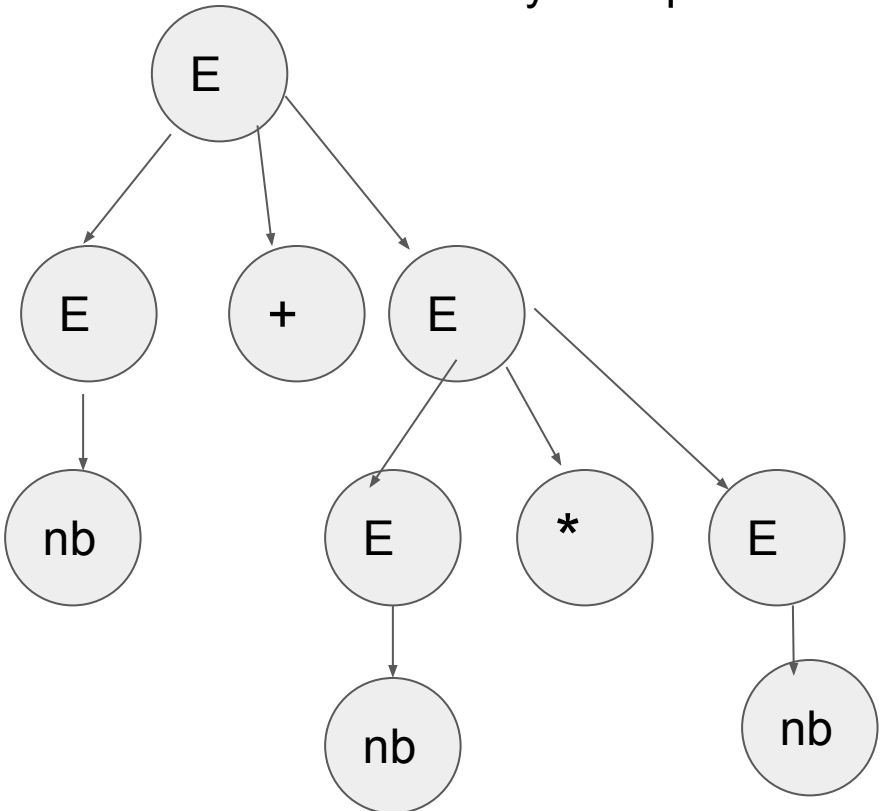
$E \Rightarrow E + E \Rightarrow nb + E \Rightarrow nb + E * E \Rightarrow nb + nb * E \Rightarrow nb + nb * nb$

La séquence d'états :

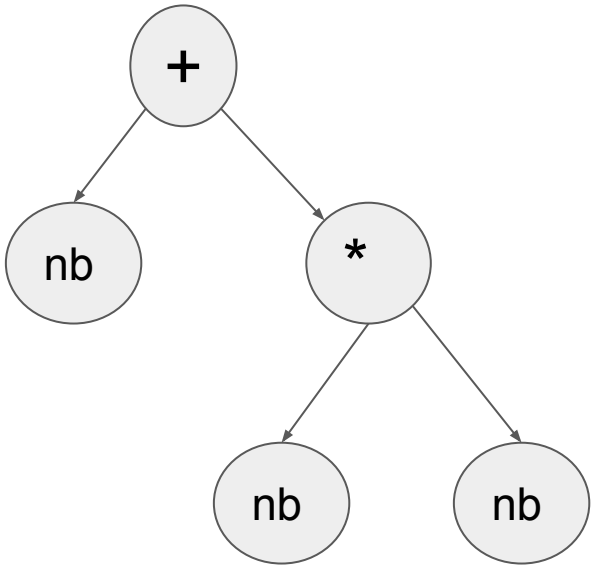
0, 8, 9, 1, 2, 4, 8, 9, 5, 6, 8, 9, 7, 3

On a besoin de sauvegarder l'état à partir duquel on va continuer l'analyse :
Structure arborescente - Automate + Pile

Arbre syntaxique

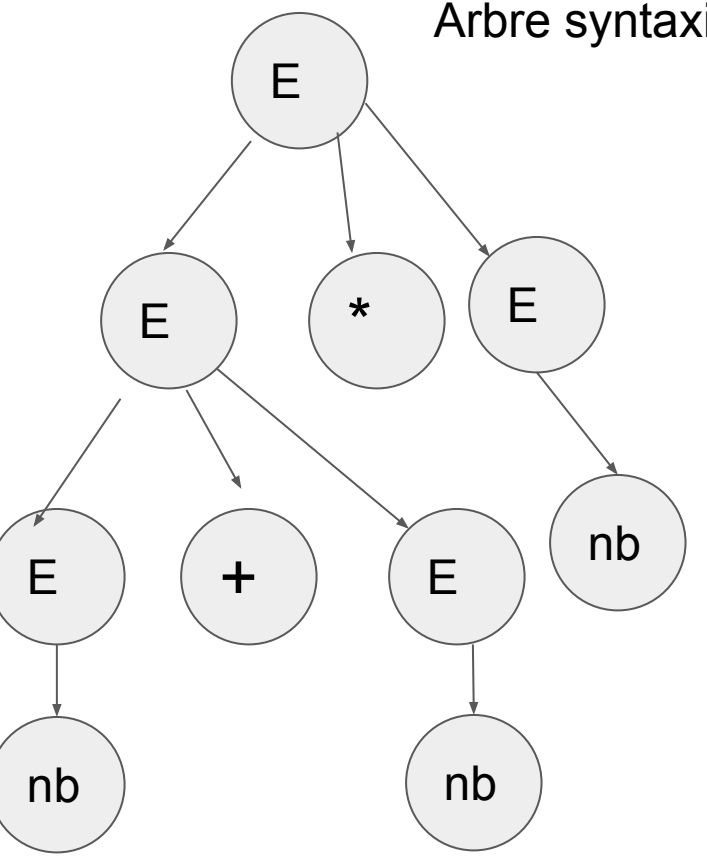


Arbre abstrait

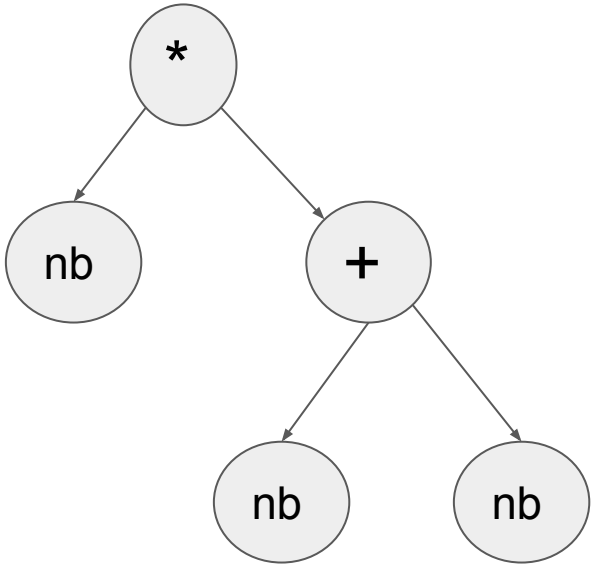


$E \Rightarrow E * E \Rightarrow E + E * E \Rightarrow nb + E * E \Rightarrow nb + nb * E \Rightarrow nb + nb * nb$

Arbre syntaxique



Arbre abstrait



Deux choix doivent être faits :

- Choix concernant le sens de lecture de la chaîne en entrée : de la gauche vers la droite ou l'inverse : left to right

- Choix concernant l'utilisation des productions : de la gauche vers la droite (Dérivation) ou de la droite vers la gauche (Réduction)

  . Approche descendante (top-down) :
  L'arbre syntaxique est construit du haut vers le bas

  . Approche ascendante (Bottom-Up)
  L'arbre est construit du bas vers le haut

Analyse descendante et analyse ascendante

LL(1) : Left to Right Scanning of the Input
LeftMost Derivation
1 lookahead (un seul de prévision)

LR(1) : Left to Right Scanning of the Input
Rightmost Derivation in Reverse
1 lookahead

Souvent, on utilise les deux approches

Le langage des expressions arithmétiques, des expressions logiques, des instructions, des parenthèses équilibrées ne peuvent être définis que de façon récursive

Ils sont dits **langages algébriques** ou **indépendants du contexte** (contexte-free) et sont décrits par des **grammaires algébriques** .

Les identificateurs, les entiers, les réels, les opérateurs arithmétiques, les opérateurs logiques, les opérateurs relationnels sont réguliers

Les langages indépendants du contexte (contexte-free) :

S un langage défini par la concaténation des langages A et B qui sont indépendants les uns des autres : pas de relation de voisinage

S $\rightarrow$ A.B A $\rightarrow$ **α** B $\rightarrow$ **β**

α et **β** concaténations de terminaux et de non terminaux définissant respectivement les langage A et B

Deux exemples de contraintes contextuelles :

1- Chaque identificateur doit être déclaré avant d'être utilisé.

2- Chaque appel de fonction doit être conforme à la déclaration en terme de nombre et types de paramètres

Langages de programmation sont contextuels (contexte-sensitive)

Deux exemples de contraintes contextuelles :

1- Chaque identificateur doit être déclaré avant d'être utilisé.

2- Chaque appel de fonction doit être conforme à la déclaration en terme de nombre et types de paramètres

S désigne un langage. S est un langage contextuel si les S productions ont la forme :

$\alpha S \beta \rightarrow \alpha \gamma \beta$

α, β et γ sont des concaténations de terminaux et de non terminaux

Les mots de S ont la forme γ à condition que le contexte à droite et celui à gauche de S sont respectivement α et β.

Hiérarchie de Chomsky

TYPE 3 : Langages rationnels - Grammaires régulières - Automate fini

TYPE 2 : Langages algébriques - Grammaires algébriques - Automate à pile

TYPE 1 : Langages contextuels - Grammaires contextuels - Automate linéairement borné **(Grammaires attribuées)**

TYPE 0 : Langages récursivement énumérables - Machine de Turing

Principes de Conception d'un Compilateur (phase d'analyse)

Le sous langage régulier (LEXIQUE) : micro-syntaxe

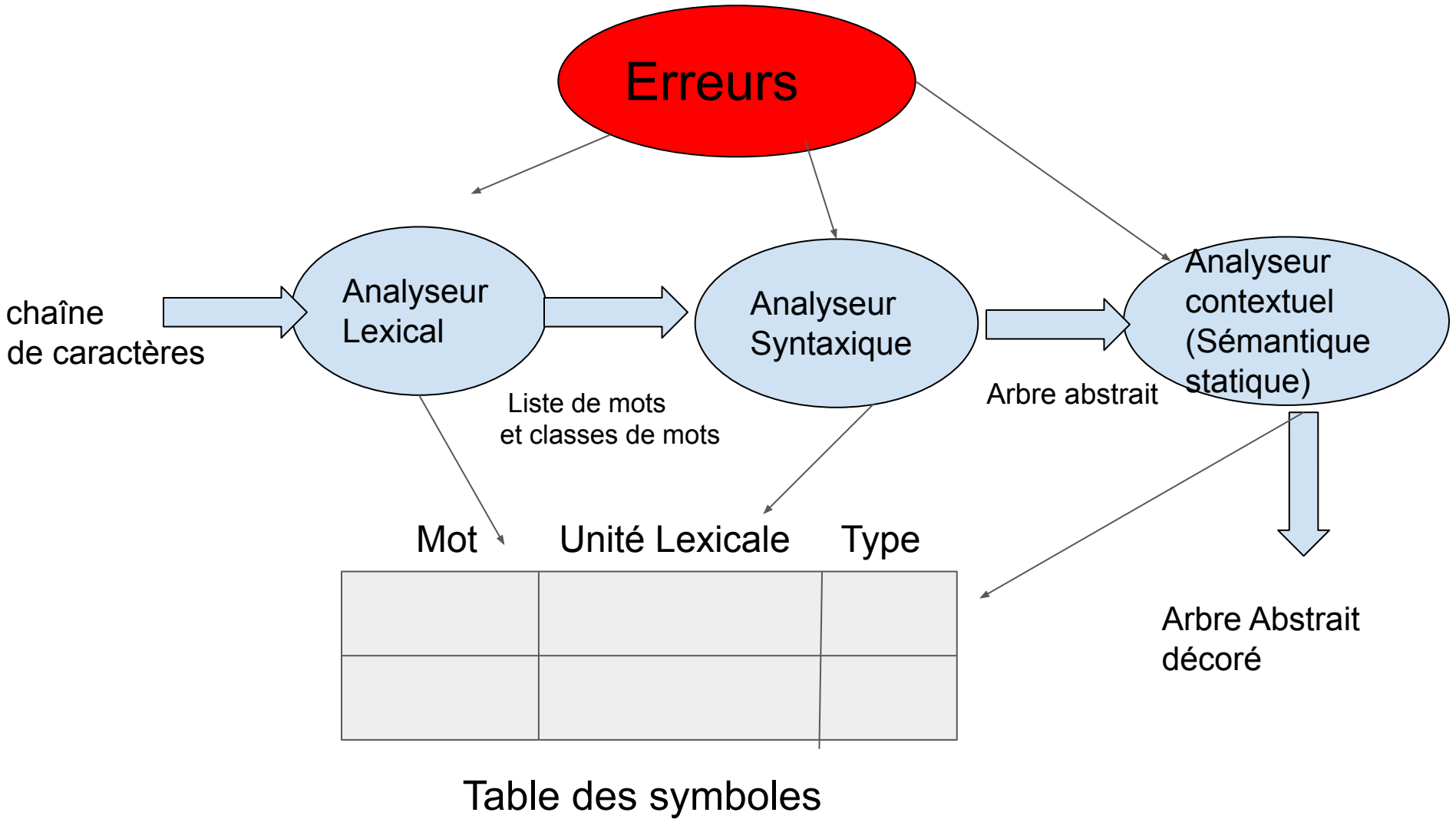
Le sous langage indépendant du contexte (SYNTAXE) : macro-syntaxe

Le sous langage contextuel (CONTEXTE) : sémantique statique

Grammaires attribuées : extension des grammaires indépendante du contexte.

On associe à chaque symbole un ensemble d'attributs

Un attribut peut être n'importe quelle type d'information

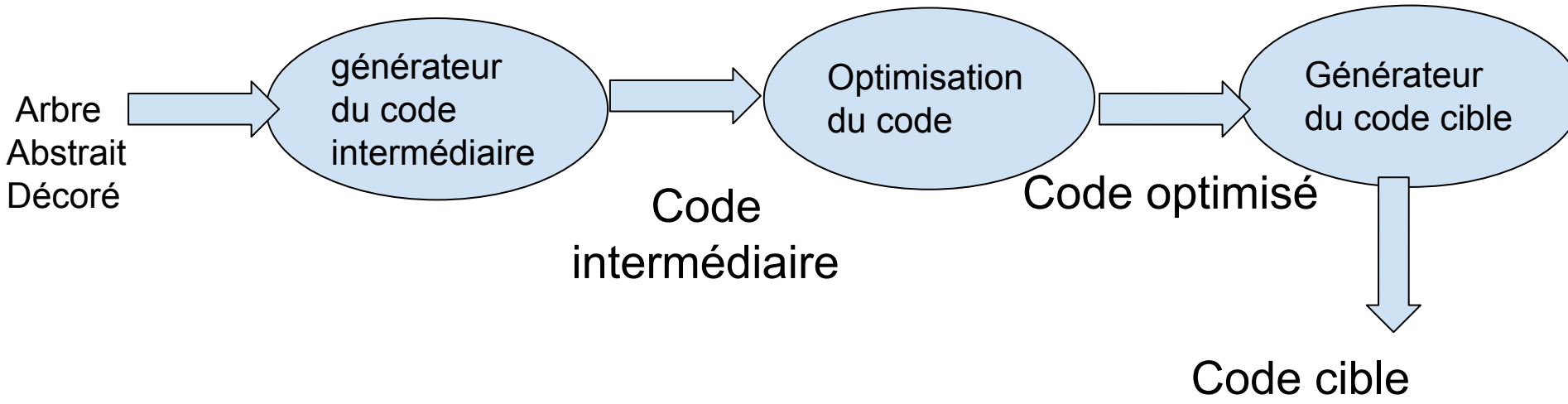

Erreurs
chaîne
de caractères
Analyseur
Lexical
Analyseur
Syntaxique
Analyseur
contextuel
(Sémantique
statique)
Liste de mots
et classes de mots
Arbre abstrait
Mot
Unité Lexicale
Type
Arbre Abstrait
décoré
Table des symboles

Principes de Conception d'un Compilateur (phase de synthèse)

- Le code doit être indépendant de toute machine particulière et de tout système d'exploitation : code intermédiaire (codes à trois adresses, C--)

- Le code doit être optimisé :
  Analyse de la complexité algorithmique
  Temporaires (évaluations des expressions)
  Les boucles imbriquées

Principes de Conception d'un Compilateur (phase de synthèse)



Sommaire :

1- Traduction des constituants de langages de programmation impérative

2- Préliminaires mathématiques

3- Langages réguliers (rationnels)

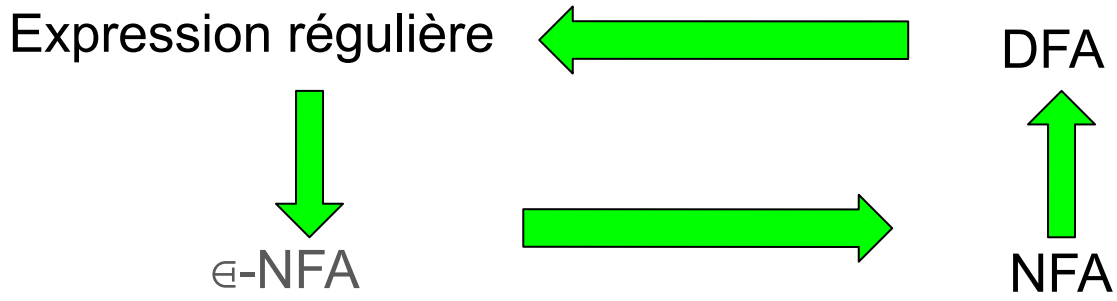
- Expressions régulières
- Automates à états finis :

. Déterministes DFA: Deterministic Finite Automaton

. Non-déterministes NFA : Non deterministic Finite Automaton

. Non-déterministes avec des transitions vides : $\in$-NFA

- Algorithmes (de transformations) :



- Automate Minimal
- Lemme de l'étoile : caractérisation des langages réguliers

- Analyseurs lexicaux manuels et automatiques (l'outil flex)

4- Langages algébriques (context free)

- Grammaires algébriques - langages algébriques
- Analyse par descente récursive : LL(1)
- Analyse ascendante : LR(1)
- Implémentation automatique : générateur d'analyseurs syntaxiques et Interprétation (l'outil BISON)