1. Introduction

1.1Qu’est ce que la compilation ?

D’un manière plus générale, un compilateur traduit un programme écrit dans un langage **source** en un programme écrit dans un langage **cible**.
Cette vision est en fait simplifiée. En effet, une phase importante qui intervient après la compilation pour obtenir un programme exécutable est la phase **d’éditions de liens**. . Le schéma 1.1 montre la chaîne de développement d’un programme.

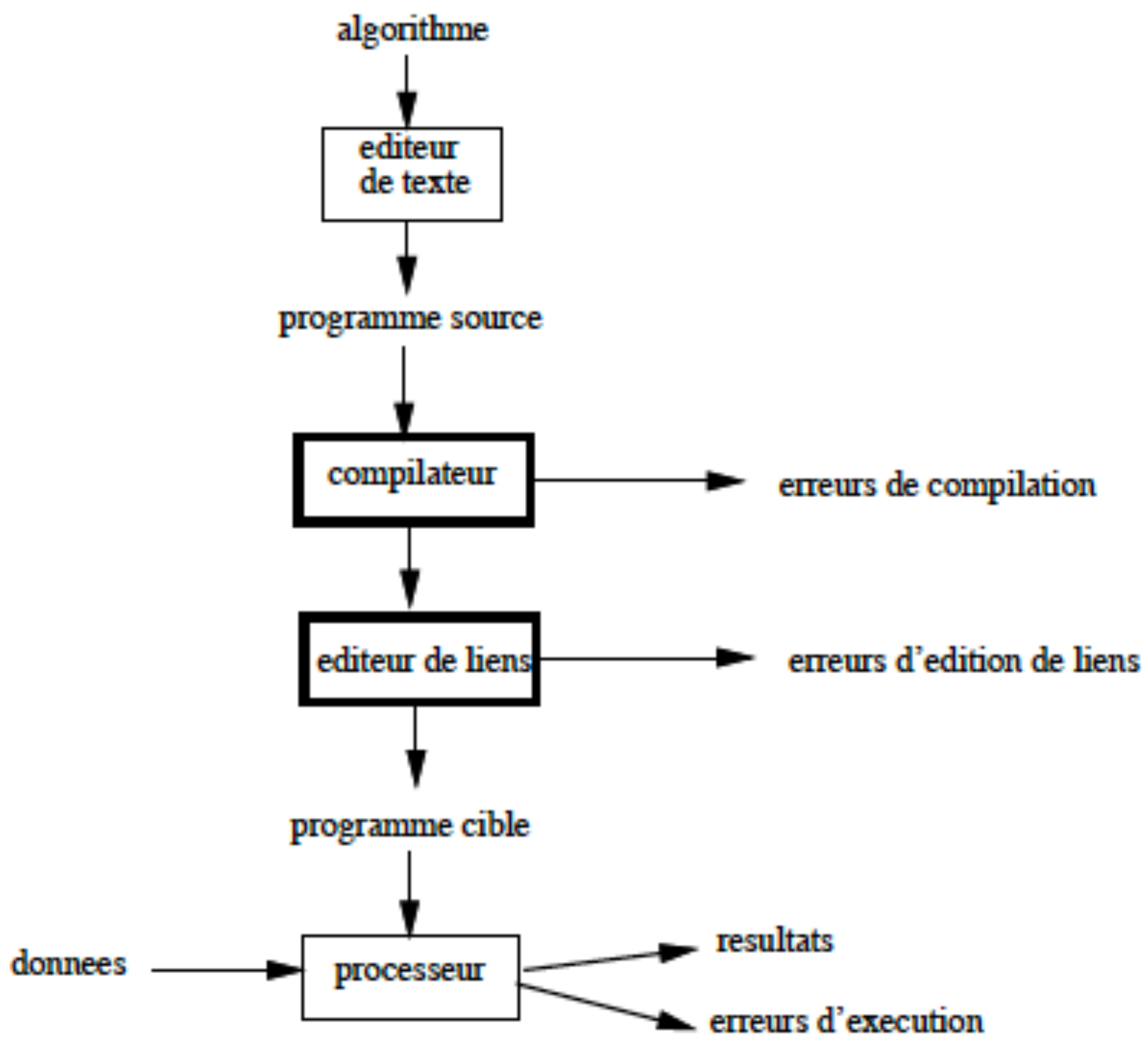

FIG. 1.1 – Chaîne de développement d’un programme

1. Introduction

1.2 L'édition de lien

L'éditeur de liens résout entre autres les références à des appels de routines dont le code est conservé dans des librairies (cf par exemple le `printf` dans un programme C : le code de cette fonction n'est pas dans le programme, mais dans la librairie standard `libc.a`).
L'éditeur des liens permet également de faire de la **compilation séparée** et de concevoir un programme comme un assemblage de briques (ou de modules). Voir le schéma 1.2.

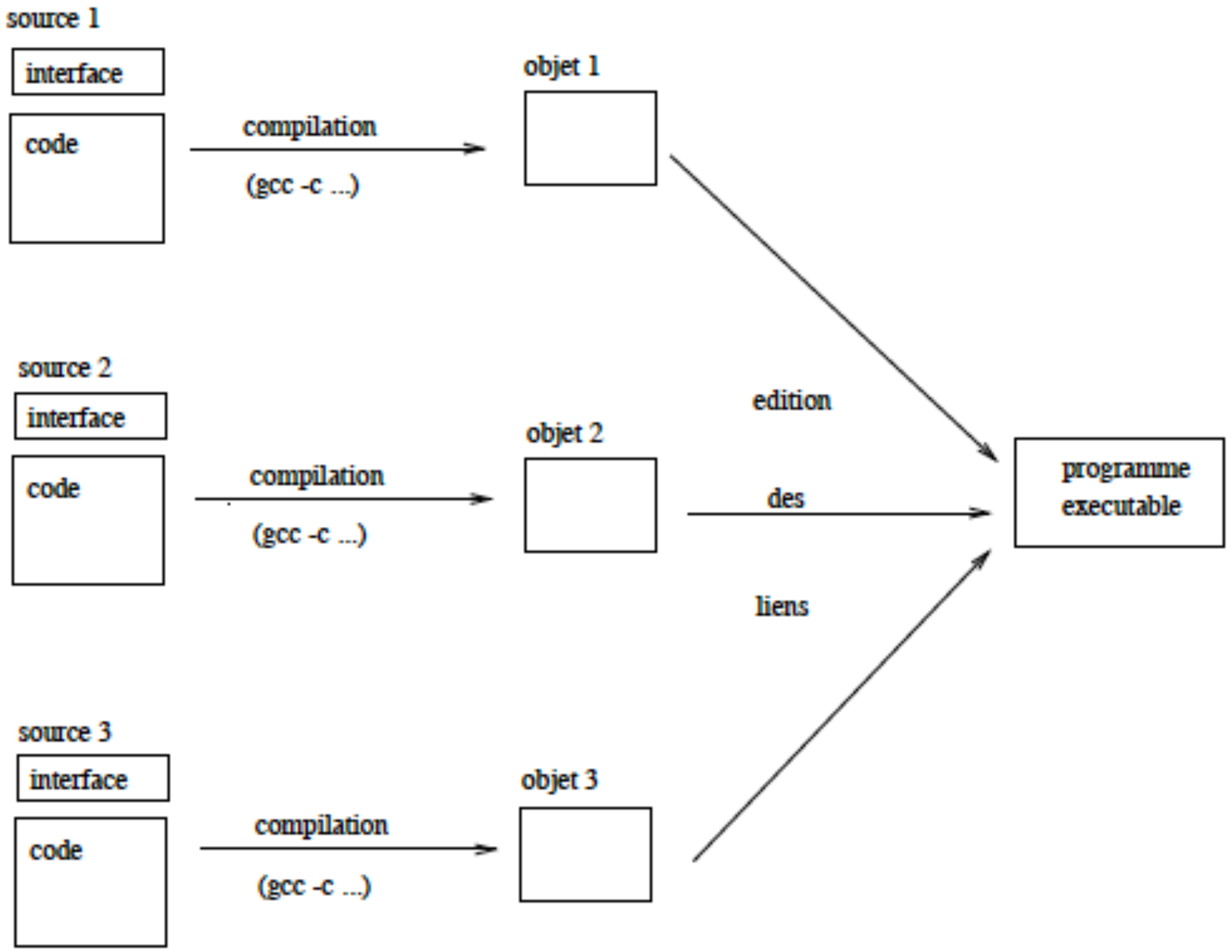

FIG. 1.2 – Compilation séparée

1. Introduction

1.3 Compilateur et interpréteur

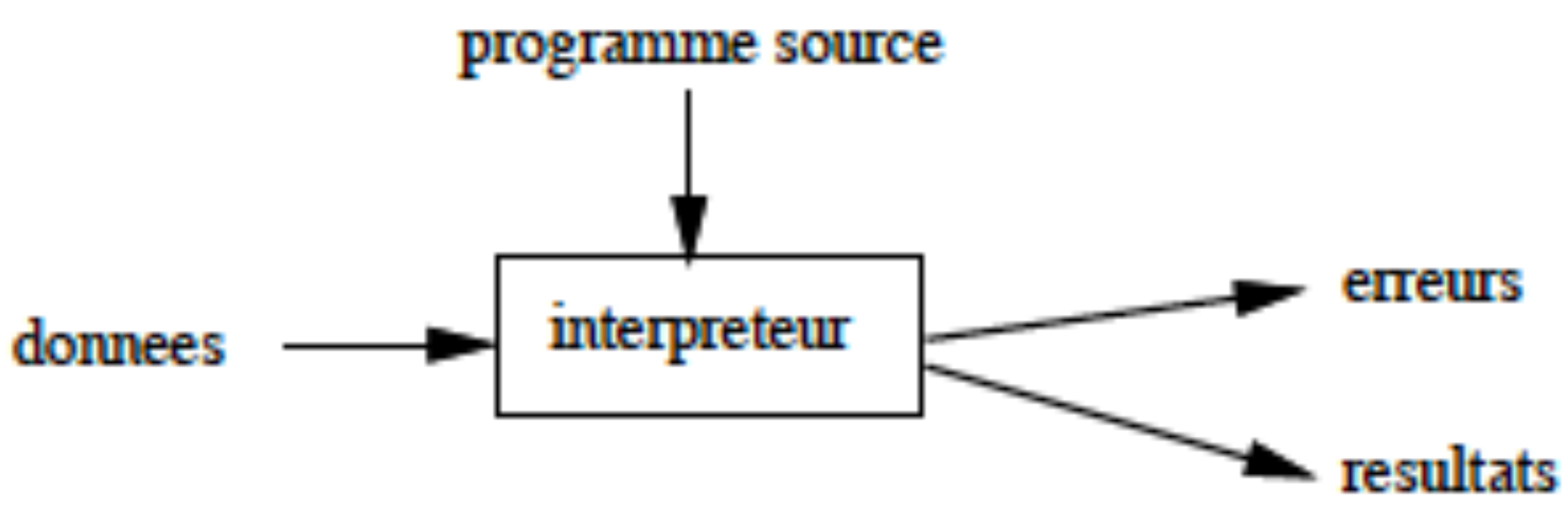

FIG. 1.3 – Interpréteur

• Il existe des langages qui sont à mi-chemin de l'interprétation et de la compilation. On les appelle **langages P-code** ou **langages intermédiaires**. Le code source est traduit (compilé) dans une forme binaire compacte (du pseudo-code ou p-code) qui n'est pas encore du code machine. Lorsque l'on exécute le programme, ce P-code est interprété. Par exemple en Java, le source est compilé pour obtenir un fichier (.class) "byte code" qui sera interprété par une **machine virtuelle**. Autre langage p-code : Python.
Les interpréteurs de p-code peuvent être relativement petits et rapides, si bien que le p-code peut s'exécuter presque aussi rapidement que du binaire compilé[1]. En outre les langages p-code peuvent garder la flexibilité et la puissance des langages interprétés. On ne garde que les avantages!!

1. Introduction

1.4 Intérêt du cours

1) La "compilation" n'est pas limitée à la traduction d'un programme informatique écrit dans un langage de haut niveau en un programme directement exécutable par une machine, cela peut aussi être :

• la traduction d'un langage de programmation de haut niveau vers un autre langage de programmation de haut niveau. Par exemple une traduction de Pascal vers C, ou de Java vers C++, ou de …
Lorsque le langage cible est aussi un langage de haut niveau, on parle plutôt de **traducteur**. Autre différence entre traducteur et compilateur : dans un traducteur, il n'y a pas de perte d'informations (on garde les commentaires, par exemple), alors que dans un compilateur il y a perte d'informations.
• la traduction d'un langage de programmation de bas niveau vers un autre langage de programmation de haut niveau. Par exemple pour retrouver le code C à partir d'un code compilé (piratage, récupération de vieux logiciels, …)
• la traduction d'un langage **quelconque** vers un autre langage quelconque (ie pas forcemment de programmation) : word vers html, pdf vers ps, …
Ce genre de travail peut très bien être confié à un ingénieur maître de nos jours.

1. Introduction

1.4 Intérêt du cours

2) Le but de ce cours est de présenter les principes de base inhérents à la réalisation de compilateurs. Les idées et techniques développées dans ce domaine sont si **générales et fondamentales** qu'un informaticien (et même un scientifique non informaticien) les utilisera très souvent au cours de sa carrière : traitement de données, moteurs de recherche, outils `sed` ou `awk`, etc.

Nous verrons

- les principes de base inhérents à la réalisation de compilateurs : analyse lexicale, analyse syntaxique, analyse sémantique, génération de code,
- les outils fondamentaux utilisés pour effectuer ces analyses : fondements de base de la théorie des langages (grammaires, automates, ...), méthodes algorithmiques d'analyse, ...

3) En outre, comprendre comment est écrit un compilateur permet de mieux comprendre les "contraintes" imposées par les différents langages lorsque l'on écrit un programme dans un langage de haut niveau.

1. Introduction

1.5 Contenu

Différentes phases de la compilation		Outils théoriques utilisés
Phases d'analyse	analyse lexicale (**scanner**)	expressions régulières automates à états finis
	analyse syntaxique (**parser**)	grammaires automates à pile
	analyse sémantique	traduction dirigée par la syntaxe
Phases de production	génération de code	traduction dirigée par la syntaxe
	optimisation de code	
Gestions parallèles	table des symboles	
	traitement des erreurs	

1.6 Elements de bibliographie

- *A. Aho, R. Sethi, J. Ullman*, **Compilateurs : principes, techniques et outils**, InterEditions 1991.
- *N. Silverio*, **Réaliser un compilateur**, Eyrolles 1995.
- *R.Wilhelm, D. Maurer*, **Les compilateurs : théorie, construction, génération**, Masson 1994.
- *J. Levine, T. Masson, D. Brown*, **lex & yacc**, Éditions O'Reilly International Thomson 1995.

et de diverses notes de cours trouvées sur le web.

2. Rappels sur la théorie des ensembles

2.1Définitions

Définition 1 : Un ensemble est une collection d'objets sans répétition.

Exemple 1 : $\{a, b, c, d, ..., z\}$: l'ensemble des lettres minuscules.

Si un objet appartient à un ensemble A, on dit qu'il est élément de A et l'on note $x \in A$. On a donc les propriétés suivantes :

- Si $\forall x \in A, x \in B$ alors on dit que A est inclus dans B et l'on note $A \subseteq B$;
- $A = B$ si $A \subseteq B$ et $B \subseteq A$ (on dit que A est un sous-ensemble de B) ;
- Par conséquent, $\emptyset \subseteq A$, A étant un ensemble quelconque.

2. Rappels sur la théorie des ensembles

2.2 Opérations sur les ensembles

Soient A et B deux sous-ensembles quelconques de Ω, on définit alors les opérations suivantes :

- Union : notée $A \cup B$ qui comporte tout élément appartenant à A *ou* B ;
- Intersection : notée $A \cap B$ qui comporte tout élément appartenant à A *et* B ;
- Différence : notée $A - B$ qui comporte tout élément appartenant à A *et* qui n'appartient pas à B ;
- Complément : notée $\overline{A} = \Omega - A$
- Cardinalité : notée $\mathrm{card}(A)$ qui est le nombre d'éléments de A ;
- Produit cartésien : noté $A \times B$ qui est l'ensemble des paires (a, b) telles que $a \in A$ et $b \in B$. Il est clair que $\mathrm{card}(A \times B) = \mathrm{card}(A).\mathrm{card}(B)$;
- L'ensemble des parties de A : notée 2^A qui est l'ensemble de tous les sous-ensembles de A. On a : $\mathrm{card}(2^A) = 2^{\mathrm{card}(A)}$.

2. Rappels sur la théorie des ensembles

2.3 Exemple

Exemple 2 : $A = \{a, b\}$, $B = \{a, c\}$, $\Omega = \{a, b, c\}$:

- $A \cap B = \{a\}$;
- $A \cup B = \{a, b, c\}$;

- $A - B = \{b\}$;
- $\overline{A} = \{c\}$;
- $A \times B = \{(a, a), (a, b), (b, a), (b, c)\}$;
- $2^A = \{\emptyset, \{a\}, \{b\}, \{a, b\}\}$

3. Théorie des langages

3.1 Notions sur les mots

Définition 2 : Un symbole est une entité abstraite. Les lettres et les chiffres sont des exemples de symboles utilisés fréquemment, mais des symboles graphiques peuvent également être employés.

Définition 3 : Un alphabet est un ensemble de symboles. Il est également appelé le vocabulaire.

Définition 4 : Un mot (ou bien une chaîne) défini sur un alphabet A est une suite finie de symboles juxtaposés de A.

Exemple 3 :
- Le mot 1011 est défini sur l'alphabet $\{0, 1\}$
- Le mot 1.23 est défini sur l'alphabet $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, .\}$;

3. Théorie des langages

3.1.1 Longueur d'un mot

Si w est un mot, alors sa longueur est définie comme étant le nombre de symboles contenus dans w, elle est noté par $|w|$. Par exemple, $|abc| = 3, |aabba| = 5$. En particulier, on note le mot dont la longueur est nulle par ε : $|\varepsilon| = 0$.

On définit également la cardinalité d'un mot w par rapport à un symbole $a \in A$: $|w|_a$ comme étant le nombre d'occurrence de a dans w. Par exemple, $|abc|_a = 1, |aabba|_b = 2$.

3.1.2 Concaténation des mots

Soient w_1 et w_2 deux mots définis sur l'alphabet A. La concaténation de w_1 et w_2 est un mot w défini sur le même alphabet. w est obtenu en écrivant w_1 suivi de w_2, en d'autres termes, on colle le mot w_2 à la fin du mot w_1 :

$$w_1 = a_1...a_n, w_2 = b_1b_2...b_m$$
$$w = a_1...a_nb_1b_2...b_m$$

La concaténation est noté par le point, mais il peut être omis. On écrira alors : $w = w_1.w_2 = w_1w_2$.

Propriété de la concaténation

Soient w, w_1 et w_2 trois mots définis sur l'alphabet A :
- $|w_1.w_2| = |w_1| + |w_2|$;
- $\forall a \in A : |w_1.w_2|_a = |w_1|_a + |w_2|_a$;
- $(w_1.w_2).w_3 = w_1.(w_2.w_3)$ (la concaténation est associative)
- $w.\varepsilon = \varepsilon.w = w$;

3.1.3 L'exposant

L'opération $w.w$ est notée par w^2. En généralisant, on note $w^n = \underbrace{w...w}_{n\ fois}$. En particulier, l'exposant 0 fait tomber sur ε : $w^0 = \varepsilon$ (le mot w est répété 0 fois).

3.1.4 Le mot miroir

Soit $w = a_1a_2...a_n$ un mot sur A ($a_i \in A$). On appelle mot miroir de w et on le note par w^R le mot obtenu en écrivant w l'envers, c'est-à-dire que $w^R = a_n...a_2a_1$. Il est donc facile de voir que $(w^R)^R = w$.

Certains mots, appelés palindromes, sont égaux à leur miroir. En d'autres termes, on lit la même chose dans les deux directions. Par ailleurs, on peut facilement vérifier que : $(u.v)^R = v^R.u^R$.

3. Théorie des langages

3.1.5Préfixe et suffixe

Soit w un mot défini sur un alphabet A. Un mot x (resp. y) formé sur A est un préfixe (resp. suffixe) de w s'il existe un mot u formé sur A (resp. v formé sur A) tel que $w = xu$ (resp. $w = vy$). Si $w = a_1a_2...a_n$ alors tous les mots de l'ensemble $\{\varepsilon, a_1, a_1a_2, a_1a_2a_3, ..., a_1a_2...a_n\}$ sont des préfixes de w. De même, tous les mots de l'ensemble $\{\varepsilon, a_n, a_{n-1}a_n, a_{n-2}a_{n-1}a_n, ...,$ $a_1a_2...a_n\}$ sont des suffixes de w.

3.1.6 Entrelacement (mélange)

Soit u et v deux mots tel que $u = u_1u_2...u_n$ et $v = v_1v_2...v_m$. On appelle entrelacement de u et v (on le note par $u \sqcup\!\sqcup v$) l'ensemble des mots w qui mélangent les symboles de u de v tout en gardant l'ordre des symboles de u et de v. Par exemple, $ab \sqcup\!\sqcup ac = \{abac, aabc, aacb, acab\}$. Cette opération est commutative.

En particulier, lorsque v se réduit un seul symbole, l'opération d'entrelacement se résume à l'insertion dudit symbole quelque part dans le mot u. Si a est un symbole, alors $u \sqcup\!\sqcup v =$ $\{u_1.a.u_2 | u_1.u_2 = u\}$. Par exemple, $ab \sqcup\!\sqcup c = \{cab, acb, abc\}$.

3.2 Notions sur les langage

Définition 5 : Un langage est un ensemble (fini ou infini) de mots définis sur un alphabet donné. Le langage qui ne contient que ε est dit langage vide.

3.2.1 Opérations sur les langages

Soit L, L_1 et L_2 trois langages définis sur l'alphabet X, on définit les opérations suivantes :
- Union : notée par $+$ ou $|$ plutôt que $\cup$. $L_1 + L_2 = \{w|w \in L_1 \vee w \in L_2\}$;
- Intersection : $L_1 \cap L_2 = \{w|w \in L_1 \wedge w \in L_2\}$;
- Concaténation : $L_1.L_2 = \{w|\exists u \in L_1, \exists v \in L_2 : w = uv\}$;
- Exposant : $L^n = \underbrace{L.L...L}_{n \text{ fois}} = \{w|\exists u_1, u_2, ...u_n \in L : w = u_1u_2...u_n\}$;
- Fermeture transitive de Kleene : notée $L^* = \bigcup_{i \geqslant 0} L^i$. En particulier, si $L = X$ on obtient X^* c'est-à-dire l'ensemble de tous les mots possibles sur l'alphabet X. On peut ainsi définir un langage comme étant un sous-ensemble quelconque de X^* ;
- Fermeture non transitive : $L^+ = \bigcup_{i > 0} L^i$;
- Le langage miroir : $L^R = \{w|\exists u \in L : w = u^R\}$;
- Le mélange de langages : $L \sqcup\!\sqcup L' = \{u \sqcup\!\sqcup v | u \in L, v \in L'\}$. En particulier, lorsque le langage L' se réduit à un seul mot composé à d'un seul symbole a, on a : $L \sqcup\!\sqcup a = \{u.a.v|(u.v) \in L\}$

3. Théorie des langages

3.2 Notions sur les langage

Définition 5 : Un langage est un ensemble (fini ou infini) de mots définis sur un alphabet donné. Le langage qui ne contient que ε est dit langage vide.

3.2.2 Propriétés des opérations sur les langages

Soit L, L_1, L_2, L_3 quatre langage définis sur l'alphabet A :
- $L^* = L^+ + \{\varepsilon\}$;
- $L_1.(L_2.L_3) = (L_1.L_2).L_3$;
- $L_1.(L_2 + L_3) = (L_1.L_2) + (L_1.L_3)$;
- $L.L \neq L$;
- $L_1.(L_2 \cap L_3) \neq (L_1 \cap L_2).(L_1 \cap L_3)$;
- $L_1.L_2 \neq L_2.L_1$.
- $(L^*)^* = L^*$;
- $L^*.L^* = L^*$;
- $L_1.(L_2.L_1)^* = (L_1.L_2)^*.L_1$;
- $(L_1 + L_2)^* = (L_1^* L_2^*)^*$;
- $L_1^* + L_2^* \neq (L_1 + L_2)^*$

4. Expressions et langages rationnels

4.1 Propriétés des opérations sur les langages

L'un des objectifs de ce point est d'introduire un formalisme efficace nous permettant de décrire certaines catégories de mots (autrement dit des langages) que l'on peut être amené à rechercher dans un texte (trouver par exemple tous les mots correspondant à une adresse mail valide dans une page internet). La caractéristique de ces langages est la possibilité de les décrire par des motifs (patterns en anglais), c'est-à-dire par des formules qu'on appelle des *expressions rationnelles.*

Définition. — *Soit Σ un alphabet. Les* expressions rationnelles [5] *sont définies inductivement par les propriétés suivantes :*

- *$\emptyset$ et ε sont des expressions rationnelles ;*
- *$\forall a \in \Sigma$, a est une expression rationnelle ;*
- *Si e_1 et e_2 sont des expressions rationnelles, alors $e_1 + e_2$, $e_1 e_2$ et e^* sont des expressions rationnelles.*

4. Expressions et langages rationnels

4.1 Propriétés des opérations sur les langages

L'un des objectifs de ce point est d'introduire un formalisme efficace nous permettant de décrire certaines catégories de mots (autrement dit des langages) que l'on peut être amené à rechercher dans un texte (trouver par exemple tous les mots correspondant à une adresse mail valide dans une page internet). La caractéristique de ces langages est la possibilité de les décrire par des motifs (patterns en anglais), c'est-à-dire par des formules qu'on appelle des *expressions rationnelles.*

Remarque. On peut aussi noter $e_1|e_2$ au lieu de e_1+e_2 et $e*$ au lieu de e^*.

L'interprétation d'une expression rationnelle est définie par les règles inductives suivantes :

- $\emptyset$ dénote le langage vide et ε le langage $\{\varepsilon\}$;
- $\forall a \in \Sigma$, a dénote le langage $\{a\}$;
- $e_1 + e_2$ dénote l'union des langages dénotés par e_1 et e_2 ;
- $e_1 e_2$ dénote la concaténation des langages dénotés par e_1 et e_2 ;
- e^* dénote la fermeture de Kleene du langage dénoté par e.

Un langage dénoté par une expression rationnelle sera qualifié de *langage rationnel*.

On notera Σ est une langage rationnel puisque dénoté par $\sum_{a \in \Sigma} a$. Il en est de même de Σ^* et $\Sigma^+ = \Sigma\Sigma^*$.

4. Expressions et langages rationnels

4.1 Propriétés des opérations sur les langages

L'un des objectifs de ce point est d'introduire un formalisme efficace nous permettant de décrire certaines catégories de mots (autrement dit des langages) que l'on peut être amené à rechercher dans un texte (trouver par exemple tous les mots correspondant à une adresse mail valide dans une page internet). La caractéristique de ces langages est la possibilité de les décrire par des motifs (patterns en anglais), c'est-à-dire par des formules qu'on appelle des *expressions rationnelles.*

Exemples. Sur l'alphabet $\Sigma = \{a, b\}$:

- a^*b^* dénote le langage des mots débutant par un certain nombre de a suivi d'un certain nombre de b ;
- $(ab)^*$ dénote le langage des mots alternant la lettre a et la lettre b. Notons que l'expression $\varepsilon + a(ba)^*b$ dénote le même langage ;

- $(a+b)^*aaa(a+b)^*$ dénote le langage des mots dont aaa est un facteur ;
- $(a+ba)^*$ dénote le langage des mots dans lesquels chaque b est suivi d'un a ;
- $(a^*b)^*$ dénote le langage formé du mot vide et de tous les mots qui se terminent par un b. Il peut aussi être dénoté par $\varepsilon + (a+b)^*b$.

4. Expressions et langages rationnels

4.1 Propriétés des opérations sur les langages

L'un des objectifs de ce point est d'introduire un formalisme efficace nous permettant de décrire certaines catégories de mots (autrement dit des langages) que l'on peut être amené à rechercher dans un texte (trouver par exemple tous les mots correspondant à une adresse mail valide dans une page internet). La caractéristique de ces langages est la possibilité de les décrire par des motifs (patterns en anglais), c'est-à-dire par des formules qu'on appelle des *expressions rationnelles.*

Théorème. — *L'ensemble* $\mathrm{Rat}(\Sigma)$ *des langages rationnels sur l'alphabet* Σ *est la plus petite partie de* $\mathscr{P}(\Sigma^*)$ *(au sens de l'inclusion) contenant* $\emptyset$, $\{\varepsilon\}$, $\{a\}$ *pour tout* $a \in \Sigma$ *et stable par réunion, concaténation et passage à l'étoile de* Kleene.

4. Expressions et langages rationnels

4.2 Expressions rationnelles étendues

Les expressions rationnelles sont utilisées par des langages de programmation (Perl, PHP, module re de Python), des éditeurs de texte (emacs, vim), des commandes unix (grep). Malheureusement, malgré un effort de normalisation chacun de ces langages ou programmes utilise en partie ses propres notations. Celles présentées ici répondent au standard de normalisation POSIX ; certaines d'entre elles demandent à être adaptées suivant le langage ou le programme utilisé.

4. Expressions et langages rationnels

4.2 Expressions rationnelles étendues

quantificateurs	
`n*`	0 ou plus *n*
`n+`	1 ou plus *n*
`n?`	0 ou 1 *n*
`n{2}`	exactement 2 *n*
`n{2,}`	au moins 2 *n*
`n{2,4}`	2, 3 ou 4 *n*

caractères spéciaux	
`\`	caractère d'échappement
`\n`	nouvelle ligne
`\s`	espace
`\w`	un caractère a-z, A-z ou 0-9 ou _
`\d`	un chiffre 0-9

intervalles	
`.`	tout caractère
`(a|b)`	*a* ou *b*
`[abc]`	*a*, *b* ou *c*
`[^abc]`	tout caractère sauf *a*, *b*, *c*
`[a-z]`	caractère alphabétique en minuscule
`[A-Z]`	caractère alphabétique en majuscule
`[0-9]`	chiffre
`(...)`	délimite un groupe
`\n`	le *n*ᵉ groupe

méta-caractères
`^ [ . $ { * ( ) + \ | ? < >`

Les méta-caractères doivent être précédés du caractère d'échappement.
Par exemple, l'expression régulière ci-dessous peut être utilisée pour reconnaître une adresse mail valide :

```
\w+([-+.']\w+)*@\w+([-.]\w+)*\.[a-zA-Z]{2,6}
```