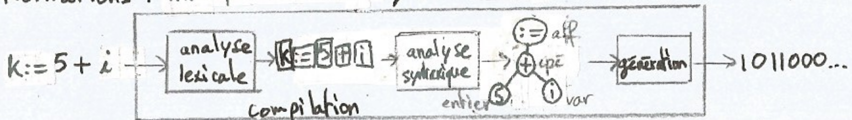
923

Analyse lexicale et syntaxique . Application .

*Introduction : La compilation .

Motivations : interpréter un code / convertir du LaTex en HTML / ...



I] Rappels de théorie des langages

1) Langages rationnels (RAT)

def 1 : un automate est un quintuplet $\mathcal{A}=(Q,\Sigma,\delta,I,F)$ où Q est l'ensemble fini des états, Σ est l'alphabet (ensemble fini), $\delta\subseteq Q\times\Sigma\times Q$ est l'ensemble des transitions, $I\subseteq Q$ les états initaux, $F\subseteq Q$ les états finals.

déf 2 : un chemin dans $\mathcal{A}$ est une suite finie de transitions $q_0\xrightarrow{x_1}q_1\xrightarrow{x_2}\cdots\xrightarrow{x_n}q_n$, où $q_0\in I$. On dit que le mot $x_1\cdots x_n$ est l'étiquette du chemin et q_n l'état d'arrivée.

déf 3 : un mot $x\in\Sigma^*$ est accepté par l'automate $\mathcal{A}$ s'il est l'étiquette d'un chemin dans $\mathcal{A}$ d'état d'arrivée $q_n\in F$. On note $L(\mathcal{A})$ le langage des mots acceptés par $\mathcal{A}$.

ex 4 : (automate du motif). Soit $x=x_1\cdots x_m\in\Sigma^*$. On peut construire l'automate suivant en $O(m)$:

$\mathcal{A}^x_{motif} = \to(\varepsilon)\xrightarrow{x_1}(x_1)\xrightarrow{x_2}(x_1x_2)\xrightarrow{x_3}\cdots\xrightarrow{x_m}((x))$. De plus, $L(\mathcal{A}^x_{motif})=\{x\}$.

ex 5 : (automate des occurrences). Soit $x\in\Sigma^*$, P l'ensemble des préfixes de x. L'automate $\mathcal{A}_x=(P,\Sigma,\delta,\{\varepsilon\},\{x\})$, où $\delta=\{(u,a,f_x(ua)),u\in P,a\in\Sigma\}$, avec $f_x(v)$ le plus petit suffixe de v préfixe de x, est l'automate minimal du langage Σ^*x.

Rq 6 : l'automate du motif sera utilisé pour l'analyse lexicale ; l'automate des occurrences est utile pour la recherche de motif (langage impératif avec Label / Goto).

déf 7 : on considère des symboles frais $\varepsilon,*,|,+,(,)\notin\Sigma$. Le langage $\mathcal{E}$ des expressions régulières est le plus petit langage contenant $\Sigma\cup\{\varepsilon\}$ tel que : si $x\in\mathcal{E},y\in\mathcal{E}$ alors $x^*,(x|y),x^+,xy\in\mathcal{E}$.

déf 8 : soit $e\in\mathcal{E}$. Le langage rationnel $L(e)$ est défini par induction :

- $\forall a\in\Sigma\cup\{\varepsilon\}$, $L(a)=\{a\}$
- $\forall x\in\mathcal{E}$, $L(x^*)=\bigcup_{n\in\mathbb{N}}\{x^n,n\in\mathbb{N}\}$
- $\forall x\in\mathcal{E}$, $L(x^+)=\bigcup_{n\in\mathbb{N}^*}\{x^n,n\in\mathbb{N}\}$
- $\forall x,y\in\mathcal{E}$, $L(x|y)=L(x)\cup L(y)$
- $\forall x,y\in\mathcal{E}$, $L(xy)=\{w_1w_2,\ w_1\in L(x),\ w_2\in L(y)\}$

Rq 9 : on omet les parenthèses en cas de non-ambiguïté.

thm 10 : (Kleene) Soit $\mathcal{L}\subseteq\Sigma^*$ un langage. On a l'équivalence :

(i) Il existe un automate $\mathcal{A}$ tq $\mathcal{L}=L(\mathcal{A})$ [$\mathcal{L}$ est reconnaissable par automate].

(ii) Il existe $e\in\mathcal{E}$ tq $\mathcal{L}=L(e)$ [$\mathcal{L}$ est rationnel].

Rq 11 : on dispose d'un algorithme pour passer d'une expression régulière à un automate.

2) Langages algébriques (ALG).

déf 12 : une grammaire algébrique sur un alphabet Σ est un quadruplet $G=(\mathcal{N},T,S,R)$ où $\mathcal{N}$ est l'ensemble fini des non-terminaux, $T\subseteq\Sigma$ est l'ensemble des terminaux, $S\subseteq\mathcal{N}$ est l'axiome et $R\subseteq\mathcal{N}\times(\mathcal{N}\cup T)^*$ l'ensemble des règles.

déf 13 : Soient $u,v\in(\mathcal{N}\cup T)^*$. On dit que u se dérive en v $(u\to v)$ s'il existe $\alpha,\beta\in(\mathcal{N}\cup T)^*$ et $X\in\mathcal{N}$ tq $\begin{cases}u=\alpha X\beta\\ v=\alpha\omega\beta\end{cases}$ et $(X\to\omega)\in R$.

déf 14 : un mot $x\in\Sigma^*$ est engendré par une grammaire $G=(\mathcal{N},T,S,R)$ si $S\xrightarrow{G}x$, où $\xrightarrow{G}$ est la clôture réflexive transitive de $\to$.

On note $L(G)$ le langage des mots engendrés par la grammaire G.

ex 15 : le langage $L(a^*b)$ est engendré par la grammaire $\begin{cases}S\to aS\\ S\to b\end{cases}$

ex 16 : (Langage de Dyck sur n paires de parenthèses).

La grammaire G_{Dyck} : $\begin{cases}S\to a_1S\bar{a}_1\\ S\to a_2S\bar{a}_2\\ \vdots\\ S\to a_nS\bar{a}_n\\ S\to\varepsilon\end{cases}$ engendre le langage des mots bien parenthésés avec au plus n types de parenthèses.

ex 17 : (expressions arithmétiques préfixes) G_{pref} : $\begin{cases}S\to +SS\\ S\to c\end{cases}$

ex 18 : (expressions arithmétiques postfixes) G_{post} : $\begin{cases}S\to SS+\\ S\to c\end{cases}$

thm 19 : (RAT $\subseteq$ ALG).

Tout langage rationnel est algébrique (ie : engendré par une grammaire algébrique)

déf 20 : un arbre de dérivation est un arbre fini étiqueté par $\mathcal{N}\cup T\cup\{\varepsilon\}$ tq :

- l'arbre est enraciné en S.
- Si T est l'étiquette d'un noeud interne et $a_1\cdots a_n$ ses fils, alors $(T\to a_1\cdots a_n)\in R$.

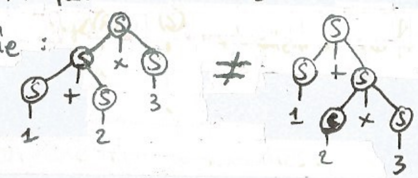
→ on appellera frontière d'un tel arbre le mot obtenu par concaténation de ses feuilles de gauche à droite.

déf 21 : une grammaire est ambigüe s'il existe un mot admettant deux arbres distincts.

déf 22 : un langage algébrique est non-ambigü s'il est engendré par une grammaire non-ambigüe. Dans le cas contraire, il est dit "intrinsèquement ambigu".

ex 23 : les grammaires G_{Dyck}, G_{pref}, G_{post} sont non-ambigües.

La grammaire $\begin{cases}S\to S+S\\ S\to S\times S\\ S\to c\end{cases}$ est ambigüe :



II] Analyse lexicale

But : transformer un texte en une liste de lexèmes (terminaux d'une grammaire).

Intérêts : -> filtration des commentaires
-> mise en page ("1+2" et "1   +2" représentent le même programme)
-> repérage des types.
-> décoration des lexèmes (nom du fichier, n° de ligne, nbre de caractères, ...)
-> repérage des erreurs lexicales (motif interdit).

def 24 : (règle d'analyse lexicale). Soit T l'ensemble fini des types de lexèmes.
Une règle d'analyse lexicale sur l'alphabet Σ est la donnée :
- d'une expression régulière e sur Σ tq $\varepsilon \notin L(e)$.
- d'un type de lexème $t \in T$.

Cette règle est notée $e \to t$.

Algo 25 : Entrée : texte $\in \Sigma^*$; liste de règles $(e_j \to t_j)_{j \in \{1...n\}}$
Sortie : liste de lexèmes correspondant au texte.

```
Analyse_lexicale(texte) :
    si texte = "" retourner [ ]
    sinon :
        (lexème, suite) <- Analyse_aux(texte)
        retourner lexème :: Analyse_lexicale(suite)
```

```
Analyse_aux(texte) :
    fin <- 0        {position de fin du plus long préfixe connu}
    i_Aut <- -∞     {indice de l'automate du plus long préfixe}
    i_texte <- 1
    Tant que (il existe un A_motif^{e_j} non-bloqué) et (i_texte ≤ longueur(texte)) :
        pour j <- 1 à n :
            Faire lire texte[i_texte] à l'automate A_motif^{e_j}
            Si A_motif^{e_j} est dans un état final :
                si fin < i_texte :
                    fin <- i_texte
                    i_Aut <- j
        incrémenter i_texte
    Si i_Aut = 0 : erreur ("le texte ne contient aucun préfixe connu")
    préfixe <- texte[1 ... fin]
    lexème <- (préfixe, t_{i_Aut})
    retourner (lexème, texte - préfixe)
```

-> Complexité temporelle : $O(\text{longueur(texte)})$

ex 26 : pour analyser le texte a^n avec les règles $\begin{cases} a \to \text{lettre\_a} \\ a^*b \to \text{a-itérée-b} \end{cases}$ il faut $\frac{n(n+1)}{2}$ lectures.

Rq 27 : A la fin de l'analyse lexicale, on dispose d'une table des symboles dans laquelle on a stocké les informations souhaitées (type, lignes ...).

III) Analyse syntaxique

But : déterminer si un mot est reconnu par une grammaire, et construire une dérivation ou l'arbre syntaxique.

1) Algorithmes génériques.

thm 28 : Soit $G = (\mathcal{N}, T, S, R)$ une grammaire algébrique sur l'alphabet Σ et $m \in \Sigma^*$. Notons a le nombre maximal de symboles d'une partie droite d'une règle dans R. Alors :
$m \in L(G) \iff$ il existe une dérivation reconnaissant m dans G, de longueur inférieure à $a^{|m| \times |\mathcal{N}|}$

-> Il suffit de parcourir les dérivations de longueur $\leq a^{|m| \times |\mathcal{N}|}$

def 29 : une grammaire est en forme normale de Chomsky si ses règles sont de la forme "$N \to ML$" ou "$N \to a$", $N, M, L \in \mathcal{N}$, $a \in T$.

prop 30 : Pour toute grammaire G, il existe une grammaire G' en forme normale de Chomsky tq $L(G') = L(G) \setminus \{\varepsilon\}$ de taille $|G'| = O(|G|^2)$

algo 31 : (CYK) Entrée : une grammaire G en forme normale de Chomsky, un mot $w \in \Sigma^*$.
Sortie : oui si $w \in L(G)$, non sinon
Algorithme, correction, terminaison, complexité.

DEV1

2) Algorithmes linéaires.

a/ Analyse descendante.

Principe 32 : on part de l'axiome et on le dérive pour arriver au texte.

ex 33 : un algorithme simple permet d'analyser la grammaire G_{pref} [cf Annexe 1]
La lecture du caractère courant permet de choisir la règle à appliquer.

⚠ ce n'est pas le cas pour toutes les grammaires : $G^2_{pref} \begin{cases} S \to TSS \\ T \to + \\ S \to c \end{cases}$ | $G^3_{pref} \begin{cases} S \to U+SS \\ U \to \varepsilon \\ S \to c \end{cases}$

def 34 : (premier($\vec{\alpha}$))
Soit $G = (\mathcal{N}, T, S, R)$ une grammaire. Pour $\vec{\alpha} \in (\mathcal{N} \cup T)^*$, on définit :

$$\text{premier}(\vec{\alpha}) = \begin{cases} \{a \in T \text{ tq } \vec{\alpha} \to^* a\vec{\beta},\ \vec{\beta} \in (T \cup \mathcal{N})^*\} & \text{si } \vec{\alpha} \not\to^* \varepsilon \\ \{a \in T \text{ tq } \vec{\alpha} \to^* a\vec{\beta},\ \vec{\beta} \in (T \cup \mathcal{N})^*\} \cup \{\bowtie\} & \text{si } \vec{\alpha} \to^* \varepsilon \end{cases}$$

où $\bowtie \notin \Sigma$ est un symbole frais.

def 35 : (suivant(N))
Soit $G = (\mathcal{N}, T, S, R)$ une grammaire. Pour $N \in \mathcal{N}$, on définit :

$$\text{suivant}(N) = \begin{cases} \{a \in T \text{ tq } S \to^* \vec{\alpha} N a \vec{\beta},\ \vec{\alpha}, \vec{\beta} \in (T \cup \mathcal{N})^*\} & \text{si } S \not\to^* \vec{\alpha} N \\ \{a \in T \text{ tq } S \to^* \vec{\alpha} N a \vec{\beta},\ \vec{\alpha}\vec{\beta} \in (T \cup \mathcal{N})^*\} \cup \{\text{⧗}\} & \text{si } S \to^* \vec{\alpha} N \end{cases}$$

où $\text{⧗} \notin \Sigma \cup \{\bowtie\}$ est un symbole frais.

thm 36 : (calcul de premier)
La fonction premier est la plus petite partie de $(T \cup \mathcal{N})^* \times (T \cup \{\bowtie\})$ tq
(i) $\forall a \in T$, $\text{premier}(a) = \{a\}$
(ii) $\text{premier}(\varepsilon) = \{\bowtie\}$
(iii) $\forall (N \to \varepsilon) \in S$, $\text{premier}(\varepsilon) \subseteq \text{premier}(N)$
(iv) $\forall (N \to \alpha_1 \dots \alpha_n) \in S$, $\text{premier}(\alpha_1 \dots \alpha_n) \subseteq \text{premier}(N)$
(v) Si $\bowtie \notin \text{premier}(\alpha_1)$, $\text{premier}(\alpha_1) \subseteq \text{premier}(\alpha_1 \dots \alpha_n)$
(vi) Si $\bowtie \in \text{premier}(\alpha_1)$, $\text{premier}(\alpha_1) \setminus \{\bowtie\} \cup \text{premier}(\alpha_2 \dots \alpha_n) \subseteq \text{premier}(\alpha_1 \dots \alpha_n)$

DEV2

def 37 : une grammaire G est dite $LL(1)$ si pour tout $N \in \mathcal{N}$, en notant $N \to \vec{\alpha_1}, \dots, N \to \vec{\alpha_n}$ les règles dont la partie gauche est N, on a :
(i) les ensembles $\text{premier}(\vec{\alpha_1}), \dots, \text{premier}(\vec{\alpha_n})$ sont deux à deux disjoints.
(ii) Si $N \to^* \varepsilon$, alors suivant(N) est disjoint de chaque $\text{premier}(\vec{\alpha_i})$

ex 38 : G^2_{pref}, G^3_{pref}, $G_{Dyck} \in LL(1)$. $G_{post} \notin LL(1)$

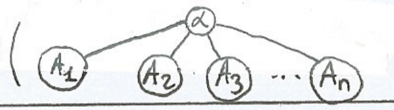
def 39 : Un langage est $LL(1)$ s'il existe une grammaire $LL(1)$ qui le reconnaît.

ex 40 : $L(G_{post}) \in LL(1)$ [cf Annexe 2]

thm 41 : $\text{RAT} \subseteq LL(1) \subseteq \{\text{langages non-ambigus}\}$

```
Analyse_LL(1) (texte) : arbre
    A, texte' = Analyse_LL(1)_aux (S, texte)
    si texte' ≠ ε : erreur "le mot n'est pas reconnu"
    retourner (A)
```

```
Analyse_LL(1)_aux (α, texte) : arbre x texte
    si α ∈ T :
        si texte[1] = α : retourner ( [texte[1]] , texte[2...] )
    sinon : erreur ("le mot n'est pas reconnu")
    # les règles sont de la forme α → β⃗1, ..., α → β⃗n
    Si texte = ε :
        Si ⋈ ∈ premier(α) et ⧗ ∈ suivant(α) :
            i ← l'indice tq ⋈ ∈ premier(β⃗i)
        sinon : erreur ("le mot n'est pas reconnu")
    Sinon :
        Si (⋈ ∈ premier(α)) et (texte[1] ∈ suivant(α)) :
            i ← l'indice tq ⋈ ∈ premier(β⃗i)
        sinon i ← l'indice tq texte[1] ∈ premier(β⃗i)
    si β⃗i = ε : retourner (
```

```
    , texte )
    Sinon : A1, texte1 ← Analyse_LL(1)_aux ( β⃗i[1], texte )
        A2, texte2 ← Analyse_LL(1)_aux ( β⃗i[2], texte1 )
        ⋮
        Retourner (
```

```
        , texten )
```

b/Analyse ascendante

Principe : on part du texte et on remonte les règles pour obtenir l'axiome.

def 43 : une situation $LR(0)$ d'une grammaire G, notée $N \to \alpha_1 \dots \alpha_j {}_\uparrow \alpha_{j+1} \dots \alpha_n$ est :
- une règle $(N \to \alpha_1 \dots \alpha_n) \in R$ avec $N \in \mathcal{N}$, $\alpha_1 \dots \alpha_n \in \mathcal{N} \cup T$.
- un entier $j \in [\![0, n]\!]$.

→ on note $\mathcal{S}_G$ l'ensemble des situations.

def 44 : la fermeture de $E \subseteq \mathcal{S}_G$ est la plus petite partie $Cl_G(E)$ de $\mathcal{S}_G$ tq :
- $E \subseteq Cl_G(E)$
- $(N \to \alpha {}_\uparrow L \beta) \in Cl_G(E)$ et $(L \to \gamma) \in R \Rightarrow (L \to {}_\uparrow \gamma) \in Cl_G(E)$.

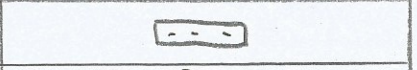
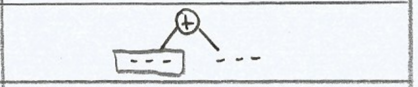
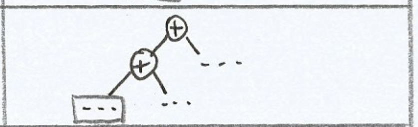
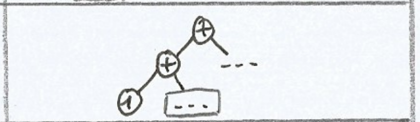
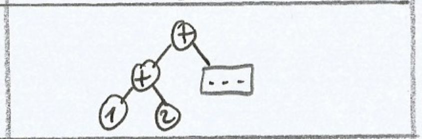
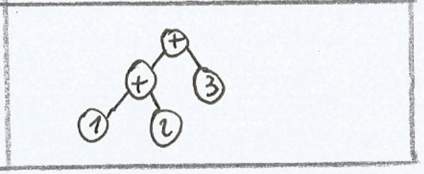
def 45 : l'automate $LR(0)$ d'une grammaire G est défini par :
- $Q = \{Cl_G(E),\ E \subseteq \mathcal{S}_G\}$
- $I = \{Cl_G(\{S \to {}_\uparrow \vec{\alpha} \text{ tq } (S \to \vec{\alpha}) \in R\})\}$
- $\delta : (E, \alpha) \mapsto Cl_G(\{N \to \vec{\gamma} \alpha {}_\uparrow \vec{\beta} \text{ tq } N \to \vec{\gamma} {}_\uparrow \alpha \vec{\beta} \in E\})$

ex 46 : Automate $LR(0)$ de la grammaire G_{post} [cf Annexe 3]

def 47 : une grammaire G est dite $LR(0)$ si tout état de son automate qui contient une situation de la forme $N \to \alpha_1 \dots \alpha_n {}_\uparrow$ ne contient que celle-ci

prop 48 : schéma des inclusions RAT, $LL(1)$, $LR(0)$, ALG [cf Annexe 4]

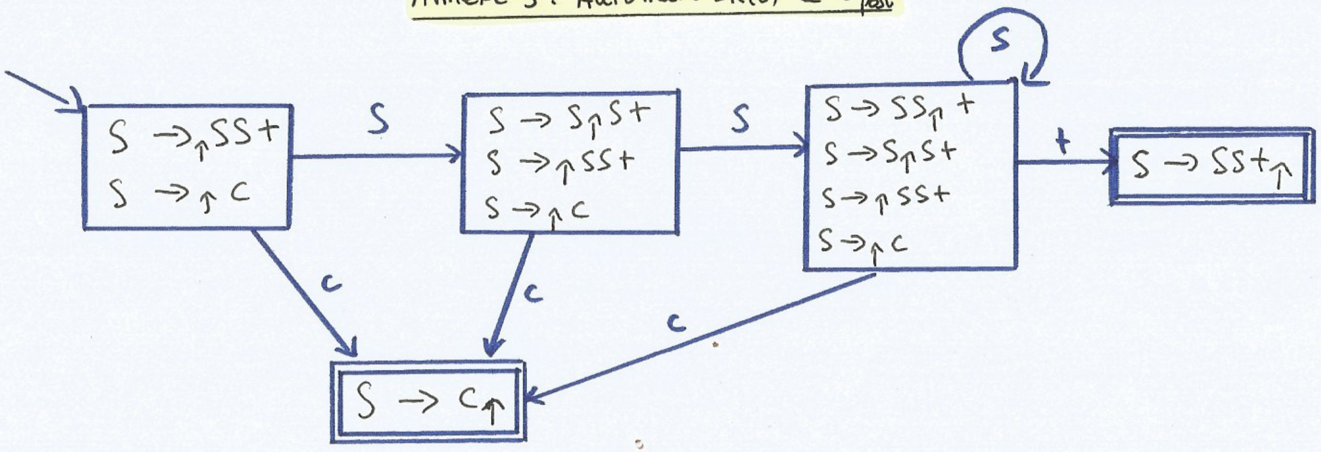
Annexe 1 : Analyse G_{pref}

mot lu	arbre en cours de construction
$\uparrow *****$	
$+\uparrow ****$	
$++\uparrow ***$	
$++1\uparrow **$	
$++12\uparrow *$	
$++123\uparrow$	

Annexe 2 : une grammaire LL(1) reconnaissant $L(G_{post})$.

$$G_{post}^{LL(1)} : \begin{cases} S \rightarrow cT \\ T \rightarrow S+T \\ T \rightarrow \varepsilon \end{cases}$$

Annexe 3 : Automate LR(0) de G_{post}.



Annexe 4 : schéma des inclusions.