923

Analyses Lexicale et Syntaxique, Applications.

Idées Générales

yacc → LALR ?

Langage du Programmeur $\xrightarrow{\text{Code}}$ Langage de Programmation — Compilation (AL → AS) → Langage Machine $\xrightarrow{\text{Traduction Machine}}$ Instructions machine $\xrightarrow{\text{Exécution}}$ Langage du Programme $\xrightarrow{\text{Programme}}$ Langage Utilisateur

I- Compilation

La compilation assure la traduction entre un langage haut niveau utilisable par le programmeur en un langage bas niveau utilisable pour la machine. Elle doit aussi détecter les erreurs du programmeur et lui permettre de les résoudre.

1) Analyse lexicale

- Regroupe les caractères du mot d'entrée (programme source) en séquences significatives : les lexèmes.
- Pour chaque lexème, transmet à l'analyseur syntaxique l'unité lexicale correspondante.
- Un pré-traitement peut éliminer les blancs et les commentaires du programme.
- A chaque occurrence d'une unité lexicale est associée un pointeur vers le lexème correspondant.

Exemple

Unité lexicale : <id> "nom du langage"

motif associé : $[a\text{-}z\,A\text{-}Z].([a\text{-}z\,A\text{-}Z]+[0\text{-}9]+\{\_\})^*$ "langage"

lexème du motif : Variable_1 "mot"

Autre unités lexicales :

- mots clés - entiers - booléens ... - parenthèse, - ; ...
- opérateurs

- let chaine = "Hello world!"; $\xrightarrow{A.L}$ <let><id><=><string><;>

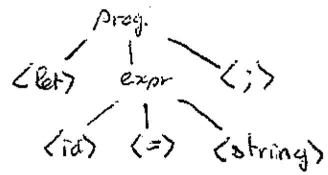
App° : latex, html, xml, ...

2) Analyse syntaxique

Fait 1. Pour les langages de programmation classiques, les chaines d'unité lexicales produites à partir d'un code source correct sont les mots d'une grammaire interne au langage.

- L'analyse syntaxique revient à résoudre le problème du mot pour cette grammaire, par des méthodes plus adaptées que l'algorithme général (CYK).
- Production d'un arbre de dérivation associé.

Exemple



prog, expr = symboles non-terminaux.

3) Analyse sémantique

- Ajoute des infos à l'arbre abstrait
- Effectue le contrôle de type du code : nature des opérandes, nombre et nature des instructions d'une fonction, etc..

4) Table de symboles

Lieu de la mémoire où sont stockées les infos relatives à chaque lexème : nature, instruction nécessaires, première apparition dans le code, etc.. Et surtout : Valeur !!

→ Les trois analyseurs lisent, écrivent, communiquent par la table de symboles.

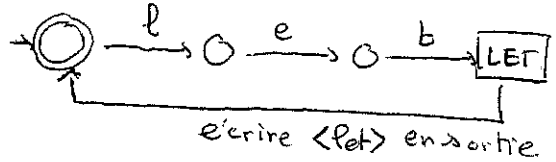
App° : colorat° syntaxique / linguistiq informatiq

II- Analyse Lexicale

Fait 2 : Le motif de chaque unité lexicale peut être décrit par une expression régulière.

Corollaire 3 : Un analyseur lexical est un automate fini déterministe auquel on ajoute une fonction d'écriture.

Exemple :



Il y a erreur lexicale si on cherche à prendre une transition inexistante ou si on ne se retrouve pas dans l'état final (= initial = "juste après écriture") à la fin de la lecture du mot.

Question ouverte : Retour = lecture d'un "blanc" ?

- Deux principes :
  - Plus long préfixe : 12 devient <nb> et pas <nb><nb>
  - Priorités : let devient <let> et pas <id>.

Algorithme de Hopcroft [DEV]

On utilise la puissance de la théorie des automates pour optimiser l'analyseur lexical : Calcul de l'automate minimal.

III Analyse Syntaxique Descendante

- Partir de l'axiome S de la grammaire pour en dériver la suite d'unités donnée en entrée :

1) Deux problèmes :

→ quelle règle choisir ? → quelle occurrence dans l'entrée correspond aux terminaux créés par la règle ?

A part cela, l'idée générale est :

Proc_A() : * A non-terminal de G *
  Choisir $A \to X_1 \dots X_k$ dans G
  Pour i de 1 à k * Dérivation gauche ! *
    Si X_i non terminal, faire Proc_X_i() (1)
    Si $X_i = w[p]$ faire $p \leftarrow p+1$ (2)
    Sinon signaler une erreur.
Où $w[1..N]$ est le mot d'entrée.

- Ceci correspond à un automate à pile, dit expand/check : le mot à lire est w, la pile, initialisée à S, est la pile d'appels récursifs.

Les transitions sont $(A, \varepsilon \mid X_1 \dots X_k)$ "expand" (1)
$(a, a \mid \varepsilon)$ "check" (2)

Un seul état, accepte par pile vide.

2) Résolution des problèmes :

On veut avoir un choix déterministe des transitions à effectuer dans l'automate.

Fonction Premier :

$$\text{Premier}(\alpha) = \{a \in T,\ \alpha \xrightarrow{*} a\gamma\} \cup \{\varepsilon \text{ si } \alpha \xrightarrow{*} \varepsilon\}$$

α chaîne de terminaux et non-terminaux.

Proposition 4 : On peut effectuer un calcul récursif de Premier et l'étendre aux ensembles de chaînes.

Fonction Suivant : A non-terminal

$$\text{Suivant}(A) = \{a \in T,\ S \xrightarrow{*} \alpha A a \beta\} \cup \{\varepsilon \text{ si } S \xrightarrow{*} \alpha A\}$$

Proposition 5 : On peut calculer récursivement Suivant à l'aide de Premier.

Table d'analyse syntaxique

Tableau $M[A, a]$, A non-terminal, a = terminal ou ε

$M[A, a]$ contient les productions $A \to \alpha$ telles que

- $a \in$ Premier (α) (y compris si $a = \varepsilon$??)
- $\varepsilon \in$ Premier (α) et $a \in$ Suivant (A). (aussi si $a = \varepsilon$!!)

Définition 6 : Une grammaire est dite LL(1) si chaque entrée de la table d'analyse contient au plus une production.

- Nos problèmes sont alors résolus
- Ça arrive !!

Caractérisation 7 : Une grammaire est LL(1) ssi pour toutes règles $A \to \alpha$ et $A \to \beta$, $\alpha \neq \beta$, on a

$$\text{Premier}(\alpha\,\text{Suivant}(A)) \cap \text{Premier}(\beta\,\text{Suivant}(A)) = \emptyset$$

IV - Analyse Syntaxique Ascendante

- Partir de w et remonter à S par réductions successives : Forme : $\alpha \leftarrow A$
- On obtient une dérivation droite
- On lit w de gauche à droite jusqu'à reconnaître un "manche", i.e. une chaîne à réduire. Le tout est de le reconnaître...
- On a une entrée w est une pile. On peut :
  - Décaler = empiler le symbole lu
  - Réduire : Remplacer les ? symboles au sommet de la pile par un unique non-terminal correspondant.
  - Accepter
  - Relever une erreur.

Remarque : Pas une vraie pile, puisqu'on a accès au contenu qui se trouve à l'intérieur

Les problèmes : Décaler ou Réduire ? Que réduire ?

Items : A chaque production $A \to XYZ$ on associe les items $A \to \bullet XYZ$ $A \to X \bullet YZ$ $A \to XY \bullet Z$ et $A \to XYZ \bullet$

Les items permettent de construire un automate fini déterministe appelé Automate des Items, utilisé par l'analyseur syntaxique pour prendre des décisions.

Les grammaires pour lesquelles on peut construire et utiliser l'automates des items sont les grammaires LR(0)

Exemple et Explications = [DEV].

[Dragon] = Aho, Sethi, (Lam), Ullman : Compilateurs

Livre très clair, vraiment pédagogique.

Algorithme de Hopcroft. ref. Carton.

calcul de la congruence de Nerode en $O(|\Sigma||Q|\log|Q|)$

<u>principe de la coupure :</u>

$a \in \Sigma$, $B, C \subseteq Q$ alors B est stable par (C, a) si

$B.a \subseteq C$ ou $B.a \cap C = \emptyset$.

sinon on coupe B en l'union disjointe de B_1 et B_2

$B_1 = \{q \in Q,\ q.a \in C\}$ $\quad B_2 = \{q \in Q,\ q.a \notin C\}$

Une partition de Q est stable pour (C, a) si chacune de ses partie l'est.

résultat évident : Une partition de Q compatible avec F est une congruence ssi elle est stable pour chaque paire (P, a) avec P partie de la partition et $a \in \Sigma$.

deuxième lemme évident :

$B \subseteq Q$, $C = C_1 \mathring{\cup} C_2$ alors

- Si B stable pour (C_1, a) et (C_2, a) alors B stable pour (C, a)
- Si B stable pour (C_1, a) et (C, a) alors B stable pour (C_2, a)

Algo : entrée automate déterministe $\mathcal{A} = (Q, \Sigma, \delta, I, F)$

$\mathcal{P} \leftarrow (F, Q \setminus F)$

$S \leftarrow \{(\min(F, Q\setminus F), a) \mid a \in \Sigma\}$

tant que $S \neq \emptyset$ faire

$(C, a) \leftarrow$ un élément de S

$S \leftarrow S \setminus (C, a)$

pour chaque B coupé par (C,a) en B_1 et B_2 faire

remplacer B par B_1 et B_2 dans $\mathcal{P}$

pour tout $b \in \Sigma$ faire

si $(B, b) \in S$ alors

remplacer (B, b) par (B_1, b) et (B_2, b) dans S

sinon

ajouter $(\min(B_1, B_2), b)$ à S

;;

terminaison : à chaque étape soit on coupe un élément de $\mathcal{P}$ soit on diminue le cardinal de S de 1, deux évènements qui ne peuvent avoir lieu qu'un nombre fini de fois.

Ensuite, $\mathcal{P}$ est la congruence de Nérode, en effet la congruence de Nérode raffine $\mathcal{P}$ nécessairement car chaque coupure de $\mathcal{P}$ est nécessaire, il reste à montrer que aucune coupure n'a été oubliée.

lemme : $\forall a \in \Sigma$ et $P \in \mathcal{P}$, P est combinaison de

- parties C telles que $\mathcal{P}$ stable pour (C, a)
- parties C telles que $(C, a) \in S$

preuve par induction sur le nombre d'itération de la boucle principale.

et lorsque S vide, alors P est combinaison de parties C telles que $\mathcal{P}$ stable pour (C,a) il découle d'un du lemme précédent que $\mathcal{P}$ stable pour (P, a).

Exemple de grammaire LR(0), automate des items

ref. Hopcroft / Ullman.

Or une grammaire non contextuel, les items de G sont les triplets (A, α, β) pour chaque règle de la grammaire $A \to \alpha\beta$.
on note les items $[A \to \alpha . \beta]$ un item de type $[A \to \alpha .]$ est dit complet.

L'automate des items. On se donne un état initial q_0 et les autres états sont formé des items de G. La table de transition est donnée par

$$\delta(q_0, \varepsilon) = \{[S \to .\alpha] \mid \text{quand } S \to \alpha \text{ est une règle de } G\}$$

$$\delta([A \to \alpha . B\beta], \varepsilon) = \{[B \to .\gamma] \mid B \to \gamma \text{ est une règle de } G\}$$

$$\delta([A \to \alpha . X\beta], X) = \{[A \to \alpha X . \beta]\}$$

⚠ Pour l'instant il n'est pas encore question d'automate à pile.

On déterminise ensuite cet automate.

Une fois cela fait on ajoute la fonction pile.

Initialise à la pile $\boxed{0}$ et lorsque l'on lit une lettre du mot et qu'on avance on stock dans la pile la lettre lue puis l'état dans lequel on arrive.

par exemple : $q_0 \xrightarrow{a} q_1$ la pile passe de $\boxed{0}$ à $\begin{array}{|c|}\hline 1 \\ \hline a \\ \hline 0 \\ \hline\end{array}$

Si on arrive dans un état avec un items du type complet $[A \to \alpha .]$ qui sera alors le seul items de l'état si la grammaire est bien LR(0).
La fonction de transition lira alors le fait d'être dans cet état en tête de pile et saura donc quel réduction faire. On remontera donc dans la pile et on stockera à la place du α la lettre A et le nouvel état courant.
La reconnaissance se fait alors par pile vide.

exemple : Pour la grammaire

$S \to S_1 c \qquad S_1 \to S_1 A \mid A \qquad A \to a S_1 b \mid a b$ qui est une grammaire LR(0)

on a les items

$S \to \cdot S_1 c \qquad S_1 \to \cdot S_1 A \qquad A \to \cdot a S_1 b$

$S \to S_1 \cdot c \qquad S_1 \to S_1 \cdot A \qquad A \to a \cdot S_1 b$

$S \to S_1 c \cdot \qquad S_1 \to S_1 A \cdot \qquad A \to a S_1 \cdot b$

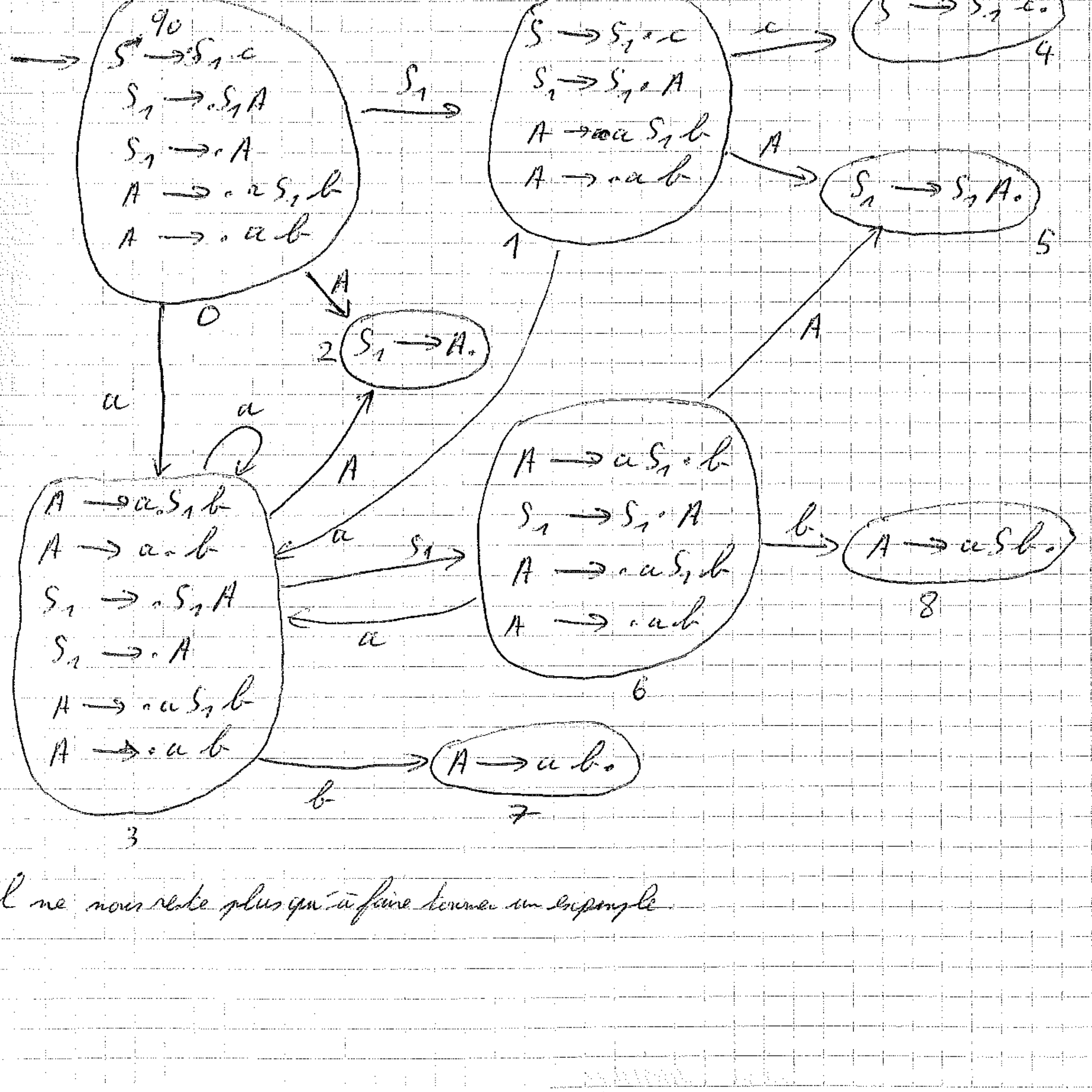
$S_1 \to \cdot A \qquad A \to a S_1 b \cdot$

$S_1 \to A \cdot \qquad A \to \cdot a b$

$A \to a \cdot b$

$A \to a b \cdot$

l'automate (déjà déterminisé) est alors :



Il ne nous reste plus qu'à faire tourner un exemple

pile	mot
0	a a b a b b c
0 a 3	a b a b b c
0 a 3 a 3	b a b b c
0 a 3 a 3 b 7	a b b c
0 a 3 A 2	a b b c
0 a 3 S_1 6	a b b c
0 a 3 S_1 6 a 3	b b c
0 a 3 S_1 6 a 3 b 7	b c
0 a 3 S_1 6 A 5	b c
0 a 3 S_1 6	b c
0 a 3 S_1 6 b 8	c
0 A 2	c
0 S_1 1	c
0 S_1 1 c 4	—

accept.

réduction.
pour avoir la fonction de transition d'une réduction $A \to \alpha$
on remonte de α dans la pile on cherche alors dans le dernier état avant la lecture de α où on serait allé si on avait lu A
on va dans cette état et on insert dans la pile A et le nouvel état courant