afcet

les techniques de l'informatique

CONGRÈS AFCET 72

Grenoble
6-9 novembre 1972

BROCHURE 1

UN ANALYSEUR SYNTAXIQUE POUR GRAMMAIRES HORS CONTEXTE ASCENDANT SELECTIF ET GENERAL .

P. COUSOT
Institut de mathématiques appliquées de Grenoble

Construire un analyseur syntaxique c'est passer d'un algorithme de génération du langage (grammaire hors contexte) à un algorithme de reconnaissance du langage. Pour des raisons d'efficacité le modèle de reconnaissance (précédence d'opérateur, matrices de transition, langage de production...) n'est pas équivalent au modèle de génération. Le passage du premier au second consiste à ne retenir qu'un certain type d'informations données par la grammaire. Ce filtrage correspond à une perte d'informations. On définit alors la classe des grammaires pour lesquelles l'analyseur fonctionne correctement, c'est-à-dire la classe des grammaires qui ne contiennent pas d'informations utiles éliminées à priori par le constructeur de l'analyseur. Suivant la qualité d'information perdue par différents constructeurs, les classes de grammaires correspondantes peuvent se hiérarchiser dans un arbre d'inclusion ([1] p191). On ne peut malheureusement pas donner de critères permettant d'écrire une grammaire entrant dans l'une ou l'autre des classes.

Notre approche respecte trois principes:

- l'analyseur doit être sélectif. Il est le résultat d'un algorithme dit 'constructeur' qui effectue un prétraitement de la grammaire pour réduire le nombre d'opérations élémentaires de l'analyseur. (DE REMER |3|, BACKES |4|).
- l'algorithme est général et multiple. (EARLEY |2|).

A ce stade la donnée de la grammaire ou de l'analyseur est équivalente, (il existe un algorithme de passage de l'une à l'autre forme, les informations étant plus explicites dans l'analyseur).

- l'analyseur est filtré : les informations non utiles pour l'analyse sont éliminées par un optimiseur. (ICHBIAH & MORSE |7|).

II - Présentation informelle de l'algorithme d'analyse.

Le lecteur non initié trouvera dans [1] une présentation des notions de syntaxe et d'analyse syntaxique.

Considérons la grammaire G1 dont l'ensemble des productions est:

P1: S -> |- A -|
P2: A -> a A
P3: A -> a

L'analyse de la chaine : '|- a a -|' commence par la 'lecture' de '|-': '|- / a a -|'. Le symbole terminal 'a' est le seul successeur terminal possible de '|-'. Le 'a' que nous avons reconnu est soit 'a' de la régle P3 (que nous notons a(3,1)), soit celui de la régle P2 (noté a(2,1)). La lecture de 'a' conduit à:

|- a(3,1) / a -| ou à |- a(2,1) / a -|

- Dans le premier cas, l'auteur a remplacé 'A' par 'a' dans la 'séquence de génération canonique'.En analyse ascendante il faut remplacer 'a' par 'A'. (Nous appelons cette opération 'réduction').
- Dans le deuxième cas , l'auteur a utilisé la régle:

A -> a A

Ayant reconnu 'a', il faut reconnaitre 'A', c'est-à-dire lire l'un de ses débutants terminaux soit 'a(2,1)' ou 'a(3,1)'. D'après la grammaire a(2,1) est suivi de 'a' (a(2,1) ou a(3,1)) alors que a(3,1) a comme 'contexte droit': '-|'. Nous avons donc reconnu a(2,1):

|- a(2,1) / a -|

La lecture de 'a' suivi de '-|' permet de reconnaitre a(3,1):

|- a(2,1) a(3,1) / -| .

la réduction P3 consiste alors à remplacer sur la 'pile syntaxique' (à gauche du délimiteur '/' précédant le 'symbole d'entrée'), le membre droit de la régle P3 par son membre gauche :

|- a(2,1) A / -|

Le 'A' qui est sur la pile syntaxique peut être A(1,2) ou A(2,2). C'est A(2,2) parce que A(2,2) est précédé de 'a(2,1)' alors que A(1,2) a '|-' en 'contexte gauche':

|- a(2,1) A(2,2) / -|

Réduction P2:

|- A / -|

'reconnaissance' de A(1,2) ('|-' en contexte gauche):

|- A(1,2) / -|

Lecture de l'un des suivants terminaux de A(1,2) soit '-|':

|- A(1,2) -| /

Réduction P1:

S / ,fin.

L'analyseur syntaxique est une table qui permet d'après les conditions de contexte de reconnaitre un symbole de la grammaire. Pour chaque symbole on indique une réduction, une lecture de l'un des successeurs terminaux, ou une reconnaissance. (Voir la table syntaxique de la grammaire G1 à la figure 2 pour suivre l'exemple ci-dessus, avec les conventions suivantes:

Dans la colonne de droite:

* signifie 'lire la chaine d'entrée donnée en troisième colonne'

X(p,1) 'mettre X(p,1) sur la pile syntaxique et aller à l'étiquette X(p,1)',

X ou 'aller à E ' signifie 'aller respectivement à l'étiquette X ou E'.

Dans la quatrième colonne (reduction):

-Lp-> signifie 'enlever Lp symboles sur la pile syntaxique').

III - Définitions.

1) Etant donné une grammaire hors contexte G = { Vt , Vnt , S , Π}, nous ordonnons les régles du système de productions. Le 1-ème symbole X du membre droit de la p-ème régle est noté X(p,1) , (ouΠ(p,1)), X(p,1) caractérise le premier symbole à gauche.

Nous supposons qu'il y a n régles et notons par Li la longueur de la i-ème régle.

le 'vocabulaire terminal indexé' est:

$$\overline{Vt} = \{ a(p,1) \mid (a \in Vt) \wedge (\Pi(p,1) = a) \}$$

le 'vocabulaire non terminal indexé' est:

$$\overline{Vnt} = \{ A(p,1) \mid (A \in Vnt) \wedge (\Pi(p,1) = A) \}$$

D'après sa construction le 'système de production indexé', vérifie la propriété suivante: $\{ \overline{\pi} \in Vnt \times (\overline{Vt} \cup \overline{Vnt})^* \}$.
Soit $X \in (Vt \cup Vnt)$, 'l'ensemble des occurences de X dans $\overline{\pi}$' est :

$$O(X) = \{ X(p,k) \mid X(p,k) \in (\overline{Vt} \cup \overline{Vnt}) \}$$

Exemple : pour la grammaire G1, le système de production indexé est:

S -> |-(1,1) A(1,2) -|(1,3)
A -> a(2,1) A(2,2)
A -> a(3,1)

$O(A) = \{A(1,2) , A(2,2)\}$.

Pour une présentation formelle des grammaires indexées voir [5].

2) Sans perte de généralité nous supposerons:
- Que la grammaire ne contient pas de symboles inaccessibles ou parasites.
- Qu'aucune régle n'est vide (les résultats peuvent être facilement étendus à ce cas).
- Qu'il y a une régle de la forme S -> |- S' -| où 'S'' est un axiome subordonné et 'S' et les terminaux '|-' et '-|' n'apparaissent dans aucune autre régle.
- Qu'il y a K symboles '|-' au début de la chaine d'entrée à analyser et K' symboles '-|' à la fin pour permettre l'utilisation de K symboles en contexte à gauche et de K' à droite. (K et K' sont choisis par l'utilisateur).

3) Nous notons '=>' la 'dérivation' au sens classique, '≡>' en est la forme induite quand on utilise une grammaire indexée. '$\overset{*}{\equiv}>$' est la fermeture transitive de '≡>'.

le 'k-contexte droit terminal indexé' de X(p,1) est:

$$\overline{RTCk}(X(p,1)) = \{ \bar{x} \mid (\bar{x} \in \overline{Vt}^*) \wedge (|\bar{x}| = k) \wedge (\exists\, \bar{y},\bar{z} : S \overset{*}{\equiv}> \bar{y}\, X(p,1)\, \bar{x}\, \bar{z}) \}$$

le 'k-contexte droit terminal de X(p,1)' est :

$$RTCk(X(p,1)) = \{ x \mid (x \in Vt^*) \; (|x| = k) \; (\exists\, \bar{y},\bar{z} : S \overset{*}{\equiv}> \bar{y}\, X(p,1)\, \bar{x}\, \bar{z}) \}$$

Le 'k-contexte gauche indexé de X(p,1)' est :

$$\overline{LCk}(X(p,1)) = \bar{x} \mid (\bar{x} \in (\overline{Vt} \cup \overline{Vnt})^*) \; (|\bar{x}| = k) \; (\exists\, \bar{y},\bar{z} : S \overset{*}{\equiv}> \bar{y}\, \bar{x}\, X(p,1)\, \bar{z})$$

(dans la séquence de génération ci-dessus, on effectue toujours la dérivation du symbole non terminal le plus à droite).

Pour illustrer ce dernier point prenons l'exemple de la grammaire G2:

S -> |-(1,1) A(1,2) -|(1,3)
A -> A(2,1) A(2,2)
A -> a(3,1)

En analyse ascendante nous n'aurons jamais '|-(1,1) a(3,1) A(2,2)' sur la pile syntaxique, parce qu'on effectue d'abord les réductions les plus à gauche: '|-(1,1) a(3,1)' puis '|-(1,1) A(2,1)'

puis '|-(1,1) A(2,1) a(3,1)' puis '|-(1,1) A(2,1) A(2,2)'. c'est pourquoi nous définissons LC de façon que a(3,1) n'appartienne pas à $\overline{LC1}(A(2,2)) = \{A(2,1)\}$.

IV - Le constructeur

1) Le constructeur reçoit en données la grammaire et fournit la table d'analyse syntaxique. Le lecteur est supposé avoir compris le fonctionnement de la table (fig. 2) pour la grammaire G1, (exemple du Ch. II). Nous présentons notre algorithme avec un seul symbole en contexte,(K=K'=1).(EARLEY ([2],p 101) justifie ce point de vue).

2) Le constructeur fonctionne comme suit:

A) Pour chaque symbole X(p,1) de chaque régle nous générons l'étiquette X(p,1) dans la première colonne de la table puis:

a) Si 1<Lp (longueur de la p-ème régle)

a-1) soit $\overline{U} = \overline{RTC1}(X(p,1)) = \{a(p',1')\}$

soit $\overline{U}' = \{a(p',1') \mid (a(p',1') \in \overline{U}) \wedge ((\forall\, b(q,r) \in \overline{U}) \wedge ((q,r) \neq (p',1')) \rightarrow a \neq b)\}$

soit $\overline{U}'' = \{a(p',1') \mid (a(p',1') \in \overline{U}) \wedge (\exists\, b(q,r) \in \overline{U} : ((q,r) \neq (p',1') \wedge (a = b))\}$

Nous avons $\overline{U} = \overline{U}' \cup \overline{U}''$.

a-2)

a-2-1) Pour chaque a(p',1') de $\overline{U}'$ nous générons sur la ligne X(p,1) et éventuellement les suivantes:

- a dans la colonne symbole terminal,
- * a(p',1') dans la dernière colonne.

a-2-2) Pour chaque a(p',1') de $\overline{U}''$ nous déterminons les ensembles V(p',1') = $\overline{RTC1}$ (a(p',1')).Nous obtenons m ensembles si m est le nombre d'éléments de $\overline{U}''$. Nous dirons que l'ensemble Vi est disjoint des autres si:

$$\{\forall\, x \in Vi, \nexists\, y \in V1 \cup V2 \cup \ldots \cup Vi-1 \cup Vi+1 \ldots \cup Vm : x = y\}$$

a-2-2-1) Pour chaque ensemble Vi = V(p',1') disjoint des autres, nous générons:

- a /V(p',1')/ en colonne symbole terminal,
- * a(p',1') dans la dernière colonne.

a-2-2-2) Si un ensemble Vj = V(pj,1j) n'est pas disjoint de V(p1,11)...V(pq,1q), nous essayons de les séparer en une partie disjointe pour laquelle la génération est similaire a a-2-2-1 et en une partie non disjointe V.Pour V(p1,11)...V(pq,1q) nous générons une fois:

- a / V / en colonne symbole terminal,
- * i: a(p1,11),...,a(pq,1q) en dernière colonne.

Les symboles a(p1,11),...,a(pq,1q) sont dits inadéquats. Intuitivement, X(p,1) peut être suivi de terminaux différents (a-2-1) ou de mêmes terminaux (a-2-2) dont il faut déterminer l'index dans la grammaire. Pour cela on utilise le contexte droit(a-2-2-1), mais ceci ne suffit pas toujours pour lever l'ambiguité locale (a-2-2-2).

b) Si 1 = Lp

b-1) Si p = 1, nous générons '-3->' dans la colonne réduction et 'fin' en dernière colonne.

b-2) Si p > 1, Nous générons '-Lp->' en colonne réduction. Appelons Y le membre gauche de la p-ème régle, nous générons 'Y' dans la dernière colonne si 0(Y) a plus d'un élément et sinon 0(Y).

Pour la grammaire G1 ceci donne: figure 1

étiquette	pile syntaxique	symbole terminal	réduction	fin,saut lecture
|-(1,1):		a /a / a /-|/		* a(2,1) * a(3,1)
A(2,1):		-|		* -|(1,3)
-|(1,3):			-3->	fin
a(2,1):		a /a/ a /-|/		* a(2,1) * a(3,1)
A(2,2):			-2->	A
a(3,1):			-1->	A

B) Pour chaque non terminal A de {Vnt - S} tel que O(A) a plus d'un élément, nous procédons comme suit: Nous générons l'étiquette A en colonne 'étiquette'. Nous déterminons les paires (contexte gauche,contexte droit): U(p,1)=(LC1(A(p,1),RTC1=(A(p,1)) pour chaque A(p,1) de O(A). Nous séparons U(p,1) en un sous-ensemble disjoint D(p,1) et un sous-ensemble non disjoint ND(p,1). Pour chaque D(p,1) nous générons:

- D(p,1)|1| (contexte gauche) en colonne 'pile syntaxique',
- D(p,1)|2| (contexte droit) en colonne 'symbole terminal',
- A(p,1) en colonne 'fin,saut,lecture'.

Nous rassemblons les ND(p,1) identiques soit ND(p1,l1)...ND(pq,lq) , et générons:

- ND(p1,l1)|1| en colonne 'pile syntaxique',
- ND(p1,l1)|2| en colonne 'symbole terminal',
- i : A(p1,l1),...,A(pq,lq) en dernière colonne.

Les non-terminaux A(p1,l1),...,A(pq,lq) sont dits inadéquats. Le constructeur permet de regrouper les tests communs à différentes étiquettes, lors du processus de construction. La table syntaxique pour la grammaire G1 est donc:

	étiquette	pile syntaxique	symbole terminal	réduction	fin,saut, lecture
1	E1:|-(1,1):		a /a / a /-|/		* a(2,1) * a(3,1)
3	A(1,2):		-|		* -|(1,3)
4	-|(1,3):			-3->	fin
5	a(2,1):				allera E1
6	A(2,2):			-2->	A
7	a(3,1):			-1->	A
8	A:	|-(1,1)	-|		A(1,2)
9		a(2,1)	-|		A(2,2)

figure 2

V - fonctionnement de l'analyseur

La table syntaxique est utilisable telle quelle par un interpréteur,ou peut être livrée à un compilateur qui la transforme en un programme (ensemble d'appels de sous-routines).

1 - fonctionnement de l'analyseur déterministe.

Nous analysons la phrase: |- a a -| , relativement à la table de la figure 2. L'analyse commence par la lecture de '|-' et le symbole |-(1,1) sur la pile syntaxique. L'histoire de l'analyse est alors la suivante:

pile syntaxique	chaine d'entrée	ligne f.2	arbre syntaxique résultat
|-(1,1)	a a -|	1	|-(1,1)
|-(1,1) a(2,1)	a -|	5	a(2,1)
|-(1,1) a(2,1) a(3,1)	-|	2	a(3,1)
|-(1,1) a(2,1)	-|	7	↑
|-(1,1) a(2,1) A(2,2)	-|	9	A(2,2)
|-(1,1)	-|	6	↑
|-(1,1) A(1,2)	-|	8	A(1,2)
|-(1,1) A(1,2) -|(1,3)		3	↑ -|(1,3)
———	__	4	S

La condition de fin:' la pile syntaxique et la chaine d'entrée sont vides' est vraie, la phrase '|- a a -|' appartient au language généré par la grammaire G1.

Pour illustrer l'intérêt d'indexer la grammaire prenons l'exemple G3 (|4| p.176) :

```
S -> |-(1,1) S'(1,2) -|(1,3)
S'-> a(2,1) A(2,2)   |   b(3,1) B(3,2)
A -> c(4,1) A(4,2)   |   X(5,1)
B -> c(6,1) B(6,1)   |   X(7,1)
X -> d(8,1)
```

Dans l'analyse de la phrase '|- a c c c c d -|' l'algorithme de DE REMER ne peut prendre la décision de réduire 'X' en 'A' ou 'B' car les symboles essentiels 'a' ou 'b' sont arbitrairement loins sur la pile syntaxique. Dans le cas d'indexation, la table se présente comme suit:

X	a(2,1),c(4,1)			X(5,1)
	b(3,1),c(6,1)			X(7,1)

et le contexte gauche c(4,1) sur la pile syntaxique:

|-(1,1) a(2,1) c(4,1) c(4,1) c(4,1) c(4,1)

permet de reconnaitre c(4,1) donc x(5,1) donc A.

2 - fonctionnement de l'analyseur non déterministe.

Considérons la grammaire ambigue G2:

```
S -> |-(1,1) A(1,2) -|(1,3)
A -> a(2,1) A(2,2)  |  a(3,1)
```

La table syntaxique est donnée par le constructeur:

	étiquette	pile syntaxique	symbole terminal	réduction	saut,fin, lecture.
1	E1:|-(1,1):		a		* a(3,1)
2	A(1,2):		-|		* -|(1,3)
3	-|(1,3):			-3->	fin
4	A(2,1):				aller à E1
5	A(2,2):			-2->	A
6	a(3,1):			-1->	A
7	A:	|-(1,1)	-|		A(1,2)
8			a		A(2,1)
9		A(2,1)	-|		A(2,2)
10			a		i:A(2,1),A(2,2)

figure 3

A la ligne 10 de la fig. 3, A(2,1) et A(2,2) sont inadéquats. Les informations en contexte gauche et droit ne permettent pas de choisir entre A(2,1) et A(2,2). Pour résoudre ce problème nous utilisons la 'méthode paralléle' de GREIBACH [8]. Nous ne faisons pas de choix entre les états inadéquats (ce qui revient à choisir un arbre syntaxique particulier), mais nous considérons 'simultanément' toutes les analyses possibles. En fait, les fonctions primitives de l'analyseur sont modifiées, de facon à fonctionner par 'étapes' sur chaque analyse. Une étape est la période de fonctionnement de l'analyseur commencant par une opération de lecture et se terminant avant l'opération de lecture suivante. (la lecture est notée '*' dans la table syntaxique).

La pile syntaxique doit pouvoir contenir plus d'une analyse à la fois: la pile est donc multiple et chaque élément de cette pile est un ensemble de symboles. Pour maintenir les informations de contexte gauche, un ou plusieurs pointeurs sont associés à chaque symbole. On appelle 'état' l'ensemble 'symbole et pointeurs associés'. On ajoute un état au sommet de la pile, à moins qu'il n'ait déjà été ajouté dans une même position de la fenêtre d'entrée,c'est-à-dire qu'on ajoute un symbole et un pointeur ou un pointeur ou rien. Considérant successivement, en position 'i' de la fenêtre d'entrée, chaque état de l'ensemble Si (sommet de pile), l'analyseur fait une étape de facon déterministique sur cet etat(en maintenant correctement les pointeurs de contexte gauche). Si l'étape se termine avant une opération de lecture, on ajoute alors l'état correspondant dans l'ensemble Si+1, et on passe à l'état suivant. Si l'étape se termine par une tentative d'ajouter à Si+1 un état existant, on passe à l'état suivant dans Si. Quand tous les états de l'ensemble Si ont été considérés, on avance la fenêtre d'entrée et on passe à l'ensemble Si+1. Si l'ensemble Si+1 est vide la phrase est incorrecte. L'analyse se termine avec succès si la pile et la chaine d'entrée sont vides à la rencontre de l'opération 'fin'.

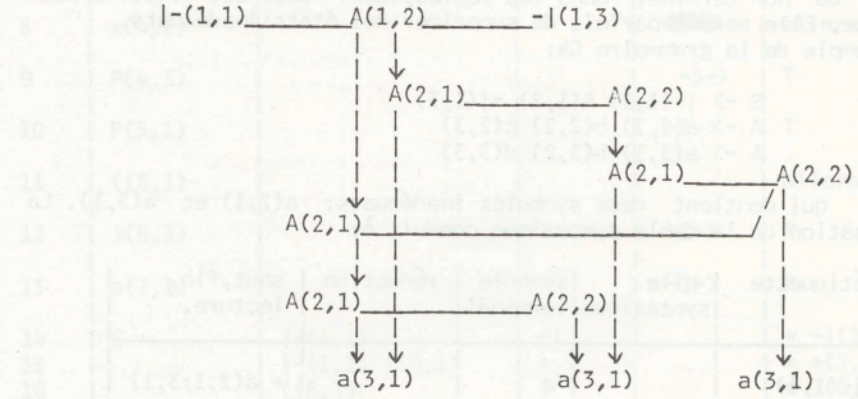
Nous montrons maintenant l'histoire de la reconnaissance de la phrase: '|- a a a -|' pour la grammaire G2. Nous repérons par '.' l'analyse en cours:

pile syntaxique	chaine d'entrée	ligne fig.3	commentaires
|-(1,1) ∘	a a a -|		état initial, mode determinis.
|-(1,1)←a(3,1) ∘	a a -|	1	
|-(1,1) ∘	a a -|	6	
|-(1,1)←A(2,1) ∘	a -|	8	
|-(1,1)←A(2,1)←a(3,1) ∘	a -|	1	
|-(1,1)←A(2,1) ∘	a -|	6	passage en mode non déterministe
|-(1,1)←A(2,1)←A(2,1) A(2,2) ∘	a -|	10	
|-(1,1) ∘ ←A(2,1)←A(2,1)	a -|	5	
|-(1,1) A(2,1) ∘ ←A(2,1)←A(2,1)	a -|	8	
|-(1,1)←A(2,1)←A(2,1) ∘ a(3,1)	a -|	1	fin de la 3-ème étape pour une analyse
|-(1,1)←A(2,1)←A(2,1)←a(3,1)	-|	1	état existant, fin de la 3-ème étape.
|-(1,1)←A(2,1)←A(2,1)← ∘	-|	6	Contexte gauche analyse en une seule fois.
|-(1,1)←A(2,1)←A(2,1)←A(2,2) ∘	-|	9	réduction multiple
|-(1,1) ∘ ←A(2,1)←A(2,1)←A(2,2)	-|	5	
|-(1,1) A(1,2) ∘ ←A(2,1)←A(2,1)←A(2,2)	-|	7	
|-(1,1)←A(2,1)←A(2,1)←A(2,2) ∘ -J(1,3)	-|	2	fin de la 4-ème étape pour une analyse.
|-(1,1)←A(2,1) ∘ -|(1,3)	-|	5	
|-(1,1)←A(2,1)←A(2,2) ∘ -|(1,3)	-|	9	
|-(1,1) ∘ ←A(1,2)←-|(1,3)	-|	5	
|-(1,1)←A(1,2)←-|(1,3) ∘	-|	7	état existant, fin étape, passage en mode déterm.
———	———	3	fin

Pour respecter les contraintes de temps, nous conservons pour chaque ensemble d'états Si et chaque symbole X une liste des états X(p,q).

Pour obtenir l'analyseur syntaxique multiple nous transformons le reconnaisseur pour qu'il construise l'arbre syntaxique en cours d'analyse. Après chaque réduction qui ajoute un symbole C(n,m) tel que G(n,m)-->D(p,1)...D(p,p'), on construit un pointeur de cette occurence de G vers le symbole D(p,1) correspondant à D(p,p') qui a donné lieu à l'opération de réduction. Dans le cas où G est ambigu il y aura un ensemble de pointeurs fils, un pour chaque opération qui l'a fait ajouter à un ensemble d'états particulier. Chaque symbole D(p,1),...,D(p,p') aura des pointeurs fils (à moins qu'il ne soit terminal) etc...Quand on atteint l'opération de fin on obtient l'arbre syntaxique de la phrase analysée ou une représentation factorisée de tous les arbres possibles dans le cas d'une phrase ambigue.

Ceci donne pour l'exemple précédant:



VI - Optimisation de la table syntaxique

1) factorisation

Nous reprenons maintenant l'idée de factorisation de FOSTER [9] développée par BORDIER [6]. En l'appliquant à la table syntaxique et non à la grammaire, on ne fait pas disparaître arbitrairement les informations nécessaires pour l'analyse. (Dans le cas de la grammaire G3, on ne peut factoriser c(4,1) et c(6,1)).

Soit X ∈ (Vt U Vnt), on peut factoriser X(p1,q1),...,X(pn,qn) dans les conditions suivantes: (n>1)

$$\text{Soit } R(X)=\{\ X(pj,qj) \mid (qj=Lpj) \wedge (\exists y \in Vt, \exists i \in [1,n], i\neq j : (y \in RTC1(X(pj,qj))) \wedge (y \in RTC1(X(pi,qi)))\ \}$$

(Eléments demandant une réduction et ayant un contexte droit non disjoint).

$$\text{Soit } R'(X) = \{\ X(pi,qi) \mid \exists X(pj,qj) \in R\ ,\ i\neq j : qi\neq qj\ \}$$

(R' est vide si les réductions précédentes sont toutes de même longueur).

- Il ne faut pas que plus d'un X(pi,qi),(i ∈ [1,n]) soit utilisé en contexte gauche (colonne pile syntaxique) sous une même étiquette, ou qu'ils soient distinguables dans le cas contraire par contexte droit.

- Il faut que:

$\{ RTC1(X(pi,qi) \neq RTC1(X(pj,qj)) , \forall (i,j) \in [1,n]2 , i \neq j \}$

ou sinon,(après élimination des X(pi,qi) vérifiant la condition ci-dessus):

Soit $S=\{ y \mid \exists (i,j) \in [1,n]2 : (y \in RTC1(X(pj,qj)) \wedge (y \in RTC1(X(pi,qi)) \}$ il faut alors que:

. $R(X) = \phi$ et on peut factoriser O(S).

. ou que $\{(R'(X)= \phi) \wedge (R(X)=X(p1,q1),...,X(pn,qn)) \wedge$ (on peut factoriser O(D))$\}$ où D est l'ensemble des membres gauches des régles p1,...,pn).

La factorisation réduit la taille de la table syntaxique, en éliminant la distinction entre les différentes occurences d'un même terminal ou non terminal dans les régles,quand ceci est inutile pour l'analyse. Elle permet parfois de supprimer des états inadéquats.

Exemple de la grammaire G4:

S -> |-(1,1) A(1,2) -|(1,3)
A -> a(2,1) b(2,2) c(2,3)
A -> a(3,1) b(3,2) d(3,3)

qui contient deux symboles inadéquats: a(2,1) et a(3,1). La factorisation de la table syntaxique conduit à:

étiquette	pile syntaxique	symbole terminal	réduction	saut,fin, lecture.
|-(1,1)		a		* a(2,1;3,1)
A(1,2)		-|		* -|(1,3)
-|(1,3)			-3->	fin
a(2,1;3,1)		b		* b(2,2;3,2)
b(2,2;3,2)		c d		* c(2,3) * d(3,3)
c(2,3)			-3->	A(1,2)
d(3,3)			-3->	A(1,2)

2) substitution

La phase d'optimisation commence par une tentative de factorisation des éléments de O(X), pour tout X $\in$ (Vt U Vnt).

Pour la grammaire arithméthique G5:

S -> |-(1,1) E(1,2) -|(1,3)
E -> E(2,1) +(2,2) T(2,3)
E -> T(3,1)
T -> T(4,1) x(4,2) P(4,3)
T -> P(5,1)
P -> ((6,1) E(6,2))(6,3)
P -> a(7,1)

Ceci donne:

	étiquette	pile syntaxique	symbole d'entrée	réduction	fin,saut, lecture.
1	E1:|-(1,1):		a		* a(7,1)
2			(		* ((6,1)
3	-|(1,3)			-2->	fin
4	+(2,2)				allera E1
5	T(2,3)			-2->	E
6	T(3,1)			-1->	E
7	T(4,1)		x		* x(4,2)
8	x(4,2)				allera E1
9	P(4,3)			-3->	T
10	P(5,1)			-1->	T
11	((6,1)				allera E1
12	)(6,3)			-2->	P
13	a(7,1)			-1->	P
14	E	|-(1,1)	-|		* -|(1,3)
15		|-(1,1) ((6,1)	+		* +(2,2)
16		((6,1)	)		*)(6,3)
17	T	+(2,2)	-| +)		T(2,3)
18		|-(1,1)+(2,2)((6,1)	x		T(4,1)
19		|-(1,1) ((6,1)	-| +)		T(3,1)
20	P	x(4,2)	x -| +)		P(4,3)
21		|-(1,1)+(2,2)((6,1)	x -| +)		P(5,1)

figure 4

La factorisation de E(1,2),E(2,1),E(6,2) nous a conduit à supprimer ces étiquettes qui ne figureront plus sur la pile syntaxique, donc à diminuer de un les longueurs des réductions correspondantes. La factorisation de P(4,3),P(5,1) et de T(2,3),T(3,1),T(4,1) est impossible. D'après la construction de la table les étiquettes indexées de la dernière colonne (saut,fin,lecture) non précédées de 'allera' ne figurent qu'une seule fois dans cette colonne, si elles sont adéquates. Quand c'est possible on a intérêt à remplacer un saut à une ligne par la ligne elle-même, ce qui ne peut que diminuer la taille de la table et le nombre d'éléments à placer sur la pile syntaxique.

La ligne 17 de la figure 4 est un saut à la ligne 6, T(2,3) n'étant pas utilisé en contexte gauche, on peut supprimer la ligne 6 et remplacer la ligne 17 par:

17	T	+(2,2)	-| +)	-2-1->	E

La table prend alors la forme suivante:

	étiquette	pile syntaxique	symbole terminal	réduction	saut, fin, lecture.
1	E1:|-(1,1):		a		* P
2			(		* ((6,1)
3	+(2,2)				allera E1
4	x(4,2)				allera E1
5	((6,1)				allera E1
6	E	|-(1,1)	-|	-1->	* fin
7		|-(1,1) ((6,1)	+		* +(2,2)
8		((6,1)	)	-1->	* P
9	T	+(2,2)	-| +)	-1->	E
10		|+(2,2)|-(1,1)((6,1)	x		* x(4,2)
11		|-(1,1) ((6,1)	-| +)		E
12	P	x(4,2)	|x -| +)|	-1->	T
13		||-(1,1)+(2,2)((6,1)	|x -| +)|		T

figure 5

3) Restructuration

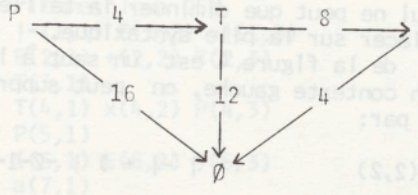
La phase finale de l'optimisation est une réduction de la taille de l'analyseur par l'algorithme de l'arborescence inverse à coût minimum [7]. Il s'agit de fusionner différents groupes de tests de contexte.

Considérons les tests de contexte (couples pile syntaxique - symbole d'entree) pour les étiquettes non indexées, en première colonne. Si deux étiquettes ou plus ont des tests de contexte et des actions respectives identiques elles sont considérées comme une seule étiquette. Le graphe G = (X,V) ou X est un ensemble de noeuds et V un ensemble d'arcs est défini comme suit:

- Un noeud de l'ensemble X est associé à chaque étiquette restante. En plus, X contient un noeud fictif Ø associé à une étiquette fictive correspondant à une ligne vide.
- Soient deux noeuds distincts n1 et n2 de X. L'arc (n1,n2) appartient à V si les tests de contexte de n2 sont inclus dans ceux de n1 et si les tests de contexte de n2 ne correspondent sous n1 à aucune modification de contexte. (Pas de modification de la pile syntaxique, ni déplacement de la fenêtre d'entrée).

Un coût est associé à l'arc (n1,n2) qui représente le nombre de tests associés à n1 qui ne sont pas sous l'étiquette n2.

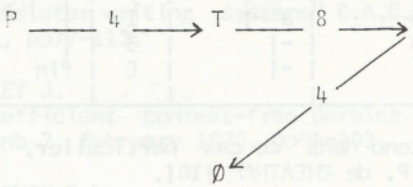
Exemple,(fig. 5):



Il y a quatre tests de contexte sous E { (|-(1,1),-|) ; (|-(1,1),+) ; (((6,1),+) ; (((6,1),)) } qui sont sous T. Ces conditions ne correspondent sous T à aucune modification de contexte (ligne 6,figure 5).

L'arborescence inverse à coût minimum A = (X,W) de G = (X,V) se construit de la façon suivante:

- W ne contient pas d'arc d'origine Ø,
- pour chaque noeud n1 dans X autre que Ø, W contient un arc unique d'origine n1. C'est l'arc de V qui a le coût minimum parmi les arcs d'origine n1.



La restructuration de l'analyseur syntaxique (figure 5) conduit au résultat suivant:

	étiquette	pile syntaxique	chaine d'entrée	réduction	saut,lecture fin,erreur
1		|-(1,1)+(2,2)	a		* P
2		x(4,2)((6,1)x|	(		* ((6,1)
					erreur
3	P:	x(4,2)	x -| +)	-1->	T
4	T:	+(2,2)	+) -|	-1->	E
5		+(2,2)|-(1,1)((6,1)	x		* x(4,2)
6	E:	|-(1,1)	-|	-1->	* fin
7		|-(1,1) ((6,1)	+		* +(2,2)
8		((6,1)	)	-1->	* P
9					erreur

figure 6

Pour chaque ligne de la colonne 'pile syntaxique', nous éliminons les symboles de la colonne 'chaine d'entrée' si aucun des symboles en contexte gauche ne se retrouve sur les lignes précédant la prochaine ligne d'erreur. En répétant cette opération pour la colonne 'chaine d'entrée' nous obtenons:

3	P:	x(4,2)	
4	T:	+(2,2)	+) -|
5			x
6	E:		-|
7			+
8		((6,1)	

figure 7

Nous présentons maintenant l'histoire de l'analyse de la phrase '|- (a + a) x a -|' relativement à la grammaire G5. L'arbre résultat a des noeuds de type P,+,x,fin si l'utilisateur a écrit des fonctions sémantiques pour P, +(2,2), x(4,2) et fin:

pile syntaxique	chaine d'entrée	ligne fig.6	arbre résultat
|-(1,1)	(a+a)xa-|	2	
|-(1,1) ((6,1)	a+a)xa-|	1	P
|-(1,1) ((6,1)	+a)xa-|	7	+
|-(1,1) ((6,1) +(2,2)	a)xa-|	1	P
|-(1,1) ((6,1) +(2,2)	)xa-|	4	
|-(1,1) ((6,1)	)xa-|	8	P
|-(1,1)	xa-|	5	x
|-(1,1) x(4,2)	a -|	1	P
|-(1,1) x(4,2)	-|	3	
|-(1,1)	-|	6	fin

L'analyseur obtenu dans ce cas particulier, est une forme simplifiée du système R.A.P. de CHEATHAM |10|.

Reprise d'erreur.

Une erreur se produit quand le symbole terminal sous la 'fenêtre d'entrée' ne correspond à aucun des terminaux attendus en colonne 'chaine d'entrée'. Nous supposerons les mots clés réservés. Parmi les symboles attendus, on choisit, par l'algorithme de MORGAN [11], celui qui se rapproche le plus du symbole sous la 'fenêtre d'entrée'. Deux cas sont à distinguer:

- Parmi les symboles attendus, il y a un terminal servant en contexte gauche. Un choix du 'symbole d'entrée' est impératif pour conserver la 'pile syntaxique' dans un état ne conduisant jamais à une erreur en contexte gauche. Exemple:
"* a -|" est remplacé arbitrairement par "x a -|" ou "+ a -|".
- L'algorithme de MORGAN échoue, et il n'y a aucun terminal servant en contexte gauche parmi les symboles attendus. L'analyseur lexicographique avance la 'fenêtre d'entrée' sur le premier mot clé correct identique à un symbole X de la colonne 'chaine d'entrée'. La 'pile syntaxique' est consultée jusqu'à ce que le contexte gauche soit compatible avec le X retenu. Si on atteint le fond de la pile, on avance à nouveau la 'fenêtre d'entrée', sinon on élimine la partie consultée pour continuer l'analyse. Exemple de la grammaire G4:
"|-(1,1) a(2,1;3,1) b(2,2;3,2) / 7 -|" ==> " |-(1,1) / -|"

VII - Conclusion.

Quand on écrit un compilateur pour un langage, pourquoi choisit-on une forme particulière de la grammaire?
1) parce que l'analyseur syntaxique utilise n'accepte que les grammaires d'une classe donnée.
2) parce que l'important dans l'écriture d'un compilateur étant l'aspect semantique, la forme de la grammaire est determinée par des considérations non syntaxiques.

Dans le dernier cas, notre méthode a l'avantage d'être générale et permet d'obtenir automatiquement un résultat efficace pour les grammaires courantes. (pas de symboles inadéquats, et raisonnable dans l'autre cas). Ceci a été obtenu par la synthèse d'une méthode générale |2| et d'optimisations classiques [3],[7]. Le coût d'analyse pour une grammaire donnée est "proportionnel" à la "complexité" de la grammaire.

Remerciements.
Je remercie S. SCHUMAN pour les idées de base et son aide dans cette recherche. J'ai également profité des conseils de P. JORDAND et d'une critique detaillée de M. GRIFFITHS.

VIII - Bibliographie

[1] FELDMAN J. & GRIES D.
"translator writing systems" C.A.C.M.,vol 11,nb 2, february 1968, pp77-112.

[2] EARLEY J.
"an efficient context-free parsing algorithm", C.A.C.M., vol 13, nb 2, february 1970, pp94-102.

[3] DE REMER F.L.
"simple LR(k) grammars", C.A.C.M., vol 14, nb 7, july 1971, pp 453-460.

[4] BACKES S.
"top down syntax analysis and FLOYD-EVANS production language", I.F.I.P. congress 1971, Ljubljana, august 1971, booklet TA-3 pp 173-177.

[5] TERRINE G.
"construction automatique d'analyseurs syntaxiques ascendants déterministes à partir de C.F. grammaires éventuellement non de contexte borné", Thése de troisième cycle, Université scientifique et médicale de Grenoble. 11 mars 1972.

[6] BORDIER J.
"méthodes pour la mise au point de grammaires LL(1)", Thése de troisième cycle, Faculté des Sciences de l'Université de Grenoble. Juillet 1971.

[7] ICHBIAH J.D. & MORSE S.P.
"a technique for generating almost optimal FLOYD-EVANS productions for precedence grammaires", C.A.C.M., vol 13, nb 8, august 1970, pp 501-508

[8] GREIBACH S.A.
"formal parsing systems", C.A.C.M., 1964.

[9] FOSTER J.M.
"a syntax improving device", computer journal, may 1968.

[10] CHEATHAM T.E.
"the introduction of definitional facilities into higher level programming languages", Proceedings, fall joint computer conference, 1966.

[11] MORGAN H.L.
"spelling correction in system programs", C.A.C.M., vol 13, nb 2, february 1970, pp 90-94