



Sémantique

Mini-projet: Parties I et II

Yvette Le Bras, Jean-Marie Le Yaouanc

Brest, 7 avril 2005

Enseignant responsable : D. Sarni

Département informatique

UFR Sciences et Techniques 6 avenue Le Gorgeu 29200 Brest

Master 1 informatique

Table des matières

1	Partie I	1
1.1	Question 1 : Syntaxe d'une expression fonctionnelle	1
1.2	Question 2 : L'arbre syntaxique	1
1.3	Question 2 bis : Réduction	2
1.4	Question 3 : Codage matriciel d'une EFS	2
1.5	Question 4 : Codage de l'opérateur PROD	3
1.6	Question 5 : Sémantique dénotationnelle interprétée	4
	1.6.1 Modifications à apporter	4
	1.6.2 Définition de la sémantique dénotationnelle L'	4
2	Partie II	6
2.1	États de la machine	6
2.2	Les instructions de la machine	6
2.3	Algorithme de compilation COMP	8
	2.3.1 Le programme	8
	2.3.2 Les traces d'exécution	10
2.4	Comportement de la machine sur l'EFS Ξ de la section 1.4	14

Chapitre 1

Partie I

1.1 Question 1 : Syntaxe d'une expression fonctionnelle

$\langle CAT \rangle \rightarrow \langle EFS \rangle \langle seqE \rangle$
 $\langle EFS \rangle \rightarrow (\ \langle seqN \rangle \ M)$
 $\langle seqN \rangle \rightarrow \langle varN \rangle \ | \ \langle seqN \rangle \langle varN \rangle$
 $\langle M \rangle \rightarrow (\ \langle M \rangle) \ | \ \langle M \rangle \langle op \rangle \langle varN \rangle \ | \ \langle varN \rangle$
 $\langle seqE \rangle \rightarrow \langle ent+ \rangle \ | \ \langle seqE \rangle \langle ent+ \rangle$
 $\langle op \rangle \rightarrow$ ensemble des opérateurs arithmétiques

On remarquera que le nombre d'arguments de $seqE$ est égal au nombre de variables de la séquence $seqN$ soit $\text{card}(seqE) = \text{card}(seqN)$. Plutôt que de préciser cette condition, une autre solution est d'utiliser une syntaxe qui à chaque ajout de $\langle varN \rangle$ en début de l'**EFS** ajoute aussi $\langle ent+ \rangle$ en fin d'**EFS**. Cependant, cette seconde solution n'utilise pas les données du problème et nous paraît donc moins judicieuse.

1.2 Question 2 : L'arbre syntaxique

L'arbre syntaxique permettant la construction de l'expression $(x_1 x_2 (x_1 + x_2) * x_2)$ est illustré à la figure 1.2 page 2.

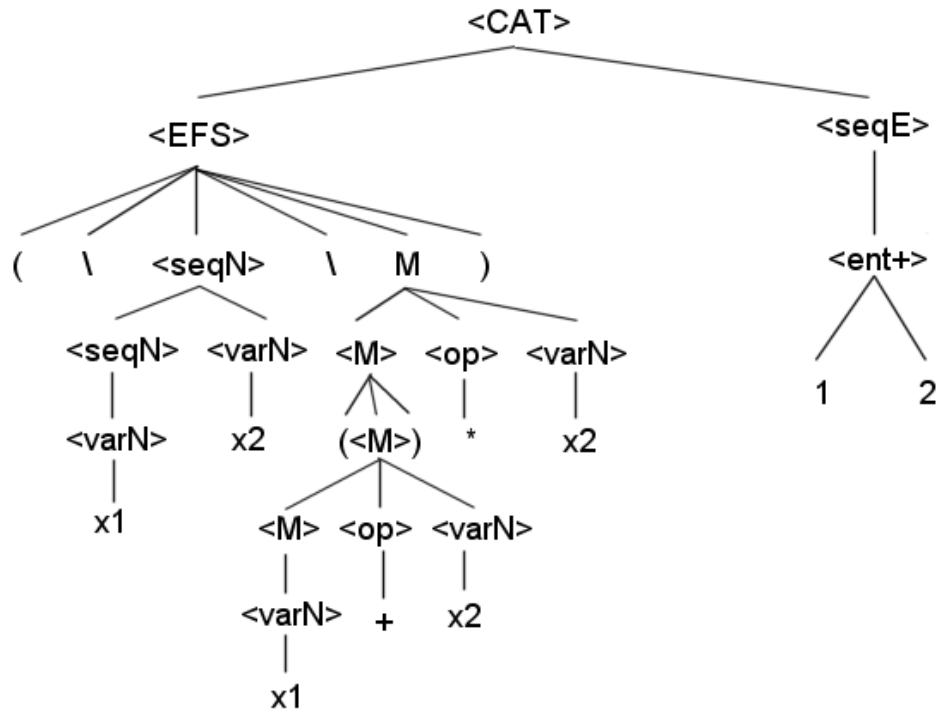


FIG. 1.1 – Arbre syntaxique

1.3 Question 2 bis : Réduction

$\langle CAT2 \rangle \rightarrow \langle PROD \rangle \langle vectE \rangle \langle vectS \rangle$
 $\langle PROD \rangle \rightarrow PROD$
 $\langle vectE \rangle \rightarrow [\langle seqE \rangle]$
 $\langle seqE \rangle \rightarrow \langle ent+ \rangle \mid \langle seqE \rangle, \langle ent+ \rangle$
 $\langle vectS \rangle \rightarrow [\langle seqS \rangle]$
 $\langle seqS \rangle \rightarrow \langle Symb \rangle, \langle seqE \rangle$
 $\langle Symb \rangle \rightarrow \langle op \rangle \mid \langle Symb \rangle, \langle op \rangle$

1.4 Question 3 : Codage matriciel d'une EFS

On remarque que la matrice ne contient qu'un 1 au maximum par colonne donc on peut représenter $[\Xi]$ par un simple vecteur dont ses données représentent les numéros de lignes des 1 dans la matrice.

$$\Xi = \begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

La dimension de la matrice est de 9×15 soit 135. Elle peut être représentée par le vecteur $V(\Xi)$ et

$$V(\Xi) = (1 \ 1 \ 7 \ 3 \ 8 \ 2 \ 5 \ 9 \ 2 \ 6 \ 1 \ 7 \ 3 \ 9 \ 2)$$

1.5 Question 4 : Codage de l'opérateur PROD

Si A est la séquence de 3 arguments, alors $PROD([A][\Xi])$ représente le produit des matrices $[A]$ et $[\Xi]$.

$$PROD([A][\Xi]) = [() + - * / 1 2 3] * [\Xi] = ((1 + 2) * 3) / (1 + 3).$$

On remarque que le résultat obtenu correspond à la réduction de notre expression Ξ . On généralise cette remarque en admettant que le résultat du produit par l'opérateur $PROD$ d'une **EFS** et de ses arguments correspond à la réduction de la dite **EFS**.

En langage algorithmique, le programme réalisant l'opération $PROD$ est alors :

```
PROD(vecteur ENTIER A, vecteur ENTIER E, vecteur ENTIER P, ENTIER taille)
{
  ENTIER i;
  POUR i = 1 jusqu'à taille
  {
    P<i> = A< E<i> >
  }
}
```

1.6 Question 5 : Sémantique dénotationnelle interprétée

1.6.1 Modifications à apporter

A partir de L, il est possible de définir une sémantique dénotationnelle L'. Cependant, quelques modifications restent à apporter : une fonction de conversion d'un tableau de caractères vers un tableau d'entiers est nécessaire ; De plus, on suppose possible l'existence de tableau d'entiers à deux dimensions.

1.6.2 Définition de la sémantique dénotationnelle L'

Domaine syntaxique

- une expression fonctionnelle simple E
- un ensemble d'arguments A
- le résultat d'une EFS appliquée à A

Équivalences de types

- vectE \equiv tableau[ENTIER]
- vectS \equiv tableau[CHARACTERE]
- op \equiv varCar
- ent+ \equiv entier

Le programme RED est le suivant :

```
(chaîne) RED (EFS E, ENTIER f, tableau[ENTIER] A)
{
  tableau[ENTIER] tabPar0,tabParF,tabPlus,tabMoins,tabMult,tabDiv,tabXi,tabRed;
  tableau[ENTIER][ENTIER] tabMatrice = 0;
  tableau[CHARACTERE] tabA = [(, ), +, -, *, /, A];

  si (f == taille(tabA)){

    tabPar0 = occ(E; '(');
    tabParF = occ(E; ')');
    tabPlus = occ(E; '+');
    tabMoins = occ(E; '-');
    tabMult = occ(E; '*');
    tabDiv = occ(E; '/');
    Pour i=1 jusqu'à taille(A){
      tabXi = occ(chaine E; 'Xi');
    }
  }
}
```

```

    Pour j=1 jusqu'à taille(tabPar0) { tabMatrice [0][tabPar0[j]] = 1 ;}
    Pour j=1 jusqu'à taille(tabParF) { tabMatrice [1][tabParF[j]] = 1 ;}
    Pour j=1 jusqu'à taille(tabPlus) { tabMatrice [2][tabPlus[j]] = 1 ;}
    Pour j=1 jusqu'à taille(tabMoins) { tabMatrice [3][tabMoins[j]] = 1 ;}
    Pour j=1 jusqu'à taille(tabMult) { tabMatrice [4][tabMult[j]] = 1 ;}
    Pour j=1 jusqu'à taille(tabDiv) { tabMatrice [5][tabDiv[j]] = 1 ;}
    Pour k=6 jusqu'à (6+taille(A)) {
        Pour i=1 jusqu'à taille(A) {
            Pour j=1 jusqu'à taille(tabXi) { tabMatrice [k][tabXi[j]] = 1 ;}
        }
    }

    tabRed = PROD(tabMatrice; tabA);
    retourner (CONC(tabRed));
}

sinon écrire 'Nombre d'arguments incorrect';
}

```

Précision : dans ce programme **RED**, on crée la matrice $[\Xi]$ composée de 1 et de 0. On considère que A ne contient que les arguments numériques, $tabA$ contient l'ensemble de la matrice $[\mathbf{A}]$; Ξ ne contient pas $\backslash x_1 \dots x_n \backslash$ en son début.

Chapitre 2

Partie II

2.1 États de la machine

Schématiquement, une pile contenant l'item A en sommet, puis l'item B est représenté par $A/B/$. Un registre vide est représenté par $\dots/$. La machine est définie par deux piles P_1, P_2 et une 3^{ème} pile C contenant l'expression à réduire.

Variante : Puisque les arguments $(,), +, -, *, /$ sont identiques quelque soit l'**EFS**, on les place initialement dans la pile P_2 . L'instruction **LGARG** est alors uniquement associée à la séquence d'arguments numériques.

État initial :

P_1	P_2	C
$\dots/$	$($	$\dots/$
	$)$	
	$+$	
	$-$	
	$*$	
	$/$	

2.2 Les instructions de la machine

Exemple pour une expression simple $\Xi = \backslash X_1 X_2 \backslash (x_1 + x_2)$ avec comme arguments 1 et 2 : $V(\Xi) = (1\ 7\ 3\ 8\ 2)$.

P_1	P_2	C
...	(LD 1 7 3 8 2 : LDARG 1 2 : AP : EVAL
/)	
	+	
	-	
	*	
	/	

L'instruction **LD** est appliquée au vecteur $V(\Xi)$ et empile dans P_1 les composantes de $V(\Xi)$ dans le même ordre.

P_1	P_2	C
1	(LDARG 1 2 : AP : EVAL
7)	
3	+	
8	-	
2	*	
	/	

L'instruction **LDARG** est appliquée au vecteur $[A]$ associé à la séquence d'arguments 1 2 et empile dans P_2 les composantes de $[A]$ dans le même ordre.

P_1	P_2	C
1	(AP : EVAL
7)	
3	+	
8	-	
2	*	
	/	
	1	
	2	

L'instruction **AP** réalise la réduction en traitant les piles comme expliqué dans le sujet.

P_1	P_2	C
)	...	EVAL
2	/	
+		
1		
(

Puis l'instruction **EVAL** réalise le calcul de l'expression de P_1 et empile dans P_1 vidée le résultat de l'évaluation.

P_1	P_2	C
3	.../	.../

2.3 Algorithme de compilation COMP

2.3.1 Le programme

```
(tableau[CARACTERE]) COMP (EFS E, tableau[ENTIER] T)
{
  ENTIER index=0, i=0;
  CARACTERE lexeme;
  tableau[CARACTERE] code;
  tableau[ENTIER][CARACTERE] tabIdent;

  tant que (lectureLexeme != FIN)
  {

    //Lecture des Indent
    si (lexeme != '\') retourner 'Erreur de syntaxe';
    tant que (lexeme == IDENT)
    {
      si (chercherTabIdent(lexeme) == FAUX)
      {
        ajouterTabIdent(lexeme); nbIdent++;
      }
      sinon retourner 'Erreur: identifiant déjà présent'
    }
    si (lexeme != '\') retourner 'Erreur de syntaxe';

    //Codage de M
    code[index] = LD;
    index++;

    si (lexeme == '(' ) code[index] = 1;
    si (lexeme == ')' ) code[index] = 2;
    si (lexeme == '+' ) code[index] = 3;
    si (lexeme == '-' ) code[index] = 4;
    si (lexeme == '*' ) code[index] = 5;
    si (lexeme == '/' ) code[index] = 6;
```

```
    si (lexeme == IDENT) {
        si (i=chercherTabIdent(lexeme) == FAUX) {
            retourner 'Erreur: variable inconnue';
        }
        code[index] = 6+i;
    }
    sinon retourner 'Caractere inconnu';
    index++;
}

//Codage des arguments
code[index] = : LDARG;
index++;

tant que (T != VIDE)
{
    si (taille(T) != nbIdent)
    {
        retourner 'Erreur: Nombre d'arguments incorrect';
    }
    pour (j=1 jusqu'à taille(T))
    {
        code[index] = T[j];
        index++;
    }
}

//Finalisation de l'expression
code[index] = : AP;
index++;
code[index] = : EVAL;
index++;

retourner code;

}

#chercherTabIdent et ajouterTabIdent sont des fonctions correspondant
#à la recherche et l'ajout d'un identifiant dans une table. (idem TDS)
#chercherTabIdent retourne FAUX si l'IDENT n'est pas present, son indice
# sinon.
```

2.3.2 Les traces d'exécution

Rappel : $\Xi = \backslash X_1 X_2 X_3 \backslash ((X_1 + X_2) * X_3) / (X_1 + X_3)$

 Lecture de l'EFS, vérification de la syntaxe, génération de code.

 Lecture des variables et de M

Lexeme lu: '\'
 Commentaire: Syntaxe correcte
 Action: Continuer la lecture

Lexeme lu: 'X1'
 Commentaire: type: IDENT
 Action: Vérifier présence dans tabIdent.
 Lexeme non present.
 Ajouter Lexeme a tabIdent
 Continuer la lecture.

Lexeme lu: 'X2'
 Commentaire: type: IDENT
 Action: Vérifier présence dans tabIdent.
 Lexeme non present.
 Ajouter Lexeme a tabIdent
 Continuer la lecture.

Lexeme lu: 'X3'
 Commentaire: type: IDENT
 Action: Vérifier présence dans tabIdent.
 Lexeme non present.
 Ajouter Lexeme a tabIdent
 Continuer la lecture.

Lexeme lu: '\'
 Commentaire: Syntaxe correcte
 Action: Ajouter LD au code
 Continuer la lecture
 Etat du code généré: LD

Lexeme lu: '('
Action: Ajouter 1 au code
Continuer la lecture
Etat du code généré: LD 1

Lexeme lu: '('
Action: Ajouter 1 au code
Continuer la lecture
Etat du code généré: LD 1 1

Lexeme lu: 'X1'
Action: Vérifier présence dans tabIdent
Lexeme présent.
Ajouter 7 au code
Continuer la lecture
Etat du code généré: LD 1 1 7

Lexeme lu: '+'
Action: Ajouter 3 au code
Continuer la lecture
Etat du code généré: LD 1 1 7 3

Lexeme lu: 'X2'
Action: Vérifier présence dans tabIdent
Lexeme présent.
Ajouter 8 au code
Continuer la lecture
Etat du code généré: LD 1 1 7 3 8

Lexeme lu: ')'
Action: Ajouter 2 au code
Continuer la lecture
Etat du code généré: LD 1 1 7 3 8 2

Lexeme lu: '*'
Action: Ajouter 5 au code

Continuer la lecture
Etat du code généré: LD 1 1 7 3 8 2 5

Lexeme lu: 'X3'
Action: Vérifier présence dans tabIdent
Lexeme présent.
Ajouter 9 au code
Continuer la lecture
Etat du code généré: LD 1 1 7 3 8 2 5 9

Lexeme lu: ')'
Action: Ajouter 2 au code
Continuer la lecture
Etat du code généré: LD 1 1 7 3 8 2 5 9 2

Lexeme lu: '/'
Action: Ajouter 6 au code
Continuer la lecture
Etat du code généré: LD 1 1 7 3 8 2 5 9 2 6

Lexeme lu: '('
Action: Ajouter 1 au code
Continuer la lecture
Etat du code généré: LD 1 1 7 3 8 2 5 9 2 6 1

Lexeme lu: 'X1'
Action: Vérifier présence dans tabIdent
Lexeme présent.
Ajouter 7 au code
Continuer la lecture
Etat du code généré: LD 1 1 7 3 8 2 5 9 2 6 1 7

Lexeme lu: '+'
Action: Ajouter 3 au code
Continuer la lecture
Etat du code généré: LD 1 1 7 3 8 2 5 9 2 6 1 7 3

Lexeme lu: 'X3'
 Action: Vérifier présence dans tabIdent
 Lexeme présent.
 Ajouter 9 au code
 Continuer la lecture
 Etat du code généré: LD 1 1 7 3 8 2 5 9 2 6 1 7 3 9

Lexeme lu: ')'
 Action: Ajouter 2 au code
 Continuer la lecture
 Etat du code généré: LD 1 1 7 3 8 2 5 9 2 6 1 7 3 9 2

 Fin de la lecture de M
 Lecture des arguments numériques

Action: Ajouter :LDARG au code
 Continuer la lecture
 Etat du code généré: LD 1 1 7 3 8 2 5 9 2 6 1 7 3 9 2 : LDARG

Lexeme lu: '1'
 Action: Ajouter 1 au code
 Continuer la lecture
 Etat du code généré: LD 1 1 7 3 8 2 5 9 2 6 1 7 3 9 2 : LDARG 1

Lexeme lu: '2'
 Action: Ajouter 2 au code
 Continuer la lecture
 Etat du code généré: LD 1 1 7 3 8 2 5 9 2 6 1 7 3 9 2 : LDARG 1 2

Lexeme lu: '3'
 Action: Ajouter 3 au code
 Continuer la lecture
 Etat du code généré: LD 1 1 7 3 8 2 5 9 2 6 1 7 3 9 2 : LDARG 1 2 3

 Fin de la lecture de T
 Finalisation de l'expression

Action: Ajouter :AP au code

Etat du code généré: LD 1 1 7 3 8 2 5 9 2 6 1 7 3 9 2 : LDARG 1 2 3 : AP

Action: Ajouter :EVAL au code

Etat du code généré: LD 1 1 7 3 8 2 5 9 2 6 1 7 3 9 2 : LDARG 1 2 3 : AP : EVAL

 Fin de la génération de code

2.4 Comportement de la machine sur l'EFS Ξ de la section 1.4

P_1	P_2	C
.../	(.../
)	
	+	
	-	
	*	
	/	

P_1	P_2	C
.../	(LD 1 1 7 3 8 2 5 9 2 6 1 7 3 9 2 : LDARG 1 2 3 : AP : EVAL
)	
	+	
	-	
	*	
	/	

P_1	P_2	C
1	(LDARG 1 2 3 : AP : EVAL
1)	
7	+	
3	-	
8	*	
2	/	
5		
9		
2		
6		
1		
7		
3		
9		
2		

P_1	P_2	C
1	(AP : EVAL
1)	
7	+	
3	-	
8	*	
2	/	
5	1	
9	2	
2	3	
6		
1		
7		
3		
9		
2		

P_1	P_2	C
)	.../	 EVAL
3		
+		
1		
(
/		
)		
3		
)		
2		
+		
1		
(
(
P_1	P_2	C
2	.../	.../

Le résultat à obtenir est $\frac{9}{4}$ mais on suppose que notre machine formelle ne traite que les entiers lorsqu'elle évalue.