Cours de Compilation-Exercices

Master d'Informatique M1 2008–2009

29 septembre 2008

1 Analyse syntaxique

1.1 Analyse descendante et ascendante

(Partiel novembre 2000)

On se donne la grammaire suivante avec comme ensemble de symboles terminaux $\{(, \texttt{a}, \texttt{;},)\}$, comme ensemble de symboles non-terminaux $\{S, L, A\}$ et S le symbole de départ:

S	::=	(L
S	::=	a
L	::=	SA
A	::=	;SA
A	::=	)

1. Donner les étapes de l'analyse descendante du mot `(a;(a;a))`.
2. Donner les étapes de l'analyse ascendante du mot `(a;(a;a))`.
3. En déduire les dérivations gauche et droite de ce mot.

2 Analyse ascendante

(D'après examen septembre 2000)

On se donne un langage de types permettant de décrire le type des entiers ou celui de fonctions à valeur entière, prenant en argument des entiers ou d'autres fonctions de même nature.
Un type est donc soit la constante `int` soit de la forme $\tau_1 * \ldots * \tau_n \rightarrow$ `int` avec τ_i des types.
Pour reconnaître ce langage de types, on se donne la grammaire suivante avec comme ensemble de terminaux $\{\sharp, \texttt{->}, *, \texttt{int}\}$ et comme ensemble de non-terminaux $\{S, A, T\}$ avec S le symbole de départ:

S	::=	T $\sharp$
T	::=	int
T	::=	A -> int
A	::=	T
A	::=	T * A

1. Donner l'arbre de dérivation syntaxique pour l'expression `int -> int * int -> int` $\sharp$
2. Quels terminaux peuvent suivre les symboles non-terminaux T et A dans une dérivation.

3. Les états de l'analyse SLR(1) de cette grammaire sont les suivants:

s_1: $S \to .T\sharp$, $T \to .\texttt{int}$, $T \to .A\texttt{->int}$, $A \to .T$, $A \to .T * A$, $T \to \texttt{int}.$

s_3: $S \to T.\sharp$, $A \to T.$, $A \to T. * A$, $T \to A.\texttt{->int}$

s_5: $A \to T * .A$, $T \to .\texttt{int}$, $T \to .A\texttt{->int}$, $A \to .T$, $A \to .T * A$, $T \to A\texttt{->}.\texttt{int}$

s_7: $A \to T.$, $A \to T. * A$

s_8: $A \to T * A.$, $T \to A.\texttt{->int}$

s_9: $T \to A\texttt{->int}.$

   - Construire la table de transition.
   - Indiquer les états présentant des conflits et dans ces états, les différentes actions possibles.
4. Expliquer la nature du conflit obtenu en donnant un exemple d'entrée où ce conflit se produit. La grammaire donnée est-elle ambiguë ?
5. Que suggérez-vous pour remédier à ce problème ?
6. En caml, la grammaire des types inclut les règles

```
S  := T ♯
T  := int
T  := T -> T
T  := T * T
```

   La précédence de `*` est plus importante que celle de `->`, le symbole `->` associe à droite. Donner l'arbre de dérivation pour l'expression:
   - `int -> int * int -> int ♯`

2.1 Conflits

On se donne un langage pour reconnaître des expressions de type de tableaux soient effectives comme `int[4]` soient virtuelles comme `int[]` ou `void[]`.

Les terminaux de la grammaire sont {`int`,`char`,`void`,`[`,`]`,`number`}

Les règles de la grammaire sont :

```
S    := TE | TV
TE   := TET TEA
TET  := int | char
TEA  := [ number ]
TV   := TVT TVA
TET  := int | char | void
TEA  := [ ]
```

Si cette grammaire est entrée dans yacc, on obtient le message suivant:

```
2 rules never reduced
2 reduce/reduce conflicts.
```

   - Expliquer pourquoi
   - Proposer une grammaire équivalente dans lequel ce problème est résolu.