TD d'analyse syntaxique : feuille d'exercices

Exercice 1. On considère les terminaux `NAT`, `MINUS` qui correspondent aux langages de lexèmes du TD1, plus les terminaux `SHARP`, `OPAR` et `CPAR` qui correspondent respectivement aux singletons {`#`}, {`(`} et {`)`}. On considère aussi la BNF attribuée suivante avec les non-terminaux $\mathbf{S}\uparrow\mathbb{Z}$ et $\mathbf{exp}\downarrow\mathbb{N}\uparrow\mathbb{Z}$:

(1)		$\mathbf{S}\uparrow n$	::=	$\mathbf{exp}\downarrow 1\uparrow n$
(2)	$\mathbf{exp}\downarrow p\uparrow n$		::=	`NAT`$\uparrow n$
(3)			|	`OPAR` $\mathbf{exp}\downarrow p\uparrow n$ `CPAR`
(4)			|	$\mathbf{exp}\downarrow(p+1)\uparrow n_0$ `SHARP` $\quad n := n_0 \times 2^p$
(5)			|	$\mathbf{exp}\downarrow p\uparrow n_1$ `MINUS` $\mathbf{exp}\downarrow p\uparrow n_2 \quad n := n_1 - n_2$

▷ **Question 1.** Cette BNF étant ambiguë, donner deux arbres d'analyse du mot "`3 # - 3 #`" pour lesquels le calcul d'attributs donne deux résultats différents. Décorer ces 2 arbres d'analyse avec leur calcul d'attributs.

▷ **Question 2.** Calculer les directeurs LL(1) de chacune des règles, y compris la règle (1). La BNF est-elle LL(1) ?

Exercice 2. Calculer les directeurs LL(1) des BNFs suivantes. Sont-elles LL(1) ? ambiguës ?

1. $\mathbf{S}$ ::= `a`$\mathbf{X}$ $\quad$ $\mathbf{X}$ ::= $\mathbf{S}$`b` | `b`$\mathbf{S}$ | ε
2. $\mathbf{S}$ ::= $\mathbf{XX}$ | ε $\quad$ $\mathbf{X}$ ::= `b`$\mathbf{S}$
3. $\mathbf{S}$ ::= $\mathbf{Y}$`a`$\mathbf{X}\mathbf{Y}$`c`$\mathbf{Y}$ $\quad$ $\mathbf{X}$ ::= `a` | ε $\quad$ $\mathbf{Y}$ ::= `b`$\mathbf{S}$ | ε

Exercice 3. On considère la BNF attribuée suivante, sur le vocabulaire terminal $\{a, b, c\}$, avec des non-terminaux de profil $\mathbf{S}\downarrow\mathbb{N}\uparrow\mathbb{N}$, $\mathbf{X}\downarrow\mathbb{N}\uparrow\mathbb{N}$ et $\mathbf{Y}\downarrow\mathbb{N}\uparrow\mathbb{N}$:

(1)	$\mathbf{S}\downarrow h\uparrow\max(r_1, r_2)$		::=	`a` $\mathbf{X}\downarrow h\uparrow r_1$ $\mathbf{Y}\downarrow h\uparrow r_2$
(2)	$\mathbf{X}\downarrow h\uparrow r$		::=	$\mathbf{S}\downarrow h+1\uparrow r$ `b`
(3)			|	$\varepsilon \quad r := 3 \times h$
(4)	$\mathbf{Y}\downarrow h\uparrow r$		::=	`c` $\mathbf{Y}\downarrow h+1\uparrow r$ `a`
(5)			|	$\varepsilon \quad r := 2^h$

▷ **Question 1.** La BNF est-elle LL(1) ? Est-elle ambiguë ? Justifier.

▷ **Question 2.** Le mot "`aaacabbca`" est-il accepté par la BNF ? Si oui, en dessiner un arbre d'analyse avec la propagation d'attributs quand l'attribut hérité h à la racine vaut initialement 0.

▷ **Question 3.** On suppose définie ci-dessous la machine à états des analyseurs syntaxiques LL(1) vue en cours et en TD, qui modifie une variable globale `current` contenant le token de pré-vision.

```
TOKENS = tuple(range(4))
a, b, c, END = TOKENS   # END = token spécial de fin

def init_parser():   # initialise 'current' sur le premier token
def parse_token(t):  # vérifie 'current==t' et fait avancer 'current' sur le prochain token
```

Écrire le code Python d'une fonction "`parse()`" qui implémente l'analyseur spécifié par la BNF attribuée ci-dessus : elle retourne un entier r correspondant à celui calculé par le système d'attributs lorsque h est initialisé à 0. On fera bien attention à rejeter un mot comme "`acaa`" qui n'est pas reconnu par la BNF. On pourra introduire des fonctions auxiliaires.

Exercice 4. (Optionnel) Pour chacun des 2 langages suivants, donner une BNF LL(1)

$$L_1 = \{a^n b^m \mid 0 \le n \le m\} \quad \text{et} \quad L_2 = \{a^n b^m \mid 0 \le m \le n \le m+2\}$$

Exercice 5. Dans les questions ci-dessous, on étudie des BNFs reconnaissant des fragments du langage C. Pour chacune des BNFs, on aimerait trouver une (E)BNF LL(1) engendrant le même langage que la BNF initiale. On justifiera le caractère LL(1) des (E)BNFs proposées.

▷ **Question 1.** En C, les instructions peuvent commencer par des labels de "`goto`". Exemple :

```
etat1: if (cc=='a') goto etat1;
```

Voici la BNF à transformer en (E)BNF LL(1) :

inst ::= `idf` : **inst** | **exp** ;
exp ::= `idf` = **exp** | `num`

▷ **Question 2.** La syntaxe du langage C définit la notion de *lvalue*, pour "valeur à gauche" d'une affectation. C'est une catégorie d'expressions dont la valeur a une adresse mémoire. Typiquement, un littéral entier "`1`" n'est pas une lvalue. Mais une variable "`x`" en est une. Voici la BNF à transformer :

list ::= **inst** | **inst list**
inst ::= **exp** ;
exp ::= `num` | **lvalue** | **lvalue** = **exp** | **lvalue** ++
lvalue ::= **lvalue** . `idf` | **lvalue** [**exp**] | `idf`

▷ **Question 3.** En C, une branche d'un "`if/else`" peut ne pas être délimitée par des accolades lorsqu'elle ne contient qu'une seule instruction (comme dans l'exemple de la question 1). Mais cela peut introduire des contre-sens sur la signification du programme (notamment si l'indentation est incorrecte). Voici ci-dessous une BNF G_1 qui reconnaît un fragment du langage C avec "`if/else`". Le symbole **exp** correspond à celui de la question 2. Les symboles **L** et **I** remplacent respectivement les **list** et **inst** précédents.

(1) **L** ::= **I L** (3) **I** ::= **exp** ; (5) **O** ::= `else` **B** (7) **B** ::= { **L** }
(2) | ε (4) | `if` (**exp**) **B O** (6) | ε (8) | **I**

1. Montrer que la BNF G_1 est ambiguë.
2. Comme G_1 est ambiguë, elle n'est pas LL(1). Indiquer une paire de règles dont les directeurs sont en conflit.
3. On considère la BNF G_2 obtenu en supprimant la règle (8) de G_1. Calculer les directeurs de G_2. Est-elle LL(1) ? Est-elle ambiguë ?
4. Dans la sémantique du langage C, les ambiguïtés des `if/else` sont éliminées en rattachant le "`else`" au "`if`" le plus proche (parmi les "`if`" ambiguës). Par exemple, la sémantique de "`if`(e_1)`if`$(e_2)e_3$`;else` e_4`;`" est équivalente à "`if`$(e_1)\{$`if`$(e_2)e_3$`;else` e_4`;}`;"
   Expliquer comment écrire un analyseur récursif inspiré d'une analyse LL(1) qui reconnaît le langage de G_1 et dont l'arbre des appels récursifs applique la règle de désambiguation ci-dessus : on donnera en particulier du pseudo-code pour `parse_O` et `parse_B`.

▷ **Question 4** (avancée)**.** Il n'existe en fait pas de BNF LL(1) équivalente à G_1. Par contre, il existe une BNF non ambiguë équivalente (et même LALR).

L ::= **L I** | ε **I** ::= **I**$_0$ | **I**$_1$ **I**$_0$ ::= **exp** ; | `if` (**exp**) **B**$_0$ `else` **B**$_0$ **B**$_0$ ::= { **L** } | **I**$_0$

I$_1$ ::= `if` (**exp**) **I** | `if` (**exp**) { **L** } | `if` (**exp**) **B**$_0$ `else` **I**$_1$

1. Attacher sur cette BNF un système d'attributs qui associe à tout programme reconnu par G_1, un programme de même sémantique mais sans ambiguïté. Autrement dit, ce système d'attributs "ajoute" des accolades, comme dans l'exemple précédent, pour qu'on puisse comprendre le code sans avoir à appliquer la règle de désambiguation. On supposera écrit le non-terminal de profil **exp**$\uparrow w$ où w est une chaîne de caractères correspondant à l'expression reconnue. On essaiera de minimiser les accolades ajoutées.
2. Appliquer votre système d'attributs sur le mot w_1="`if`(e_1)`if`$(e_2)e_3$`;else if`$(e_4)e_5$`;`" et le mot w_1 suivi de "`else` e_6`;`".