Compilation EISE4 – TD

Exercice 1 : Grammaire Hors-Contexte et Ambiguïté

Soit la grammaire suivante :
$G_1 =< \{a, b\}, \{S\}, S, R >$ et $R : S \rightarrow aSb|aS|\varepsilon$

- Quel est le langage engendré par la grammaire ?
- Montrer que cette grammaire est ambigüe
- Donner une grammaire équivalente non-ambigüe

Mêmes questions avec la grammaire $G_2 =< \{a, b\}, \{S\}, S, R >$ et $R : S \rightarrow SaSaS|bS|\varepsilon$

Exercice 2 : Analyse syntaxique

Soit le programme yacc suivant, qui permet de reconnaitre les mots du langage $x^n a y^n$:

```
1  A : 'x' A 'y'     { printf("A -> x A y\n"); }
2    | 'a'           { printf("A -> a\n"); }
3    ;
```

- Qu'affiche ce programme appliqué à la chaine d'entrée `xxayy` ?
- Afficher les états successifs de la pile de yacc pour la chaine `xxayy`.

Soit le langage $\texttt{ab}^*\texttt{c}$ engendré par la grammaire :

```
1  A : 'a' B
2    ;
3  B : 'b' B
4    | 'c'
5    ;
```

- Dessiner les états successifs de la pile de yacc pour la chaine d'entrée `abbbc`
- Quel problème cela peut-il poser ? Donner une grammaire équivalente qui résout le problème.

Exercice 3 : MiniC

Soit le programme MiniC suivant :

```
1  int a = 1, b;
2  bool c = true;
3  void main() {
4  }
```

- Dessiner l'arbre de ce programme à la fin de l'analyse syntaxique, en numérotant les noeuds selon l'ordre de leur création
- Dessiner l'arbre de dérivation correspondant à ce programme dans la grammaire hors-contexte de MiniC (règles qui sont prise depuis l'axiome pour arriver à ce programme)
- Pour chaque noeud de l'arbre du programme, indiquer dans l'arbre de dérivation la réduction à l'origine de la création du noeud
- Donner le code assembleur mips correspondant à ce programme

Exercice 4 : MiniC

Soit le programme MiniC suivant :

```
1  void main() {
2    int a = 120, b = 80;
3    if (a > b) {
4      a = a - b;
5    }
6    print("a = ", a, " - b = ", b);
7  }
```

- Dessiner l'arbre de ce programme après la première passe
- Donner toutes les conditions, explicites ou implicites, vérifiées par la grammaire attribuée de MiniC pour ce programme
- Dessiner l'état de la pile au moment du `if`
- Donner le code assembleur mips correspondant à ce programme

Exercice 5 : Grammaire attribuée de MiniC

Donner, pour chacune des règles suivantes de la grammaire attribuée, un programme MiniC minimal ne vérifiant pas la condition de la règle :

- 1.8
- 1.12 (partie de la condition : $type = type_1$)
- 1.20
- 1.61

Exercice 6 : Génération de code assembleur Mips

Soit le programme MiniC suivant :

```
1  void main() {
2     int a = 136;
3     int b = 80;
4     while (a != b) {
5        if (a > b) {
6           a = a - b;
7        }
8        else {
9           b = b - a;
10       }
11    }
12    print(a);
13 }
```

- Que calcule ce programme ?
- Écrivez un programme assembleur mips correspondant

Exercice 7 : Génération de code assembleur Mips

Soit le programme MiniC suivant :

```
1  void main() {
2     int i;
3     int masque = 1;
4     int mot = 0x46;
5     int res = 0;
6     int temp = 0;
7
8     for (i = 0; i < 32; i = i + 1) {
9        temp = mot & masque;
10       temp = temp >>> i;
11       res = res + temp;
12       masque = masque << 1;
13    }
14    print(res);
15 }
```

- Qu'affiche ce programme ?
- Écrivez un programme assembleur mips correspondant

Exercice 8 : Langages sous-contexte (optionnel, hors programme)

Rappel : Une grammaire est dite sous-contexte si toutes ses règles sont de la forme $\alpha A \beta \to \alpha \omega \beta$, avec $\alpha, \beta \in V^*, A \in V_N, \omega \in V^+$.

Montrer que la définition au-dessus est équivalente à la définition suivante : une grammaire sous-contexte est une grammaire dans laquelle toutes les règles sont de la forme $\alpha \to \beta$, où $|\alpha| \leq |\beta|$ ($|.|$ désigne la longueur d'un mot, c'est-à-dire son nombre d'éléments terminaux et non terminaux)