Exercice A014 *Grammaires*

1. Soit $\Sigma_0 = \{\texttt{a}, \texttt{b}, ..., \texttt{z}\}$ l'ensemble des caractères alphanumériques minuscules. Quel est le langage engendré par la grammaire suivante de symbole initial S et d'alphabet terminal Σ_0 ?

$$S \to \varepsilon \,|\, AS \qquad A \to \sigma \text{ avec } \sigma \in \Sigma_0$$

On considère un langage de programmation $\hat{\texttt{C}}$ (prononcé *C chapeau*) qui peut être vu comme un sous ensemble de `C`. On rappelle qu'un *identificateur* est un nom choisi par le programmeur qui peut être utilisé pour désigner une variable ou une fonction.

En $\hat{\texttt{C}}$, un identificateur est une séquence de caractères de $\Sigma_1 = \{\texttt{a}, ...\texttt{z}, \texttt{A}, ..., \texttt{Z}, \texttt{0}, ..., \texttt{9}, \texttt{\_}\}$ ne commençant pas par un chiffre. Exemples d'identificateurs : `x` `Tea4Two` `_foo_bar`

2. Donner une grammaire de symbole initial X qui engendre exactement le langage des mots qui sont des identificateurs valides en $\hat{\texttt{C}}$.

Dans toute la suite, on suppose que E est le symbole initial d'une grammaire permettant d'engendrer les expressions en $\hat{\texttt{C}}$. On considère à présent un non-terminal I destiné à engendrer les instructions de $\hat{\texttt{C}}$. Ces dernières sont : soit une expression (éventuellement absente dans le cas d'une instruction vide) suivie du caractère '`;`', soit une instruction `while`, soit un *bloc*. On obtient ainsi les règles suivantes :

(1) $I \longrightarrow E\,\texttt{;}$
(2) $I \longrightarrow \texttt{;}$
(3) $I \longrightarrow \texttt{while (}\,E\,\texttt{)}\ I$
(4) $I \longrightarrow B$

Le non-terminal B est destiné à engendrer les blocs de $\hat{\texttt{C}}$, sachant qu'un bloc est une liste (éventuellement vide) d'instructions délimité par une accolade ouvrante et une accolade fermante. Exemple :
`{ while (a!=1){ a= 3*a+1; {} ;; while (a%2 ==0) a= a/2; } }`

3. Montrer que tout mot engendré par I se termine par un point-virgule ou une accolade fermante.
4. Indiquer quelles règles ajouter aux règles (1) à (4) pour que le langage engendré par B soit celui des blocs en $\hat{\texttt{C}}$.
5. Le langage engendré par I est-il rationnel ?

Similairement à `C`, $\hat{\texttt{C}}$ a une instruction conditionnelle. Elle est définie syntaxiquement par l'ajout de la règle (5) à I : le non-terminal O désigne une branche sinon (`else`) :

(5) $I \longrightarrow \texttt{if (}\,E\,\texttt{)}\ I\ O$
(6) $O \longrightarrow \texttt{else}\ I$
(7) $O \longrightarrow \varepsilon$

6. Après avoir ajouté ces règles, la grammaire de symbole initial I est-elle ambiguë ? Justifier votre réponse en considérant l'instruction suivante : `if (x) if (y) a = a+1; else a = a+2;`

Similairement à `C`, la sémantique de $\hat{\texttt{C}}$ élimine les ambiguités des `if`/`else` en rattachant le `else` au `if` le plus proche parmi les `if` ambigus. Par exemple, `if (e1) if (e2) e3; else e4;` équivaut sémantiquement à `if (e1) {if (e2) e3; else e4;}`

7. Donner une grammaire non-contextuelle non ambiguë rattachant le `else` au `if` le plus proche.