Examen du cours "Introduction à la compilation"
Durée: 3 heures

Tout document autorisé.

Le soin apporté à la rédaction et à la présentation ainsi que la rigueur des réponses seront pris en compte dans la notation. La première partie est une application directe du cours d'analyse syntaxique. Les sections suivantes portent sur une extension du langage impératif étudié en travaux dirigés.

1 Analyse syntaxique

Soit la grammaire suivante :

$$
\begin{array}{lcl}
P & \rightarrow & T\,\# \\
T & \rightarrow & a \mid T.T \mid (TL) \\
L & \rightarrow & \epsilon \mid T\;L
\end{array}
$$

où :

- P, T et L sont des symboles non-terminaux ;
- $\#$, $(,)$ et a sont des symboles terminaux.

Exercise 1

1. *Pourquoi cette grammaire n'est-elle pas adaptée à l'analyse syntaxique LL ?*
2. *Appliquez les transformations vues en cours pour essayer de transformer cette grammaire en une grammaire équivalente adaptée à l'analyse LL.*
3. *Calculez la fonction FIRST des non-terminaux de cette grammaire.*
4. *Calculez la fonction FOLLOW des non-terminaux de cette grammaire.*
5. *Calculez la table LL(1) de cette grammaire. Y-a-t-il des conflits dans cette table ?*
6. *Quelle associativité avez vous donné à l'opérateur "." ?*

Exercise 2 (Syntaxe)

1. *Écrivez un programme dans ce langage qui calcule $\sum_{i=0}^{k} i^2$ à l'aide d'une boucle.*
2. *On rappelle qu'un « sucre syntaxique » est une extension de la syntaxe concrète qui ne nécessite pas de modification de la syntaxe abstraite pour être traitée. On reconnaît un sucre syntaxique dans un compilateur au fait qu'il est implémenté dans les actions sémantiques de l'analyse syntaxique.*

   Pour chacune des constructions "classiques" suivantes, indiquez si elle peut ou non être vue comme un sucre syntaxique pour la syntaxe abstraite définie plus haut. Dans le cas d'une réponse positive, vous donnerez la traduction de la construction en termes des constructions de la syntaxe abstraite. Dans le cas contraire, vous justifierez votre réponse.

   1. *"**do** { c } **while** e"* *(comme en C)*
   2. *"**fun** x → e"* *(comme en O'Caml)*
   3. *"**for** (c; e; c) { c }"* *(comme en C)*

□

Ce langage est impératif : la sémantique d'un programme est la transformation d'un état initial, formé d'une mémoire, en un nouvel état, dans lequel les valeurs associées aux variables peuvent avoir été modifiées et de nouvelles variables ont pu être allouées. On modélise une mémoire de la même façon qu'un environnement à l'aide d'un dictionnaire M dont la syntaxe est :

M	$::=$		**Mémoire**
	$\mid$	$\bullet$	Mémoire vide
	$\mid$	$M + \{x \mapsto n\}$	Mémoire où la variable x vaut n

Exercise 3 (Mémoire)

1. *En vous inspirant de définitions du cours, donnez les règles du jugement « $x \in M$ » qui se lit « La variable x est bien définie dans la mémoire M. ».*
2. *D'après vous, comment peut-on représenter efficacement une mémoire M dans un interprète écrit en OCaml ?*

□

La sémantique opérationnelle à petits pas du langage est spécifiée par la sémantique à petits pas des commandes qui repose sur la sémantique à grands pas des expressions.

Exercise 4 (Sémantique des expressions)

1. *Donnez la sémantique à grands pas des expressions en donnant les règles d'un jugement « $M \vdash e \Downarrow n$ » qui se lit*

« Dans la mémoire M, l'expression e s'évalue en l'entier n. ».

Exercise 5 (Sémantique à petits pas des commandes)

1. *Parmi ces règles, quelles sont les règles de réduction ? Quelles sont les règles de passage au contexte ?*
2. *En utilisant ces règles de sémantique, donnez la chaîne de réduction du programme de la question 2.1, en supposant*

que la valeur initiale de k est 1 dans la mémoire.

3. *Quels sont les programmes bloqués pour cette sémantique ?*
4. *Décrivez, à l'aide de règles d'inférence ou d'un programme O'Caml, une analyse statique non triviale[1] permettant de n'accepter que des programmes qui ne vont pas bloquer.*

□

3 Branchement multiple

On étend la syntaxe du langage en introduisant une commande effectuant un branchement multiple :

c	$::=$	$\ldots$	**Commande**
	$\mid$	**switch** e **on** $\vec{b}$	Branchement multiple
b	$::=$		**Branche**
	$\mid$	$n \Rightarrow c$	Cas d'une constante entière
	$\mid$	$\_ \Rightarrow c$	Cas par défaut

On écrit « $n \sim b$ » le jugement qui se lit « la branche b correspond à l'entier n » par les deux axiomes suivants :

$$\frac{n = m}{n \sim m \Rightarrow c} \qquad \frac{}{n \sim \_ \Rightarrow c}$$

Un branchement multiple de la forme "**switch** e **on** $\vec{b}$" s'évalue en réduisant l'expression e en un entier n. On choisit ensuite la première branche de la liste de branches $\vec{b}$ qui correspond à l'entier n. On évalue enfin la commande de cette branche.

Exercise 6 (Sémantique des branchements multiples)

1. *Donnez un exemple de programme utilisant un branchement multiple.*
2. *Donnez des règles de sémantique à petits pas pour le branchement multiple. Vous pouvez étendre la syntaxe des*