

Ce sujet comporte 2 pages. Tous les documents de cours, TD et TP sont autorisés. Le barème est indicatif.

Justifier toutes les réponses de façon concise mais précise. Utiliser de façon cohérente les notations du cours. Ne pas répondre au hasard.

Partie I – La notion de réduction (3 points)

Soit le problème P_0 suivant :

- Entrée : une liste l triée par ordre croissant et un entier n .
- Sortie : vrai si le plus petit élément de l est plus grand que n , faux sinon.

Soit le problème P_1 suivant :

- Entrée : une liste l non-triée et un entier n .
- Sortie : vrai si le plus petit élément de l est plus grand que n , faux sinon

Question 1

Qu'est-ce qui fait que les fonctions P_0 et P_1 sont des « problèmes » ?

Question 2

Donner des majorations de la complexité de P_0 en utilisant la notation « grand O ». Distinguer la plus précise trouvée.

Question 3

Trier la liste l entrée de P_1 permet-il de réduire P_1 vers P_0 ou de réduire P_0 vers P_1 ?

Question 4

Peut-on en déduire que la complexité de P_0 est supérieure à celle de P_1 ?

Partie II – Problème de l'arrêt (6 points)

Contexte : la division par 0 n'étant pas définie, tout programme qui contient une opération de division présente le risque de causer une erreur à l'exécution. Savoir prouver sans l'exécuter qu'un programme qui contient une opération de division ne peut pas causer de division par 0 est évidemment désiré.

Soit le problème P_2 suivant :

- Entrée : un programme p qui contient une opération de division et une entrée v de ce programme.
- Sortie : vrai si p n'exécute pas l'opération de division lorsqu'il est exécuté avec l'entrée v , faux sinon.

Question 5

On veut montrer que P_2 est indécidable. Quelle technique de preuve employer ?

Question 6

Développer la preuve de l'indécidabilité de P_2 selon la technique choisie.

Partie III – Sémantique du langage WHILE (2 points)

Contexte : on ajoute au langage WHILE l'opération ++ qui calcule le successeur d'un entier dans sa représentation par un arbre binaire.

Question 7

Indiquer les modifications à apporter à la sémantique formelle du langage WHILE pour prendre en compte cette nouvelle opération.

Partie IV – L'interpréteur du langage WHILE en Scheme (9 points)

Contexte : certaines expressions du langage WHILE sont d'un usage si fréquent qu'on peut souhaiter leur donner un nom afin d'éviter les répétitions fastidieuses et augmenter la lisibilité des programmes. Par exemple, on souhaite pouvoir écrire vrai plutôt que (cons nil nil). On utilise pour cela des définitions de la forme

```
define vrai (cons nil nil)
```

L'analyseur syntaxique qui lit un programme WHILE et ses définitions rassemble toutes les définitions dans une liste de la forme

```
(list ... (cons "vrai" (CONS NIL NIL)) ...)
```

et fabrique une représentation de l'arbre de syntaxe abstraite du programme où les noms définis sont représentés comme

```
(DEF "vrai")
```

On suppose aussi que sont fournis un prédicat DEF? pour déterminer si un arbre de syntaxe abstraite est celui d'un nom défini et une fonction DEF->name pour obtenir le nom défini.

Question 8

Indiquer les grandes lignes des modifications à apporter à l'interpréteur réalisé en travaux pratiques, et détailler celles qui concernent la fonction EVAL.

Contexte : on appelle fonction WHILE un programme WHILE qui ne retourne qu'un résultat. On souhaite pouvoir donner un nom à une fonction WHILE et l'appeler dans un autre programme WHILE. On convient que l'analyseur syntaxique lit les définitions des fonctions et d'un programme qui les utilise, et fabrique une liste de définitions de fonction de la forme

```
(list ... (cons "f" arbre-de-syntaxe-abstraite-de-f) ...)
```

et une représentation de l'arbre de syntaxe abstraite du programme où les appels de fonctions sont représentés comme

```
(CALL "f" args)
```

On suppose aussi que sont fournis un prédicat CALL? pour déterminer si un arbre de syntaxe abstraite est celui d'un appel de fonction WHILE et deux fonctions CALL->fun et CALL->args pour obtenir le nom de la fonction et la liste des paramètres de l'appel.

Évaluer un appel (CALL "f" args) revient essentiellement à exécuter le programme f avec les paramètres args et à retourner l'unique résultat de l'exécution de f.

Question 9

Indiquer les grandes lignes des modifications à apporter à l'interpréteur réalisé en travaux pratiques, et détailler celles qui concernent la fonction EVAL.