ISTIC
UFR INFORMATIQUE - ELECTRONIQUE

Licence STS 2ème année mentions Informatique & Mathématiques

Contrôle continu n°1 - jeudi 3 mars 2022 - durée : 1h30

Exercice 1 (Dénombrabilité) :

Question 1.1 : Montrez que $\mathbb{Z} \times \mathbb{Z}$ est dénombrable.

Nous avons vu en TD que $\mathbb{Z}$ est dénombrable. Il existe donc une bijection f de $\mathbb{Z}$ dans $\mathbb{N}$.

Nous avons vu en cours que $\mathbb{N} \times \mathbb{N}$ est dénombrable. Il existe donc une bijection g de $\mathbb{N} \times \mathbb{N}$ dans $\mathbb{N}$.

Par composition de f et g, on peut définir la bijection h suivante :

$$\begin{array}{rccc} h: & \mathbb{Z} \times \mathbb{Z} & \rightarrow & \mathbb{N} \\ & (x, y) & \mapsto & g(f(x), f(y)) \end{array}$$

$\mathbb{Z} \times \mathbb{Z}$ est en bijection avec $\mathbb{N}$, $\mathbb{Z} \times \mathbb{Z}$ est donc dénombrable.

Exercice 2 (Réduction calculatoire) :

Question 2.1 : Notons Multiple$_\exists$ le problème de déterminer s'il existe une valeur d'entrée pour laquelle un programme p_1 rend un résultat multiple de celui d'un programme p_2. Donnez une spécification formelle (typage et valeur du résultat en fonction des paramètres) du problème Multiple$_\exists$.

La spécification du problème Multiple$_\exists$ est la suivante :

$$\begin{array}{rccc} \text{Multiple}_\exists: & \mathbb{P} \times \mathbb{P} & \rightarrow & \mathbb{B} \\ & (\mathtt{p_1}, \mathtt{p_2}) & \mapsto & \exists \mathtt{d} \in \mathbb{D} : \mathcal{SEM}(\mathtt{p_1})(\mathtt{d}) \,\%\, \mathcal{SEM}(\mathtt{p_2})(\mathtt{d}) = 0 \end{array}$$

où % est l'opérateur *modulo*.

Question 2.2 : Montrez par réduction du problême de l'arrêt que le problème Multiple$_\exists$ est indécidable.

Supposons Multiple$_\exists$ décidable. Soit *multiple* un programme résolvant le problème Multiple$_\exists$. On peut alors construire un programme *halt* résolvant le problème de l'arrêt comme suit :

```
algo halt(p,v)
    Construction des deux programmes suivants :
        p1(x) : p(v); 4
        p2(x) : 2
    Renvoyer la valeur :    multiple(p1,p2)
fin
```

Le problème de l'arrêt étant indécidable, ce programme *halt* ne peut pas exister. Le programme *multiple* n'existe donc pas et le problème Multiple$_\exists$ est indécidable.

Exercice 3 (Programmes et fonctions) :

Question 3.1 : Rappelez ce qu'est le point de vue extensionnel d'une fonction.

Le point de vue extensionnel d'une fonction f est de voir celle-ci comme entièrement définie par son *graphe*, c'est-à-dire par l'ensemble des couples (v, w) de $\mathbb{D} \times \mathbb{D}$ où w est l'image de v par f. Ces couples représentent le lien entre une entrée v et une sortie w sans préciser comment w est calculée.

Question 3.2 : Rappelez ce qu'est une propriété de programme extensionnelle.

Tout d'abord, précisons qu'une propriété de programme est un ensemble de programmes, c'est-à-dire un sous-ensemble de $\mathbb{P}$. Une propriété A est *extensionnelle* si et seulement si

$$\forall \mathtt{p}, \mathtt{q} \in \mathbb{P}. \left[\begin{array}{c} \mathcal{SEM}(\mathtt{p}) = \mathcal{SEM}(\mathtt{q}) \\ \Longrightarrow \\ \mathtt{p} \in A \Leftrightarrow \mathtt{q} \in A \end{array} \right].$$

A est *intensionnelle* sinon. Une propriété de programme est extensionnelle si les programmes ne sont pas distingués par leur syntaxe mais uniquement par leur sémantique.

Question 3.3 : Soit la propriété de programme $\exists x.\, \forall y : \mathcal{SEM}(p)(y) < x$. Que décrit-elle ? Est-elle extensionnelle ? Justifiez votre réponse.

Cette propriété de programme décrit l'ensemble des programmes dont les résultats sont bornés. Chaque programme p possède sa propre borne x. Elle est extensionnelle car deux programmes ayant la même sémantique renverront les mêmes résultats pour chacune des valeurs d'entrée. Ces résultats seront inférieurs à la même borne.

Question 3.4 : Soit la propriété de programme $\forall x.\, \neg\exists y : \mathcal{SEM}(p)(x) = y$. Que décrit-elle ? Est-elle extensionnelle ? Justifiez votre réponse.

Cette propriété de programme décrit l'ensemble des programmes qui ne terminent pour aucune valeur d'entrée. Leurs exécutions bouclent quelle que soit l'entrée. Elle est extensionnelle car deux programmes ayant la même sémantique boucleront toujours, quelle que soit leur entrée, ou ne boucleront pas toujours, puisque si l'un renvoie un résultat pour une entrée, l'autre renverra le même résultat pour cette même entrée.

Exercice 4 (Syntaxe et sémantique) :
Le langage WHILE contient une commande dont la syntaxe est `while e do c od` et dont la sémantique informelle est celle d'une boucle qui répète la commande c tant que l'expression e est vraie.

La sémantique formelle de cette boucle *while* est la suivante :

$$\mathcal{SEM}_c : \text{commande} \times \textit{mémoire} \to \textit{mémoire}$$
$$\mathcal{SEM}_c(\texttt{while e do c od}, m) = \begin{cases} m & \text{si } \mathcal{SEM}_e(e,m) = \otimes \\ \mathcal{SEM}_c(\texttt{while e do c od}, \mathcal{SEM}_c(c,m)) & \text{sinon} \end{cases}$$

Nous introduisons une nouvelle commande dont la syntaxe est `iterate c until e od` et dont la sémantique informelle est celle d'une boucle qui répète la commande c jusqu'à ce que l'expression e soit vraie. Le test de la valeur de l'expression e se fait après l'exécution de la commande c.

Question 4.1 : Proposez une modification de la grammaire du langage WHILE.

On ajoute dans la partie commande de la grammaire la règle suivante :

commande → `iterate` commande `until` expression `od`

Question 4.2 : Nous allons définir la sémantique formelle de cette nouvelle boucle.

a) Lors du calcul de $\mathcal{SEM}_c(\texttt{iterate c until e od}, m)$, sur quelle mémoire s'effectuera l'évaluation de l'expression e ?

La sémantique informelle précise que le test de la valeur de l'expression e se fait après l'exécution de la commande c. La mémoire sur laquelle est évaluée l'expression e est donc $\mathcal{SEM}_c(c, m)$.

b) Si l'évaluation de l'expression e provoque la sortie de la boucle, quelle sera, en fonction de m, la valeur de la mémoire à l'issue de la boucle ?

L'évaluation de l'expression se fait en dernier dans la boucle, il n'y a pas d'autre commande exécutée après le test s'il provoque la sortie de boucle. La mémoire à l'issue de la boucle est donc $\mathcal{SEM}_c(c, m)$ également.

c) Donnez la sémantique formelle de la boucle `iterate c until e od`.

La sémantique formelle de cette boucle *iterate* peut être la suivante :

$$\mathcal{SEM}_c : \text{commande} \times \textit{mémoire} \to \textit{mémoire}$$
$$\mathcal{SEM}_c(\texttt{iterate c until e od}, m) = \begin{cases} \mathcal{SEM}_c(c, m) & \text{si } \mathcal{SEM}_e(e, \mathcal{SEM}_c(c, m)) \neq \otimes \\ \mathcal{SEM}_c(\texttt{iterate c until e od}, \mathcal{SEM}_c(c, m)) & \text{sinon} \end{cases}$$

On aurait pu aussi réutiliser la sémantique de la boucle *while* pour définir celle de la boucle *iterate* :

$$\mathcal{SEM}_c(\texttt{iterate c until e od}, m) = \mathcal{SEM}_c(c \texttt{ ; while e =? nil do c od}, m)$$