Calculabilité
R. Rioboo

Travaux Pratiques URM

Nous allons iimplanter un simulateur pour les URMs. Une machine à registres exécute des instructions, assigne des registres et lit un programme. Nous représenterons les instructions par le type

```
type instruction =
  | Reset of int
  | Incr of int
  | Set of int*int
  | Jump of int*int*int
```

Nous stockerons les registres dans des tableaux que nous initialiserons à une taille fixe `max_registers` pour chaque programme, comme dans le cours nous utiliserons le registre 0 pour désigner l'instruction en cours. Un programme sera alors un tableau d'instructions.

```
type programme = instruction array
```

Exercice 1
Écrire la fonction `run_instruction` qui prend en paramètres un ensemble de registres, une instruction ainsi que son indice dans un programme et retourne un entier correspondant au prochain numéro d'instruction à exécuter dans le programme. Vous pourrez retourner l'exception `Memory_exhausted` si l'instruction affecte un registre plus grand que `max_registers` ainsi que l'exception `Segmentation_fault` si on consulte le contenu d'un registre plus grand que `max_registers`.

Afin de s'assurer que tous les programmes terminent nous limiterons leur temps d'exécution à l'aide d'un compteur entier que nous décrémenterons à chaque fois qu'une isntruction est exécutée par la machine. Pour chaque programme nous initialiserons ce compteur à un entier `max_steps`.

Exercice 2
Écrire la fonction `step_program` qui prend en paramètres un ensemble de registres, un programme et le nombre maximum d'étapes à exécuter. On retournera l'ensemble des registres à la fin de l'exécution. Vous pourrez déclencher l'exception `Resources_exhausted` si le programme ne termine pas.

Nous nous limiterons à implémenter les fonctions qui prennent des entiers en paramètres et retournent des entiers. Une fonction sera codée par un programme, l'argument devra être le contenu du premier registre, les autres registres seront devront tous être initialisés à 0. Le résultat de la fonction sera le contenu du premier registre à la fin de l'exécution.

Exercice 3
Écrire la fonction `run_function` qui prend en paramètres un programme (une fonction), un entier (l'argument) et retourne la valeur du résultat.

Exercice 4
Écrire la fonction qui calcule l'indice du plus grand registre modifié par un programme. Écrire la fonction `normalize` qui prend un programme P ayant N instructions et retourne un programme P' où toutes les instructions d'arret `Jump`(n, m, p) avec $p > N$ sont remplacées par `Jump`$(n, m, N + 1)$.

Exercice 5
Lorsque l'on veut combiner deux programmes on doit exécuter le premier, remettre à 0 les registres qui ont été modifiés lors de l'exécution puis exécuter le deuxième programme. Écrire une fonction `clean_program` qui prend en argument un programme P et retourne un programme P' qui s'exécute comme P mais remet à 0 les registres $R_2, \ldots R_{\max}$ où $R_{\max}$ est le plus grand registre utilisé par P.