

Travaux Machines Encadrés URM

Nous allons implanter quelques outils pour manipuler des URM. On reprend les représentations du TME précédent. Nous aurons besoin des utilitaires écrits pendant la séance précédente, notamment :

- la fonction `rho` qui calcule le plus petit indice « libre ». C'est à dire non accédé ni modifié par un programme.
- la fonction `normalize` qui remplace tous les sauts provoquant l'arrêt d'un programme par un saut à l'instruction qui suit la dernière.

Exercice 1

Pour tester nos fonctions nous utiliserons quelques programmes de base. Par exemple le programme `succ` retourne le successeur du premier registre, le programme `somme` retourne la somme de ses deux arguments, le programme `constant` prend en argument un entier et retourne cet entier.

Nous pourrions aussi implémenter des fonctions à valeurs booléennes comme le programme `bigger` qui retourne 1 si le premier registre est supérieur au deuxième et 0 sinon.

Implémenter ces programmes.

Exercice 2

Pour nous assurer de la corrections des programmes que nous produisons nous devrions parfois les lire. Écrire la fonction `string_of_prog` qui produit une chaîne de caractères lisible.

Adapter la fonction `run_program` du TME précédent pour écrire une fonction `debug_program` qui affiche le contenu des registres 1 à $\rho(P)$ avant et après l'exécution d'un programme P .

Exercice 3

Nous voulons d'abord implémenter la composition de programmes. Étant donnés deux programmes P_1 et P_2 nous souhaitons exécuter P_1 puis P_2 en séquence. La fonction `compose` prendra donc deux programmes et retournera un nouveau programme exécutant les deux en séquence.

Nous devons remplacer les instructions $J(r_1, r_2, i)$ de saut de P_2 par des instructions $J(r_1, r_2, n_1 + i)$ où n_1 est la longueur de P_1

Exercice 4

Nous souhaitons maintenant implémenter le translaté $P[a_1, \dots, a_n, \rightarrow r]$ d'un programme P . La fonction `translate` prendra en paramètres un programme un vecteur d'entiers désignant les registres a_1, \dots, a_n où se trouvent les paramètres d'entrée et un entier désignant le registre r où on souhaite placer le résultat. Nous devons pour cela composer les opérations suivantes :

- transférer le contenu des registres a_1, \dots, a_n dans les registres 1 à n ,
- initialiser à 0 les registres de $n + 1$ à $\rho(P)$,
- exécuter P (on n'oubliera pas de compter que les étapes précédentes prennent $\rho(P)$ instructions),
- transférer le contenu du premier registre dans le r -ième registre.

Exercice 5

Nous allons maintenant implanter la composition généralisée. Étant donné un programme F implantant une fonction à n arguments et n programmes G_i implémentant des fonctions g_i à k arguments nous voulons implémenter le programme implémentant la fonction à k arguments x_1, \dots, x_k donnée par

$$f(g_1(x_1, \dots, x_k), \dots, g_n(x_1, \dots, x_k))$$

Notre fonction `compose` prendra en paramètres le programme F , un vecteur de programmes pour représenter les G_i ainsi que l'arité k des fonctions g_i (l'arité n de f est bien sur la taille du tableau des G_i). Elle retournera un programme.

Nous devons composer les opérations suivantes :

- calculer un registre N « libre » à partir de $n, k, \rho(F), \max(\rho(G_i))$.
- sauvegarder la donnée qui se trouve dans les registres 1 à k dans les registres $N + 1, \dots, N + k$.
- Exécuter les programmes traduits des G_i qui prennent leurs arguments dans les registres $N + 1, \dots, N + k$ et stockent le résultat dans le registre $N + k + i$.
- Exécuter le traduit de F qui prend ses arguments dans les registres $N + k + 1, \dots, N + k + n$ et place le résultat dans le premier registre.

Exercice 6

Nous prendrons un petit langage très simple d'expressions formé de constantes entières (positives) et de sommes d'expressions.

```
type expr =
  | Int of int
  | Plus of expr*expr
```

Nous voulons traduire ces expressions en des programmes qui les calculent.

Pour traduire les constantes nous utiliserons le programme `constant`.

Pour traduire une somme nous devons composer la somme avec chacun des résultats des traductions des programmes arguments. Écrire la fonction `prog_of_expr`

Exercice 7

On se propose d'étendre le type `expr` avec un champ :

```
| Param of int
```

et les valeurs de la forme `Param n` désigneront le n -ième paramètre d'une fonction. Nous souhaitons étendre la fonction `prog_of_expr` pour qu'elle puisse traiter ce nouveau cas.

1. Écrire la fonction `number_of_params` qui prend en argument une expression e et retourne le nombre v_e de paramètres d'une expression.

Nous allons produire une URM qui trouvera son i -ième paramètre dans le i -ième registre. Nous souhaitons connaître le nombre de registres utilisés par notre traducteur pour traduire la structure arithmétique d'une expression.

On rappelle que l'URM `somme` utilise 3 registres, et que si P_1 et P_2 sont deux URMs utilisant respectivement n_1 et n_2 registres la composée généralisée que nous utilisons pour traduire l'expression arithmétique d'une somme utilise $\max(3, n_1, n_2) + 2$ registres.

2. En déduire une fonction `bound_arith` qui prend en argument une expression e et retourne l'indice n_e d'un registre tel que la traduction de e n'interfère pas avec les registres suivant n_e .

Pour traduire la fonction implantée par une expression arithmétique e avec v_e paramètres et utilisant n_e registres nous allons :

- (a) transférer les registres 1 à v_e dans les registres r_e à $r_e + v_e - 1$,
- (b) coller le résultat de la traduction de l'expression e dans laquelle on traduit les occurrences de `Param i` par le transfert du registre $r_e + i - 1$ dans le registre 1 . Le reste de la fonction `prog_of_expr` n'étant pas modifié.

3. Écrire la nouvelle version de `prog_of_expr`. On n'oubliera pas que l'étape (a) prend v_e instructions.

Exercice 8

Étant donnés trois programmes P_t, P_v et P_f nous voulons écrire un programme P qui exécute P_v ou P_f suivant que l'exécution de P_t donne un résultat non nul.

Implémenter la fonction `if_then_else` qui prend trois programmes en arguments et retourne un programme qui s'exécute conditionnellement.

Comment enrichir le langage d'expressions et la fonction `prog_of_expr` pour autoriser des conditionnelles dans les expressions ?