Projet Deuxième partie

Sémantique des langages informatiques

Table des matières

Exercice 1

Comme dans la première partie du projet, nous allons séparer la partie sur papier en trois sous-partie. Ces dernières comporteront :

- la syntaxe
- le domaine sémantique
- la sémantique d'évaluation

1.1 Syntaxe

Nous allons donc définir la syntaxe du langage. Tout d'abord, intéressons-nous aux instructions :

$$I \in \{ \text{declare(\_), delcare(\_,\_), copy(\_,\_),jump(\_,\_,\_), incr(\_), beginbloc, endbloc,add(\_,\_,\_),sub(\_,\_,\_), if(\_,=,\_),then,else, endif} \}$$

Voici donc notre ensemble d'instructions, où "_" signifie "quelque que soit le paramètre de la fonction".

Tout comme le précédent projet, la programme est défini comme :

$$\text{Programme} \in I^*$$

Avec l'opérateur * qui est la fermeture de Klenne. La définition du pointeur reste :

$$\text{pointeur} \in \mathbb{N} \setminus 0$$

Car, on veut que les instructions commencent à la ligne un et on ne travaille que dans les entiers naturels. Dû à l'apparition des blocs, la définition des registres va elle changer :

$$\text{registre} = \{ (nom, valeur, degre) \} \ ,$$

où *nom* est un String, $valeur \in \mathbb{Z}$ car nous travaillons sur les entiers relatifs à cause de la soustraction, et $degre \in \mathbb{N}$ qui représente le degré de profondeur. C'est-à-dire dans combien de bloc(s) imbriqué(s) nous trouvons nous.
On reprendra la notation telle que R est l'ensemble des registres. La définition de la mémoire change quelque peut, par l'apparition de la notion de degré :

$$mem \in \{ (nom_i, valeur_i, degre_i) \mid \forall i, j, nom_i \neq nom_j, \text{ si } deg_i = deg_j \}$$

C'est donc un ensemble de registre, et il peut exister plusieurs registres portant le même nom, pour autant qu'ils n'aient pas le même degré.

Comme nouvelle formulation du contexte, nous prendrons cette représentation :

$$Ctx = (mem, prog, index, degre)$$

Où *mem* sera la mémoire, *prog* le programme, *index* le pointeur d'instruction et *degre* le degré actuel.

1.2 Domaine Sémantique

En changeant la définition du contexte et des registres, on peut réutiliser les domaines sémantiques définis dans l'ancien projet. En effet, la modification de ces deux éléments, fait que nous n'avons pas besoin de répercuter le changement sur les anciennes définitions.
Ce qui nous donne [1]: "

- incr : $registre \times Ctx \rightarrow Ctx'$, le fait d'incrémenter un registre implique que l'on modifie la mémoire et donc le contexte. Comme il est nécessaire de connaître la valeur du registre pour la changer, on a besoin du contexte pour l'obtenir.
- declare : $String \times \mathbb{N} \times Ctx \rightarrow Ctx'$, dans ce cas , pour déclarer un registre, nous avons besoin d'un nom et d'une valeur, ainsi que du contexte pour vérifier que le nom n'existe pas déjà. Cette déclaration entraîne un changement dans la mémoire et donc du contexte.
- declare : $Nom \times Ctx \rightarrow Ctx'$, ici, nous n'avons besoin que du nom, car on affectera une valeur initiale par défaut, néanmoins le contexte reste nécessaire pour les mêmes raisons que l'autre variante de la fonction *declare*.
- copy : $R \times R \times Ctx \rightarrow Ctx'$,la fonction *copy* nécessite d'avoir deux registres, ainsi que le contexte pour obtenir les valeurs de ces registres. L'évaluation de cette fonction va modifier la mémoire, et nous aurons un nouveau contexte.
- jump : $R \times R \times N \times Ctx \rightarrow Ctx'$, comme pour la fonction ci-dessus nous avons besoin de deux registres et de connaître leurs valeurs. Nous avons aussi besoin de la nouvelle valeur du pointeur si le test est juste. Le résultat de *jump* sera un nouveau contexte dont le pointeur d'instruction aura été modifié."

1 Source : rapport_projet_1_MencattiniRomain.pdf

Par conséquent, je vais présenter le domaine sémantique des nouvelles opérations :

- *beginbloc* : $Ctx \rightarrow Ctx'$. En effet, le *beginbloc* doit disposer d'un contexte pour fonctionner. Lorsqu'on utilise cette instruction, elle va modifier le contexte en changeant le degré et le pointeur d'instruction, d'où le nouveau contexte.
- *endbloc* : $Ctx \rightarrow Ctx'$. Pour les mêmes raisons que ci-dessus, cette fonction a besoin d'un contexte en entrée, et elle va retourner un nouveau contexte avec un degré, une mémoire et un index différent.
- *add* : $r_i \, x \, r_j \, x \, r_k \, x \, Ctx \rightarrow Ctx'$. L'opération *add* a besoin de trois registres (ils ne sont pas forcé d'être différents) et d'un contexte. Elle va entrainer un changement de contexte en modifiant la mémoire et l'index.
- *sub* : $r_i \, x \, r_j \, x \, r_k \, x \, Ctx \rightarrow Ctx'$. Son domaine est le même que celui de *add*.
- *if* : $r_i \, x \, r_j \, x \, Ctx \rightarrow Ctx'$. L'instruction *if* a besoin de deux registres pour effectuer la comparaison. En retour, elle va changer l'état du pointeur et donc le contexte.
- *then* : $Ctx \rightarrow Ctx'$. L'instruction *then* va juste changer le pointeur d'instruction.
- *else :* $Ctx \rightarrow Ctx'$. Même domaine que *then*.
- *endif* : $Ctx \rightarrow Ctx'$. On va change le pointeur d'instruction, ainsi que le degré et la mémoire. Ceci sera plus visible dans la sémantique d'évaluation.

1.3 Sémantique d'évaluation

Ici , nous ne pouvons nous contenter de remettre la même sémantique d'évaluation, nous allons donc redéfinir la sémantique des instructions en entier.

- $$\frac{r \in R, Ctx = (mem \cup \{r\}, prog, index, deg), r = (n_1, v_1, d_1)}{incr(r) \Rightarrow r' = (n_1, v_1 +_{\mathbb{N}} 1, d_1), Ctx' = (mem \cup \{r'\}, prog, index +_{\mathbb{N}} 1, deg)}$$,

  on a besoin d'avoir les informations du registres (soit le nom, la valeur et le degré), ainsi qu'un contexte. Ensuite, on augmente d'un la valeur du registre, et on l'ajoute à la mémoire. On devra également changer l'index. Ce qui entrainera un changement de contexte.

- $$\frac{r_1, r_2 \in R, Ctx = (mem \cup \{r_1, r_2\}, prog, index, deg), r_1 = (n_1, v_1, d_1), r_2 = (n_2, v_2, d_2)}{copy(r_1, r_2) \Rightarrow r_2' = (n_2, v_1, d_2), Ctx' = (mem \cup \{r_1, r_2'\}, prog, index +_{\mathbb{N}} 1, deg)}$$,

  dans ce cas-ci, nous avons besoin de deux registres ainsi ,que de leurs informations, et d'un contexte. On met la valeur du premier registre dans le deuxième. On modifie le contexte en ajoutant ce nouveau registre à la mémoire et en changeant l'index.

- $$\frac{n \in String, v \in \mathbb{Z}, Ctx = (mem, prog, index, deg), \nexists (n, k, deg) \in mem \; \forall \; k \in \mathbb{N}}{declare(n, v) \Rightarrow Ctx' = (mem \cup \{(n, v, deg)\}, prog, index +_{\mathbb{N}} 1, deg)},$$

  on prend un nom et une valeur. On doit vérifier qu'il n'existe aucun registre avec un nom pareil dans le même degré. Si c'est le cas, alors on ajoute le registre dans la mémoire, tout en augmentant l'index.

- $$\frac{n \in String, Ctx = (mem, prog, index, deg), \nexists (n, k, deg) \in mem \; \forall \; k \in \mathbb{N}}{declare(n) \Rightarrow Ctx' = (mem \cup \{(n, 0, deg)\}, prog, index +_{\mathbb{N}} 1, deg)},$$

  même principe que ci-dessus, sauf qu'au lieu d'une valeur *v*, on met une valeur par défaut, ici 0.

- $$\frac{r_1, r_2, r_3 \in R, r_i = (n_i, v_i d_i), \text{pour i = 1..3}, Ctx = (mem, prog, index, deg_1), v_1 = v_2}{jump(r_1, r_2, r_3) \Rightarrow Ctx' = (mem, prog, v_3, deg_2)},$$

  on prend trois registres ainsi que leurs informations et on dispose d'un contexte. Il faut vérifier que la valeur du premier registre et du deuxième soient égales. On change ensuite le contexte en mettant la valeur du registre 3 comme index, et on change le degré. Car le saut peut nous amener dans un autre bloc. Le degré peut changer.
  Nous allons également faire les autres variantes du *jump*. C'est-à-dire avec un entier, et les cas où ce n'est pas égale.

- $$\frac{r_1, r_2 \in R, r_i = (n_i, v_i d_i), \text{pour i = 1..2 }, v_3 \in \mathbb{N}, Ctx = (mem, prog, index, deg_1), v_1 = v_2}{jump(r_1, r_2, v_3) \Rightarrow Ctx' = (mem, prog, v_3, deg_2)},$$

  il s'agit du cas, où, on utilise une valeur. Cela reste la même chose, sauf qu'on ne doit pas rechercher la valeur d'un registre.

- $$\frac{r_1, r_2, r_3 \in R, r_i = (n_i, v_i d_i), \text{pour i = 1..3}, Ctx = (mem, prog, index, deg_1), v_1 \neq v_2}{jump(r_1, r_2, r_3) \Rightarrow Ctx' = (mem, prog, index +_{\mathbb{N}} 1, deg)},$$

  si l'égalité n'est pas vérifiée, alors, on doit juste passer à la ligne suivante. Donc on garde le même degré et on augmente l'index d'un.

- $$\frac{r_1, r_2 \in R, r_i = (n_i, v_i d_i), \text{pour i = 1..2 }, v_3 \in \mathbb{N}, Ctx = (mem, prog, index, deg_1), v_1 \neq v_2}{jump(r_1, r_2, v_3) \Rightarrow Ctx' = (mem, prog, index +_{\mathbb{N}} 1, deg_2)},$$

  même chose que ci-dessus, sauf qu'on a une valeur à la place d'un registre.

- $$\frac{Ctx = (mem \cup \{r_i | (n_i, v_i, deg)\}, prog, index, deg)}{endbloc \Rightarrow Ctx' = (mem, prog, index +_{\mathbb{N}} 1, deg -_{\mathbb{N}} 1)},$$

  pour le *endbloc*, on prend un contexte dont la mémoire contient tout les registres et plus particulièrement ceux du degré du contexte actuel. L'instruction *endbloc* consiste

à retirer les éléments du degré actuel, à augmenter l'index et à diminuer d'une unité le degré.

- $\frac{Ctx=(mem, prog, index, deg)}{beginbloc \Rightarrow Ctx'=(mem, prog, index +_{\mathbb{N}} 1, deg +_{\mathbb{N}} 1)}$,

on prend un contexte en entrée, et on retourne un nouveau contexte, avec un index augmenté d'un et un degré plus grand d'une unité.

- $\frac{r_1, r_2, r_3 \in R, r_i=(n_i, v_i, d_i), \text{pour i=1..3}, Ctx=(mem \cup \{r_3\}, prog, index, deg)}{add(r_1, r_2, r_3) \Rightarrow r_3'=(n_3, v_1 +_{\mathbb{N}} v_2, d_3), Ctx'=(mem \cup \{r_3'\}, prog, index +_{\mathbb{N}} 1, deg)}$,

nous avons besoin de trois registres ainsi que de leurs informations et d'un contexte. Grâce à cela, on peut additionner les valeurs des deux premiers registres et assigner ce résultat comme valeur du troisième. Tout en modifiant la mémoire et l'index.

- $\frac{r_1, r_2, r_3 \in R, r_i=(n_i, v_i, d_i), \text{pour i=1..3}, Ctx=(mem \cup \{r_3\}, prog, index, deg)}{sub(r_1, r_2, r_3) \Rightarrow r_3'=(n_3, v_1 -_{\mathbb{N}} v_2, d_3), Ctx'=(mem \cup \{r_3'\}, prog, index +_{\mathbb{N}} 1, deg)}$,

exactement le même principe que pour *add* sauf qu'on soustrait au lieu d'additionner.

- $\frac{r_i \in R, r_i=(n_i, v_i, d_i), \text{pour i =1..2}, Ctx=(mem, prog, index, degre), index(then)=t}{if(r_{1,}'=', r_2), v_1=v_2 \Rightarrow Ctx'=(mem, prog, t, degre +_{\mathbb{N}} 1)}$,
- $\frac{r_i \in R, r_i=(n_i, v_i, d_i), \text{pour i =1..2}, Ctx=(mem, prog, index, degre), index(else)=t}{if(r_{1,}'=', r_2), v_1 \neq v_2 \Rightarrow Ctx'=(mem, prog, t, degre +_{\mathbb{N}} 1)}$,

nous avons les deux règles correspondant aux deux cas du *if*, si les valeurs des registres sont égales , on saute à l'index de *then*, si non on va à l'index du *else.*

La fonction $index : I \rightarrow \mathbb{N}$, fait correspondre à une instruction, son index dans le programme.

- $\frac{Ctx=(mem, prog, index, deg)}{then \Rightarrow Ctx'=(mem, prog, index +_{\mathbb{N}} 1, deg)}$,
- $\frac{Ctx=(mem, prog, index, deg)}{else \Rightarrow Ctx'=(mem, prog, index +_{\mathbb{N}} 1, deg)}$,

les instructions *then* et *else* ne modifient que l'index et n'ont donc pas d'incidence autre que sur le contexte.

- $\frac{Ctx=(mem \cup \{r_i | r_i=(n_j, v_k, d_l), \forall i, j, k, l, d_l=deg\}, prog, index, deg)}{endif \Rightarrow Ctx'=(mem, prog, index +_{\mathbb{N}} 1, deg -_{\mathbb{N}} 1)}$,

l'instruction *endif* marque la fin du bloc du *if*. On retrouve donc le même principe que pour le *endbloc* où , nous enlevons les registres du degré du bloc, on augmente l'index

et on diminue le degré.

Voici donc, les différentes sémantiques d'évaluation des instructions. Il me reste néanmoins une chose à expliquer : la manière de choisir des registres. En effet, du au masquage des registres par des registres de plus bas niveau, il faut établir une relation de priorité.

Ce qui nous donne cette règle :

$$\frac{r_1 \in mem}{\max_i(r_{1,} v_j, i), \forall j},$$

ce qui signifie qu'on prend toujours le registre qui porte un même nom, du plus haut degré. Quelque soit sa valeur.

Il nous reste à traiter le cas de l'affectation automatique. C'est-à-dire si un registre n'existe pas et est utilisé, il est déclaré dans le bloc le plus à l'extérieur. On le définit comme :

$$\frac{r_1 \notin mem, Ctx = (mem, prog, index, degre)}{(r_1, 0, 1), Ctx \Rightarrow Ctx' = (mem \cup (r_{1,} 0, 1), index, degre)},$$

qui pour un registre qui n'existe pas, et un certain registre, avec l'insérer dans la mémoire , avec une valeur par défaut et le degré un (le plus à l'extérieur). Ce qui aura pour effet de changer le contexte.

Ces deux règles s'appliquent à l'intérieur des autres règles de sémantique d'évaluation et nous permettent de choisir le registre adéquat selon l'énoncé.

Exercice 2

J'ai entièrement commenté mon code. Dans ces commentaires se trouvent également mes choix d'implémentation ainsi que le sens des fonctions. Ces commentaires sont suffisants pour comprendre le code.