Deuxième partie, cahier des charges

29 août 2001

Dans cette partie, on vérifiera que le programme est correctement typé. De plus on calculera les informations sémantiques associées aux différents identificateurs.

1 Correction du programme

Un programme, pour être correct, doit respecter les points suivants :

- L'utilisation des variables doit respecter les règles de portée habituelles.
- Le programme doit être bien typé, que ce soit pour les appels de fonctions, les affectations, ou l'utilisation des opérateurs.
- Lorsqu'une fonction attend un argument en passage par référence, l'argument réel de la fonction doit être une valeur gauche, c'est-à-dire soit une variable, soit un accès à un tableau.
- Il ne doit pas y avoir deux fonctions de même nom dans le programme.
- Il ne doit pas y avoir deux variables globales de même nom dans le programme.
- Deux variables ne peuvent avoir le même nom que si l'une des deux est une variable globale ou que leurs portées sont disjointes.
- Une fonction n'est connue qu'après la déclaration de son prototype (ainsi on peut écrire des fonctions récursives mais pas des fonctions mutuellement récursives).
- Un certain nombre de fonctions sont prédéfinies (voir annexe). Ces fonctions ne peuvent être redéfinies.

2 Informations à calculer

On vous demande de calculer pour chaque variable, la taille de sa représentation en mémoire. Cette place pourra être calculée de la façon suivante :

- une référence a toujours la taille 1 ;
- un type atomique (entier, booléen) a la taille 1 ;
- un tableau a une taille égale à son nombre d'éléments multiplié par la taille des éléments stockés.

Pour chaque fonction vous calculerez la somme des tailles de ses paramètres ainsi que la somme des tailles de ses variables locales.

Pour chaque variable locale (resp. paramètre) d'une fonction, on appelle décalage la somme des tailles des variables locales (resp. paramètres) déclarées dans la même fonction avant cette variable (resp. donnés dans le prototype de la même fonction avant ce paramètre). Vous calculerez les décalages de tous les paramètres et de toutes les variables locales.

Enfin, on vous demande d'attribuer à chaque nom de fonction et à chaque variable globale un *label* unique. Un label est une suite de caractères respectant les même règles que les identificateurs du langage Scalpa.

3 Méthodes proposées

Deux méthodes vous sont proposées : la construction de tables de symboles et la décoration de l'arbre de syntaxe abstraite. Vous pouvez choisir d'utiliser l'une de ces deux méthodes mais vous êtes également libre de prendre toute autre méthode raisonnable pour résoudre le problème qui vous est posé.

3.1 Construction d'une table des symboles

Cette méthode consiste à transformer tout d'abord l'arbre de syntaxe abstraite pour y remplacer tous les identificateurs par des numéros de façon que deux identificateurs représentant la même variable ou fonction soient remplacés par le même numéro et que deux identificateurs représentant des variables ou fonctions différentes (éventuellement portant le même nom) soient remplacés par des numéros différents.

Dans un second temps, on passe sur l'arbre de syntaxe abstraite ainsi modifié pour construire une table indiquant pour chaque "numéro" les informations qui lui sont associées (type, décalage, variable par référence ou par valeur, ...). Durant cette seconde passe, ou bien lors d'une troisième, on pourra vérifier que l'ensemble du programme est correct.

3.2 Décoration de l'arbre de syntaxe abstraite

Cette méthode consiste à transformer l'arbre de syntaxe abstraite en un arbre contenant à chaque nœud une table associant aux variables visibles à cet endroit du programme les informations sur ces variables.

La vérification du programme peut alors se faire en utilisant, pour typer chaque nœud, la table stockée à ce nœud.

4 Options supplémentaires du compilateur

Vous ajouterez à votre compilateur l'option `-type-only` qui demandera à votre compilateur de n'effectuer que les phases d'analyses lexicale, syntaxique et sémantique. En cas d'erreur les erreurs seront rapportées de la façon suivante :

```
File "toto.bla", line 23, characters 10-32 : Type error.
This expression has type bool but is used here with type int.

File "toto.bla", line 13, characters 12-13 : Unbound variable.
Variable x is unbound.
```

Vous pourrez également implanter une option `-dump-symbols` qui dira pour chaque variable si elle est globale. Si c'est le cas, elle donnera son label ; sinon elle donnera son décalage et dira s'il s'agit d'une variable locale ou d'un paramètre d'une fonction. Cette option donnera de plus pour chaque fonction son type ainsi que la liste de ses paramètres et la liste de ses variables locales.

Annexe : liste des fonctions prédéfinies

`print(`*int v*`)` : *unit*	`random(`*int v*`)` : *int*
`move()` : *bool*	`x_pos()` : *int*
`y_pos()` : *int*	`power()` : *int*
`look()` : *int*	`where()` : *int*
`turn_left()` : *unit*	`turn_right()` : *unit*
`x_door()` : *int*	`y_door()` : *int*