ECOLE SUPERIEURE
D'INGENIEURS DE RENNES

Université de Rennes

Documentation utilisateur du compilateur et du langage

Auteurs :

Noam GEFFROY
Pol JAOUEN
Flavien LEBORGNE
Stevan METAYER
Fanny SHEHABI

2023 - 2024

Comment écrire un programme en langage While ?

Le langage While est un langage de programmation fonctionnel basé sur la manipulation d'arbres binaires. Les types standards sont encodés sous la forme d'un arbre binaire interprété en fonction des besoins.

Expressions et simulation des types

Les expressions dans le langage While sont utilisées pour décrire des arbres binaires.

- **nil** correspond à l'arbre vide
- **Symb** correspond à une feuille
- **cons** permet la construction des arbres binaires
  - **(cons) = nil** correspond à l'arbre vide
  - **(cons T) = T** représente l'arbre T
  - **(cons A B)** représente un arbre ayant A pour fils gauche et B pour fils droit
  - **(const T_1 T_2 … T_n) = (cons T_1 (cons T_2 … (cons T_{n-1} T_n) …))**
- **list** permet la construction des listes
  - **(list) = nil** correspond à la liste vide
  - **(list T) = (const T nil)** construit une liste à un élément
  - **(list T_1 T_2 … T_n) = (cons T_1 (cons T_2 … (cons T_n nil) …)**
- **(hd T)** permet de retourner le fils gauche de l'arbre T
  - Si **T = (cons A B)** alors cela retourne A
  - Si **T = Symb** alors cela retourne nil
  - Si **T = nil** alors retourne nil
- **(tl T)** permet de retourner le fils droit de l'arbre T
  - Si **T = (cons A B)** alors cela retourne B
  - Si **T = Symb** alors cela retourne nil
  - Si **T = nil** alors retourne nil
- **(f V_1 … V_n)** correspond à un appel de fonction f avec les paramètres v_1 à v_n
  - **f** est une fonction définie au préalable
  - l'appel de **f** doit comporter le bon nombre de paramètres

Un arbre binaire est le seul type de données manipulable en langage While. Les types de base comme les entiers, les booléens ou les chaînes de caractères sont simulées grâce aux arbres binaires.

Un entier est alors représenté par la **hauteur du sous-arbre droit + 1**. 0 correspond à l'arbre vide ou bien à une feuille. Par exemple, `(cons nil (cons nil nil))` correspond à 2.

Le booléen `False` est représenté par un arbre vide ou bien une feuille. Le booléen `True` correspond alors aux autres cas. Par exemple, `(cons nil nil)` vaut `True`.

Une chaîne de caractère est représentée par la concaténation des symboles de son sous-arbre gauche et de son sous-arbre droit. L'arbre vide correspond alors au mot vide. Ainsi, on peut représenter le mot 'SI' de la façon suivante : `(cons S I)`.

Prenons l'arbre binaire suivant :

```
T = (cons E (cons S (cons I (cons R nil))))
```

Il peut être interprété :

- Comme un entier, et valoir 4.
- Comme une chaîne de caractères et valoir "ESIR"
- Comme un booléen et ainsi valoir `True`.
- Par ailleurs, `(list T)` correspond également à une liste à 4 éléments

Variables

Les variables sont locales à la fonction où elles apparaissent. Il n'y a pas de variable globale.

Les variables ne permettent pas les effets de bords. Par exemple,

```
X := (cons nil nil)
Y := X
X := nil
```

Après l'exécution de la ligne 3, Y vaut `(cons nil nil)`. Si l'on souhaitait modifier la valeur de Y, il faudrait utiliser une syntaxe du style `Y := nil`.

Commandes et structures de contrôle

La seule chose modifiable via les commandes est la relation entre les variables et les arbres binaires. Voici une liste des commandes et des structures de contrôle en While :

- `X := Expr` correspond à une assignation où Expr correspond à une expression comme expliqué ci-dessus. Par exemple, `X := (cons nil nil)`.
- `nop` correspond à une commande qui ne fait rien.
- `C1; C2` correspond à la syntaxe pour exécuter C1 puis C2. Le point virgule est un opérateur permettant l'enchaînement de commandes.
  Par exemple, `X := nil; Y := (cons nil nil)`
- `if E then C1 fi` : si $[E]_{bool}$ alors exécute C1 sinon rien.
- `if E then C1 else C2 fi` : si $[E]_{bool}$ alors exécute C1 sinon exécute C2.
  Par exemple, `if X then Y := nil else Y := (cons nil nil) fi` .

- `while E do C od` : si $[E]_{bool}$ alors exécute C, répète tant que $[E]_{bool}$
  /!\ Cette structure de contrôle peut boucler indéfiniment
- `for E do C od` : répète $[E]_{int}$ fois la commande C
  /!\ L'exécution de C ne perturbera pas le décompte du for :
  Soit v = $[X]_{int}$.
  `for X do X := (cons nil X) od` double la longueur de X
  En sortie de boucle X -> (cons nil (cons nil … X) …) de taille 2v
- `foreach X in E do C od` : pour chaque élément X de E, exécuter C.
  Exécute $[E]_{int}$ fois la commande C.
  Si on pose v, la valeur de E en entrée dans la boucle, on exécute C sur le fils gauche de v. La deuxième itération exécutera C sur le fils gauche du fils droit de v et ainsi de suite.
  Par exemple, pour l'arbre `(cons (cons A B) (cons D nil))`, C sera exécuté sur `(cons A B)` et sur `D`.

Fonctions

Les programmes While sont généralement définis à l'intérieur de **fonctions**, dont la principale est appelée `main`. Une fonction est définie comme suit :

```
function nom_de_la_fonction:
read Parametres_Entree
%
    Commandes
%
write Valeurs_Sortie
```

- `nom_de_la_fonction` : Le nom de la fonction.
- `Parametres_Entree` : Les paramètres en entrée de la fonction.
- `Commandes` : Les commandes à exécuter dans la fonction.
- `Valeurs_Sortie` : Les valeurs de sortie de la fonction.

Paramètre d'entrée

Les programmes While acceptent 2 types de paramètres d'entrée, les entiers et les chaînes de caractère qui décrivent un arbre, de la forme : "(cons ES (cons IR nil))".

Exemples

```
function add :
read Op1, Op2
%
    Result := Op1;
    for Op2 do
        Result := ( cons nil Result )
    od
%
write Result
```

```
function mul :
read Op1, Op2
%
    for Op1 do
        Result := (add Result Op2)
    od
%
write Result
```

```
function main :
read N1, N2, N3
%
    B := (add N1 N2) ;
    A := (mul B N3) ;
    Result := (cons int A)
%
write Result
```

Comment utiliser le compilateur ?

Après avoir décompresser l'archive du projet dans un dossier suivez les étapes suivantes :

- être sur un système qui peut exécuter des scripts bash et jar.
- écrire votre code while dans un fichier txt. exemple : "nomdufichier.txt" ou utiliser les fichiers du répertoire Exemples
- ouvrir une console et aller dans le projet
- lancer ' **./compilateur.sh"**' cela vas compiler le compilateur
- lancer ' **./programme.sh "chemin/vers/mon/fichier/while.txt" [param1 param2 …]** afin d'exécuter votre programme
  (si vous n'avez pas fait l'étape précédente, pas de panique, une sécurité est la pour le faire a votre place ;p)

Ce qui sur un programme d'addition donne '**./programme.sh "Exemples/add.txt" 1 1**' et devrais ressortir 2 dans la console.