PLP - Laboratoire 2

Alex, Happy, Evaluateur

18 décembre 2019

Étudiants
Doran Kayoumi
Chau Ying Kot

Professeur
Marc Dikötter
Assistant
Nadir Benallal

heig-vd
HAUTE ÉCOLE
D'INGÉNIERIE ET DE GESTION
DU CANTON DE VAUD
www.heig-vd.ch

1 Lexer

Pour notre analyser lexicale, nous avons défini, en plus de ce qui a été données en cours, des tokens pour des mots clés généraux, des opérateurs, des symboles spéciaux, constante, nom de variable et nom de fonction.

1.1 Tokens

TLet	TIn	TSym	TInt	TVar
TIf	TThen	TElse	TFunc	TDef

1.2 Symboles

Pour les symboles, nous avons des opérateurs de comparaisons, opérateurs arithmétiques, les parenthèses et la virgule [1].

1.3 Fonction & Variable

Les noms de fonctions et variables peuvent contenir des lettres majuscules ou minuscule, des digits ou le caractère '_'. La différentiation se fait sur le premier caractère.

- Les fonctions débutent obligatoirement par une majuscule
- Les variables débutent obligatoirement par une minuscule

2 Grammaire

2.1 Terminaux

Les éléments terminaux de notre grammaire sont les tokens que nous avons défini dans notre analyseur lexicale. (c.f. section 1)

2.2 Règles

Nous avons définis 5 règles pour notre grammaire.
La première `Line` qui nous permet de déterminer si nous devons traiter une définition de fonction ou une expression. Ensuite `Def` pour les définitions de fonction et `Exp` pour les expression (c.f. section 2.2.1). Et finalement les règles `Exps` et `Vars` pour des listes d'expression et listes de variables repectivement.

2.2.1 Expression

- `Let Name Exp Exp` : Permet de faire une définition dans une expression
- `Bin [Char] Exp Exp` : Exprime un opérateur binaire, l'opérateur est de type chaîne de caractère
- `Cst Int` : Une constante littérale entière
- `Var Name` : Un nom de variable
- `If Exp Exp Exp` : Structure de contrôle, la première expression correspond à un prédicat, si le prédicat est rempli la seconde expression est évalué, sinon la dernière
- `Func Name [Exp]` : Fonction à plusieurs paramètre de type expression
- `Def Name [Name] Exp` : Défini une fonction en indiquant le nom, la liste de ses arguments et le corps de la fonction

1. La virgule est utilisé comme symbole séparateur des paramètres d'une fonction

3 Évaluateur

Nous avons crée un environnement de test qui définie :

- Variables
  - `a` qui possède la valeur 1
  - `b` qui possède la valeur 2
  - `c` qui possède la valeur 3
- Fonctions
  - `Succ n` : Retourne $n - 1$
  - `Pred n` : Retourne $n + 1$
  - `Geoseries n` : Retourne la série géométrique de n
  - `Fibonacci n` : Retourne le suite de Fibonacci pour n
  - `Fact n` : Calcule $n!$

4 Résultats

4.1 Arbre syntaxique

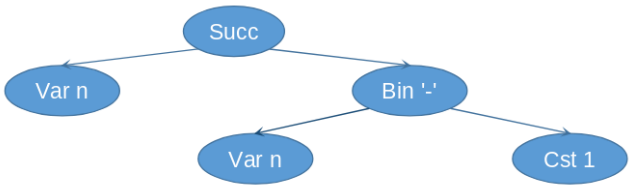

FIGURE 1 – Arbre syntaxique de notre fonction `Succ n`

4.2 Factorielle

```
*Lab8> main ([],[])
SLP>def Fact a = if a then a*Fact(a-1) else 1
[TDef "def",TFunc "Fact",TVar "a",TSym "=",TIf,TVar "a",TThen,TVar "a",TSym "*",TFunc "Fact",TSym "(",TVar "a",TSym "-",TInt 1,TSym ")",TElse,TInt 1]
Def "Fact" ["a"] (If (Var "a") (Bin "*" (Var "a") (Func "Fact" [Bin "-" (Var "a") (Cst 1)])) (Cst 1))
"Fact"
SLP>Fact (4)
[TFunc "Fact",TSym "(",TInt 4,TSym ")"]
Func "Fact" [Cst 4]
24
```

FIGURE 2 – Définition de la fonction factorielle et l'appelle de celle-ci

4.3 Notre fonction - Suite de Fibonacci

```
*Lab8> main ([],[])
SLP>def Fact a = if a then a*Fact(a-1) else 1
[TDef "def",TFunc "Fact",TVar "a",TSym "=",TIf,TVar "a",TThen,TVar "a",TSym "*",TFunc "Fact",TSym "(",TVar "a",TSym "-",TInt 1,TSym ")",TElse,TInt 1]
Def "Fact" ["a"] (If (Var "a") (Bin "*" (Var "a") (Func "Fact" [Bin "-" (Var "a") (Cst 1)])) (Cst 1))
"Fact"
SLP>Fact (4)
[TFunc "Fact",TSym "(",TInt 4,TSym ")"]
Func "Fact" [Cst 4]
24
```

FIGURE 3 – Définition de la fonction Fib et l'appelle de celle-ci