PROYECTO FINAL

Tina María Torres - 2259729

Marlon Astudillo Muñoz - 2259462

Juan José Gallego Calderón - 2259433

Juan José Hernández Arenas - 2259500

Universidad Del Valle

Ingeniería De Sistemas

FUNDAMENTOS DE INTERPRETACIÓN Y COMPILACIÓN DE LENGUAJES DE PROGRAMACIÓN

Tuluá, Valle Del Cauca

2024

Para crear este lenguaje de programación, primero definimos la parte léxica del código, llevando a cabo el espacio en blanco, comentarios, identificadores, digitos binarios positivos y negativos, digitos decimales, flotantes, hexadecimales y octales. Luego de definir la parte léxica definimos la gramática teniendo en cuenta a-programa, bool-exp, var-exp, num-exp, cadena-exp, decl-exp, lista-exp, cons-exp, empty-list-exp, array-exp y todas las demás expresiones requeridas para el proyecto.

Realizamos una función llamada eval-program, la cual envía los datos de a-programa y el ambiente inicial a evaluar-expresion.

Posteriormente a ello, definimos la función evaluar-expresion, dentro de esta función realizamos un cases que valida que tipo de expresión es,por ejemplo: Si la expresión es bool-exp, verifica si es un true-exp o false-exp, luego de eso se toma el valor #T o #F dependiendo de la expresión verificada. Tenemos var-exp, en num-exp tenemos las expresiones numéricas, aunque no realizamos los números octales y hexadecimales, los plasmamos.

En cadena-exp realizamos un let-loop, el cual convierte los símbolos a string. En decl-exp realizamos un cases, el cual evalua lavar-exp y también revisa el set para que dentro de él no se permita la expresión set. En esta función evaluamos todas las expresiones, pero no realizamos todas por cuestiones de tiempo, como: primitivas booleanas, primitivas de listas, booleanas, de array y cadenas. Tampoco realizamos algunas expresiones como funciones recursivas, switch y otras. Específicamente tenemos realizados los set-exp, var-exp, begin, set, if-exp, números flotantes, decimales y binarios.

Realizamos dos funciones para aplicar primitivas, para los números decimales y flotantes, y para los binarios. En la función para aplicar primitivas a los binarios, en cada primitiva realizamos lo siguiente: Realizamos condicionales para verificar si las dos expresiones son negativas, si la primera o segunda expresión es negativa, o si las dos son positivas. Para realizar las operaciones con binarios, sustraemos el identificador “b” para luego cambiar la expresión a number (string->number), posteriormente realizamos la operación y convertimos el resultado a string nuevamente (number->string), y verificamos utilizando la función sacarPrimero si el resultado es positivo o negativo, si es positivo, realizamos un string-append para solo ingresar en la cadena el identificador “b”, si es negativo ingresamos “-b”.

La función sacarPrimero verifica si el primer elemento de la cadena es negativo y retorna un booleano dependiendo de la verificación.
Tenemos el scan&parse y el interpretador, los cuales fueron dados por el profesor; definimos también nuestro ambiente inicial.

La función eval-rand mapea cada rand y los envía a la función eval-rand, la cual llama a evaluar-expresión y retorna la expresión ya evaluada.
Definimos también el tipo de dato ambiente, creamos ambientes vacios con empty-env, creamos ambientes extendidos con extend-env y definimos una función que busca un símbolo en un ambiente.

Como último tenemos las funciones auxiliares para encontrar la posición de un símbolo, las cuales se necesitan para apply-env.

Algunas pruebas

```
--> (b11 - -b1)
b100
--> (b111 + -b1101)
-b110
--> (22 + 22)
44
--> (22.2 - 33.2)
-11.000000000000004
```

```
--> let x = 0 in set x = 10
"variable cambiada"
```

```
--> var x = 10 in begin set x = 20; x end
20
```

Enlace GitHub
[MARAST01/PROYECTOFLP (github.com)]