Autómatas, Teoría de Lenguajes y Compiladores

Trabajo Práctico Especial

2018 2do Cuatrimestre

Integrantes del Grupo:

Bianca Ritorto, 57082
Clara Guzzetti, 57100
Constanza De Rienzo, 56659
Lucas Sanz Gorostiaga, 56312

Índice

Índice	1
Introducción al lenguaje CASTLE	2
Traducción	2
Desarrollo del lenguaje	2
Dificultades	3
Extensiones	3
Gramática	4, 5, 6
Referencias	7

Introducción al lenguaje CASTLE

La creación del lenguaje CASTLE (de CASTELlano) surge como una ayuda para poder introducir a aquellos interesados en la programación pero que tengan poco conocimiento del inglés o carezcan de ello. Es por esto que el lenguaje consiste en una simplificación de C traducida al castellano, sin librerías estándar. De esta forma, veremos que instrucciones típicas como el *do-while* aparecerán como un *hacer-mientras*.

Traducción

Se eligió que el código de salida del compilador se traduzca a Java ya que el grupo se sentía más cómodo con este lenguaje. También se había podido encontrar una muy útil librería externa que permite realizar operaciones sobre autómatas y expresiones regulares sin mayor dificultad. Sin esta librería, se debería haber programado todo a mano.

Desarrollo del lenguaje

Se utilizaron las herramientas aprendidas en clase, Yacc y Lex. Primero se determinaron las palabras correspondientes al nuevo lenguaje, y se creó con ellas el archivo lexa.l. Una vez determinados todos los símbolos a utilizar, se tuvo que considerar la estructura que se iba a implementar para construir el árbol AST.

Se buscó una implementación de árbol con nodos en C en internet, ya que era lo que más naturalmente sería capaz de representar el AST. Se adaptó dicha implementación para las necesidades del lenguaje, creando así los archivos node.c y node.h.

Luego de tener implementado el árbol, se desarrolló el funcionamiento en sí de la gramática. Se debieron profundizar los conocimientos del lenguaje Yacc, y repasar la teórica vista en clase.

Dificultades

Muchas de las dificultades durante el proceso se debieron a que el grupo no estaba familiarizado con el lenguaje Yacc ni el uso de Lex. Se debió implementar el procedimiento de prueba y error, lo que atrasó el desarrollo del lenguaje.

Se puede notar que al hacer "make" en el proyecto, salen varios *warnings* acerca de conflictos de *shift/reduce*. No se pudieron revisar todos los conflictos ya que tampoco se entendió muy bien la descripción de los errores por parte de Bison y Lex. Se tuvieron que colocar precedencias indicando con *left* y *right* a la mayoría de los operadores en un intento de reducir estos conflictos.

Irónicamente, se encontraron dificultades a la hora de escribir ejemplos para testear el funcionamiento del programa ya que los integrantes del grupo no estaban acostumbrados a las reglas de redacción y sintáxis del nuevo lenguaje creado, en cambio estaban acostumbrados a los lenguajes "de siempre", como C o Java.

Extensiones

La primera extensión en mente para un futuro desarrollo del lenguaje sería el agregado de la declaración de funciones en castellano. Así como también el agregado de tipos de datos como *char, array* (denotado con []), *boolean,* y/o algún tipo de dato más complejo.

Se deberían arreglar todos los *warnings* de los conflictos de *shift/reduce* o *reduce/reduce* para que el lenguaje sea completamente no ambiguo, y también para emprolijar el código de la gramática.

Un mejor y más detallado manejo de errores también sería muy útil a la hora de *debuggear* el funcionamiento del lenguaje.

Gramática

La gramática del lenguaje consiste en la traducción de los operandos *if, else, then, do, while, and, or* e *int* correspondientes al lenguaje C. Básicamente se intenta traducir este lenguaje al español. Y se le agregan las palabras *imprimir, leer, verdadero, falso,* y *cadena* para representar las funciones *printf* y *read*, los valores booleanos (para imprimir), y los strings respectivamente. El resto de los operandos que no están en inglés como los de comparación, negación, asignación, operaciones matemáticas, y los tokens como paréntesis, llaves y punto y coma, se dejan tal y como están.

Un ejemplo del uso del lenguaje CASTLE puede ser:

```
entero x = 10;

hacer {
	si (x % 2 == 0) {
		imprimir x;
	}
	x = x - 1;
} mientras (x >= 0);
```

Este código nos imprimiría los números pares en orden descendente desde el 10 al 0. Por supuesto hay más ejemplos que este, los mismos se encuentran en el mismo repositorio git, en la carpeta *examples*, y se verán en la presentación del trabajo práctico.

Las producciones de la gramática se implementaron de la siguiente forma:

```
program     : statement

statement   : statement statement

statement   : ENDLINE

            | IF LPAREN expression RPAREN LBRACK statement RBRACK
```

```
            | IF LPAREN expression RPAREN LBRACK statement RBRACK ELSE
LBRACK statement RBRACK

            | WHILE LPAREN expression RPAREN LBRACK statement RBRACK

            | DO LBRACK statement RBRACK WHILE LPAREN expression RPAREN

            | definition ENDLINE

            | assignment ENDLINE

            | print ENDLINE

            | scan ENDLINE

            | expression ENDLINE

expression  : operand comparator operand

            | NOT expression

            | LPAREN expression RPAREN AND LPAREN expression RPAREN

            | LPAREN expression RPAREN OR LPAREN expression RPAREN

            | TRUE

            | FALSE

definition : INT_T assignment

            | STRING_T assignment

print       : PRINT arguments

scan        : SCAN ID

operand     : INT

            | STRING

            | ID

            | operand operator operand

            | LPAREN operand RPAREN

comparator  : EQUALSCMP
```

```
            | DIFF
            | LTHAN
            | LETHAN
            | GTHAN
            | GETHAN
assignment : ID EQUALS operand
arguments  : expression
            | operand
operator    : PLUS
            | MINUS
            | AND
            | OR
            | MULT
            | DIV
            | MOD
```

Referencias

https://github.com/faturita/YetAnotherCompilerClass
https://gnuu.org/2009/09/18/writing-your-own-toy-compiler/
https://github.com/cs-au-dk/dk.brics.automaton