Universidad Rafael Landívar
Facultad de Ingeniería
Lenguajes formales y autómatas, Sección: 01
Catedrático: Ing. Moisés Alonso

Fase II
"Generador de scanner"

Roberto Solares (1173318)

Guatemala, 4 de Marzo del 2020

Objetivo

Dentro del análisis de los lenguajes Formales es importante conocer las fases del proceso de compilación, la finalidad del proyecto es que el estudiante comprenda la función de los analizadores léxico y sintáctico de un compilador a través de la generación de un programa que sea capaz de reconocer un lenguaje y finalmente evaluar si las palabras utilizadas están bien formadas de acuerdo con una gramática.

El proyecto consta de 3 fases, las cuales son dependientes, es decir, que para poder terminar la fase III, las anteriores deben estar completas, de lo contrario no se podrá entregar el funcionamiento completo.

Análisis

- **Entradas:**Un archivo de texto.
- **Salidas:**Mensaje informativo, que indica si está bien o mal.
- **Procesos:**
    - Leer el archivo.
    - Generar los árboles de expresión.
    - Evaluar el texto utilizando los árboles.
    - Mostrar algún mensaje al usuario.
- **Restricciones:**Ninguna.

Expresiones Regulares

SETS

((S·E·T·S)·((\t|\s)*·(\n))+·((\t|\s)*·[A-Z]+·(\t|\s)*·=·(\t|\s)*·((('·.·')|(C·H·R·\(·[0-9]+·\)))·((\.·\.)·(('·.·')|(C·H·R·\(·[0-9]+·\))))?·(\+·(('·.·')|(C·H·R·\(·[0-9]+·\)))·((\.·\.)·(('·.·')|(C·H·R·\(·[0-9]+·\))))?)*)·((\t|\s)*·(\n))+)+)?

TOKENS

(\n)*·(T·O·K·E·N·S)·((\t|\s)*·(\n))+·((\t|\s)*·(T·O·K·E·N)·(\t|\s)+·[0-9]+·(\t|\s)*·=·(\t|\s)*·(\s|*|\+|\?|\(|\)|\||'·.·'|[A-Z]+|{|})+·((\t|\s)*·(\n))+)+

ACTIONS

(\n)*·(A·C·T·I·O·N·S)·((\t|\s)*·(\n))+·((\t|\s)*·(R·E·S·E·R·V·A·D·A·S·\(·\))·((\t|\s)*·(\n))+·{·((\t|\s)*·(\n))+·((\t|\s)*·[0-9]+·(\t|\s)*·=·(\t|\s)*·'·[A-Z]+·'·((\t|\s)*·(\n))+)+·}·((\t|\s)*·(\n))+)·((\t|\s)*·[A-Z]+·\(·\)·((\t|\s)*·(\n))+·{·((\t|\s)*·(\n))+·((\t|\s)*·[0-9]+·(\t|\s)*·=·(\t|\s)*·'·[A-Z]+·'·((\t|\s)*·(\n))+)+·}·((\t|\s)*·(\n))+)*

ERRORS

((\t|\s)*·(\n))+·([A-Z]+·(E·R·R·O·R)·(\t|\s)*·'·(\t|\s)*·=·(\t|\s)*·[0-9]·((\t|\ s)*·(\n))+)+

Expresión final

^·(" + SETS + ")·(" + TOKENS + ")·(" + ACTIONS + ")·(" + ERRORS + ")·$

Diagramas de clases

Regex
public string Expression public Node Tree public Dictionary<Tuple<int, string>, Tuple<List<int>[], bool>> FirstLastTable public Dictionary<Tuple<int, string>, List<int>> FollowsTable public Dictionary<Tuple<string, List<int>, bool>, Dictionary<string, List<int>>> Transitions private readonly Tokenizer tokenizer private readonly Utils utils private int count
public Regex(string regex) public bool Evaluate(string text) private Node AddTerminal(Node node) private void CreateTree(string regex) private void CreateFirstlastsTable(Node node) private void CreateFollowsTable(Node node) private void CreateTransitions()

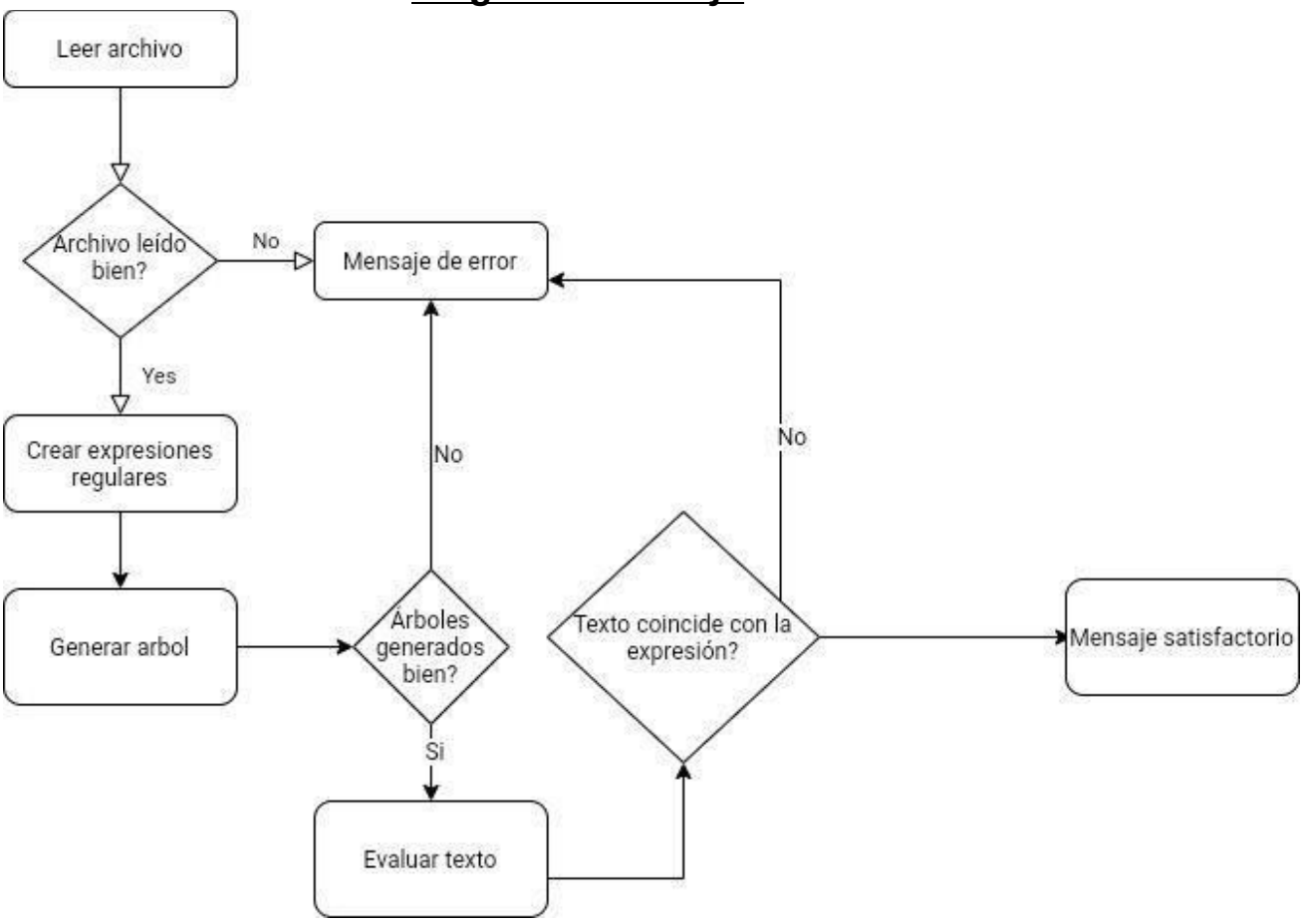
Node
public string Value public int Identifier public Node LeftChild public Node RightChild public Node Parent
public Node(string value, int identifier)

Tokenizer

public List<string> TokenizeExpression(string regex) private bool DigitInterval(string token) private bool DigitInterval(string token) private bool UpperInterval(string token)

Utils
public bool CheckPrecedence(string token, string lastOperator, List<string> operators) public Dictionary<Tuple<int, string>, List<int>> CreateDictionary(Node node) private void PopulateDictionary(Node node, ref List<Tuple<int, string>> list)

Diagramas de flujo



Algoritmos

Table de First y Last

- Entradas
    - El nodo raiz del arbol “node”
    - El diccionario para guardar los datos “table”

- Proceso

1. Si node no es null
    a. Llamar a la misma función con el hijo izquierdo del nodo
    b. Llamar a la misma función con el hijo derecho del nodo
    c. Si el hijo izquierdo del nodo es null y el hijo derecho del nodo es null
        i. Inicializar lista “first” con el identificador del node
        ii. Inicializar lista “last” con el identificador del node
        iii. Inicializar arreglo de listas “values” con los valores de first y last
        iv. Inicializar tupla “identifier” con los valores identificador del nodo y el valor del nodo
        v. Inicializar tupla “data” con los values y false.
        vi. Agregar a table el valor de identifier y data
    d. Sino
        i. Crear la lista “first” vacia
        ii. Crear la lista “last” vacia
        iii. Inicializar la variable “nullable” con falso
        iv. Inicializar la tupla “key” con los valores identificador del hijo izquierdo del nodo y el valor del hijo izquierdo del nodo
        v. Inicializar la tupla “c1” con el valor que se encuentre en table que poseen key
        vi. Si el valor del nodo es igual a |
            1. Asignar con los valores identificador del hijo derecho del nodo y el valor del hijo derecho del nodo
            2. Inicializar la tupla “c2” con el valor que se encuentre en table que poseen key
            3. Agregar a first los items 1[0] de c1 y c2
            4. Agregar a last los items 1[1] de c1 y c2
            5. Si el item 2 de c1 es true o el item 2 de c2 es true
                a. Asignar a nullable true
        vii. Sino si el valor del nodo es igual a ·
            1. Asignar con los valores identificador del hijo derecho del nodo y el valor del hijo derecho del nodo
            2. Inicializar la tupla “c2” con el valor que se encuentre en table que poseen key
            3. Si item 2 de c1 es true
                a. Agregar a first los items 1[0] de c1 y c2
            4. Sino

a. Agregar a first el items 1[0] de c1
5. Si item 2 de c2 es true
    a. Agregar a last los items 1[1] de c1 y c2
6. Sino
    a. Agregar a first el items 1[1] de c2
7. Si el item 2 de c1 es true y el item 2 de c2 es true
    a. Asignar a nullable true

viii. Sino si el valor del nodo es igual a *
1. Agregar a first el item1[0] de c1
2. Agregar a last el item1[1] de c1
3. Asignar a nullable true

ix. Sino si el valor del nodo es igual a +
1. Agregar a first el item1[0] de c1
2. Agregar a last el item1[1] de c1

x. Sino si el valor del node es igual a ?
1. Agregar a first el item1[0] de c1
2. Agregar a last el item1[1] de c1
3. Asignar a nullable true

xi. Inicializar arreglo de listas “values” con los valores de first y last

xii. Inicializar tupla “identifier” con los valores identificador del nodo y el valor del nodo

xiii. Inicializar tupla “data” con los values y false.

xiv. Agregar a table el valor de identifier y data

Table de Follows

- Entradas
    - El nodo raiz del arbol “node”
    - El diccionario para guardar los datos “table”

- Proceso

2. Si node no es null
    a. Llamar a la misma función con el hijo izquierdo del nodo
    b. Llamar a la misma función con el hijo derecho del nodo
    c. Si el hijo izquierdo del nodo no es null o el hijo derecho del nodo no es null
        i. Si el valor del nodo es igual a ·
            1. Inicializar la tupla “key” con los valores identificador del hijo izquierdo del nodo y el valor del hijo izquierdo del nodo
            2. Inicializar la tupla “c1” con el valor que se encuentre en table que poseen key
            3. Asignar con los valores identificador del hijo derecho del nodo y el valor del hijo derecho del nodo
            4. Inicializar la tupla “c2” con el valor que se encuentre en table que poseen key
            5. Por cada “element” que esta en el item1[1] de c1

a. Asignara a la tupla “insert” el valor de la llave de table que posea element
b. Agregar a la tabla en la posicion element el item 1[0] de c1

ii. Sino si el valor del nodo es igual a * o el valor del nodo es igual a +
1. Asignar con los valores identificador del hijo derecho del nodo y el valor del hijo derecho del nodo
2. Inicializar la tupla “c2” con el valor que se encuentre en table que poseen key
3. Por cada “last” que esta en el item1[1] de c1
a. Asignara a la tupla “insert” el valor de la llave de table que posea element
b. Agregar a la tabla en la posicion element el item 1[0] de c1