JOSE CARLOS GIRON MARQUEZ

**Universidad Rafael Landívar**  **Lenguajes Formales y Autómatas**

**GENERADOR SCANNER**

**INTRODUCCION**

Dentro del análisis de los lenguajes Formales es necesario conocer las fases del proceso de compilación, el presente proyecto incluye la función de 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 (Generador Scanner).

El proyecto consta de 3 fases, la primera fase es el análisis léxico de la gramática, la segunda, el análisis sintáctico y la ultima el propio scanner.

**DEFINICIONES IMPORTANTES**

### Cadena

* **Definición**: una cadena es una secuencia finita de símbolos tomados de ∑.
* **Ejemplo**: 'cabcad' es una cadena válida en el conjunto de alfabeto ∑={a, b, c, d}

### Alfabeto

* **Definición**: un alfabeto es cualquier conjunto finito de símbolos.
* **Ejemplo**: ∑ = {a, b, c, d} es un conjunto alfabético donde 'a', 'b', 'c' y 'd' son símbolos.

**Expresión Regular**

Una expresión regular es una forma de representar los lenguajes regulares (finitos o infinitos) y se construye utilizando caracteres del alfabeto sobre el cual se define el lenguaje.

**Gramática**

Una gramática regular es una gramática formal (N, Σ, P, S) que puede ser clasificada como regular izquierda o regular derecha. Las gramáticas regulares sólo pueden generar a los lenguajes regulares de manera similar a los autómatas finitos y las expresiones regulares

**Lenguaje**

Un lenguaje formal es un lenguaje cuyos símbolos primitivos y reglas para unir esos símbolos están formalmente especificados. Es un subconjunto de ∑ \* para algún alfabeto ∑. Puede ser finito o infinito.

Ejemplo: si el idioma toma todas las cadenas posibles de longitud 2 sobre ∑ = {a, b}, entonces L = {ab, aa, ba, bb}

**Autómata**

El término "Autómatas" se deriva de la palabra griega "αὐτόματα" que significa "autoactuante". Un autómata (Automata en plural) es un dispositivo informático autopropulsado abstracto que sigue automáticamente una secuencia predeterminada de operaciones. Un autómata finito (AF) o máquina de estado finito es un modelo computacional que realiza cómputos en forma automática sobre una entrada para producir una salida. Este modelo está conformado por un alfabeto, un conjunto de estados finito, una función de transición, un estado inicial y un conjunto de estados finales. Su funcionamiento se basa en una función de transición, que recibe a partir de un estado inicial una cadena de caracteres pertenecientes al alfabeto (la entrada), y que va leyendo dicha cadena a medida que el autómata se desplaza de un estado a otro, para finalmente detenerse en un estado final o de aceptación, que representa la salida.

**Autómatas finitos no deterministas (NFA)**

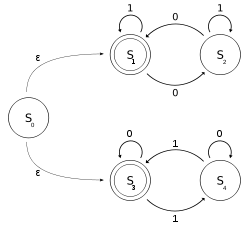
Un autómata finito no determinista es un modelo matemático que consiste en:

1. Un conjunto de estados S;
2. Un conjunto de símbolos de entrada, ∑ , llamado alfabeto de símbolos de entrada.
3. Un movimiento de función de transición que asigna pares de símbolos de estado a conjuntos de estados.
4. Un estado inicial.
5. Un conjunto de estados F llamado estado de aceptación o final.

Un NFA puede describirse mediante un gráfico de transición (gráfico etiquetado) donde los nodos son estados y los bordes muestran la función de transición.

El etiquetado en cada borde es un símbolo en el conjunto del alfabeto, ∑ , o  denota una cadena vacía.

Ejemplo:



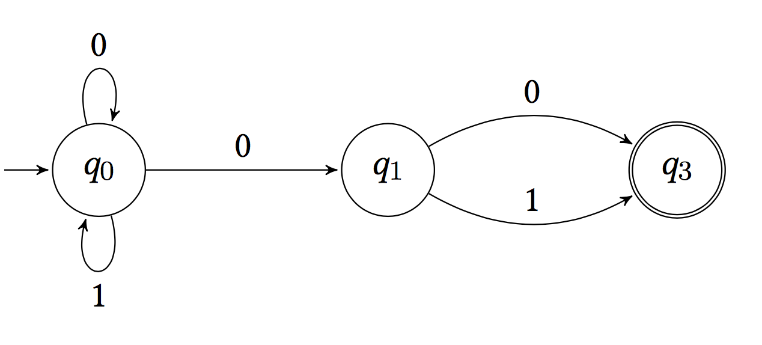
**Autómata finito determinista (DFA)**

En DFA, para cada símbolo de entrada, se puede determinar el estado al que se moverá la máquina. Por lo tanto, se llama Autómata determinista. Como tiene un número finito de estados, la máquina se llama Deterministic Finite Machine o Deterministic Finite Automaton.

Definición formal de un DFA

Un DFA puede ser representado por una tupla de 5 (Q, ∑, δ, q 0 , F) donde -

* Q es un conjunto finito de estados.
* ∑ es un conjunto finito de símbolos llamado alfabeto.
* δ es la función de transición donde δ: Q × ∑ → Q
* q 0 es el estado inicial desde donde se procesa cualquier entrada (q 0 ∈ Q).
* F es un conjunto de estados / estados finales de Q (F ⊆ Q).



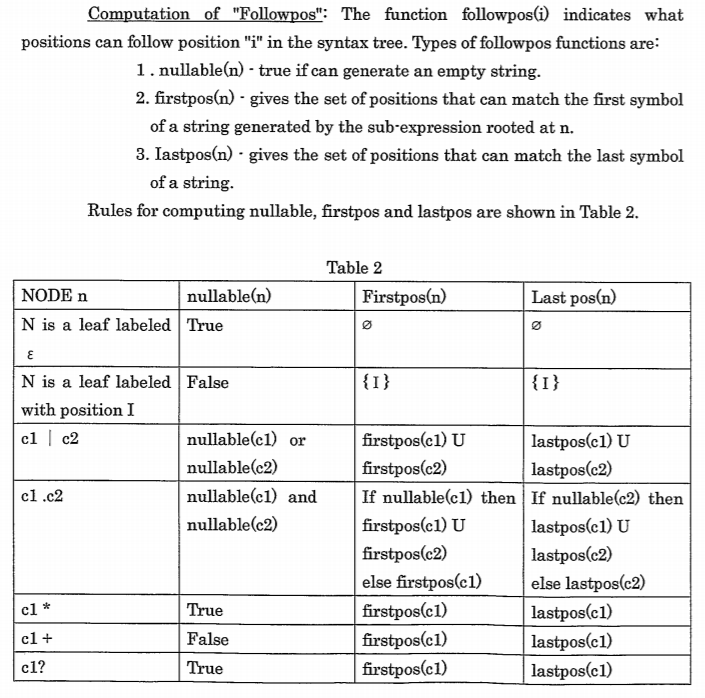
**ALGORITMOS Y DIAGRAMAS**

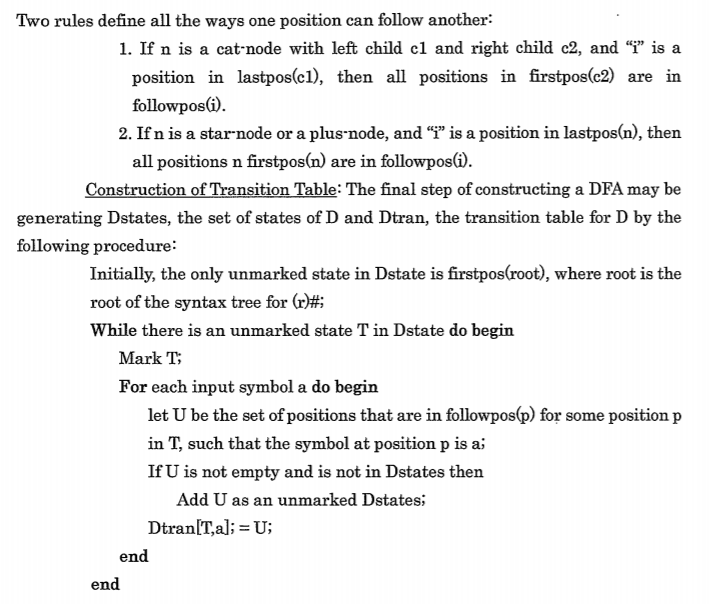
|  |
| --- |
| **ALGORITMO GENERAL** |
| **Entradas:** Expresion regular del lenguaje, Lenguaje (Conjunto de strings a validar)  **Salidas:** (bool) Validacion   * Construir árbol a partir de expresión regular. * Enumerar Nodos * Calcular FirstPos, LastPos y Nullable (Ver tabla 2). * Calcular FollowPos (Ver Algoritmo Tabla Follow). * Calcular Transiciones (Ver Algoritmo Tabla de Transiciones). * PARA CADA palabra en el Lenguaje. * Hacer Match a la palabra. (Ver Algoritmo Match del string) |
| **MATCH DEL STRING** |
| **Entrada:** Cadena (Palabra perteneciente a un lenguaje), DFA  **Salidas:** (bool) Validacion  VALIDAR CADENA   * PARA CADA símbolo de la cadena * SI el estado actual del DFA contiene al símbolo * ENTONCES el estado actual, se convertirá en el estado que tenga transición con el símbolo. * DE LO CONTRARIO * SI el estado actual es un estado de aceptación. * ENTONCES la cadena es válida. * DE LO CONTRARIO * ENTONCES la cadena NO es válida. |

|  |
| --- |
| **Algoritmo Creación de Árbol de Expresión a través de expresión regular**  **Creador: Moises Alonso** |
| Entradas   1. Tokens de la expresión regular (Símbolos terminales “**st**”, meta caracteres operadores incluyendo la concatenación “**op**”) 2. Pila de Tokens llamada “**T**” 3. Pila de árboles llamada “**S**” |
| Salidas   1. Árbol de expresión con el símbolo terminal extendido # |
| Proceso   1. Mientras existan tokens en la expresión regular 2. Obtener **token** 3. Si **token** es un carácter de escape.    1. Si a la expresión regular aún le queda al menos un **token.**       1. Obtener siguiente **token.**       2. Convertir **token** en árbol.       3. Hacer “push” a la pila **S** con el nuevo árbol.    2. De lo contrario       1. Error, se esperaban más tokens. 4. Si **token** es **st**    1. Convertir **st** en árbol    2. Hacer “push” a la pila **S** con el nuevo árbol generado de **st** 5. Sino Si **token** es **“(“**    1. Hacer “push” a la pila **T** con **token** 6. Sino Si **token** es **“)“**    1. Mientras la longitud de **T** sea mayor que **0** y el último dato insertado en **T** sea diferente de **“(“**, hacer:       1. Si Longitud de **T** es igual a **0**          1. Existe error, faltan operandos       2. Si la longitud de **S** es menor a **2**          1. Existe error, faltan operandos       3. Hacer “pop” a **T** y convertirlo en árbol llamado **temp**       4. Hacer “pop” a **S** y asignarlo al hijo derecho de **temp**       5. Hacer “pop” a **S** y asignarlo al hijo izquierdo de **temp**       6. Hacer “Push” de **temp** en la pila **S**    2. Hacer “pop” a **T** con el último dato 7. Sino si **token** es **op**    1. Si **op** es unario       1. Convertir **op** en árbol       2. Si la longitud de **S** es menor que **0**          1. Existe error, faltan operandos       3. Hacer “pop” de **S** y asignarlo como hijo izquierdo       4. Hacer “push” a la pila **S** con el nuevo árbol generado de **op**    2. Sino si **T** no está vacia y el “top” **op** en **T** es diferente a **“(“** y precedencia de **token** es menor o igual a último **op** en **T**       1. Extraer de **T** a **op**, convertirlo en árbol y llamarlo **temp**       2. Si cantidad de elementos en en **S** es menor a 2          1. Existe error, faltan operandos       3. Extraer último árbol de **S** y asignarlo al hijo derecho de **temp**       4. Extraer último árbol de **S** y asignarlo al hijo izquierdo de **temp**       5. Push de **temp** en la pila **S**       6. Push del **token** en la pila **T**    3. Si **op** no es unario Hacer “push” en la pila **T** con **token** 8. De lo contrario    1. Error, no es token reconocido 9. Si aún existen tokens en la expresión regular, ir a paso 2 10. Mientras la longitud de **T** sea Mayor que **0**     1. Hacer “pop” de **T** y crear un nuevo árbol llamado **temp**     2. Si **temp** es **“(“**        1. Existe error, faltan operandos     3. Si longitud de **S** menor que **2**        1. Existe error, faltan operandos     4. Hacer “pop” a la pila **S** y asignarlo como hijo derecho de **temp**     5. Hacer “pop” a la pila **S** y asignarlo como hijo izquierdo de **temp**     6. Hacer “push” a la pila **S** con el árbol **temp** 11. Si longitud de **T** es mayor que **0** ir a paso **9** 12. Si longitud de **S** es diferente de **1**     1. Existe error, faltan operandos 13. Hacer “pop” a **S** y retornar el valor |
|  |

**Reglas de construcción de un DFA**

Fuente: Aldous C., Richard T. & Rogelio G. (2004, 6 de febrero). System and method for securing computer against virus. Recuperado 29 febrero, 2020, de <https://patentimages.storage.googleapis.com/3e/ec/00/5e0b3704bc5c3d/WO2005076101A2.pdf>





|  |
| --- |
| **ALGORITMO TABLA FOLLOW** |
| OBTENER FOLLOW DE UN NODO   * Obtener follow del hijo izquierdo * Obtener follow del hijo derecho * SI el nodo no es hoja * SI el símbolo es de concatenación * PARA CADA ítem en lastPosition del hijo izquierdo, ENTONCES tendrá como follow todos los ítems en el firstPosition del hijo derecho. * DE LO CONTRARIO SI el símbolo es de una operación de Kleene (+) o (\*) * PARA CADA ítem en lastPosition del hijo izquierdo, ENTONCES tendrá como follow todos los ítems en el firstPosition del hijo izquierdo. |
| **ALGORITMO TABLA DE TRANSICIONES** |
| Obtener la transicion para un simbolo de un estado.  Recorre el listado de nodos, evalua el simbolo y agrega sus follows.   * Crear lista con todos los símbolos del archivo. * Crear nueva lista con que va a contener los estados. * Crear diccionario con que va a contener las transiciones. * Hacer del primer estado, el firstPosition del árbol. * MIENTRAS no existan nuevos estados * PARA CADA nodo en el estado actual * PARA CADA simbolo en el documento * SI el simbolo es igual al valor del nodo * Seleccionar todos los follows del nodo * Crear una transición con ese listado de follows para ese simbolo. * SI el conjunto de nodos aun no se ha agregado a los estados. * Agregar nuevo estado. |

|  |
| --- |
| **Expresiones Regulares:**  **Creador: José Girón** |
| **Expresiones Regulares:**  **------------------------------------------**  **SETS?**  **Ejemplo:**  **LETRA = 'A'..'Z'+'a'..'z'+'\_'**  **DIGITO = '0'..'9'**  **CHARSET = CHR(32)..CHR(254)**  **Expresion regular [Set individual]:**  **[A-Z]+ \*= \*(('[Simbolo]')|('([A-Z]|[a-z]|[0-9])+'(..'([A-Z]|[a-z]|[0-9])+')?)|(CHR\([0-9]\)(..CHR\([0-9]\))?)|(CHR\([0-9]+\))((..CHR\([0-9]+\))?))(( \*\+ \*)(('[Simbolo]')|('([A-Z]|[a-z]|[0-9])+'(..'([A-Z]|[a-z]|[0-9])+')?)|(CHR\([0-9]\)(..CHR\([0-9]\))?)|(CHR\([0-9]+\))((..CHR\([0-9]+\))?)))\***  **Expresion regular [Solucion alterna]:**  **[A-Z]+ \*= \*((('([A-Z]|[a-z]|[0-9]|[Simbolo])+')|(CHR\([0-9]+\)))(..(('([A-Z]|[a-z]|[0-9]|[Simbolo])+')|(CHR\([0-9]+\))))?)+**  **Expresion regular [De la seccion]:**  **(( \*SETS \*)([A-Z]+ \*= \*(('[Simbolo]')|('([A-Z]|[a-z]|[0-9])+'(..'([A-Z]|[a-z]|[0-9])+')?)|(CHR\([0-9]\)(..CHR\([0-9]\))?)|(CHR\([0-9]+\))((..CHR\([0-9]+\))?))(( \*\+ \*)(('[Simbolo]')|('([A-Z]|[a-z]|[0-9])+'(..'([A-Z]|[a-z]|[0-9])+')?)|(CHR\([0-9]\)(..CHR\([0-9]\))?)|(CHR\([0-9]+\))((..CHR\([0-9]+\))?)))\* \*)+)?**  **-------------------------------------------**  **TOKENS**  **Ejemplo:**  **TOKEN 1= DIGITO DIGITO \***  **TOKEN 2='"' CHARSET '"'|''' CHARSET '''**  **TOKEN 4 = '='**  **TOKEN 5 = '<''>'**  **TOKEN 3= LETRA ( LETRA | DIGITO )\* { RESERVADAS() }**  **Expresion regular [Token Individual]:**  **\*TOKEN \*[0-9]+ \*= \*(([A-Z]+)|('([Simbolo]|[A-Z]|[a-z]|[0-9])')| |\?|\||\\*|\+|\(|\)|({ \*[A-Z]+\(\) \*}))+**  **Expresion regular [Alternativa]:**  **\*TOKEN \*[0-9]+ \*= \*(([A-Z]+)|('([Simbolo]|[A-Z]|[a-z]|[0-9])')|(\([A-Z]\))| |\?|\||\\*|\+|({ \*[A-Z]+\(\) \*}))+**  **Expresion regular [De la seccion]:**  **(( \*TOKENS \*)( \*TOKEN \*[0-9]+ \*= \*(([A-Z]+)|('([Simbolo]|[A-Z]|[a-z]|[0-9])')|(\(([Simbolo]|[A-Z]|[a-z]|[0-9])\))| |\?|\||\\*|\+|({ \*[A-Z]+\(\) \*}))+ \*)+)**  **-------------------------------------------**  **ACTIONS**  **Ejemplo:**  **ACTIONS**  **RESERVADAS()**  **{**  **18 = 'PROGRAM'**  **19 = 'INCLUDE'**  **}**  **ACTIONS RESERVADAS() { 18 = 'PROGRAM' 19 = 'INCLUDE' }**  **Expresion Regular [De la seccion]:**  **( \*ACTIONS +RESERVADAS \*\( \*\) \*{( \*[0-9]+ \*= \*'[A-Z]+')+ \*}([A-Z]+ \*\( \*\) \*{( \*[0-9]+ \*= \*'[A-Z]+')+ \*})\*) \***  **-------------------------------------------**  **ERROR**  **Ejemplo:**  **ERROR = 54**  **Expresion Regular [De la seccion]:**  **( \*[A-Z]+ \*= \*[0-9]+)+**  **-------------------------------------------**  **ARCHIVO COMPLETO**  **Expresion regular:**  **(( \*SETS \*)([A-Z]+ \*= \*(('[Simbolo]')|('([A-Z]|[a-z]|[0-9])+'(..'([A-Z]|[a-z]|[0-9])+')?)|(CHR\([0-9]\)(..CHR\([0-9]\))?)|(CHR\([0-9]+\))((..CHR\([0-9]+\))?))(( \*\+ \*)(('[Simbolo]')|('([A-Z]|[a-z]|[0-9])+'(..'([A-Z]|[a-z]|[0-9])+')?)|(CHR\([0-9]\)(..CHR\([0-9]\))?)|(CHR\([0-9]+\))((..CHR\([0-9]+\))?)))\* \*)+)?(( \*TOKENS \*)(TOKEN \*[0-9]+ \*= \*(([A-Z]+)|('([Simbolo]|[A-Z]|[a-z]|[0-9])')|(\(([Simbolo]|[A-Z]|[a-z]|[0-9])\))| |\?|\||\\*|\+|({ \*[A-Z]+\(\) \*}))+ \*)+)( \*ACTIONS +RESERVADAS \*\( \*\) \*{( \*[0-9]+ \*= \*'[A-Z]+')+ \*}([A-Z]+ \*\( \*\) \*{( \*[0-9]+ \*= \*'[A-Z]+')+ \*})\*)( \*[A-Z]+ \*= \*[0-9]+)+ \*#** |

**Diagrama de Clases**

**A screenshot of a cell phone

Description automatically generated**

**REFERENCIAS**

* Aldous C., Richard T. & Rogelio G. (2004, 6 de febrero). System and method for securing computer against virus. Recuperado 29 febrero, 2020, de <https://patentimages.storage.googleapis.com/3e/ec/00/5e0b3704bc5c3d/WO2005076101A2.pdf>
* Muhammad, R. B. (s. f.). Lexical Analyzer. Recuperado 16 de abril de 2020, de <http://www.personal.kent.edu/%7Ermuhamma/Compilers/MyCompiler/chapter3.htm>
* TutorialsPoint. (s. f.). Automata Theory - Quick Guide. Recuperado 16 de abril de 2020, de <https://www.tutorialspoint.com/automata_theory/automata_theory_quick_guide.htm>
* Thakur, D. (s. f.). Convert Regular Expression to DFA - Compiler Design. Recuperado 16 de abril de 2020, de <http://ecomputernotes.com/compiler-design/convert-regular-expression-to-dfa>
* Bailey-Kellogg, C. (s. f.). *DFA* [Código Fuente]. Recuperado de <https://www.cs.dartmouth.edu/~cbk/classes/10/14winter/notes/22/DFA.java>