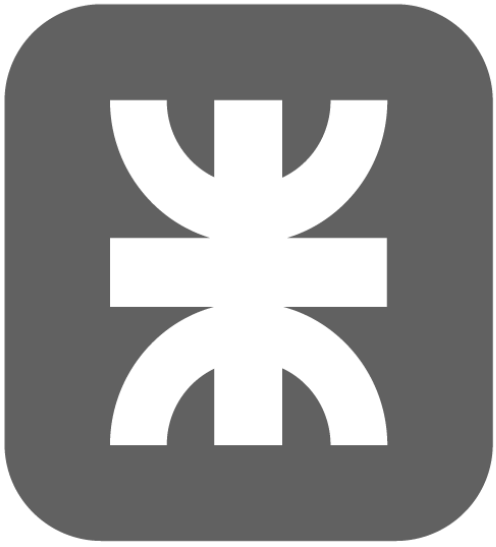
UNIVERSIDAD TECNOLÓGICA NACIONAL



FRD.**UTN**

FACULTAD REGIONAL DELTA

CARRERA: Ingeniería en sistemas de información

ASIGNATURA: Sintaxis y semántica de los lenguajes

DOCENTES:

- Miranda Hernán
- Santos Juan Miguel

TRABAJO PRACTICO Nº1: Lexer

AÑO: 2022

ALUMNOS:

- Domínguez Tomás
- González Tomás Nahuel
- Pastor Hanna Sofia
- Zaracho Julieta Mariel

Contenido

INTRODUCCIÓN

En esta experiencia, se solicitó realizar la implementación de un analizador lexicográfico, desarrollando un autómata finito (AF) por cada uno de los distintos tokens de la gramática y posteriormente modelando un autómata finito determinístico (AFD) que incluya y relacione a todos los AF construidos por cada token.

El programa realizado acepta cadenas que representan código escrito en el lenguaje generado por la gramática asignada. Este código, se convierte luego en una cadena de tokens la cual será correspondiente a la gramática mencionada.

GRAMATICA ASIGNADA

La gramática G= <VN, VT, P, S> asignada es:

```
VT = {eq, id, num, *, +, op, clp, si, entonces, sino, mostrar, aceptar,
mientras, esMenorQue, hacer, (, )}
VN = {Programa, Asignacion, Estructura, Expresion, Valor,
ListaExpresiones, Termino, Factor}
S = Program
P = {

Program → Estructura Program
        | λ

Estructura →"mientras" "id" "esMenorQue" Valor "hacer" "op" Program "clp"
           | "si" Expresion "entonces" "op" Program "clp" "sino" "op"
       Program "clp"
           | "mostrar" Expresion
           | "aceptar" "id"
           | "id" "eq" Expresion

Valor → "id"
       |"num"

Expresion → Termino Expresion2
Expresion2 → "+" Termino Expresion2
           | λ

Termino → Factor Termino2
Termino2 → "*" Factor Termino2
         | λ

Factor → "("  Expresion  ")"
       | Valor

}
```

EXPLICACIÓN Y OBSERVACIONES DEL TRABAJO REALIZADO

Archivo "automatas.py": Implementación de los autómatas

Para cada token se genera un autómata finito determinado, los cuales se formaron con una función.

Todas las funciones trabajan de manera similar, tienen un parámetro "cadena" con la que se le ingresa por parámetro el string a analizar. Este string, es recorrido carácter a carácter por medio de ciclos con el objetivo de encontrar el token correspondiente.

Para el caso de las palabras reservadas, esto se hace comparando letra por letra, verificando si se encuentra la palabra correspondiente a algún autómata. En el caso de los números (afd_num(cadena)) y de los id (afd_id(cadena)), al no ser solo una palabra específica lo que se busca verificar sino un conjunto de caracteres que cumplan con ciertas condiciones, lo que se hace es verificar que cada carácter perteneciente al string ingresado pertenezca a las listas "LETRAS_MAYUSCULAS", "LETRAS_MINUSCULAS" o "NUMEROS", según corresponda.

Estos autómatas pueden retornar tres posibles estados,

- Estado "FINAL": Los autómatas retornaran este estado si al terminar de recorrer la cadena el estado actual pertenece a la lista "estadosFinales" (es una cadena aceptada).
- Estado "NO FINAL": Los autómatas retornaran este valor si al terminar de recorrer la cadena se encuentran en algún estado no perteneciente a la lista "estadosFinales" pero no se ingresaron caracteres que lleven al autómata a un estado trampa. Significando que hay una posibilidad de que si se añaden caracteres se pueda formar una cadena aceptada.
- Estado "TRAMPA": Los autómatas retornan este valor si dentro del ciclo de recorrido de la cadena, alguno de los caracteres no cumple con las condiciones necesarias para continuar en el ciclo de búsqueda. Se le asigna al estado actual un valor arbitrario de -1.

Archivo "lexer.py": Implementación del Lexer

Como ya sabemos, la tarea del lexer es la de enviarle los tokens generados por el código fuente al parser, este no verifica el orden de las palabras, sino que solamente se limita a verificar que los distintos tokens estén bien formados.

La función principal "lexer" recibe el código fuente en forma de string y forma substrings (lexemas) mediante las variables "inicio" y "final", estas variables señalan desde que posición a que posición se considera el lexema. Estos lexemas son probados en todos los autómatas con la función "generaEstadosTrampa" para verificar que sea válido (es decir, que no genere estados trampa) en al menos uno de estos, mientras este sea el caso se continúan agregando caracteres al lexema. Esto se logra aumentando el valor de la variable "final", mientras se mantiene constante a la variable "inicio"

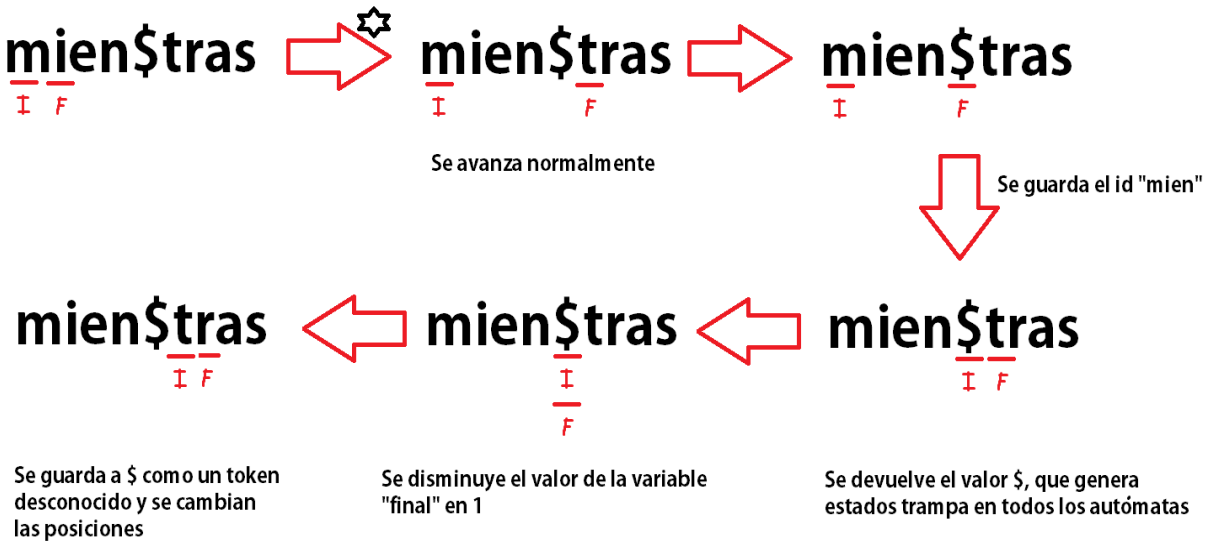
Cuando esta condición ya no se cumple, quiere decir que el lexema actual no genera un token válido en ninguno de los autómatas, pero como estamos agregando caracteres al lexema si y solo si este no genera estados trampa, quiere decir que en el estado anterior si se generaba un token válido.

Por lo tanto, se reduce el valor de la posición final en 1 y se ejecuta la función "guardarToken" que nos devuelve el token de ese lexema en específico, para ello prueba la cadena en todos los autómatas de la lista TOKENS_POSIBLES y añade el token correspondiente a la lista "tokens".

La lista TOKENS_POSIBLES esta ordenada según una pequeña jerarquía, esto es para poder considerar las palabras reservadas, ya que una vez que se quiere guardar el token generado por cierto lexema es posible que ese lexema genere estados finales en varios autómatas (por ejemplo, la palabra "mientras" generaría un estado aceptado en el autómata "mientras" y en el autómata "id") de este modo ya sabemos que cuando hay que guardar una palabra siempre hay que considerar el autómata que genera estado final que antes se encuentra en la lista. En nuestro caso, una vez que un autómata genera estado final ya dejamos de verificar en los demás, debido a que ya tenemos la respuesta que estamos buscando.

Debido al manejo de los substrings, si el programa se encuentra con un carácter desconocido las posiciones inicial y final que se toman para generar los lexemas terminarán siendo iguales, esto es debido a que una vez que nos encontramos con un carácter desconocido, el programa obtendrá el token del conjunto de caracteres que lo precede, colocará las variables "inicio" y "final" de modo que el lexema sea igual al carácter desconocido y, como ese carácter genera estados trampa en todos los autómatas, el programa va a cambiar el valor de "final" a una posición menos, generando la igualdad de los valores.

Ejemplificación del manejo de caracteres desconocidos:



CADENAS DE PRUEBA

A continuación, se presentan distintos ejemplos de cadenas introducidas en el programa, las cuales cuentan respectivamente con sus resultados:

1. **print(lexer("mientras 69 esMenorQue (contador + 45"))**

   Este ejemplo no presenta token desconocidos de la gramática

   Correspondencia de los caracteres ingresados con los tokens de la gramática:

   [('mientras', 'mientras'), ('num', '69'), ('esMenorQue', 'esMenorQue'), ('(', '('), ('id', 'contador'), ('+', '+'), ('num', '45'), (')', ')')]

2. **print(lexer("mientras eq * + ( ) si sino entonces mostrar aceptar esMenorQue hacer op clp"))**
   Este ejemplo no presenta token desconocidos de la gramática

   Correspondencia de los caracteres ingresados con los tokens de la gramática:

   [('mientras', 'mientras'), ('eq', 'eq'), ('*', '*'), ('+', '+'), ('(', '('), (')', ')'), ('si', 'si'), ('sino', 'sino'), ('entonces', 'entonces'), ('mostrar', 'mostrar'), ('aceptar', 'aceptar'), ('esMenorQue', 'esMenorQue'), ('hacer', 'hacer'), ('op', 'op'), ('clp', 'clp')]

3. **print(lexer("si (43 = 21)"))**

   Este ejemplo presenta un token desconocido por la gramática que es [‘=’]

   Correspondencia de los caracteres ingresados con los tokens de la gramática:

   [('si', 'si'), ('(', '('), ('num', '43'), ('num', '21'), (')', ')')]

4. **print(lexer("si (43=25) entonces mostrar variable"))**

   Este ejemplo presenta un token desconocido por la gramática que es [‘=’]

   Correspondencia de los caracteres ingresados con los tokens de la gramática:
   [('si', 'si'), ('(', '('), ('num', '43'), ('num', '25'), (')', ')'), ('entonces', 'entonces'), ('mostrar', 'mostrar'), ('id', 'variable')]

5. **print(lexer("si$ 67 ent#onces"))**

   Este ejemplo presenta un token desconocido por la gramática que es [‘$’, ‘#’]

   Correspondencia de los caracteres ingresados con los tokens de la gramática:

   [('si', 'si'), ('num', '67'), ('id', 'ent'), ('id', 'onces')]

6. **print(lexer("23asd"))**

   Este ejemplo no presenta token desconocidos de la gramática

   Correspondencia de los caracteres ingresados con los tokens de la gramática:

   [('num', '23'), ('id', 'asd')]

7. **print(lexer("mientras 78 esMenorQue var hacer op clp"))E**

   Este ejemplo no presenta token desconocidos de la gramática

   Correspondencia de los caracteres ingresados con los tokens de la gramática:

   [('mientras', 'mientras'), ('num', '78'), ('esMenorQue', 'esMenorQue'), ('id', 'var'), ('hacer', 'hacer'), ('op', 'op'), ('clp', 'clp')]

8. **print(lexer("si (43=25) entonces mostrar variable"))**

   Este ejemplo presenta un token desconocido por la gramática que es [‘=’]

   Correspondencia de los caracteres ingresados con los tokens de la gramática:

   [('si', 'si'), ('(', '('), ('num', '43'), ('num', '25'), (')', ')'), ('entonces', 'entonces'), ('mostrar', 'mostrar'), ('id', 'variable')]

9. **print(lexer("si (2 + 3) entonces / ()"))**

   Este ejemplo presenta un token desconocido por la gramática que es [‘/’]

   Correspondencia de los caracteres ingresados con los tokens de la gramática:

   [('si', 'si'), ('(', '('), ('num', '2'), ('+', '+'), ('num', '3'), (')', ')'), ('entonces', 'entonces'), ('(', '('), (')', ')')]

10. **print(lexer("mientras (+)"))**

    Este ejemplo no presenta token desconocidos de la gramática

    Correspondencia de los caracteres ingresados con los tokens de la gramática:

    [('mientras', 'mientras'), ('(', '('), ('+', '+'), (')', ')')]

11. **print(lexer("34 = (2 + var"))**

    Este ejemplo presenta un token desconocido por la gramática que es [‘=’]

    Correspondencia de los caracteres ingresados con los tokens de la gramática:

    [('num', '34'), ('(', '('), ('num', '2'), ('+', '+'), ('id', 'var')]

12. **print(lexer("si entonces 23 / mientras #"))**

    Este ejemplo presenta un token desconocido por la gramática que es [‘/’, ‘#’]

    Correspondencia de los caracteres ingresados con los tokens de la gramática:

    [('si', 'si'), ('entonces', 'entonces'), ('num', '23'), ('mientras', 'mientras')]

13. **print(lexer("mientras a < 97"))**

    Este ejemplo presenta un token desconocido por la gramática que es [‘<’]

    Correspondencia de los caracteres ingresados con los tokens de la gramática:

    [('mientras', 'mientras'), ('id', 'a'), ('num', '97')]