UNIVERSIDAD DE ANTIOQUIA

1803

Facultad de Ingeniería

Departamento de Ingeniería de Sistemas

PRÁCTICA 1 – TEORÍA DE LENGUAJES Y LABORATORIO

En el presente documento encontrará la explicación de los métodos que se desarrollaron para esta primera práctica

ANTONIO GONZALEZ

MATEO RIVERA

Contenido

Autómatas para reconocer palabras reservadas

Autómata para reconocer la palabra public

Símbolos de entrada = {p, u, b, l, i, c}

Estados = {

0: Estado inicial

P: Entró p estando en estado inicial

U: Entró u estando en estado P

B: Entró b estando en estado U

L: Entró l estando en estado B

I: Entró i estando en estado L

C: Entró c estando en estado I

1: Error

}

Estado de aceptación = C

Transiciones

	p	u	b	l	i	c	
0	P						0
P		U					0
U			B				0
B				L			0
L					I		0
L						C	0
C							1
1							0

Autómata para reconocer la palabra static

Símbolos de entrada = {s, t, a, i, c}

Estados = {

0: Estado inicial

S: Entró s estando en estado inicial

T1: Entró t estando en estado S

A: Entró a estando en estado T1

T2: Entró t estando en estado A

I: Entró i estando en estado T2

C: Entró c estando en estado I

1: Error

}

Estado de aceptación = C

Transiciones

	s	t	a	i	c	
0	S					0
S		T1				0
T1			A			0
A		T2				0
T2				I		0
I					C	0
C						1
1						0

Autómata para reconocer la palabra void

Símbolos de entrada = {v, o, i, d}

Estados = {

0: Estado inicial

V: Entró v estando en estado inicial

O: Entró o estando en estado V

I: Entró i estando en estado O

D: Entró d estando en estado I

1: Error

}

Estado de aceptación = D

Transiciones

	v	o	i	d	
0	V				0
V		O			0
O			I		0
I				D	0
D					1
1					0

Autómata para reconocer la palabra System.out.print

Símbolos de {'.', 'S', 'e', 'i', 'm', 'n', 'o', 'p', 'r', 's', 't', 'u', 'y'}

Estados = {

0: Estado inicial

S1: Entró s estando en estado inicial

Y: Entró y estando en estado S1

S2: Entró s estando en estado Y

T1: Entró t estando en estado S2

E: Entró e estando en estado T1

M: Entró m estando en estado E

P1: Entró . estando en estado M

O: Entró o estando en estado P1

U: Entró u estando en estado O

T2: Entró t estando en estado U

P2: Entró . estando en estado T2

P: Entró p estando en estado P2

R: Entró r estando en estado P

I: Entró i estando en estado R

N: Entró n estando en estado I

T3: Entró t estando en estado N

1: Error

}

Estado de aceptación = T3

Transiciones

	.	S	e	i	m	n	o	p	r	s	t	u	y	
0		S1												0
S1													Y	0
Y										S2				0
S2											T1			0
T1			E											0
E					M									0
M	P1													0
P1							O							0
O												U		0
U											T2			0
T2	P2													0
P2								P						0
P									R					0
R				I										0
I						N								0
N											T3			0
T3														1
1														0

Autómata para reconocer la palabra String

Símbolos de entrada = {S,t, r, i, n, g}

Estados = {

0: Estado inicial

S: Entró s estando en estado inicial

T: Entró t estando en estado S

R: Entró r estando en estado T

I: Entró i estando en estado R

N: Entró n estando en estado I

G: Entró g estando en estado n

1: Error

}

Estado de aceptación = G

Transiciones

	s	t	r	i	n	g	
0	S						0
S		T					0
T			R				0
R				I			0
I					N		0
N						G	0
G							1
1							0

Autómata para reconocer la palabra String[]

Símbolos de entrada = {S,t, r, i, n, g, [,]}

Estados = {

0: Estado inicial

S: Entró s estando en estado inicial

T: Entró t estando en estado S

R: Entró r estando en estado T

I: Entró i estando en estado R

N: Entró n estando en estado I

G: Entró g estando en estado n

C1: Entró [estando en G

C2: Entró] estando en C1

1: Error

}

Estado de aceptación = C2

Transiciones

	s	t	r	i	n	g	[	]	
0	S								0
S		T							0
T			R						0
R				I					0
I					N				0
N						G			0
G							C1		0
C1								C2	0
C2									1
1									0

Autómata para reconocer la palabra class

Símbolos de entrada = {c,l, a, s}

Estados = {

0: Estado inicial

C: Entró c estando en estado inicial

L: Entró l estando en estado C

A: Entró a estando en estado L

S1: Entró s estando en estado A

S2: Entró s estando en estado S1

1: Error

}

Estado de aceptación = S2

Transiciones

	c	l	a	s	
0	C				0
C		L			0
L			A		0
A				S1	0
S1				S2	0
S2					1
1					0

Autómata para reconocer la palabra else

Símbolos de entrada = {e,l, s}

Estados = {

0: Estado inicial

E1: Entró e estando en estado inicial

L: Entró l estando en estado E1

S: Entró s estando en estado L

E2: Entró e estando en estado S

1: Error

}

Estado de aceptación = E2

Transiciones

	e	l	s	
0	E1			0
E1		L		0
L			S	0
S	E2			0
E2				1
1				0

Autómata para reconocer la palabra char

Símbolos de entrada = {c,h, a, r}

Estados = {

0: Estado inicial

C: Entró c estando en estado inicial

H: Entró h estando en estado C

A: Entró a estando en estado H

R: Entró r estando en estado A

1: Error

}

Estado de aceptación = R

Transiciones

	c	h	a	r	
0	C				0
C		H			0
H			A		0
A				R	0
R					1
1					0

Autómata para reconocer la palabra int

Símbolos de entrada = {i,n, t}

Estados = {

0: Estado inicial

I: Entró i estando en estado inicial

N: Entró n estando en estado I

T: Entró t estando en estado N

1: Error

}

Estado de aceptación = T

Transiciones

	i	n	t	
0	I			0
I		N		0
N			T	0
T				1
1				0

Autómata para reconocer la palabra for

Símbolos de entrada = {f, o, r}

Estados = {

0: Estado inicial

F: Entró f estando en estado inicial

O: Entró o estando en estado F

R: Entró r estando en estado O

1: Error

}

Estado de aceptación = T

Transiciones

	f	o	r	
0	F			0
F		O		0
O			R	0
R				1
1				0

Autómata para reconocer la palabra if

Símbolos de entrada = {i, f}

Estados = {

0: Estado inicial

I: Entró i estando en estado inicial

F: Entró f estando en estado I

1: Error

}

Estado de aceptación = F

Transiciones

	i	f	
0	I		0
I		F	0
F			1
1			0

Autómata para reconocer OPERATOR

Símbolos de entrada = {'!', '%', '*', '+', '-', '/', '<', '>', '='}

Estados = {

0: Estado inicial

N: Negación

A1: Operación aritmética

A2: Asignación

C1: Comparador 1

C2: Comparador 2

1: Error

}

Estado de aceptación = {N, A1, A2, C1, C2}

Transiciones

	!	%	*	+	-	/	<	>	=	
0	N	A1	A1	A1	A1	A1	C1	C1	A2	0
N									C2	1
A1										1
A2									C2	1
C1									C2	1
C2										1
1										0

Autómata para reconocer numConst

Símbolos de entrada = {+, -, d, .}

$$d \in \{n | n \in \mathbb{N}_0 \wedge n \leq 9\}$$

Estados = {

0: Estado inicial

S+: Entró + estando en el Estado inicial o S-

S-: Entró - estando en el Estado inicial o S+

E: Entró dígito de la parte entera

P1: Entró . estando en estado 0, S+, S-

P2: Entró . estando en estado E

D: Entró dígito de la parte decimal

1: Error

}

Estado de aceptación = {E, P2, D}

Transiciones

	+	-	d	.	
0	S+	S-	E	P1	0
S+		S-	E	P1	0
S-	S+		E	P1	0
E			E	P2	1
P1			D		0
P2			D		1
D			D		1
1					0

Autómata para reconocer SEPARATOR

Símbolos de entrada = {(,), {, }, ;}

Estados = {

0: Estado inicial

S: Separador

1: Error

}

Estado de aceptación = R

Transiciones

	(	)	{	}	;	
0	S	S	S	S	S	0
S						1
1						0

Autómata para reconocer IDENTIFICADOR

Símbolos de entrada = {_, a, d}

$$d \in \{n | n \in \mathbb{N}_0 \wedge n \leq 9\}$$

$$a \in \{x | x \in abecedario\ inglés\ (Con\ minúsculas\ y\ mayúsculas)\}$$

Estados = {

0: Estado inicial

G: Entró _ estando en 0

A: Entró a estando en 0

B: Entró _, a, o d estando en A o G

1: Error

}

Estado de aceptación = A

Transiciones

	_	a	d	
0	G	A		0
G	A	A	A	0
A	A	A	A	1
1				0

Autómata para reconocer charConst

Símbolos de entrada = {“, a, \}

$$a \in \{x | x \in carácteres\ ASCII\} - \{",\backslash\}$$

Estados = {

0: Estado inicial

A: Entró “ estando en 0 o a estando en A

B: Entró último “

1: Error

}

Estado de aceptación = C

Transiciones

	“	a	\	
0	A			0
A	B	A		0
B				1
1				0

Tokens que reconoce

- KEYWORD:
  - public
  - class
  - static
  - void
  - int
  - String
  - char
  - System.out.print
  - for
  - if
  - else
- numConst:
  - Números reales
- charConst:
  - Cadenas de String (Todo lo que esté entre COMILLAS DOBLES, NO USAR COMILLAS SIMPLES)
- Separator:
  - (
  -)
  - {
  - }
  - ;
- Operator:
  - +
  - –
  - *
  - /
  - %
  - <
  - >
  - ==
  - !=
  - <=
  - >=
- Identificador:
  - Cualquier identificador que involucre: _, letras del abecedario inglés en mayúsculas y minúsculas, y números.

Clase Automata_Java

Atributos

Se tienen 20 atributos, de los cuales 17 están relacionados con un autómata, por ejemplo: self.public es el autómata correspondiente al autómata automata_public, self.numConst es el autómata correspondiente al autómata automata_numConst, y así sucesivamente; cada atributo relacionado con un autómata es una lista que guarda la siguiente información:

0. Estado del autómata correspondiente al atributo
1. Se especifica si el estado en el que se encuentra el autómata correspondiente al atributo es de RECHAZO o ACEPTACIÓN
2. Tipo de token que reconoce el autómata

Los demás 3 atributos corresponden a:

- **self.numbers** es la lista de dígitos del sistema numérico decimal convertidos en carácter, más específicamente: ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9'].
- **self.abecedario** es la lista de letras del abecedario inglés en mayúscula y minúscula.
- **self.A** es una lista que guarda las letras que pueden ir después de un '\', sin embargo, no se usó a lo largo de la implementación.

Métodos implementados

Constructor, __init__(self)

Este método permite inicializar los atributos, descritos anteriormente, de un objeto de esta clase; por defecto se establece que los autómatas estén en estado inicial al crearse un nuevo objeto, se estandarizó que el estado inicial de todos los autómatas se llame $\mathbf{0}$, y el estado de error de todos los autómatas se llame $\mathbf{1}$.

automata_Java(self, c)

Este método se encarga de pasarle a todos los autómatas el carácter $\boldsymbol{c}$ que se está analizando en ese instante.

estado_actual(self)

Es el encargado de decir si de lo que se lleva analizado de una cadena hasta un momento dado, hay al menos un autómata que esté en aceptación, en caso de estarlo, retorna que está en *'ACEPTACIÓN'* y el tipo de token que representa lo que se lleva de una cadena.

Para evitar confusiones entre un token de tipo *'KEYWORD'* y un token de tipo 'IDENTIFICADOR' se le dio preferencia al tipo *'KEYWORD'*, este es el único caso de que una cadena tenga más de un autómata en estado de *'ACEPTACIÓN'*.

caracter_invalido(self,c)

Este método no se usó a lo largo de la implementación, pero inicialmente su trabajo se pensó para decir si el carácter $\boldsymbol{c}$ que se está analizando, provoca que todos los autómatas entren en estado de error, es decir, retorna **True** si todos los autómatas entran en estado de error, y **False** si hay al menos un autómata que queda en estado de aceptación.

automata_public(self,c), automata_static, …, automata_charConst(self,c)

Estos métodos son la implementación de cada autómata descrito en las páginas anteriores, simplemente cambian el estado de los atributos, y si queda en *'RECHAZO'* o *'ACEPTACIÓN'*, dependiendo del valor del carácter c.

Funciones implementadas

tokenizador(secuencia) y splitter

Esta función recibe una secuencia como parámetro, es decir, todo lo que el usuario ingresa en el editor de texto.

Crea dos listas, una donde se guardan los tokens acumulados (tokensAcumulados), y otra donde se van almacenando los tokens de cada iteración (tokensIteracion)

Para comprender qué significa tokensIteracion, explicaremos el funcionamiento holístico de la función tokenizador:

1. Se crean dos listas vacías (tokensAcumulados y tokensIteracion)
2. La secuencia ingresada por el usuario se procesa por una función muy parecida al método Split de un String; lo que hace es dividir la secuencia por espacios, y eliminar los espacios, pero los espacios que estén entre comillas dobles debe mantenerlos. Por lo que si la secuencia ingresada fue: '`String cadenaText="Hola Mundo";`', la función splitter lo convierte en una lista así: `['String', 'cadenaText="Hola Mundo";']`. Por lo que cada iteración corresponde al análisis de cada cadena de la lista que retorna splitter; por lo tanto, en la primera iteración, `tokensAcumulados = [('String', 'KEYWORD')]` y en la segunda iteración `tokensAcumulados=[('cadenaText', 'IDENTIFICADOR'), ('=', 'OPERADOR'), ('"Hola mundo", 'charConst'), (';', 'SEPARATOR')]`, y es tokensAcumulados quien se encarga de guardar todos los tokens de la secuencia ingresada.
3. Se crean dos autómatas que van reconociendo los caracteres ingresados, solo que uno de ellos está 'adelantado en el tiempo', reconocedor2; esto se da con la intención de que cuando este entre en 'RECHAZO' para todos los autómatas después de estar en 'ACEPTACIÓN' analice qué token reconoció, hay unos casos particulares que se controlaron.

En esta función se tuvieron las siguientes consideraciones:

Se define como el 'antes' de un token como el token con el que formaban un mismo String después haber aplicado la función `splitter`, por ejemplo para la secuencia: `class PruebaSintaxis{public static void main(){int a=15/16;System.out.print("HOLA MUNDO")}`, tenemos que el antes de 15 es un operador, pero el identificador `PruebaSintaxis` no tiene, el antes de `System.out.print` es un 'SEPARADOR', y así sucesivamente.

- Para numConst: Antes de un número puede ir un operador, un espacio, '('.
- Para charConst: Antes puede ir un '(', '='.
- Para SEPARATOR:

    - Antes de ‘(‘ puede ir:
        - System.out.print
        - for
        - if
        - operator
    - Antes de ‘)’ puede ir:
        - (
        - numConst
        - charConst
        - identificador
    - Antes de ‘{‘ puede ir:
        -)
        - ;
        - identificador
    - Antes de ‘}’ puede ir:
        - ;
        - {
    - Antes de ‘;’ puede ir:
        -)
        - identificador
        - charConst
        - numConst
- Para OPERATOR: Antes de un operador puede ir un número, identificador, y para casos particulares como ‘!’, antes puede ir un ‘!’ o ‘(‘.
- Para identificador: Antes de un identificador puede ir un operator, ‘(‘
- Para KEYWORD: Antes de toda KEYWORD, menos `static`, `void` y `String[]` puede ir: ‘;’, ‘{‘ o ‘}’, para las demás no puede ir nada antes.