Trabajo práctico: Lexer

Materia: Sintaxis y Semántica de los Lenguajes

- **Integrantes:** Franco Agustín Valli, Felipe Roberto Miklikowski, Amadeo Pujalte, Nazareno José Rodriguez Moyano.

Año: 2025

Consigna

El objetivo de este trabajo práctico es diseñar e implementar un Analizador Lexicográfico (Lexer) para el lenguaje de programación TINY.
El analizador lexicográfico se implementará en el lenguaje de programación Python, mediante un programa principal que utilice los autómatas finitos deterministas (AFD) que se construirán para cada uno de los tokens del lenguaje de programación TINY.
El software que resulte de la implementación del lexer, deberá aceptar como entrada una cadena que representa código escrito en el lenguaje TINY, y deberá convertir este código, interpretado como una cadena de caracteres ASCII o UTF-8, a una lista de tokens correspondiente a la gramática provista, donde cada token estará representado por un par (tipo token, lexema), donde tipo token representa, para cada subcadena del código de entrada, el tipo de token correspondiente, y lexema es el valor asociado al token que generó dicha clasificación.

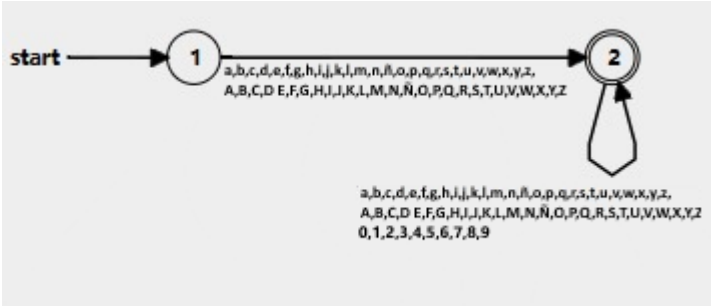
Introducción

El lenguaje regular que acepte el Analizador Lexicográfico será la unión de todas los lenguajes que generen las expresiones regulares de los tokens. No forma parte del alcance del trabajo revisar si la cadena de caracteres cumplen con la sintaxis de algún lenguaje de programación, sino se limita a identificar que cadenas pertenecen al lenguaje Tiny, además de identificar el token correspondiente a cada una.

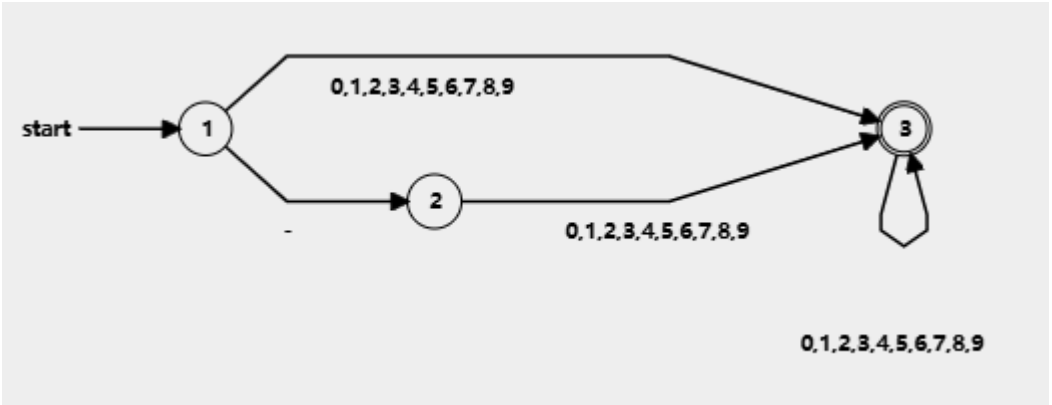
Expresiones regulares y autómatas finitos

A cada token se le fue asignada una expresión regular que acepte todas las cadenas de dicho token. A continuación se muestra un listado de los tokens con su expresión regular y automatas finitos deterministicos correspondientes[1].

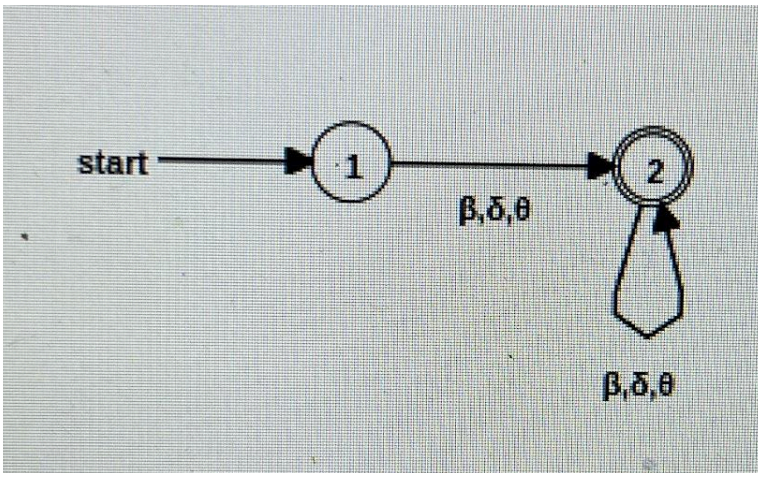
- **Id**
  ER_{id}=*(a|b|c|d|e|f|g|h|i|j|k|l|m|n|o|p|q|r|s|t|u|v|w|x|y|z|A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|Q|R|S|T|U|V|W|X|Y|Z)(a|b|c|d|e|f|g|h|i|j|k|l|m|n|o|p|q|r|s|t|u|v|w|x|y|z|A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|Q|R|S|T|U|V|W|X|Y|Z|0|1|2|3|4|5|6|7|8|9)+*



- **num** ER_{num}=*(-|ϵ)(0|1|2|3|4|5|6|7|8|9)+*

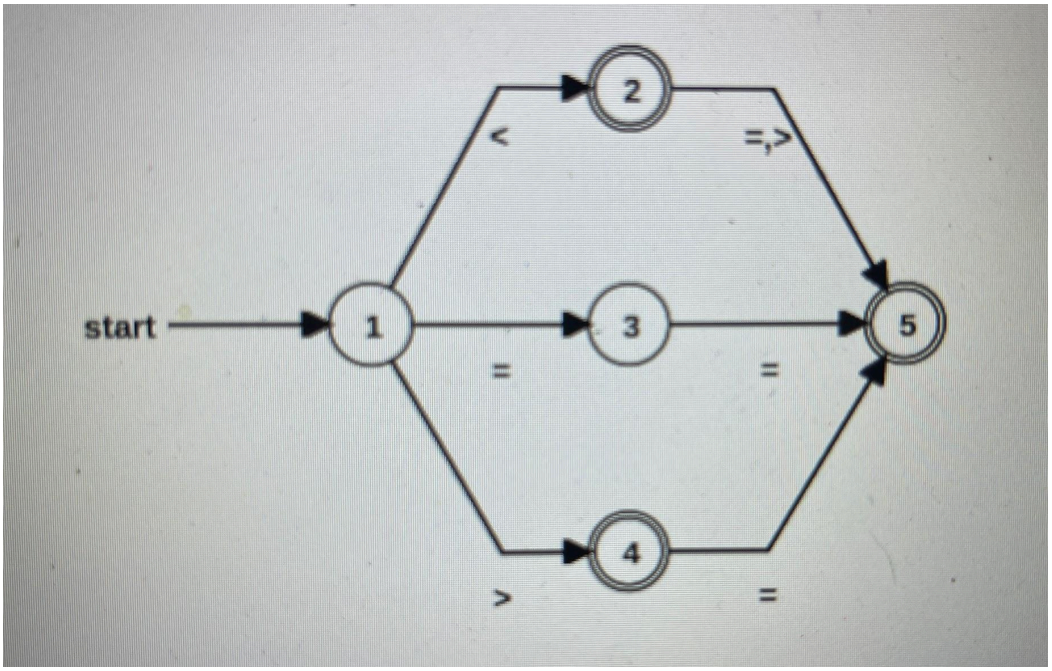


- **espacioBlanco** ER_B=*(β|δ|θ)+*
  donde β es un espacio ‘ ‘, δ es nueva línea \n, θ es tabulado \t



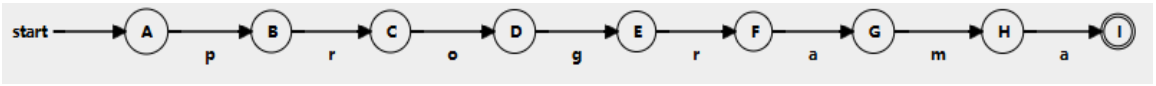
[1] Los nombres de los tokens se representan en **negrita,** la expresión regular en *italics,* y se adjunta una imágen de la transición de estados del autómata.

- **operador relacional** ER_{oprel} >|<|>=|<=|==|<>

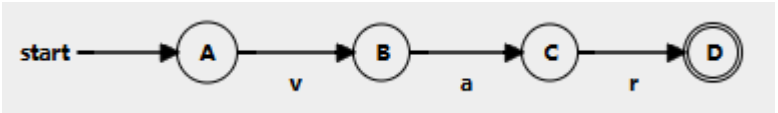


- **asignación** $ER_{asignación}$= =

- **programa** $ER_{programa}$= *programa*

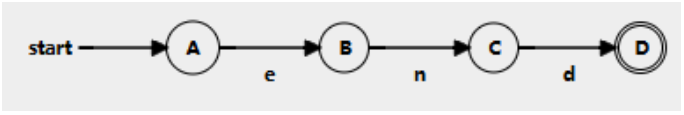


- **var** ER_{var}= *var*

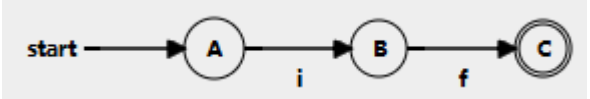


- **begin** ER_{begin}= *begin*

- **end** ER_{end}= *end*

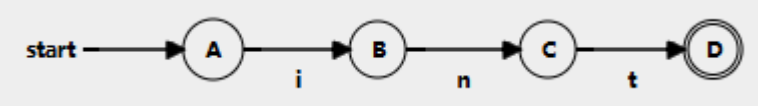


- **if** ER_{if}= *if*

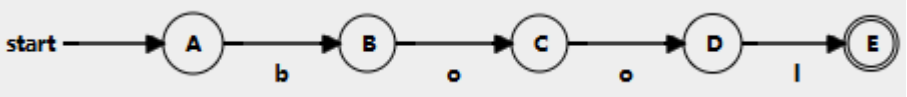


- **else** ER_{else}= *else*

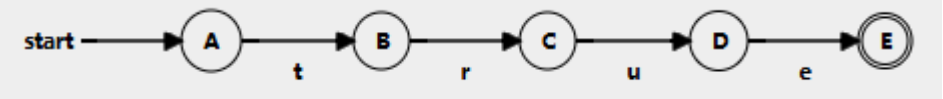
- **int** ER_{int}= *int*



- **bool** ER_{bool}= *bool*

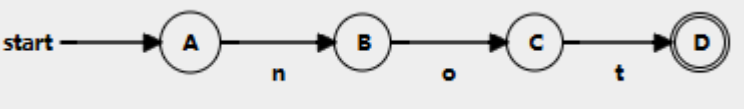


- **true** ER_{true}= *true*

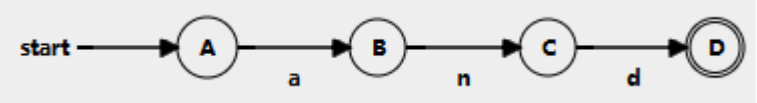


- **false** ER_{false}= *false*

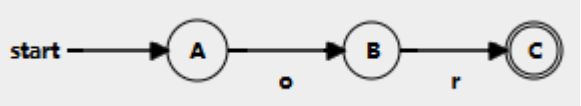
- **not** ER_{not}= *not*



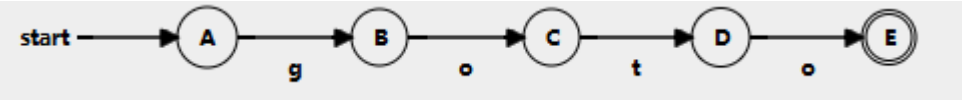
- **and** ER_{and}= *and*



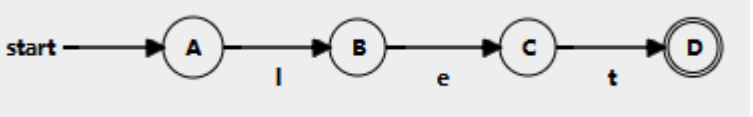
- **or** ER_{or}= *or*



- **goto**ER_{goto}= *goto*



- **let** ER_{let}= *let*

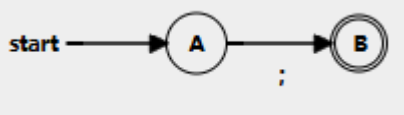


- **punto** ER_{punto}= .

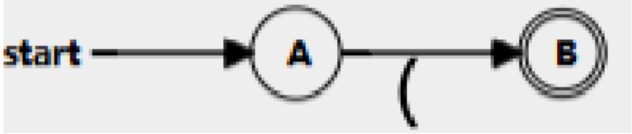
- **coma** ER_{coma}= ,

- **dosPuntos** $ER_{dosPuntos}$= :

- **puntoComa** $ER_{puntoComa}$= ;

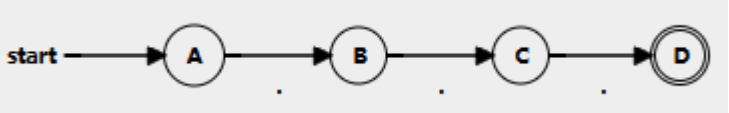


- **parentesisIzquierdo** $ER_{parentesisIzquierdo}$= (

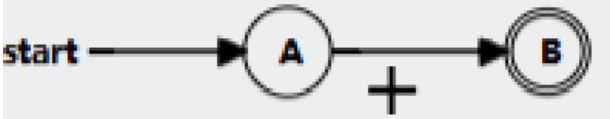


- **parentesisDerecho** $ER_{parentesisDerecho}$=)

- **triplePunto** $ER_{triplePunto}$= ...



- **mas** ER_{mas}= +



- **guion** ER_{guion}= -

- **asterisco** $ER_{asterisco}$= *