


Compiladores e Intérpretes

Análisis Léxico – Tokens – Especificación y Reconocedor

Universidad Nacional del Sur
Departamento de Ciencias e Ingeniería de la Computación
2024



Previa

- Los contenidos de **esta presentación** asumen que ya **miraste y tenes frescos** los Temas:
 - **2.1** Analizador Léxico – Esquema general
- Si no los miraste aún o no te acordás bien tal vez te sea difícil seguir algunas parte de esta presentación.



Tokens

- Las palabras validas de un lenguaje son llamas **Tokens**
- Los **Tokens** caracterizan subcadenas con un **rol** dentro del lenguaje de programación
- **Ejemplos** de Tokens:
 - Enteros, Strings, Identificadores, palabra clave IF, Igual, etc..

Cada **Token** esta caracterizado por un **patrón**

la subcadena que machea con ese patrón es denominado **Lexema**

Tokens

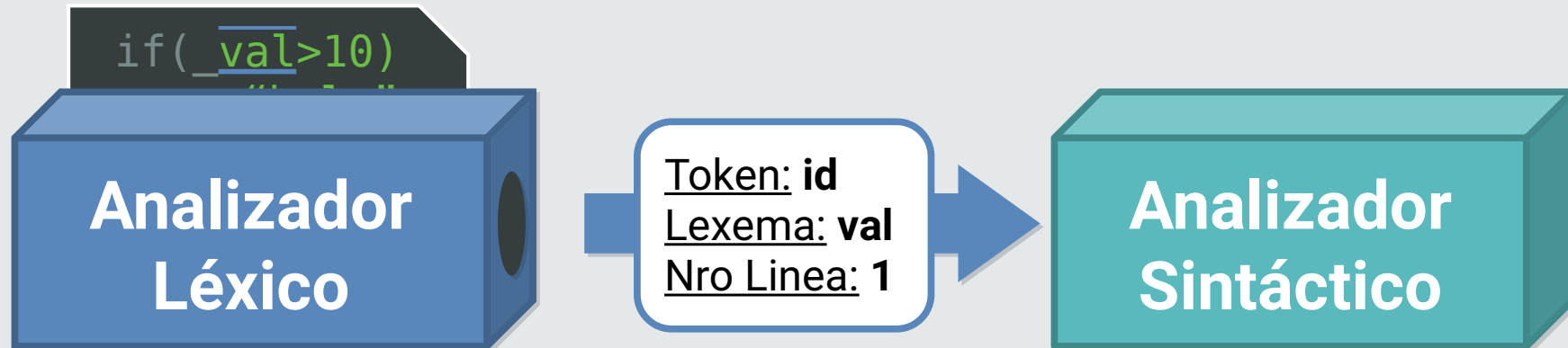
- Algunos **tokens** identifican **conjuntos** de subcadenas
- Por ejemplo:
 - **Token Identificador:** subcadena que empiezan con una letra y sigue con una secuencia de letras y/o dígitos
 - Ejemplos de Lexemas: `id1` `var` `myvar` `amigo` `a1244`
 - **Token String:** subcadena que comienza con “, sigue con una secuencia de caracteres, y termina con ”
 - Ejemplos de Lexemas: `“hola”` `“me#@!!&”` `“id1”`

Tokens

- Otros **Tokens** son identificados por una única subcadena
- Por ejemplo:
 - **Token Igual:** subcadena con el caracter =
 - **Token Mayor o Igual:** subcadena con los caracteres >=
 - **Token Palabra Clave IF:** subcadena con los caracteres if
 - **Token Palabra Clave ELSE:** subcadena con los caracteres else

Tokens

- La **salida** del analizador léxico ante un pedido del analizador sintáctico es un **token**.
- El **token retornado** está acompañado por el **lexema** caracterizado por ese token y el nro de linea donde esta en el programa fuente.
 - El **lexema** y el **nro de linea** son **atributos** del token retornado!



Tokens

- Los espacios, tabs, enters (blancos) y comentarios **NO son tokens!**
- Como los **blancos** y **comentarios** no son tokens, **no son retornados** por el analizador léxico!

Especificando Tokens – Expresiones Regulares

Especificando Tokens – Expresiones Regulares

- Para **implementar** el **analizador léxico** de un lenguaje de programación tenemos que **identificar** sus **tokens**
- Para esto, para cada token, tenemos que determinar el **patrón** que caracteriza a sus **lexemas**, es decir:

Dar una **especificación para los tokens** del lenguaje

Especificando Tokens – Expresiones Regulares

- El formalismo usual para esta tarea son los **Lenguajes Regulares**
 - ¿Por qué estos lenguajes?

Especificando Tokens – Expresiones Regulares

- El formalismo usual para esta tarea son los **Lenguajes Regulares**
 - ¿Por qué estos lenguajes?

Son lenguajes formales más **simples** y **eficientes** con el **poder expresivo** suficiente para identificar los lexemas que caracterizan a un token.

Repasemos un poco sobre los lenguajes regulares y las expresiones regulares

Especificando Tokens – Expresiones Regulares

- Los lenguajes regulares se notan con **expresiones regulares (ER)**:
- **Formalmente** dado un alfabeto una expresión regular r puede ser:
 - Un caracter c de
 - la cadena vacía ϵ
 - Dadas dos expresiones regulares $r1$ y $r2$ en se puede construir como:
 - $r1r2$ (secuencia)
 - $r1|r2$ (opción)
 - $r1^*$ (clausura: 0 o mas repeticiones de $r1$)
 - $r1^+$ (clausura positiva: 1 o mas repeticiones de $r1$)
 - Se puede usar paréntesis para indicar agrupamiento o precedencia



Especificando Tokens – Expresiones Regulares



Trata de hacerlos **antes de pasar** a la próxima slide!

- ¿Cómo es la expresión regular para un entero?
- ¿Como sería la expresión regular para un identificador?
- ¿Como serían para mayor y para mayor e igual?

Por simplicidad usamos `[0..9]`, `[a..z]` y `[A..Z]` para caracterizar dígitos, letras minúsculas y letras mayúsculas respectivamente.



Especificando Tokens – Expresiones Regulares

- ¿Cómo es la expresión regular para un entero?

Entero = $[0..9]^+$

- ¿Como sería la expresión regular para un identificador?

Identificador = $([a..z] | [A..Z])([a..z] | [A..Z] | [0..9])^*$

- ¿Como serían para mayor y para mayor o igual?

Mayor = $>$
MayorIgual = $>=$

Por simplicidad usamos $[0..9]$, $[a..z]$ y $[A..Z]$ para caracterizar dígitos, letras minúsculas y letras mayúsculas respectivamente.

Especificando Tokens – Expresiones Regulares

- ¿Cómo es la expresión regular para un entero?

Entero = `Digito+`

Digito = `[0..9]`

Letra = `[a..z] | [A..Z]`

- ¿Como sería la expresión regular para un identificador?

Identificador = `Letra(Letra|Digito)*`

Otra forma mas conveniente de notarlos usando las ER para Dígito y Letra

- ¿Como serían para mayor y para mayor o

Mayor = `>`
MayorIgual = `>=`

Por simplicidad usamos `[0..9]`, `[a..z]` y `[A..Z]` para caracterizar dígitos, letras minúsculas y letras mayúsculas respectivamente.

Reconociendo Tokens – Autómatas Finitos


Reconociendo Tokens – Autómatas Finitos

- Los **Autómatas Finitos (AF)** reconocen cadenas de los lenguajes regulares
- Un autómata finito esta formalmente caracterizado por:
 - Alfabeto Σ
 - Conjunto de estados E
 - Función de transición de estados $T: E \times \Sigma \rightarrow E$
 - Estado inicial I
 - Conjunto de estados finalizadores $F \subseteq E$



Estado 

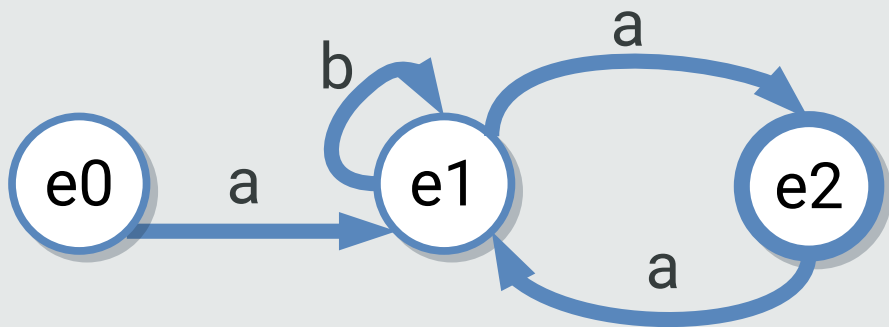
Estado Inicial 

Estado Aceptador 

Transición 

Reconociendo Tokens – Autómatas Finitos

- El AF va **aplicando transiciones** a medida que **consume** los **caracteres** de la cadena de entrada
- Partiendo del estado inicial, si el autómata puede consumir toda la cadena y queda en un estado finalizador **acepta la cadena** sino **rechaza la cadena**



<u>abba</u>	✓
<u>aabb</u>	✗
<u>abbaa</u>	✗

Reconociendo Tokens – Autómatas Finitos



Trata de hacerlos **antes de pasar** a la próxima slide!

- Dada una **ER** podemos construir un **AF** que reconoce el lenguaje generado por esa expresión
- ¿Cómo sería el autómata para reconocer X? donde X es:

Operador Mayor

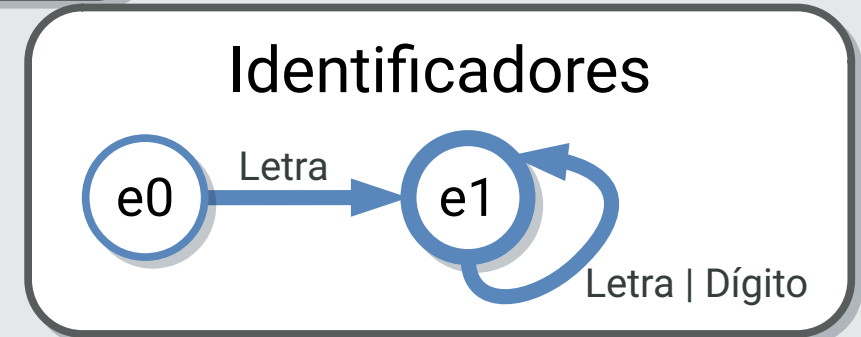
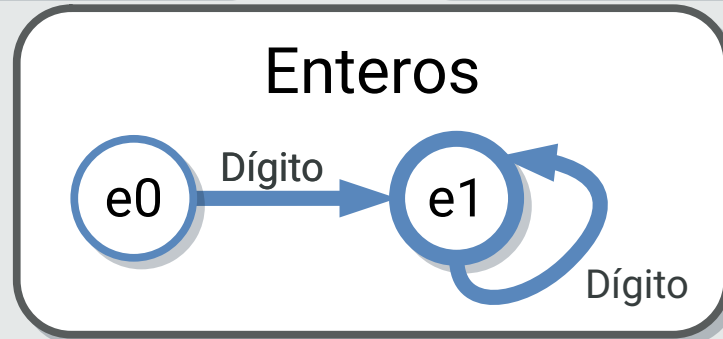
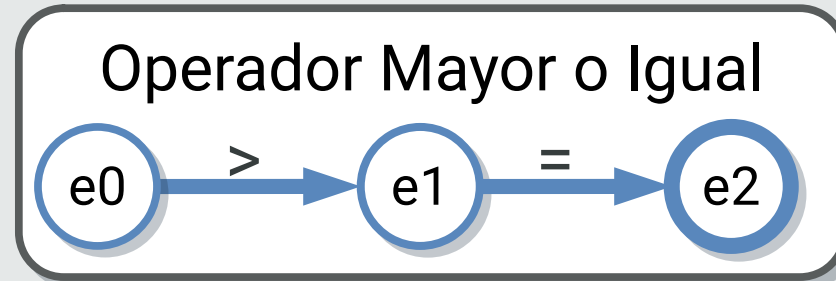
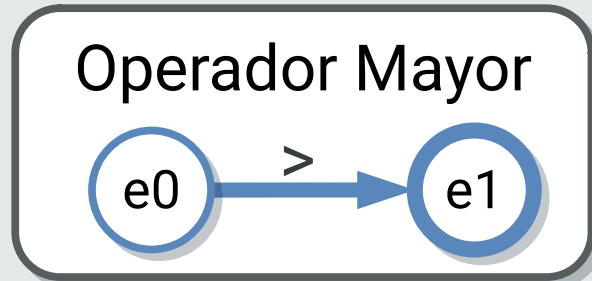
Operador Mayor o Igual

Enteros

Identificadores

Reconociendo Tokens – Autómatas Finitos

- Dada una **ER** podemos construir un **AF** que reconoce el lenguaje generado por esa expresión
- ¿Cómo sería el autómata para reconocer X? donde X es:



Como seguimos?

- Usando las **ERs** podemos **especificar** como son los **Tokens** de nuestro lenguaje
- A partir de esas **ERs** podemos armar los **AFs** capaces de **reconocer** los **Tokens** (particularmente sus lexemas)
- ¿Cómo usamos esos **AFs** para **implementar** el **Analizador Lexico**?

