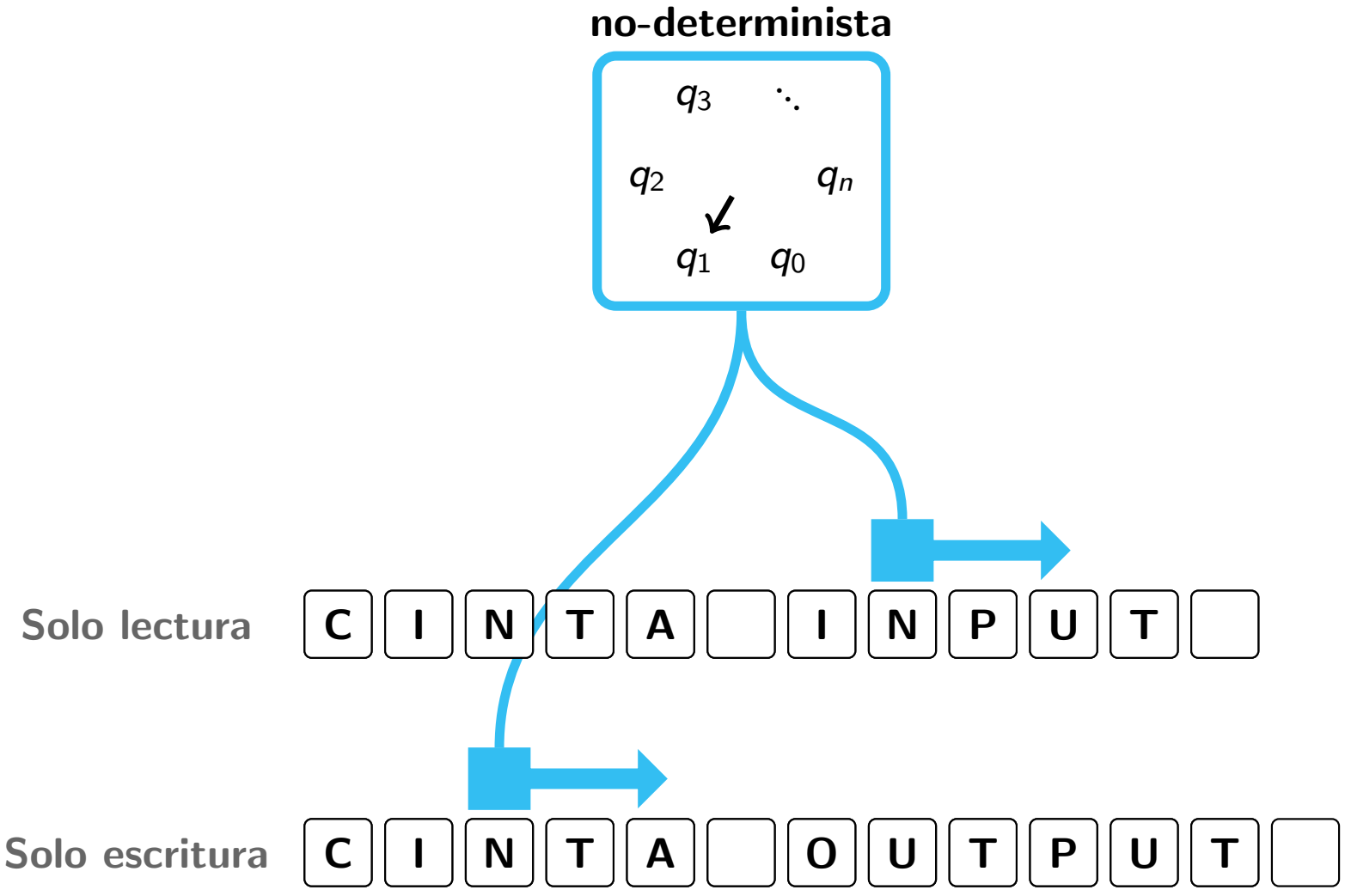
Análisis léxico

Clase 13

IIC 2223

Prof. Cristian Riveros

Transductores



Definición de transductor

Definición

Un transductor (*en inglés*, transducer) es una tupla:

$$\mathcal{T} \quad = \quad (Q, \Sigma, \Omega, \Delta, I, F)$$

- Q es un conjunto finito de estados.
- Σ es el alfabeto de input.
- Ω es el alfabeto de **output**.
- $\Delta \subseteq Q \times (\Sigma \cup \{\epsilon\}) \times (\Omega \cup \{\epsilon\}) \times Q$ es la **relación de transición**.
- $I \subseteq Q$ es un conjunto de estados iniciales.
- $F \subseteq Q$ es el conjunto de estados finales.

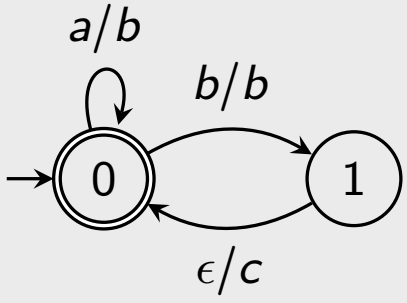
Transductores para funciones de palabras a palabras

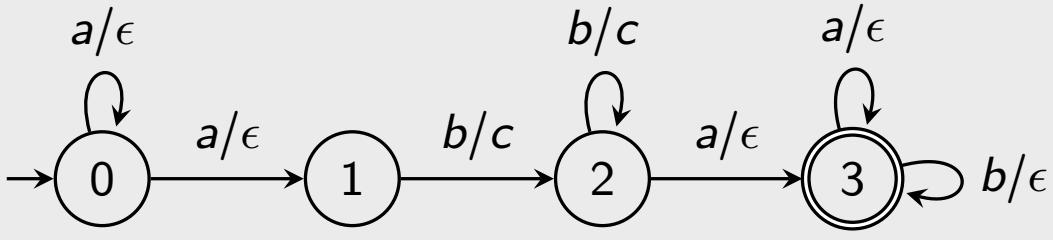
Definición

Decimos que un transductor $\mathcal{T}$ define una función (parcial) si:

$$\text{para todo } u \in \Sigma^* \text{ se tiene que } |[\![\mathcal{T}]\!](u)| \leq 1.$$

Ejemplos

- 

- 

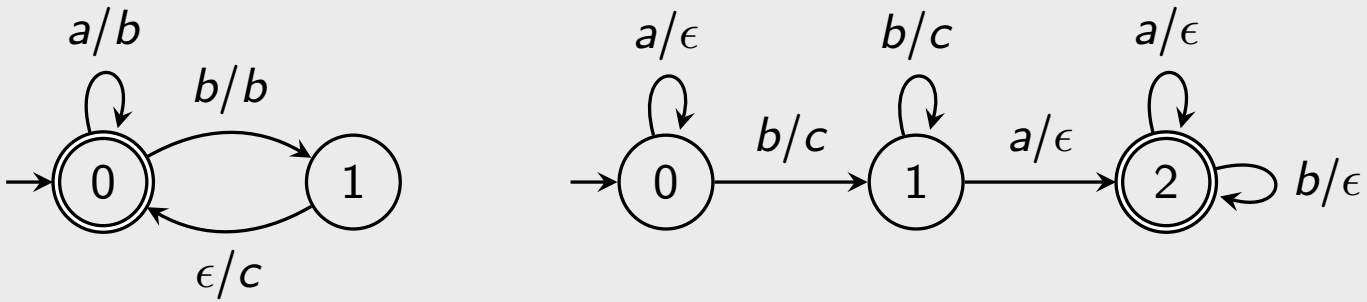
Transductores para funciones de palabras a palabras

Definición

Decimos que $\mathcal{T} = (Q, \Sigma, \Omega, \Delta, I, F)$ es **determinista** si cumple que:

1. $\mathcal{T}$ define una función $[\![\mathcal{T}]\!] : \Sigma^* \to \Omega^*$.
2. para todo $(p, a_1, b_1, q_1) \in \Delta$ y $(p, a_2, b_2, q_2) \in \Delta$,
   si $a_1 = a_2$, entonces $b_1 = b_2$ y $q_1 = q_2$.
3. si $(p, \epsilon, b, q) \in \Delta$, entonces
   para todo $(p, a', b', q') \in \Delta$, se tiene que $(a', b', q') = (\epsilon, b, q)$.

Ejemplos



Transductores para funciones de palabras a palabras

Definición

Decimos que $\mathcal{T} = (Q, \Sigma, \Omega, \Delta, I, F)$ es **determinista** si cumple que:

1. $\mathcal{T}$ define una función $[\![\mathcal{T}]\!] : \Sigma^* \to \Omega^*$.
2. para todo $(p, a_1, b_1, q_1) \in \Delta$ y $(p, a_2, b_2, q_2) \in \Delta$,
   si $a_1 = a_2$, entonces $b_1 = b_2$ y $q_1 = q_2$.
3. si $(p, \epsilon, b, q) \in \Delta$, entonces
   para todo $(p, a', b', q') \in \Delta$, se tiene que $(a', b', q') = (\epsilon, b, q)$.

¿es verdad que si $\mathcal{T}$ define una función parcial, entonces $\mathcal{T}$ es deterministas?

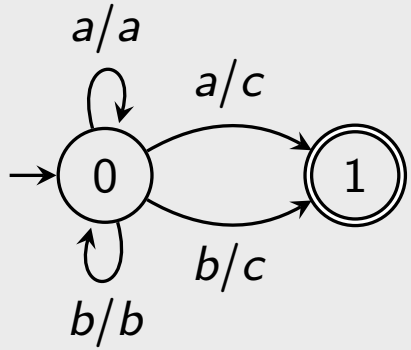
Transductores para funciones de palabras a palabras

Definición

Decimos que $\mathcal{T} = (Q, \Sigma, \Omega, \Delta, I, F)$ es **determinista** si cumple que:

1. $\mathcal{T}$ define una función $[\![\mathcal{T}]\!] : \Sigma^* \to \Omega^*$.
2. para todo $(p, a_1, b_1, q_1) \in \Delta$ y $(p, a_2, b_2, q_2) \in \Delta$,
   si $a_1 = a_2$, entonces $b_1 = b_2$ y $q_1 = q_2$.
3. si $(p, \epsilon, b, q) \in \Delta$, entonces
   para todo $(p, a', b', q') \in \Delta$, se tiene que $(a', b', q') = (\epsilon, b, q)$.

Contraejemplo



¿cuál es la ventaja de los transductores deterministas?

Outline

Outline

Sintaxis y semántica de un lenguaje de programación

Definición

1. La **sintaxis** de una lenguaje es un conjunto de reglas que describen los programas válidos que tienen significado.

¿cuáles son programas válidos en Python?

- ```
  myint = 7
  print myint
  ```

- ```
  mystring = 'hello"
  print(mystring)
  ```

Sintaxis y semántica de un lenguaje de programación

Definición

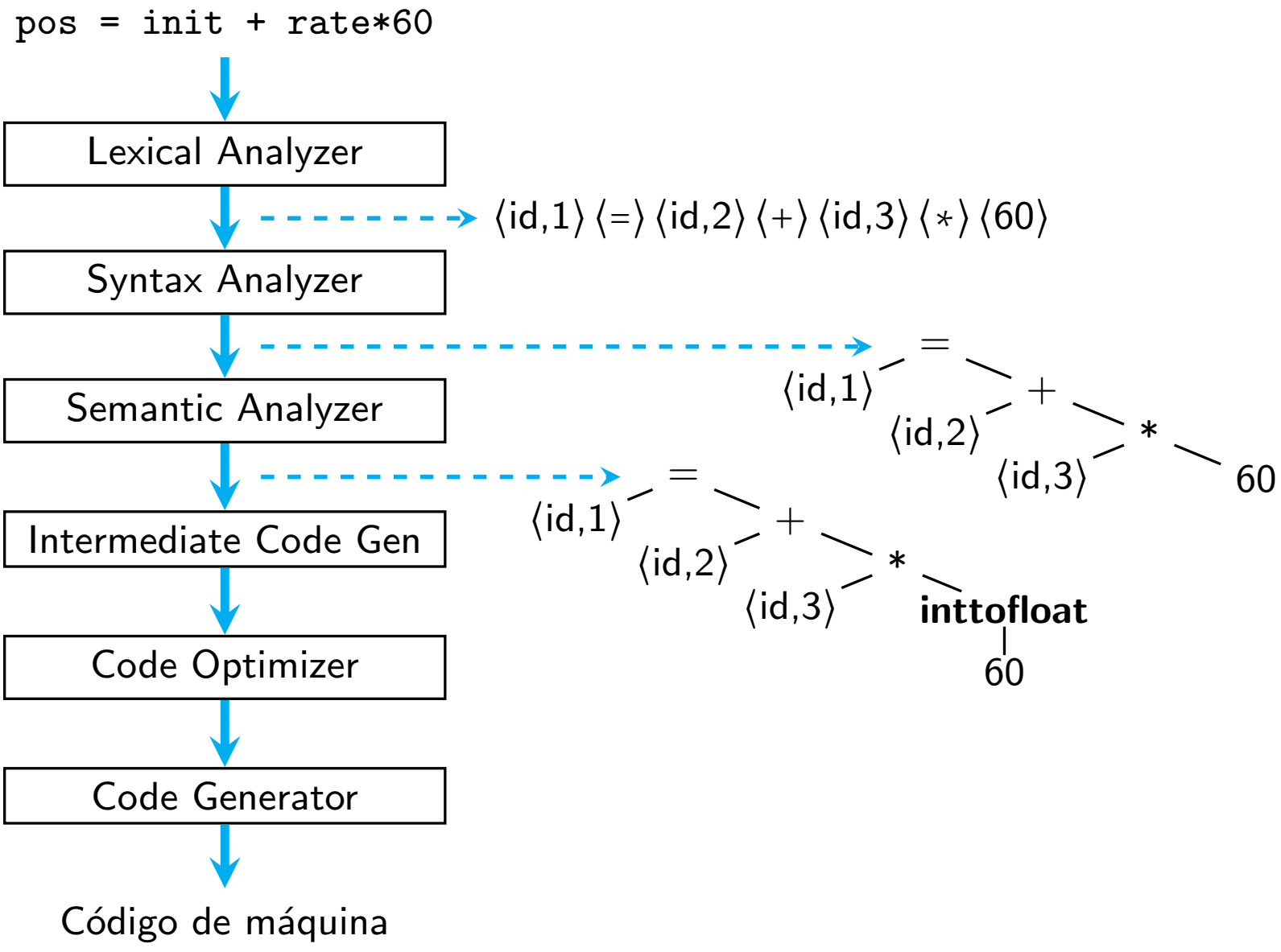
1. La **sintaxis** de una lenguaje es un conjunto de reglas que describen los programas válidos que tienen significado.

2. La **semántica** de un lenguaje define el significado de un programa correcto según la sintaxis.

¿cuál es la semántica de este programa en Python?

```
mylist = []
mylist.append(1)
mylist.append(2)
for x in mylist:
    print(x)
```

La estructura de un compilador



Verificación de sintaxis

En este proceso se busca:

- verificar la sintaxis de un programa.
- entregar la estructura de un programa (árbol de parsing).

Consta de tres etapas:

1. Análisis léxico (**Lexer**).
2. Análisis sintáctico (**Parser**).
3. Análisis semántico.

Por ahora, solo nos interesará el **Lexer**.

(el funcionamiento del **Parser** lo veremos cuando veamos gramáticas)

Análisis léxico (Lexer)

- El análisis léxico consta en dividir el programa en una sec. de **tokens**.
- Un **token** (o lexema) es un substring (válido) dentro de un programa.
- Un **token** esta compuesto por:
  - tipo.
  - valor (el valor mismo del substring).

Análisis léxico (Lexer)

Tipos usuales de **tokens** en lenguajes de programación:

- **number** (constante): 2, 345, 495, ...
- **string** (constante): 'hello', 'iloveTDA', ...
- **keywords**: `if`, `for`, ...
- **identificadores**: pos, init, rate ...
- **delimitadores**: '{', '}', '(', ')', ',', ...
- **operadores**: '=', '+', '<', '<=', ...

Análisis léxico (Lexer)

Ejemplo

```
pos = init + rate * 60
```

Tipo	Valor
id	`pos`
EQ	`=`
id	`init`
PLUS	`+`
id	`rate`
MULT	`*`
number	60

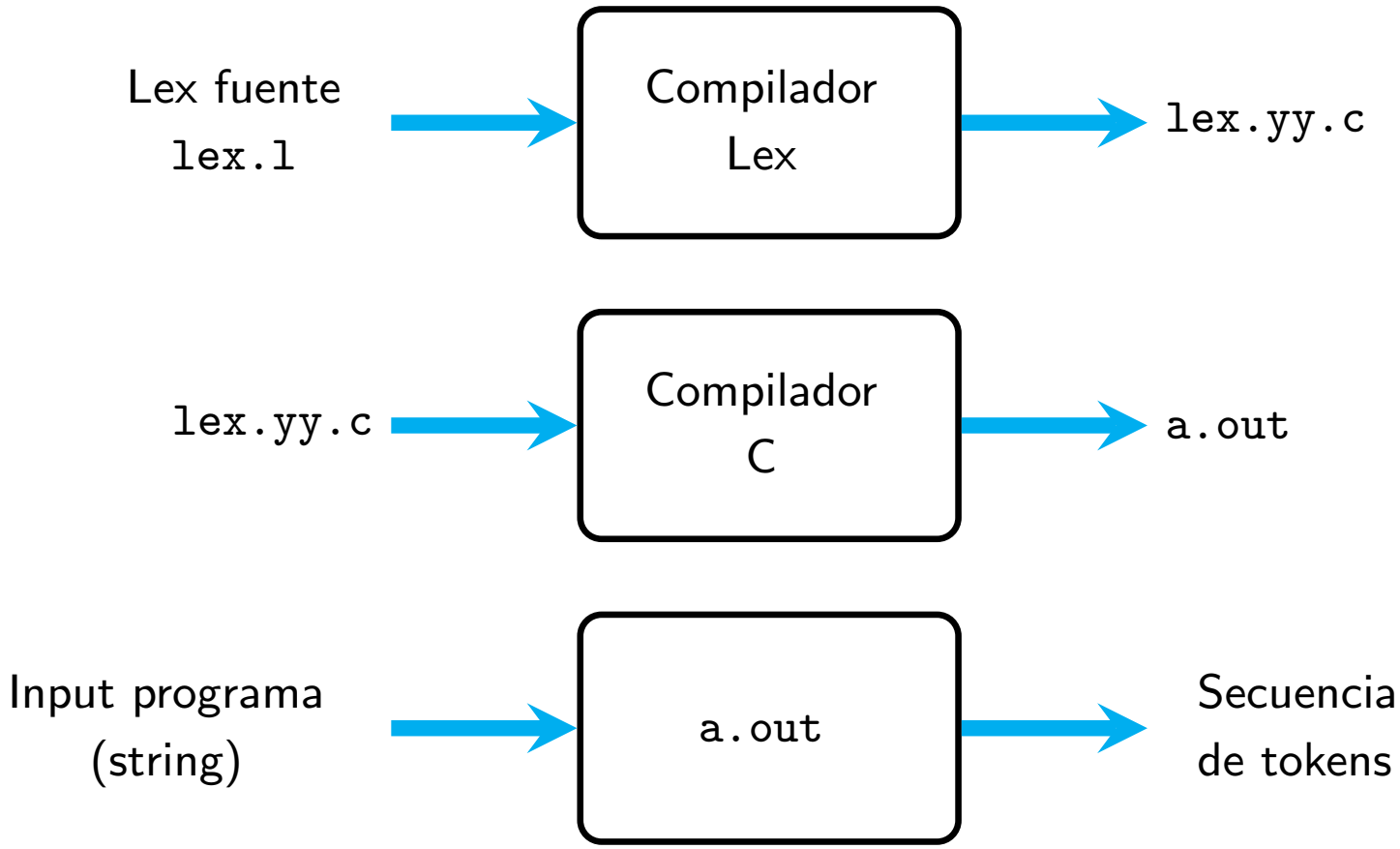
Outline

Generador de análisis léxico (Lex)

- Un **generador de análisis léxico** es un software que, a través de un programa fuente, crea el código necesario para hacer el análisis léxico.
- `Lex` es el analizador léxico estándar en Unix en lenguaje `C`.
- Versión moderna es Flex.
  - Para Java existe JFlex.
  - Para Python existe PLY.

En esta clase revisaremos `Lex` (como ejemplo práctico).

Generador de análisis léxico (Lex)



Generador de análisis léxico (Lex)

El **formato de un programa** en Lex es de la forma:

```
declaraciones
%%
reglas de traducción
%%
funciones auxiliares
```

Generador de análisis léxico (Lex)

El **formato de un programa** en Lex es de la forma:

```
declaraciones
%%
reglas de traducción
%%
funciones auxiliares
```

Ejemplo: declaraciones

```
%{
#include "misconstantes.h" \* def de IF, ELSE, ID, NUMBER *\
%}
delim   [ \t\n]
ws      {delim}+
%%

...
```

Generador de análisis léxico (Lex)

El **formato de un programa** en Lex es de la forma:

```
declaraciones
%%
reglas de traducción
%%
funciones auxiliares
```

Ejemplo: funciones auxiliares

```
...
%%
void printID(){printf("Id:  %s\n",yytext);}
void printNumer(){printf("Number:  %s\n",yytext);}
```

Lex permite usar **variables especiales** para extraer un lexema:

- `yytext`: contiene un puntero al string del token encontrado.

Generador de análisis léxico (Lex)

El **formato de un programa** en Lex es de la forma:

```
declaraciones
%%
reglas de traducción
%%
funciones auxiliares
```

Las **reglas de traducción** tienen la siguiente forma:

```
Patrón { Acción }
```

- **Patrón** esta definido por una **expresión regular**.
- **Acción** es código `C` embebido.

Generador de análisis léxico (Lex)

Las **reglas de traducción** tienen la siguiente forma:

```
Patrón { Acción }
```

- **Patrón** esta definido por una **expresión regular**.
- **Acción** es código C embebido.

Ejemplo: reglas de traducción

```
[ \t\n]+                  {\* sin accion *\}
if                        {return(IF);}
else                      {return(ELSE);}
[A-Za-z]([A-Za-z0-9])*    {printID(); return(ID);}
[0-9]+                    {printNumber(); return(NUMBER);}
```

Generador de análisis léxico (Lex)

Ejemplo completo de `lex.l`

```
%{
#include "misconstantes.h" \* def de IF, ELSE, ID, NUMBER *\
%}
delim                    [ \t\n]
ws                       {delim}+
%%
{ws}                     {\* sin accion *\}
if                       {return(IF);}
else                     {return(ELSE);}
[A-Za-z]([A-Za-z0-9])*   {printID(); return(ID);}
[0-9]+                   {printNumber(); return(NUMBER);}
%%
void printID(){printf("Id:  %s\n",yytext);}
void printNumer(){printf("Number:  %s\n",yytext);}
```

Resolución de conflictos en `Lex`

Si **varios prefijos** del input satisfacen uno o más patrones:

1. Se prefiere el **prefijo más largo** por sobre el prefijo más corto.

2. Si el prefijo más corto satisface uno o más patrones,
   se prefiere **el patrón listado primero** en el programa `lex.l`.

Outline

¿cómo evaluamos los patrones en `lex.l`?

Las **reglas de traducción** tienen la siguiente forma:

```
Patrón { Acción }
```

- **Patrón** esta definido por una **expresión regular**.
- **Acción** es código `C` embebido.

Sea $P_1, \ldots, P_k$ los **patrones** y
$C_1, \ldots, C_k$ las **acciones** en el programa "`lex.l`", respectivamente.

Primer paso

Para cada patrón P_i construimos un NFA $\mathcal{A}_i = (Q_i, \Sigma, \Delta_i, \{q_0^i\}, \{q_f^i\})$
con un solo estado inicial q_0^i y y un solo estado final q_f^i.

¿cómo **evaluamos** los autómatas $\mathcal{A}_1, \ldots, \mathcal{A}_k$ en paralelo,
encontrando todos los tokens del input?

¿cómo evaluamos los patrones en paralelo?

El **análisis léxico** tiene algunas complicaciones adicionales
que van **más allá de esta clase**.

Supondremos las siguientes **simplificaciones**:

1. Cada lexema esta separado por un símbolo de espacio "␣".
2. Documento termina con un símbolo especial `EOF`.
3. No hay conflictos entre patrones.

Ejemplo: conflictos

```
if                          {return(IF);}
else                        {return(ELSE);}
[A-Za-z]([A-Za-z0-9])*      {printID(); return(ID);}
```

¿cómo evaluamos los patrones en paralelo?

- $\mathcal{A}_i = (Q_i, \Sigma, \Delta_i, \{q_0^i\}, \{q_f^i\})$ el NFA para el **patrón** P_i y
- C_i la **acción** de P_i con $i \leq k$.

Evaluamos con el **transductor determinista**:

$$\mathcal{T} = (Q, \Sigma, \{C_i\}_{i \leq k}, \Delta, \{q_0\}, F)$$

- $Q = 2^{\left\{\bigcup_{i=1}^{k} Q_i\right\}}$
- $q_0 = \{q_0^1, q_0^2, \ldots, q_0^k\}$
- $(S, a, \epsilon, S') \in \Delta$ si, y solo si, $S' = \{q \mid \exists i.\, \exists p \in S.\, (p, a, q) \in \Delta_i\}$.
- $(S, \llcorner, C_i, q_0), (S, \texttt{EOF}, C_i, q_0) \in \Delta$ si, y solo si, $q_f^i \in S$
- $F = \{q_0\}$

El análisis léxico corresponde a **ejecutar un transductor**.

Lexer on-the-fly (simplificado)

Sea $\mathcal{A}_i = (Q_i, \Sigma, \Delta_i, \{q_0^i\}, \{q_f^i\})$ el NFA para el **patrón** P_i y $w = a_1 \dots a_n$.

Function `lexer-onthefly` $(\mathcal{A}_1, \dots, \mathcal{A}_k, w)$

- $S := \{q_0^1, q_0^2, \dots, q_0^k\}$
- **for** $j = 1$ **to** n **do**
  - **if** $a_j \neq \sqcup$ **and** $a_j \neq \mathit{EOF}$ **then**
    - $S_{\text{old}} := S$
    - $S := \varnothing$
    - **foreach** $i = 1$ **to** k **do**
      - **foreach** $p \in S_{old} \cap Q_i$ **do**
        - $S := S \cup \{q \mid (p, a_i, q) \in \Delta_i\}$
  - $\dots$

Lexer on-the-fly (simplificado)

Sea $\mathcal{A}_i = (Q_i, \Sigma, \Delta_i, \{q_0^i\}, \{q_f^i\})$ el NFA para el **patrón** P_i y $w = a_1 \dots a_n$.

Function `lexer-onthefly` $(\mathcal{A}_1, \dots, \mathcal{A}_k, w)$

- $S := \{q_0^1, q_0^2, \dots, q_0^k\}$
- **for** $j = 1$ **to** n **do**
  - ...
  - **else if** $a_j = \sqcup$ **or** $a_j = \mathit{EOF}$ **then**
    - **if** $q_f^1 \in S$ **then**
      - **execute** C_1
    - ...
    - **else if** $q_f^k \in S$ **then**
      - **execute** C_k
    - **else**
      - **ERROR**
    - $S := \{q_0^1, q_0^2, \dots, q_0^k\}$
- **return**

Sobre análisis léxico

Tiempo análisis léxico

Si $|P_i|$ es el tamaño del patrón P_i:

$$\mathcal{O}((|P_1| + \ldots + |P_k|) \cdot |w|)$$

Algunas conclusiones/observaciones

1. El análisis léxico es equivalente
   ha evaluar un **transductor** que simula los patrones en **paralelo**.
2. El análisis léxico también
   maneja **conflictos entre reglas** y otros detalles.

Estos **detalles adicionales** son cubiertos
en el curso de **IIC2323 - Construcción de Compiladores**.