Transductores y análisis léxico

Clase 12

IIC 2223

Prof. Cristian Riveros

¿cuánto se parece un autómata a un algoritmo?

¿cuáles son las diferencias?

1. Memoria. ✗
2. "Movimiento" de la máquina. ✓
3. Output. ?

En esta clase, veremos como extender autómatas con 3.

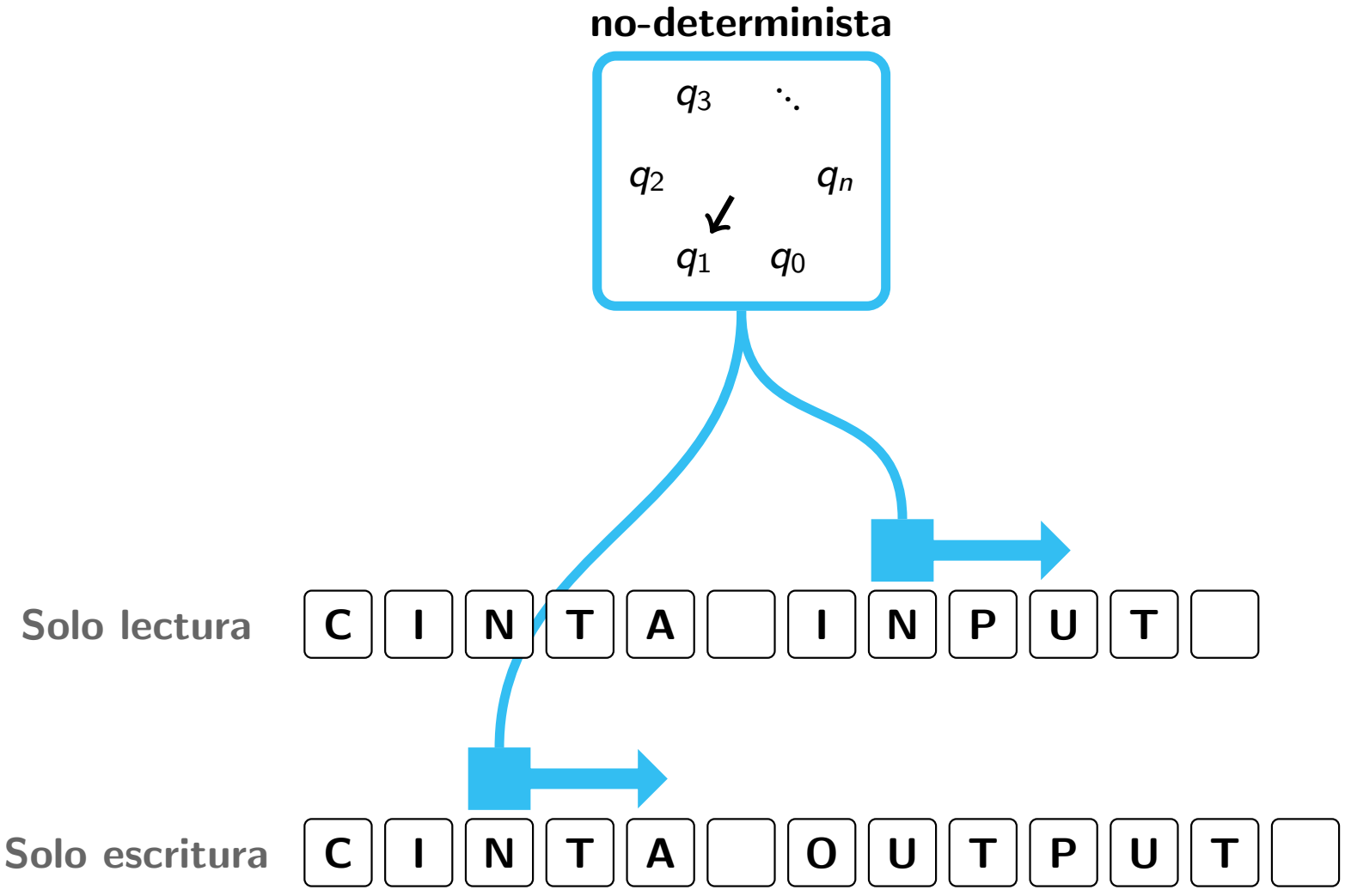
Outline

Transductores

Aplicación: Análisis léxico

Outline

Transductores



Definición de transductor

Definición

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

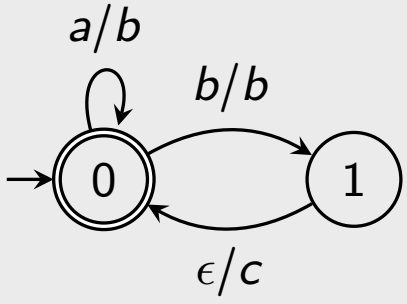
$$\mathcal{T} \quad = \quad (Q, \Sigma, \Omega, \Delta, I, F)$$

- 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.

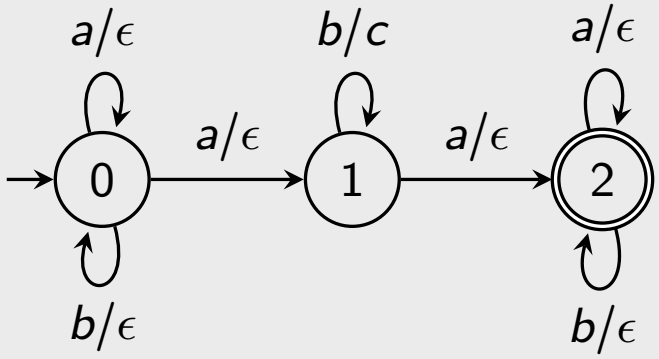
Definición de transductor

Algunos ejemplos

1.



2.



Configuración de un transductor

Sea $\mathcal{T} = (Q, \Sigma, \Omega, \Delta, I, F)$ un transductor.

Definiciones

- Un par $(q, u, v) \in Q \times \Sigma^* \times \Omega^*$ es una **configuración** de $\mathcal{T}$.
- Una configuración (q, u, ϵ) es **inicial** si $q \in I$.
- Una configuración (q, ϵ, v) es **final** si $q \in F$.

"Intuitivamente, una configuración (q, au, vb) representa que $\mathcal{T}$ se encuentra en el estado q procesando la palabra au y leyendo a, y hasta ahora grabó la palabra vb y el último símbolo impreso es b."

Ejecución de un transductor

Sea $\mathcal{T} = (Q, \Sigma, \Omega, \Delta, I, F)$ un transductor.

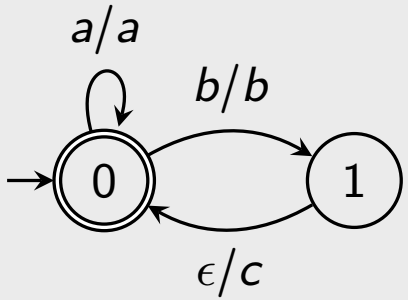
Definición

Se define la relación $\vdash_{\mathcal{T}}$ de **siguiente-paso** entre configuraciones de $\mathcal{T}$:

$$(p, u_1, v_1) \vdash_{\mathcal{T}} (q, u_2, v_2)$$

si, y solo si, existe $(p, a, b, q) \in \Delta$ tal que $u_1 = a \cdot u_2$ y $v_2 = v_1 \cdot b$.

Ejemplo



$(0, aba, \epsilon) \vdash_{\mathcal{T}} (0, ba, a)$

$(0, ba, a) \vdash_{\mathcal{T}} (1, a, ab) \vdash_{\mathcal{T}} (0, a, abc) \vdash_{\mathcal{T}} (0, \epsilon, abca)$

Ejecución de un transductor

Sea $\mathcal{T} = (Q, \Sigma, \Omega, \Delta, I, F)$ un transductor.

Definición

Se define la relación $\vdash_{\mathcal{T}}$ de **siguiente-paso** entre configuraciones de $\mathcal{T}$:

$$(p, u_1, v_1) \vdash_{\mathcal{T}} (q, u_2, v_2)$$

si, y solo si, existe $(p, a, b, q) \in \Delta$ tal que $u_1 = a \cdot u_2$ y $v_2 = v_1 \cdot b$.

$$\vdash_{\mathcal{T}} \subseteq (Q \times \Sigma^* \times \Omega^*) \times (Q \times \Sigma^* \times \Omega^*).$$

Se define $\vdash_{\mathcal{T}}^*$ como la clausura **refleja** y **transitiva** de $\vdash_{\mathcal{T}}$:

para toda configuración (q, u, v) : $\quad (q, u, v) \vdash_{\mathcal{T}}^* (q, u, v)$

si $(q_1, u_1, v_1) \vdash_{\mathcal{T}}^* (q_2, u_2, v_2)$ y
$(q_2, u_2, v_2) \vdash_{\mathcal{T}} (q_3, u_3, v_3)$: $\quad (q_1, u_1, v_1) \vdash_{\mathcal{T}}^* (q_3, u_3, v_3)$

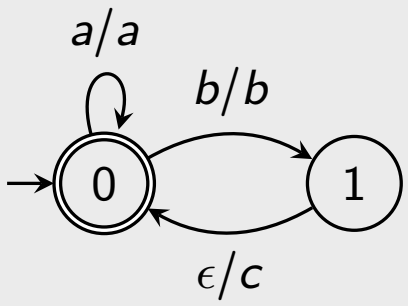
Ejecución de un transductor

Se define $\vdash_{\mathcal{T}}^{*}$ como la clausura **refleja** y **transitiva** de $\vdash_{\mathcal{T}}$:

para toda configuración (q, u, v) : $\quad (q, u, v) \vdash_{\mathcal{T}}^{*} (q, u, v)$

si $(q_1, u_1, v_1) \vdash_{\mathcal{T}}^{*} (q_2, u_2, v_2)$ y
$(q_2, u_2, v_2) \vdash_{\mathcal{T}} (q_3, u_3, v_3)$: $\quad (q_1, u_1, v_1) \vdash_{\mathcal{T}}^{*} (q_3, u_3, v_3)$

Ejemplo



$(0, aba, \epsilon) \vdash_{\mathcal{T}} (0, ba, a)$

$(0, ba, a) \vdash_{\mathcal{T}} (1, a, ab) \vdash_{\mathcal{T}} (0, a, abc) \vdash_{\mathcal{T}} (0, \epsilon, abca)$

$(0, aba, \epsilon) \vdash_{\mathcal{T}}^{*} (0, ba, a)$ y $(0, aba, \epsilon) \vdash_{\mathcal{T}}^{*} (0, \epsilon, abca)$

Función definida por un transductor

Sea $\mathcal{T} = (Q, \Sigma, \Omega, \Delta, I, F)$ un transductor y $u, v \in \Sigma^*$.

Definiciones

- $\mathcal{T}$ **entrega** v con **input** u si existe una configuración **inicial** (q_0, u, ϵ) y una configuración **final** (q_f, ϵ, v) tal que:

$$(q_0, u, \epsilon) \vdash_{\mathcal{T}}^{*} (q_f, \epsilon, v)$$

- Se define la función $[\![\mathcal{T}]\!] : \Sigma^* \to 2^{\Omega^*}$:

$$[\![\mathcal{T}]\!](u) = \{v \in \Omega^* \mid \mathcal{T} \text{ entrega } v \text{ con input } u\}$$

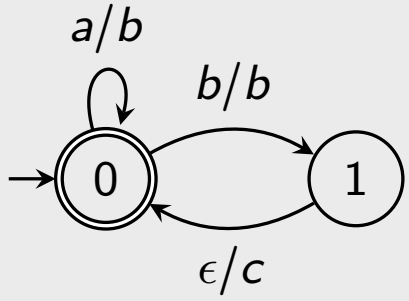
- Se dice que $f : \Sigma^* \to 2^{\Omega^*}$ es una **función racional** si existe un transductor $\mathcal{T}$ tal que $f = [\![\mathcal{T}]\!]$.

Un transductor define una función de palabras a conjunto de palabras.

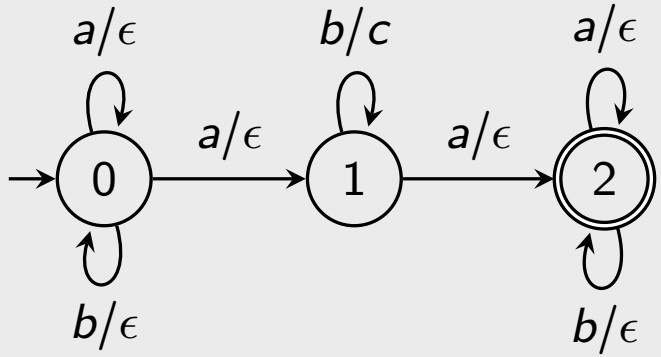
Función definida por un transductor

¿qué función define cada transductor?

1.



2.



Funciones versus relaciones

Dos interpretaciones para un transductor

1. $\mathcal{T}$ define la **función** $[\![\mathcal{T}]\!] : \Sigma^* \to 2^{\Omega^*}$:

$$[\![\mathcal{T}]\!](u) \;=\; \{v \in \Omega^* \mid \mathcal{T} \text{ entrega } v \text{ con input } u\}$$

2. $\mathcal{T}$ define la **relación** $[\![\mathcal{T}]\!] \subseteq \Sigma^* \times \Omega^*$:

$$(u, v) \in [\![\mathcal{T}]\!] \quad \text{si, y solo si,} \quad \mathcal{T} \text{ entrega } v \text{ con input } u$$

Desde ahora, hablaremos de función o relación **indistintamente** y hablaremos de las **relaciones racionales** (definidas por un transductor).

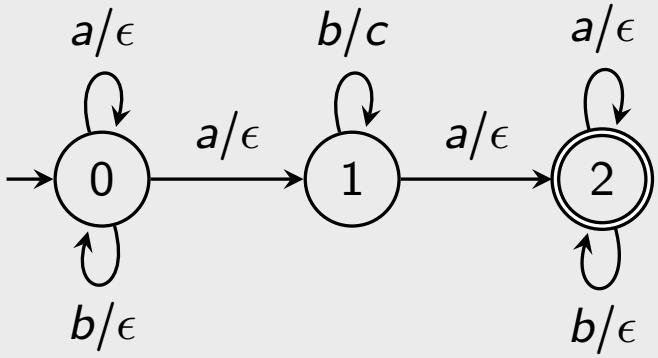
Lenguaje de input y lenguaje de output

Definiciones

Para una relación $R \subseteq \Sigma^* \times \Omega^*$ se define:

- $\pi_1(R) = \{ u \in \Sigma^* \mid \exists v \in \Omega^*.\ (u, v) \in R \}$.
- $\pi_2(R) = \{ v \in \Omega^* \mid \exists u \in \Sigma^*.\ (u, v) \in R \}$.

¿cuál es el lenguaje definido por $\pi_1([\![\mathcal{T}]\!])$ y $\pi_2([\![\mathcal{T}]\!])$?



Lenguaje de input y lenguaje de output

Definiciones

Para una relación $R \subseteq \Sigma^* \times \Omega^*$ se define:

- $\pi_1(R) = \{ u \in \Sigma^* \mid \exists v \in \Omega^*.\ (u, v) \in R \}$.
- $\pi_2(R) = \{ v \in \Omega^* \mid \exists u \in \Sigma^*.\ (u, v) \in R \}$.

Teorema

Si $\mathcal{T}$ es un transductor,
entonces $\pi_1([\![\mathcal{T}]\!])$ y $\pi_2([\![\mathcal{T}]\!])$ son lenguajes regulares sobre Σ y Ω, resp.

Demostración: $\pi_1([\![\mathcal{T}]\!])$

Para $\mathcal{T} = (Q, \Sigma, \Omega, \Delta, I, F)$, defina $\mathcal{A}_1 = (Q, \Sigma, \Delta_1, I, F)$ tal que:

$$(p, a, q) \in \Delta_1 \quad \text{si, y solo si,} \quad \exists b \in \Omega \cup \{\epsilon\}.\ (p, a, b, q) \in \Delta$$

y demuestre que $\mathcal{L}(\mathcal{A}_1) = \pi_1([\![\mathcal{T}]\!])$. □

Operaciones de relaciones

Teorema

Sea $\mathcal{T}_1$ y $\mathcal{T}_2$ dos transductores con Σ y Ω alfabetos de input y output.

Las siguientes son **relaciones racionales**.

1. $[\![\mathcal{T}_1]\!] \cup [\![\mathcal{T}_2]\!] = \{(u, v) \in \Sigma^* \times \Omega^* \mid (u, v) \in [\![\mathcal{T}_1]\!] \vee (u, v) \in [\![\mathcal{T}_2]\!]\}$.
2. $[\![\mathcal{T}_1]\!] \cdot [\![\mathcal{T}_2]\!] = \{(u_1 u_2, v_1 v_2) \in \Sigma^* \times \Omega^* \mid (u_1, v_1) \in [\![\mathcal{T}_1]\!] \wedge (u_2, v_2) \in [\![\mathcal{T}_2]\!]\}$.
3. $[\![\mathcal{T}_1]\!]^* = \bigcup_{k=0}^{\infty} [\![\mathcal{T}_1]\!]^k$.

Demostración.

Operaciones de relaciones

Teorema

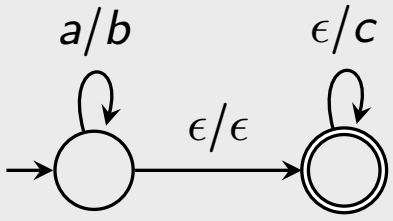
Existen transductores $\mathcal{T}_1$ y $\mathcal{T}_2$ con Σ y Ω alfabetos de input y output, tq:

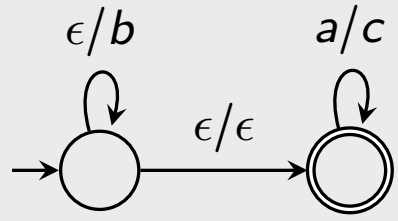
$$[\![\mathcal{T}_1]\!] \cap [\![\mathcal{T}_2]\!] \;=\; \{(u,v) \in \Sigma^* \times \Omega^* \mid (u,v) \in [\![\mathcal{T}_1]\!] \wedge (u,v) \in [\![\mathcal{T}_2]\!]\}$$

NO es una relación racional.

Demostración

Considere los siguientes transductores:





$[\![\mathcal{T}_1]\!] = \{(a^n, b^n c^m) \mid n \geq 0, m \geq 0\}$ $\quad$ $[\![\mathcal{T}_2]\!] = \{(a^n, b^m c^n) \mid n \geq 0, m \geq 0\}$

pero $[\![\mathcal{T}_1]\!] \cap [\![\mathcal{T}_2]\!] = \{(a^n, b^n c^n) \mid n \geq 0\}$,
y por lo tanto $[\![\mathcal{T}_1]\!] \cap [\![\mathcal{T}_2]\!]$ no es racional (¿por qué?) $\square$

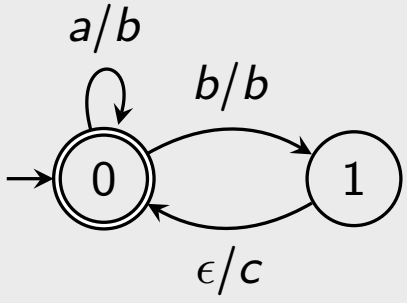
Transductores para funciones de palabras a palabras

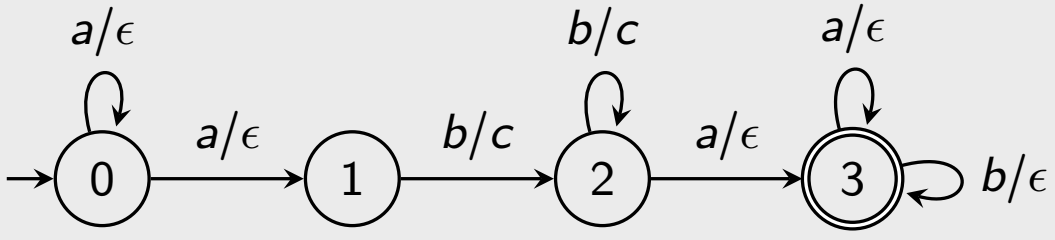
Definición

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

- 

- 

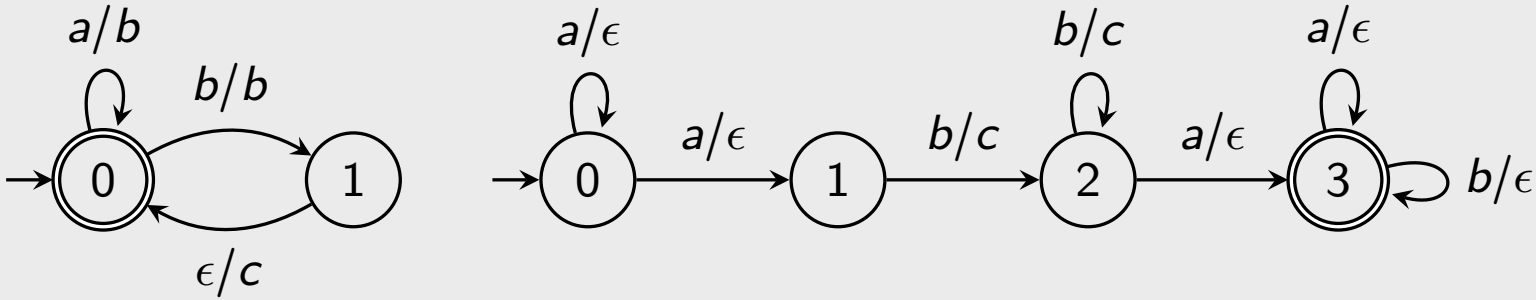
Transductores para funciones de palabras a palabras

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
   $\Delta \cap \big(\{p\} \times (\Sigma \cup \{\epsilon\}) \times (\Omega \cup \{\epsilon\}) \times Q\big) = \{(p, \epsilon, b, q)\}$.

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
   $\Delta \cap \big(\{p\} \times (\Sigma \cup \{\epsilon\}) \times (\Omega \cup \{\epsilon\}) \times Q\big) = \{(p, \epsilon, b, q)\}$.

¿son todas las funciones definidas por transductores deterministas?

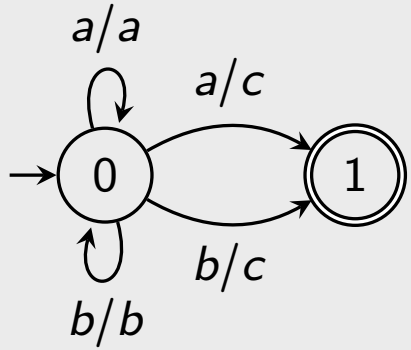
Transductores para funciones de palabras a palabras

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
   $\Delta \cap \big(\{p\} \times (\Sigma \cup \{\epsilon\}) \times (\Omega \cup \{\epsilon\}) \times Q\big) = \{(p, \epsilon, b, q)\}$.

Contraejemplo



¿cuál es la ventaja de los transductores deterministas?

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

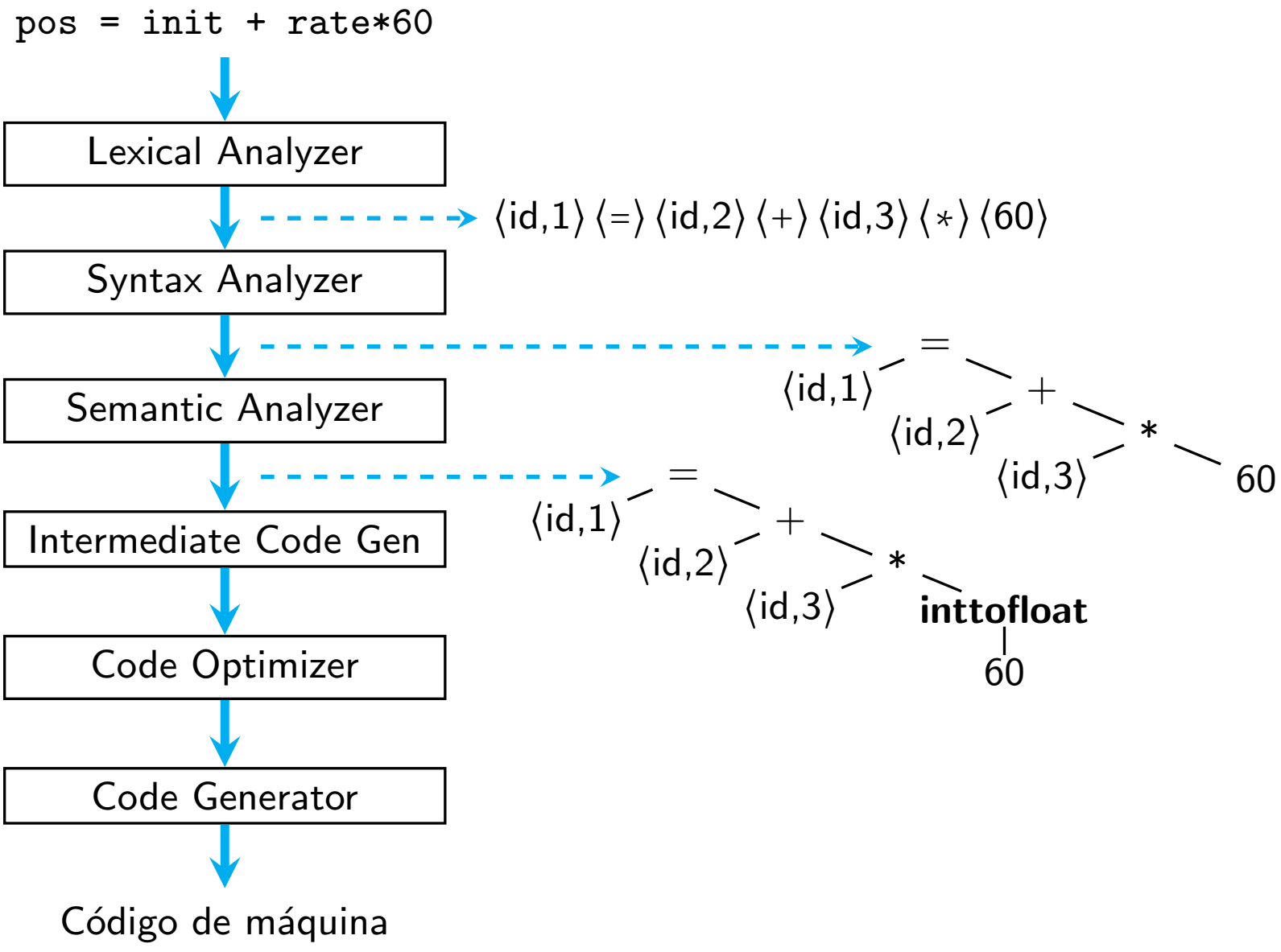
1. La **sintaxis** de una lenguaje es un conjunto de reglas que describen los programas válidos que tienen significado.

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
```

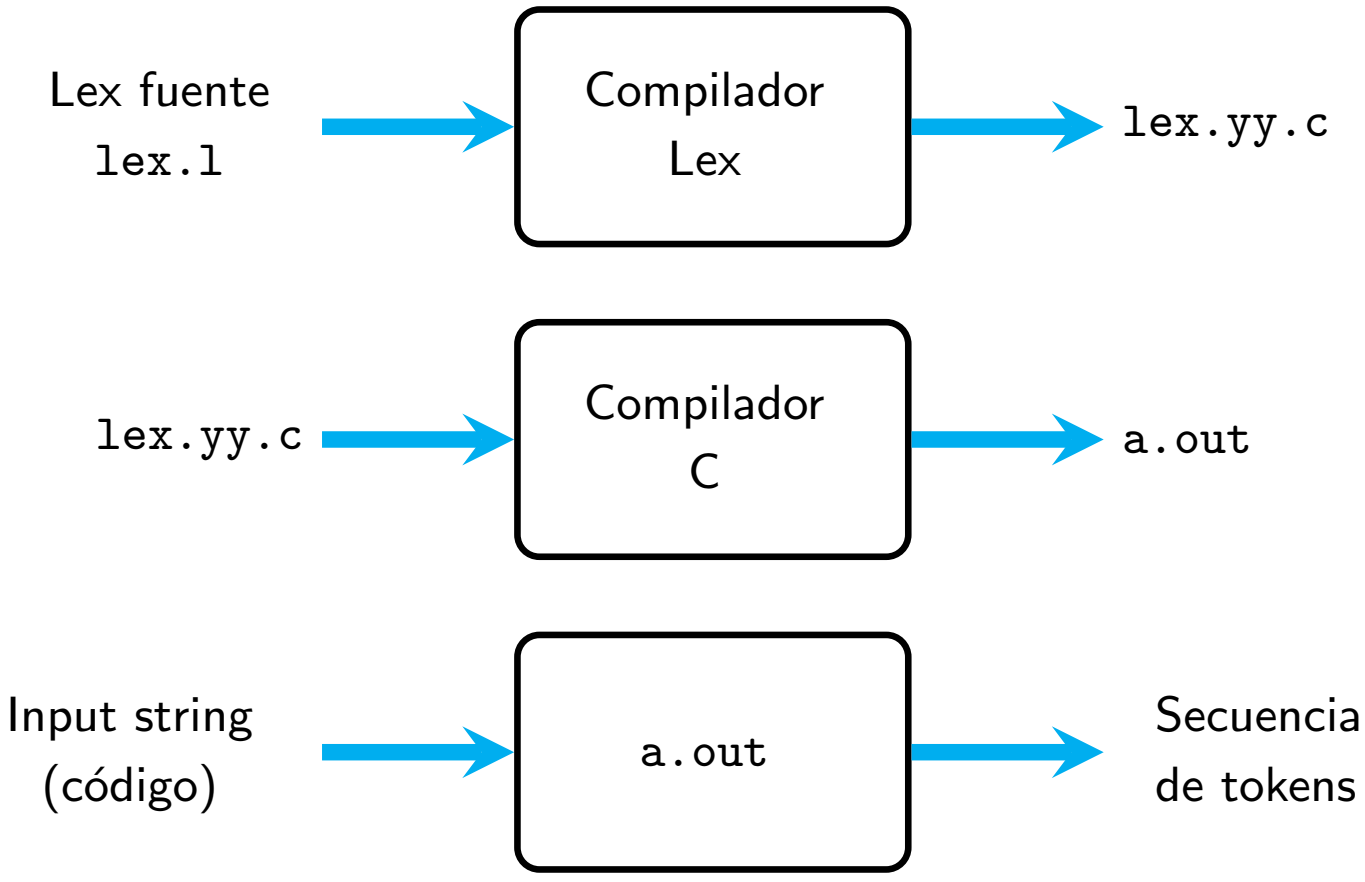
Tipo	Valor
id	`pos`
EQ	`=`
id	`init`
PLUS	`+`
id	`rate`
MULT	`*`
number	60

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.

- El más conocido es `Lex` para lenguaje `C`:
  - Versión moderna es Flex.
  - Para Java existe JFlex.
  - Para Python existe PLY.

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
```

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)

Ejemplo de `lex.l`

```
%{
#include "misconstantes.h" \* def de IF, ELSE, ID, NUMBER *\
%}
delim       [ \t\n]
ws          {delim}+
id          [A-Za-z]([A-Za-z0-9])*
number      [0-9]+
%%
{ws}        {\* sin accion *\}
if          {return(IF);}
else        {return(ELSE);}
{id}        {printID(); return(ID);}
{number}    {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`.

Para efectos del ejemplo, desde ahora supondremos
que cada patrón esta separado por un símbolo especial "␣".

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

Sea $T_1, \ldots, T_k$ los **patrones** y
$C_1, \ldots, C_k$ las **acciones** en el programa "`lex.l`", respectivamente.

Primer paso

Para cada patrón T_i construimos un NFA $\mathcal{A}_i = (Q_i, \Sigma, \Delta_i, I_i, F_i)$.

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

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

- $\mathcal{A}_i = (Q_i, \Sigma, \Delta_i, I_i, F_i)$ el NFA para el **patrón** T_i.
- C_i la **acción** de T_i.

Construimos el **transductor determinista**:

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

- $Q = 2^{\bigcup_{i=1}^{k} Q_i}$
- $(S, a, \epsilon, S') \in \Delta \quad \text{ssi} \quad S' = \{q \mid \exists i.\, \exists p \in S.\, (p, a, q) \in \Delta_i\}$.
- $(S, \llcorner, C_i, q_0) \in \Delta$ ssi $S \cap F_i \neq \varnothing$ y $(S, \llcorner, \epsilon, q_0) \in \Delta$ ssi $S \cap \bigcup_{i=1}^{k} F_i = \varnothing$.
- $q_0 = \bigcup_{i=1}^{k} I_i$
- $F = \{S \mid \exists i.\ S \cap F_i \neq \varnothing\}$

Conclusión: el análisis léxico es equivalente a **ejecutar un transductor**.