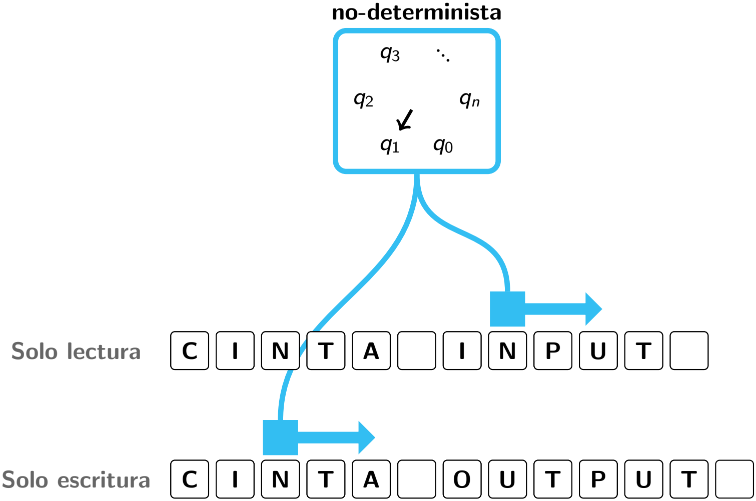
Aplicaciones de transductores

Clase 13

IIC 2223

Prof. Cristian Riveros

Transductores



Outline

Outline

Análisis léxico

Pattern matching

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

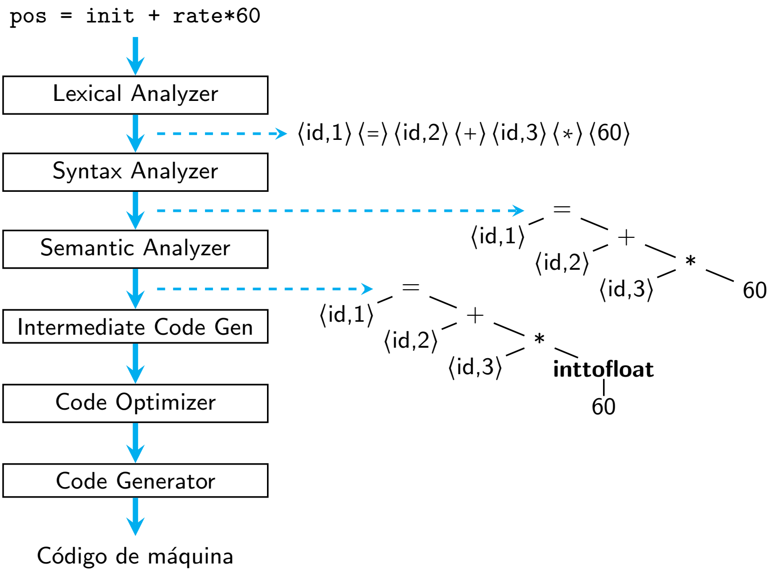
Definición

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`

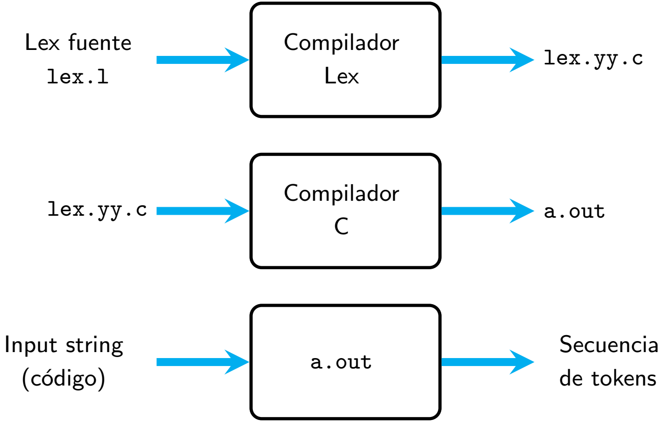
Tipo	Valor
id	`pos`
EQ	`=`
id	`init`
PLUS	`+`
id	`rate`
MULT	`*`
number	60

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**:

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

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

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

Outline

Problema de pattern matching de una palabra

Problema

Dado un **patrón** $w = w_1 \dots w_m$ y un **documento** $d = d_1 \dots d_n$, encontrar todos las posiciones donde aparece w en d, o sea, enumerar:

$$\{(i, j) \mid w = d_i d_{i+1} \dots d_j\}$$

Solución ingenua

for $i = 0$ **to** $n - m$ **do**
- $j := 1$
- **while** $j \leq m \wedge w_j = d_{i+j}$ **do**
  - $j := j + 1$
- **if** $j > m$ **then**
  - **output** $(i + 1, i + m + 1)$

¿es posible hacerlo mejor?

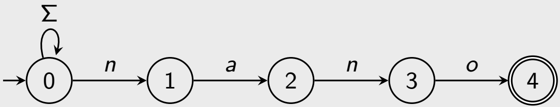
Autómata de un patrón

Definición

Dado un palabra $w = w_1 \dots w_m$, sea el NFA $\mathcal{A}_w = (Q, \Sigma, \Delta, I, F)$ tal que:

- $Q = \{0, 1, \dots, m\}$
- $\Delta = \{(0, a, 0) \mid a \in \Sigma\} \ \cup \ \{(i, w_{i+1}, i+1) \mid i < m\}$
- $I = \{0\}$ y $F = \{m\}$.

Ejemplo: palabra w = nano



¿cómo podemos usar $\mathcal{A}_w$ para encontrar todas las apariciones de w en d?

Determinización de $\mathcal{A}_w$

Sea $\mathcal{A}_w^{\text{det}} = (Q^{\text{det}}, \Sigma, \delta^{\text{det}}, \{0\}, F^{\text{det}})$ la determinización de $\mathcal{A}_w$ tal que Q^{det} contiene **solo los estados alcanzables** desde $\{0\}$.

Recordatorio

Para un autómata no-determinista $\mathcal{A} = (Q, \Sigma, \Delta, I, F)$,
se define el autómata determinista (**determinización** de $\mathcal{A}$):

$$\mathcal{A}^{\text{det}} \quad = \quad (Q^{\text{det}}, \Sigma, \delta^{\text{det}}, q_0^{\text{det}}, F^{\text{det}})$$

- $Q^{\text{det}} = 2^Q = \{S \mid S \subseteq Q\}$
- $q_0^{\text{det}} = I$.
- $\delta^{\text{det}} : 2^Q \times \Sigma \to 2^Q$ tal que:

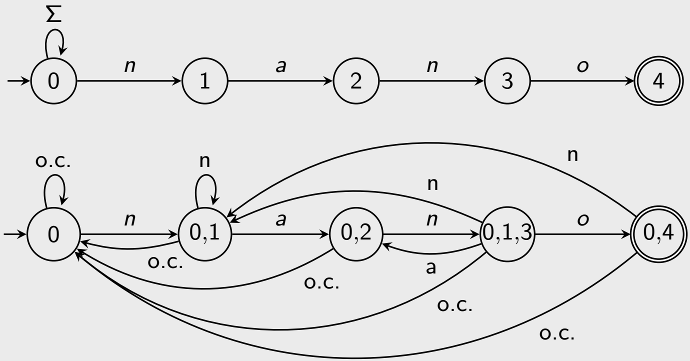
  $$\delta^{\text{det}}(S, a) = \{q \in Q \mid \exists p \in S.\ (p, a, q) \in \Delta\}$$

- $F^{\text{det}} = \{S \in 2^Q \mid S \cap F \neq \varnothing\}$.

Determinización de $\mathcal{A}_w$

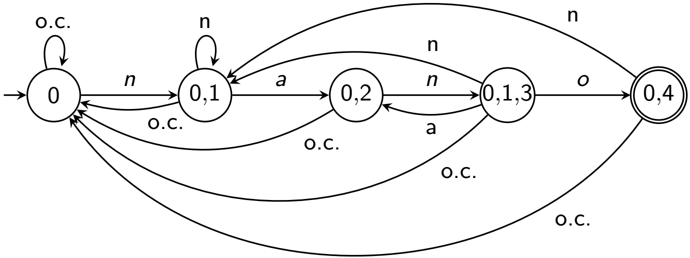
Sea $\mathcal{A}_w^{\text{det}} = (Q^{\text{det}}, \Sigma, \delta^{\text{det}}, \{0\}, F^{\text{det}})$ la determinización de $\mathcal{A}_w$ tal que Q^{det} contiene **solo los estados alcanzables** desde $\{0\}$.

Ejemplo: palabra w = nano



¿cuál es el problema de construir $\mathcal{A}_w^{\text{det}}$?

¿cómo utilizamos $\mathcal{A}_w^{\text{det}}$ para encontrar todos los matches?



q_0	q_0	q_1	q_0	q_1	q_2	q_3 ✓	q_4	q_0	q_1	q_0	q_0	q_1	q_2	q_3	q_2
u	n		n	a	n	o		n	o		n	a	n	a	
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	

¿cuál es el tiempo de este algoritmo una vez construido $\mathcal{A}_w^{\text{det}}$?

¿cuál es el tamaño de $\mathcal{A}_w^{\text{det}}$?

Sea $w = w_1 \dots w_m$ y $\mathcal{A}_w^{\text{det}} = (Q^{\text{det}}, \Sigma, \delta^{\text{det}}, \{0\}, F^{\text{det}})$ la determ. de $\mathcal{A}_w$.

Teorema

Para todo $S \in Q^{\text{det}}$ y $i \in \{0, 1, \dots, m\}$ se cumple que:

$$i \in S \quad \text{si, y solo si,} \quad w_1 \dots w_i \text{ es un sufijo de } w_1 \dots w_{\max(S)}.$$

Corolarios

- Para todo $S_1, S_2 \in Q^{\text{det}}$, si $\max(S_1) = \max(S_2)$, entonces $S_1 = S_2$.
- $\mathcal{A}_w^{\text{det}}$ tiene $|w| + 1$ estados y $\mathcal{O}(|w|^2)$ transiciones.

Por lo tanto, encontrar todos los substrings de w en d
toma tiempo $\mathcal{O}(|d| + |w|^2)$

¿cuál es el tamaño de $\mathcal{A}_w^{\det}$?

Demostración teorema

Sea $S \in Q^{\det}$ un conjunto de estados cualquiera alcanzable desde $\{0\}$.

Entonces existe una palabra $u = a_1 \dots a_k$ tal que $\hat{\delta}^{\det}(\{0\}, u) = S$.

Por la demostración que $\mathcal{L}(\mathcal{A}^{\det}) = \mathcal{L}(\mathcal{A})$ para todo NFA $\mathcal{A}$ (Clase 03), sabemos que $j \in S$ si, y solo si, existe una ejecución de $\mathcal{A}_w$ sobre u:

$$0 = q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} \dots \xrightarrow{a_k} q_k = j.$$

Por la definición de $\mathcal{A}_w$ esta ejecución es de la forma:

$$0 \xrightarrow{a_1} 0 \xrightarrow{a_2} \dots \xrightarrow{a_{k-j}} 0 \underbrace{\xrightarrow{a_{k-j+1}} 1 \xrightarrow{a_{k-j+2}} 2 \dots \xrightarrow{a_k}}_{w_1 \dots w_j} j.$$

Por lo tanto, $w_1 w_2 \dots w_j$ es sufijo de $a_1 \dots a_k$.

Usaremos este último hecho para demostrar **ambas direcciones**.

¿cuál es el tamaño de $\mathcal{A}_w^{\text{det}}$?

Propiedad

Para toda $u = a_1 \ldots a_k$ tal que $\hat{\delta}^{\text{det}}(\{0\}, u) = S$, y para todo $j \leq m$:

$$j \in S \quad \text{si, y solo si,} \quad w_1 \ldots w_j \text{ es sufijo de } a_1 \ldots a_k$$

Demostración teorema ($\Rightarrow$)

Como S es alcanzable desde $\{0\}$,
entonces existe $u = a_1 \ldots a_k$ tal que $\hat{\delta}^{\text{det}}(\{0\}, u) = S$.

Como $\max(S) \in S$, entonces $w_1 \ldots w_{\max(S)}$ es sufijo de $a_1 \ldots a_k$.

Suponga que $i \in S$. Entonces $w_1 \ldots w_i$ es sufijo de $a_1 \ldots a_k$.

Como $i \leq \max(S)$, entonces:

$$a_1 a_2 \ldots a_{k-\max(S)} \overbrace{a_{k-\max(S)+1} \ldots a_{k-i} \underbrace{a_{k-i+1} \ldots a_k}_{w_1 \ldots w_i}}^{w_1 \ldots w_{\max(S)}}$$

Por lo tanto, $w_1 \ldots w_i$ es sufijo de $w_1 \ldots w_{\max(S)}$. ✓

¿cuál es el tamaño de $\mathcal{A}_w^{\text{det}}$?

Propiedad

Para toda $u = a_1 \dots a_k$ tal que $\hat{\delta}^{\text{det}}(\{0\}, u) = S$, y para todo $j \leq m$:

$$j \in S \quad \text{si, y solo si,} \quad w_1 \dots w_j \text{ es sufijo de } a_1 \dots a_k$$

Demostración teorema ($\Leftarrow$)

Como S es alcanzable desde $\{0\}$,
entonces existe $u = a_1 \dots a_k$ tal que $\hat{\delta}^{\text{det}}(\{0\}, u) = S$.

Como $\max(S) \in S$, entonces $w_1 \dots w_{\max(S)}$ es sufijo de $a_1 \dots a_k$.

Suponga que $w_1 \dots w_i$ es sufijo de $w_1 \dots w_{\max(S)}$.

Como $w_1 \dots w_i$ es sufijo de $w_1 \dots w_{\max(S)}$ y $w_1 \dots w_{\max(S)}$ es sufijo de u,
entonces $w_1 \dots w_i$ es sufijo de $u = a_1 \dots a_k$.

Por la "Propiedad", concluimos que $i \in S$. $\square$

¿cuál es el tamaño de $\mathcal{A}_w^{\text{det}}$?

Sea $w = w_1 \dots w_m$ y $\mathcal{A}_w^{\text{det}} = (Q^{\text{det}}, \Sigma, \delta^{\text{det}}, \{0\}, F^{\text{det}})$ la determ. de $\mathcal{A}_w$.

Teorema

Para todo $S \in Q^{\text{det}}$ y $i \in \{0, 1, \dots, m\}$ se cumple que:

$$i \in S \quad \text{si, y solo si,} \quad w_1 \dots w_i \text{ es un sufijo de } w_1 \dots w_{\max(S)}.$$

Corolarios

$\mathcal{A}_w^{\text{det}}$ tiene $|w| + 1$ estados y $\mathcal{O}(|w|^2)$ transiciones.

Por lo tanto, encontrar todos los substrings de w en d toma tiempo $\mathcal{O}(|d| + |w|^2)$

¿es posible hacerlo mejor?