Teoría de Lenguajes

Analizadores Sintácticos Descendentes

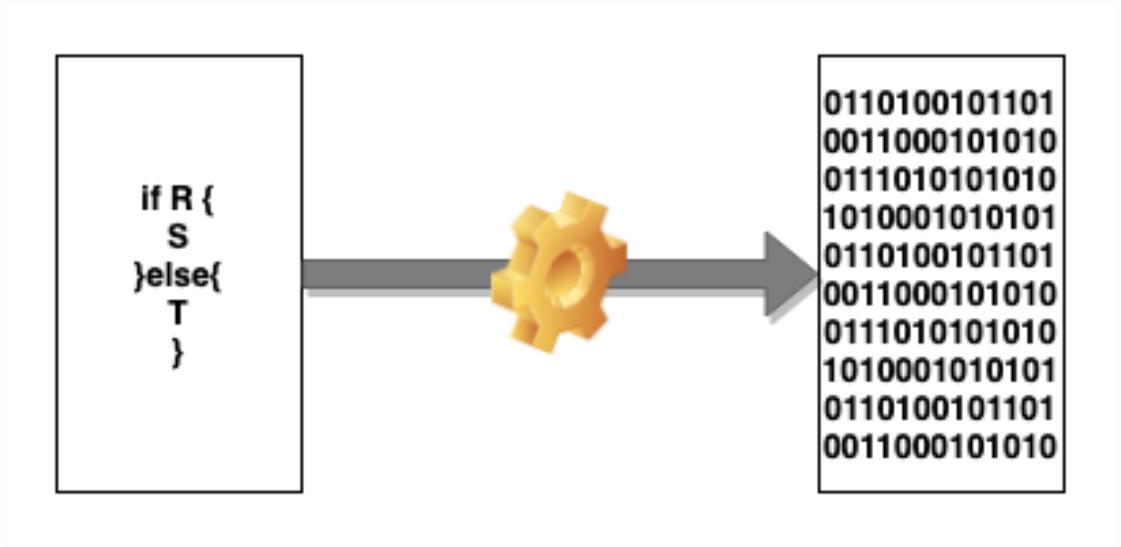
Natalia Pesaresi

DC-UBA

1er. Cuatrimestre 2020

Compiladores

- Tienen como objetivo convertir un código fuente de programa a otro lenguaje: Algunos a código máquina ejecutable, o otros intermedio interpretable (e.g., Java o .NET)
- Tendrá que poder detectar errores en el código, y poder marcarlos de forma de facilitar su corrección



Compiladores

Proceso de un Compilador

1. **Análisis Léxico**: Se encarga de procesar las cadenas y generar los lexemas correspondientes. Por cada lexema se generará las lista de tokens (par token-valor).
   `int x = 0;`
2. **Análisis Sintáctico**: A partir de la salida del analizador léxico se generará el árbol sintáctico correspondiente a la código recibido
   `if(){...}else{...}`
3. **Análisis Semántico**: Se genera información para las etapas subsiguientes, y en particular se encargará del `chequeo de tipos`
4. **Generación de Código Intermedio**
5. **Optimización**
6. **Generación de Código ejecutable**

El problema de interés

- Dado un lenguaje L definido por la gramática

$$G =< \{E, T, F\}, \{+, *, num, (,)\}, P, E >$$

$$\begin{array}{lcl} E & \rightarrow & E + T \\ E & \rightarrow & T \\ T & \rightarrow & T * F \\ T & \rightarrow & F \\ F & \rightarrow & num \\ F & \rightarrow & (E) \end{array}$$

- y dada la cadena

$$\alpha : (14 + 6) * 2$$

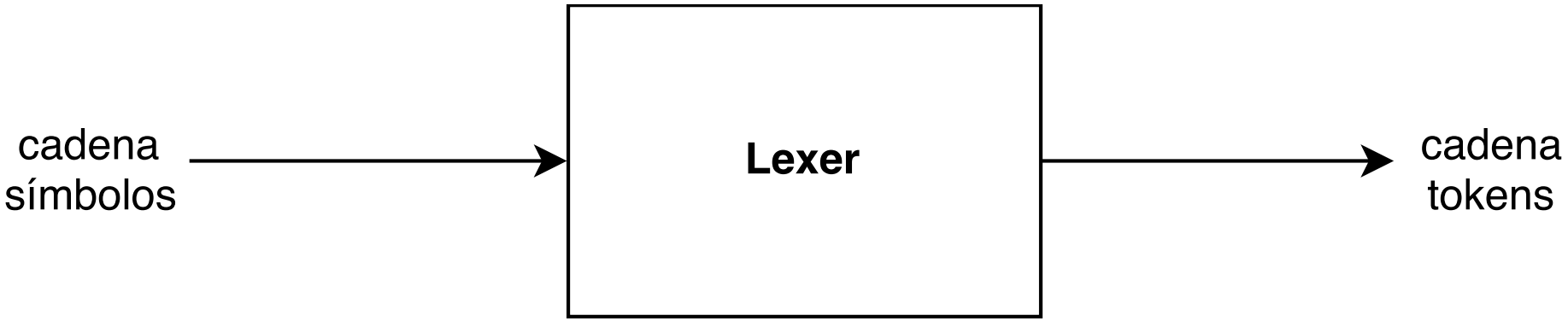
- ¿$\alpha \in L$?

¿Cómo sabe la computadora qué es un **num**?

- ¿Reemplazamos al terminal *num* por los terminales 14, 6 y 2?
- ¿Reemplazamos la producción $F \to num$ por las producciones $F \to 14$, $F \to 6$ y $F \to 2$?
- ¿Podemos hacer esto para todos los números?
- ¡Son infinitos!

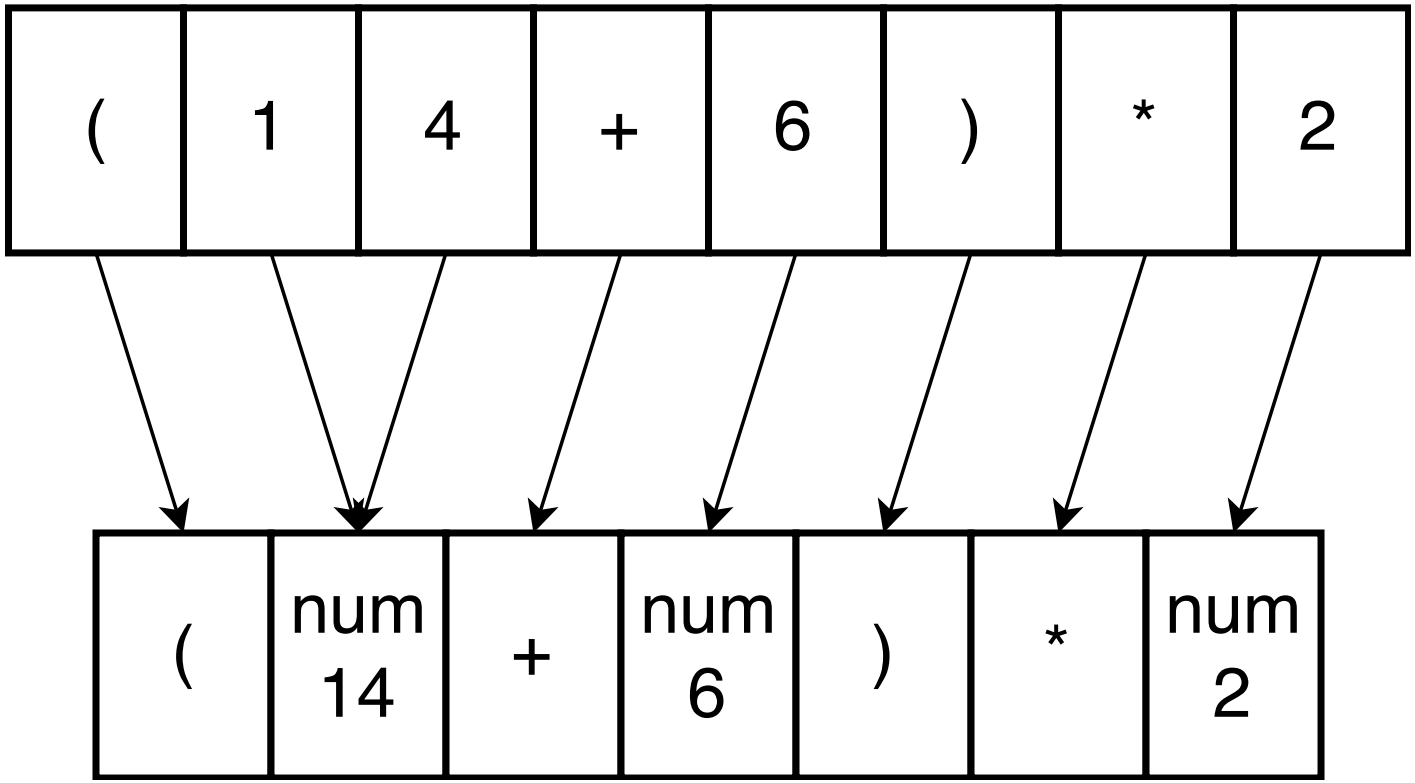
- Ahí es donde entran en juego los **analizadores léxicos** (lexers)

Analizador Léxico (Lexer)

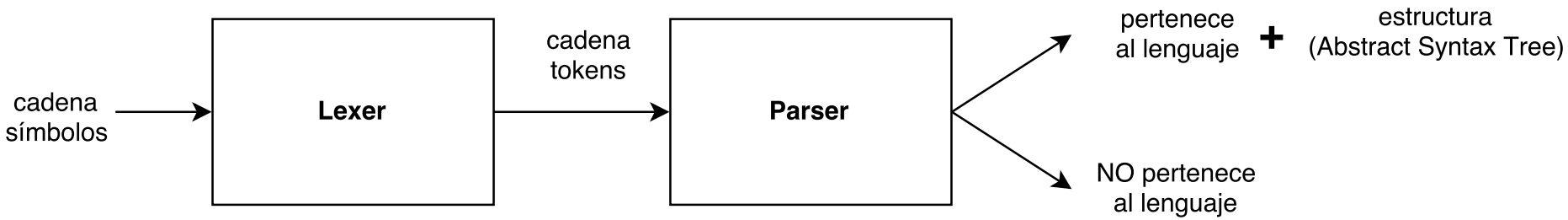


- Usa expresiones regulares
- Agrupa símbolos de la entrada en tokens
- Un token tiene un tipo y un valor

Ejemplo

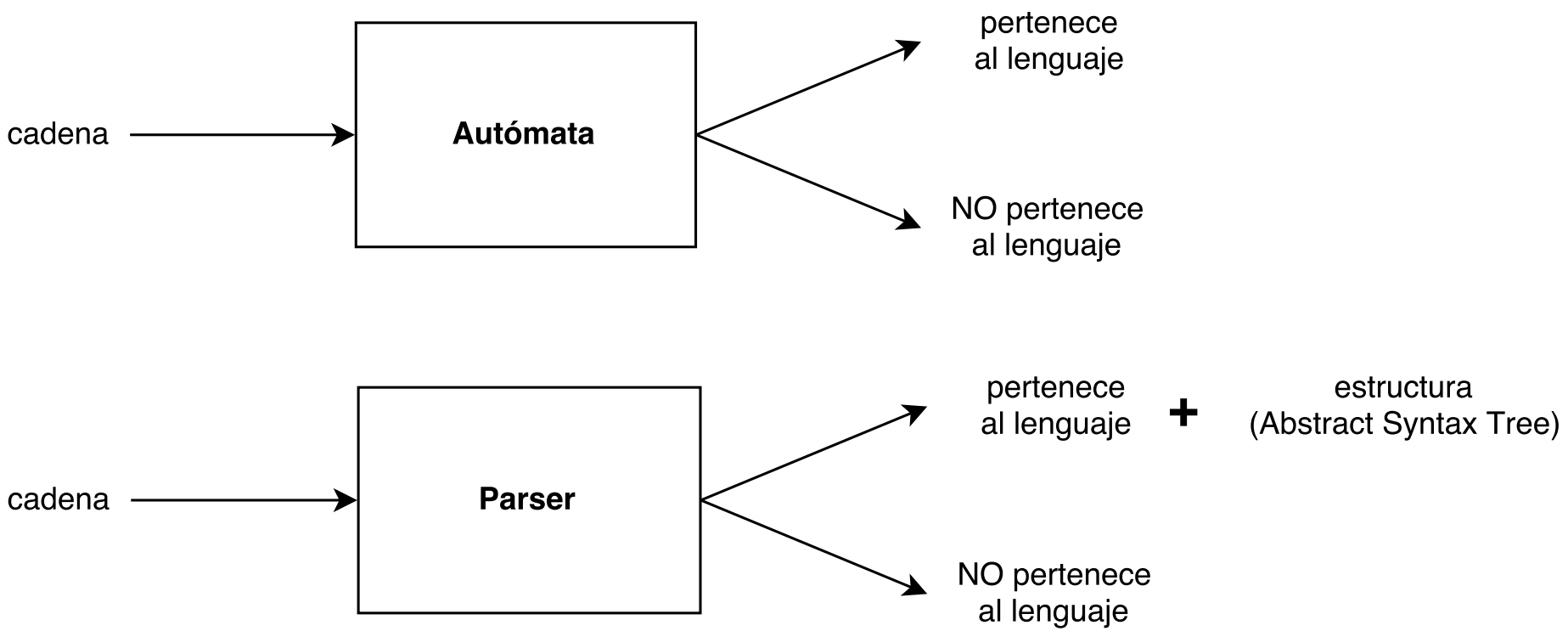


Lexer + Parser

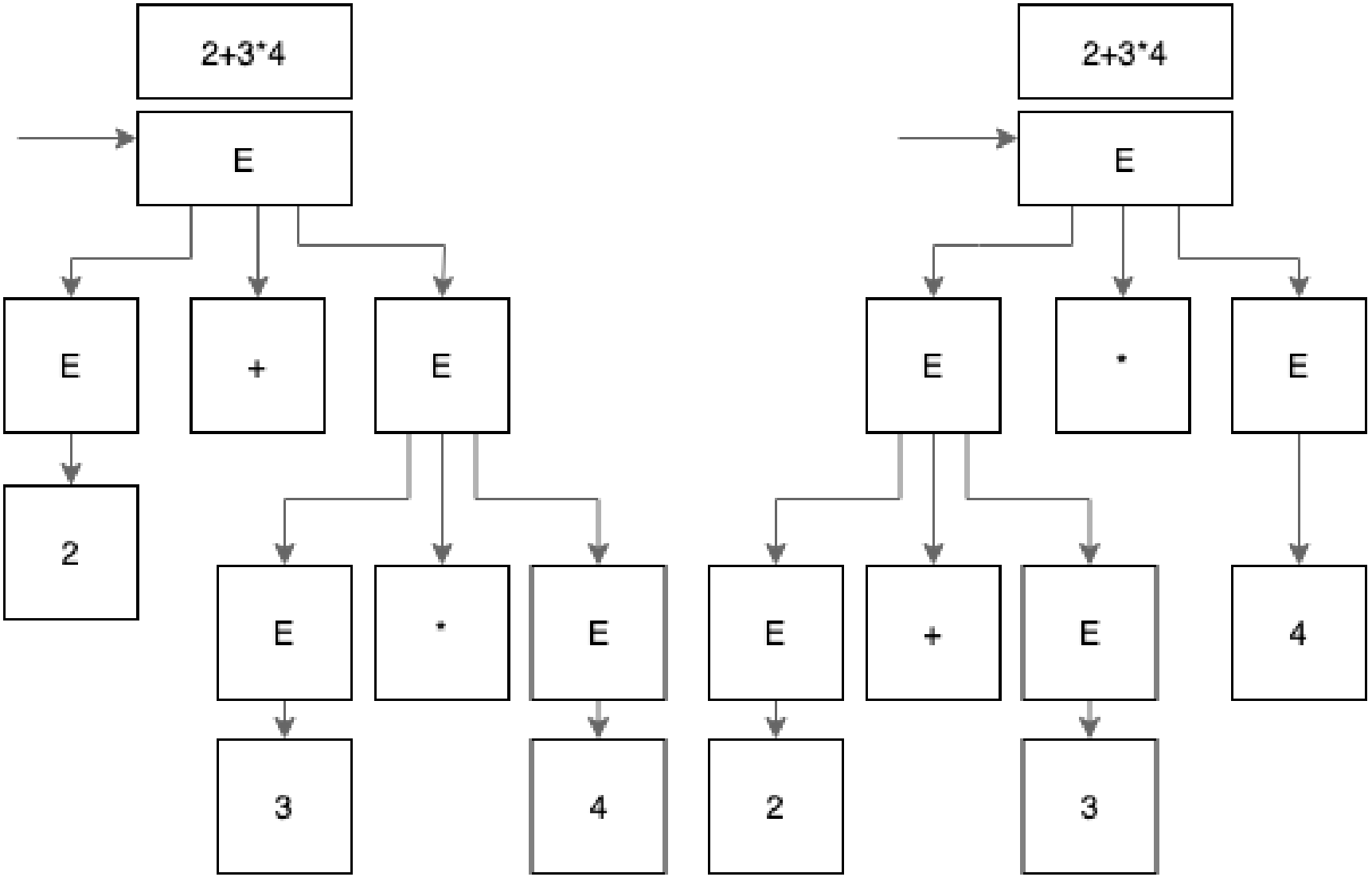


- Cadena de símbolos: `(14+6)*2`
- Cadena de tokens: `(num+num)*num`
- `Árbol sintáctico abstracto`

Autómata vs. Parser



AST



Tipos de Parsers(Analizadores Sintácticos)

- Descendentes (top-down):
  - Construyen el árbol sintáctico desde la raiz hasta las hojas (la secuencia de tokens).
  - Ejemplos: DFS, parser predictivos recursivos, LL(1), LL(k).
- Ascendentes (bottom-up)
  - Construyen el árbol sintáctico desde las hojas (la secuencia de tokens) hasta la raiz (el símbolo distinguido).
  - Ejemplos: parsers shift-reduce, LR(0), SLR, LALR, LR(K)
- Otros
  - Generalized LR (GLR), Earley, CYK

Parsers descendentes(top-down)

- Dada la siguiente gramática:

$$S \to A$$
$$S \to B$$
$$A \to a$$
$$B \to b$$

- ¿Cómo genero la cadena a?

Parsers predictivos descendentes

- Parten del símbolo distinguido. `S en nuestro ejemplo`.
- Y la secuencia que vamos a iterar es:
  - Identificar, el **token corriente, tc**. Cual es el token que queremos reconocer en este momento.
  - Y que producción lo reconoce. Para elegir la producción, calculamos el conjunto de **símbolos directrices**.

Cadena: a.

$$S \to A$$
$$S \to B$$
$$A \to a$$
$$B \to b$$

¿Cómo elegir una producción?

Primeros, Siguiente, Símbolos Directrices

Vamos a calcular dos funciones y luego SD.

Primeros: de una cadena, es un conjunto que contiene los símbolos terminales con los que 'comienza'.

$$Primeros(\alpha) : (V_N \cup V_T)^* \rightarrow \mathbf{P}(V_T)$$

$$Primeros(\alpha) = \{t \in V_T \mid \alpha \stackrel{*}{\Rightarrow} t\beta\}$$

Ej. Gramática: *(las minúsculas son tokens)*

$$S \rightarrow A|B|C$$
$$A \rightarrow a|Cm$$
$$B \rightarrow abBb|A$$
$$C \rightarrow \lambda|c$$

$P(a) = \{a\}$ $P(c) = \{c\}$ $P(abBb) = \{a\}$ $P(lambda) = \emptyset$

$P(Cm) = \{c, m\}$, m se agrega porque C es anulable.

$P(A) = P(a) \cup P(Cm) = \{a, c, m\}$

¿Cómo elegir una regla?

Primeros, Siguiente, Símbolos Directrices

Siguientes: de un **símbolo no terminal** es el conjunto de tokens, que aparecen 'después'de este símbolo entre las producciones.

$$Siguientes(N) : V_N \rightarrow \mathbf{P}(V_T)$$
$$Siguientes(N) = \{t \in V_T \mid S \stackrel{*}{\Rightarrow} ...Nt...\}$$

Ej. Gramática:

$$S \rightarrow A|B$$
$$A \rightarrow a|Cm$$
$$B \rightarrow abBb|A|C$$
$$C \rightarrow \lambda|c$$

$S(C) = \{m\}$ y $S(B) = \{b\}$

¿Cómo elegir una regla?

Primeros, Siguiente, Símbolos Directrices

Función que nos da el conjunto de de Símbolos Directrices (SD):

$$SD(N \to \beta) = \left\{ \begin{array}{ll} Primeros(\beta) & \text{Si } \lambda \notin \beta \\ Primeros(\beta) \cup Siguientes(N) & \text{Si } \lambda \in \beta \end{array} \right\}$$

Ej. Gramática:

$$S \to A|B$$
$$A \to a|Cm$$
$$B \to abBb|A|C$$
$$C \to \lambda|c$$

$SD(B \to abBb) = P(abBb) = \{a\}$
$SD(B \to C) = P(C) \cup S(B) = \{c\} \cup \{b\} = \{c, b\}$

Implementaciones de Parsers predictivos descendentes

Tenemos 2 maneras de implementar un parser descendente.

1. Programable: uso de pila explícito
2. Seguimiento: otro con uso de pila implícito

Programable (1)

Algortimo general:

- se crea un procedimiento por cada no terminal(ej. S,A,B,C). **Main** es el correspondiente al símbolo distinguido.
- dentro de cada **procedimiento** se hacen 3 cosas:
  - se invocan los procedimientos de los no terminales incluidos
  - se busca los terminales esperados (`match`)
  - o no se hace nada (si es un λ)
- si hay más de una producción para un mismo no terminal, se agregan condicionales para ejecutar sólo el procedimiento que contenga al token actual (tc) en sus SD's.
  - se agrega un `else` final con `error()`, para cubrir el caso en que el `tc` no está en los SD de ninguna de las reglas posibles

Ej. $B \rightarrow abBb$

Seguimiento (2)

- Se crea una tabla: cadena || pila || producción
- se escribe la cadena a reconocer seguida de $ en la primer columna.
- Se escribe el símbolo inicial(S) y $, con el inicial como tope de pila.
- Iteración:
  - Si el tope de pila es un es un símbolo terminal y coincide con el tc, acción match y despilar.
  - Si el tope de pila es un es un símbolo terminal y **no** coincide con el tc, ERROR().
  - Si el tope de pila es un no temrinal, elegir la producción que tiene al tc entre sus símbolos directrices.
  - Si no, ERROR().

Algunos problemas y cómo resolverlos

- Ambigüedad por SDs no disjuntos
  - factorización a la izquierda (*left-factorization*)
    Ej. $B \to abBb|am$
- Recursión a izquierda
  - eliminación de la recursión inmediata
    Ej. $B \to Ba|s|t$
  - eliminación de la recursión no inmediata
    Ej. $B \to Ca|d$ y $C \to Bf$

Factorización a la izquierda

Si tenemos producciónes de la forma
$A \to \alpha\beta_1 \mid \alpha\beta_2 \mid \ldots \mid \alpha\beta_n \mid \delta_1 \mid \delta_2 \mid \ldots \mid \delta_k$ donde $\alpha \neq \lambda$, las reemplazamos por

$$A \to \alpha A' \mid \delta_1 \mid \delta_2 \mid \ldots \mid \delta_k$$
$$A' \to \beta_1 \mid \beta_2 \mid \ldots \mid \beta_n$$

Ej. $B \to abBb|am \Rightarrow B \to aB' \quad$ y $\quad B' \to bBb|m$

Eliminación de la recursión inmediata

Si tenemos producciónes de la forma $A \to A\alpha \mid \beta_1 \mid \beta_2 \mid \ldots \mid \beta_k$ las reemplazamos por

$$A \to \beta_1 A' \mid \beta_2 A' \mid \ldots \mid \beta_k A'$$
$$A' \to \alpha A' \mid \lambda$$

Ej. $B \to Ba|s|t \Rightarrow B \to sB'|tB' \quad y \quad B' \to a|\lambda$

Eliminación de la recursión no inmediata

1. Numerar los no terminales $A_1, \ldots, A_n$
2. Para $i \to 1$ a n
   1. Para $j \to 1$ a $i-1$
      1. Reemplazar $A_i \to A_j\gamma$ por por $A_i \to \delta_1\gamma \mid \delta_2\gamma \mid \ldots \mid \delta_k\gamma$ (donde $A_j \to \delta_1 \mid \ldots \mid \delta_k \in P$)
   2. Fin Para
   3. Eliminar la recurción inmediata de A_i
3. Fin Para

Ej. $B \to Ca|d \quad y \quad C \to Bf$

$B \to Bfa|d$

$B \to dB' \quad y \quad B' \to fa|\lambda$

Ejemplo

Dada la siguiente gramática para la declaración de tipos en un lenguaje de programación:
$G_1 = \langle \{T, S\}, \{\texttt{array}, \texttt{[}, \texttt{]}, \texttt{of}, \texttt{..}, \texttt{int}, \texttt{char}, \texttt{num}, \uparrow\}, T, P \rangle$, con P:

$$
\begin{array}{lcl}
T & \longrightarrow & S \mid \uparrow S \mid \texttt{array of } T \mid \texttt{array [} S \texttt{] of } T \\
S & \longrightarrow & \texttt{int} \mid \texttt{char} \mid \texttt{num..num}
\end{array}
$$

- Hacer un parser descendente recursivo para esta gramática. Si no es posible, corregirla previamente
- Realizar el seguimiento de como se reconoce una cadena.

Ejemplo

$$
\begin{array}{lcl}
T & \longrightarrow & S \mid \uparrow S \mid \texttt{array of } T \mid \texttt{array [}S\texttt{] of } T \\
S & \longrightarrow & \texttt{int} \mid \texttt{char} \mid \texttt{num..num}
\end{array}
$$

Queremos saber,

- ¿La cadena $\texttt{array [num..num] of} \uparrow \texttt{char} \in L(G_1)$?
- ¿Es posible hacer un parser descendente recursivo para esta gramática?
  No porque hay solapamiento entre los SDs de dos reglas del no terminal T

Ejemplo: Gramática corregida

Aplicando factorización a la izquierda, para crear G_1':
$T \to \texttt{array of } T \mid \texttt{array [} S \texttt{] of } T \Rightarrow$
$T \to \textit{array} T'$
$T' \to \texttt{of } T \mid \texttt{[} S \texttt{] of } T$
entonces,
$G_1' = \langle \{T, T', S\}, \{\texttt{array}, \texttt{[}, \texttt{]}, \texttt{of}, \texttt{..}, \texttt{int}, \texttt{char}, \texttt{num}, \uparrow\}, T, P \rangle$, con P:

$$\begin{array}{lcl} T & \longrightarrow & S \mid \uparrow S \mid \texttt{array}\, T' \\ T' & \longrightarrow & \texttt{of } T \mid \texttt{[} S \texttt{] of } T \\ S & \longrightarrow & \texttt{int} \mid \texttt{char} \mid \texttt{num..num} \end{array}$$

Extendemos la gramática con,
$S_0 \to T\$$

Ejemplo: Gramática corregida y SDs

Calculemos los Símbolos directrices de todas las producciones.
$G_1' = \langle \{S_0, T, T', S\}, \{\texttt{array}, \texttt{[}, \texttt{]}, \texttt{of}, \texttt{..}, \texttt{int}, \texttt{char}, \texttt{num}, \uparrow\}, S_0, P\rangle$

	Producción		**SD**
S_0	$\longrightarrow$	T	$\{\texttt{int}, \texttt{char}, \texttt{num}, \uparrow, \texttt{array}\}$
T	$\longrightarrow$	S	$\{\texttt{int}, \texttt{char}, \texttt{num}\}$
T	$\longrightarrow$	$\uparrow S$	$\{\uparrow\}$
T	$\longrightarrow$	$\texttt{array}\, T'$	$\{\texttt{array}\}$
T'	$\longrightarrow$	$\texttt{of}\ T$	$\{\texttt{of}\}$
T'	$\longrightarrow$	$\texttt{[}S\texttt{]}\ \texttt{of}\ T$	$\{\texttt{[}\}$
S	$\longrightarrow$	$\texttt{int}$	$\{\texttt{int}\}$
S	$\longrightarrow$	$\texttt{char}$	$\{\texttt{char}\}$
S	$\longrightarrow$	$\texttt{num..num}$	$\{\texttt{num}\}$

Ejemplo: Parser descendente recursivo predictivo

```
Proc T()
    if tc in { int, char, num }
        S();
    else if tc in { ^ }
            match('^');
            S();
         else if tc in { array }
                 match('array');
                 T_p();
              else
                 error();
End

Proc T_p()
    if tc in { of }
        match('of');
        T();
    else if tc in { [ }
            match('['); S(); match(']');
            match('of');
            T();
         else
            error();
End
```

```
Proc S()
    if tc in { int }
       match('int');
    else if tc in { char }
            match('char');
         else if tc in { num }
                 match('num');
                 match('..');
                 match('num');
              else
                 error();
End

Proc Main();
    T();
    match('$');
    accept();
End
```

Seguimiento: ok

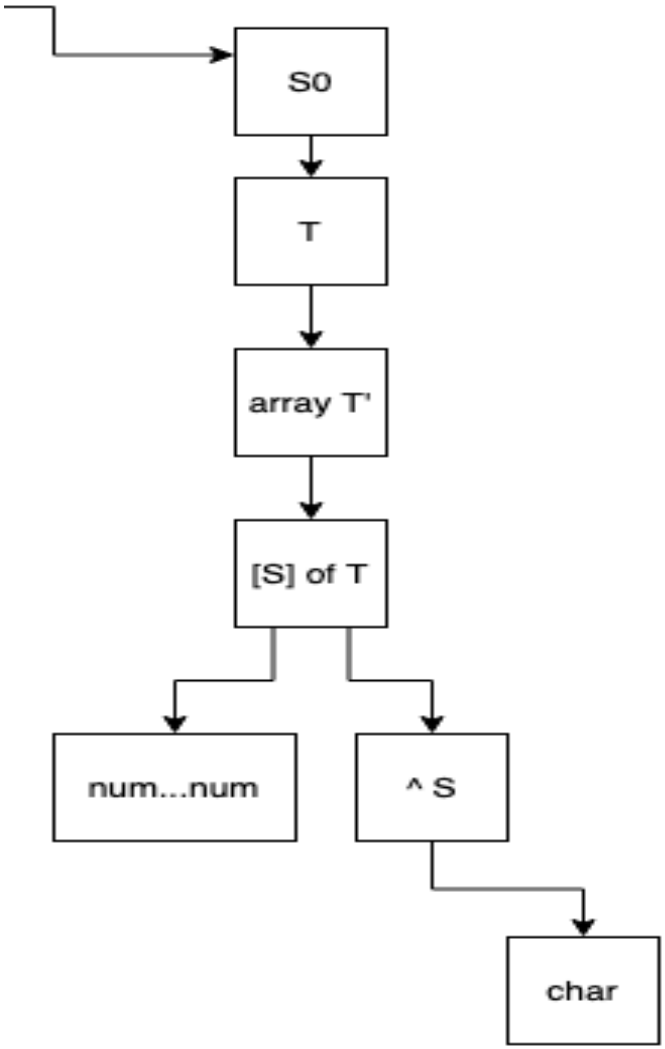
Cadena $array[num..num]of \uparrow char\$$:

Cadena	Pila	Producción / Acción
$array[num..num]of \uparrow char\$$	$T\$$	$T \to array \quad T'$
$array[num..num]of \uparrow char\$$	$array \quad T'\$$	$match\{array\}$
$[num..num]of \uparrow char\$$	$T'\$$	$T' \to \texttt{[}S\texttt{]}\ \texttt{of}\ T$
$[num..num]of \uparrow char\$$	$\texttt{[}S\texttt{]}\ \texttt{of}\ T\$$	$match\{[\}$
$num..num]of \uparrow char\$$	$S\texttt{]}\ \texttt{of}\ T\$$	$S \to \texttt{num..num}$
$num..num]of \uparrow char\$$	$num..num\texttt{]}\ \texttt{of}\ T\$$	$match\{num\}$
$..num]of \uparrow char\$$	$..num\texttt{]}\ \texttt{of}\ T\$$	$match\{..\}$
$num]of \uparrow char\$$	$num\texttt{]}\ \texttt{of}\ T\$$	$match\{num\}$
$]of \uparrow char\$$	$\texttt{]}\ \texttt{of}\ T\$$	$match\{]\}$
$of \uparrow char\$$	$\texttt{of}\ T\$$	$match\{of\}$
$\uparrow char\$$	$T\$$	

Seguimiento: ok

Cadena	Pila	Producción / Acción
$\uparrow$ *char*\$	T\$	$T \rightarrow \uparrow S$
$\uparrow$ *char*\$	$\uparrow S$\$	*match*$\{\uparrow\}$
char\$	S\$	$S \rightarrow$ `char`
char\$	*char*\$	*match*$\{char\}$
\$	\$	*match*$\{\$\}$

AST:



Seguimiento: error

Cadena $array[num..num]of$:

$array[num..num]of\$$

Cadena	Pila	Producción / Acción
$array[num..num]of\$$	$T\$$	$T \to array \quad T'$
$array[num..num]of\$$	$array \quad T'\$$	$match\{array\}$
$[num..num]of\$$	$T'\$$	$T' \to [S]\ \texttt{of}\ T$
$[num..num]of\$$	$[S]\ \texttt{of}\ T\$$	$match\{[\}$
$num..num]of\$$	$S]\ \texttt{of}\ T\$$	$S \to \texttt{num..num}$
$num..num]of\$$	$num..num]\ \texttt{of}\ T\$$	$match\{num\}$
$..num]of\$$	$..num]\ \texttt{of}\ T\$$	$match\{..\}$
$num]of\$$	$num]\ \texttt{of}\ T\$$	$match\{num\}$
$]of\$$	$]\ \texttt{of}\ T\$$	$match\{]\}$
$of\$$	$\texttt{of}\ T\$$	$match\{of\}$
$\$$	$T\$$	ERROR()

Seguimiento: ustedes!

Realizar el seguimiento de las siguientes cadenas:

- *of num...num*
- *array [char] of array of int*

Teoría de Lenguajes

Analizadores Sintácticos Descendentes

Natalia Pesaresi

DC-UBA

1er. Cuatrimestre 2020