Tema 3
Análisis sintáctico

3.1 El analizador sintáctico

3.2 Definición sintáctica

3.3 Análisis descendente

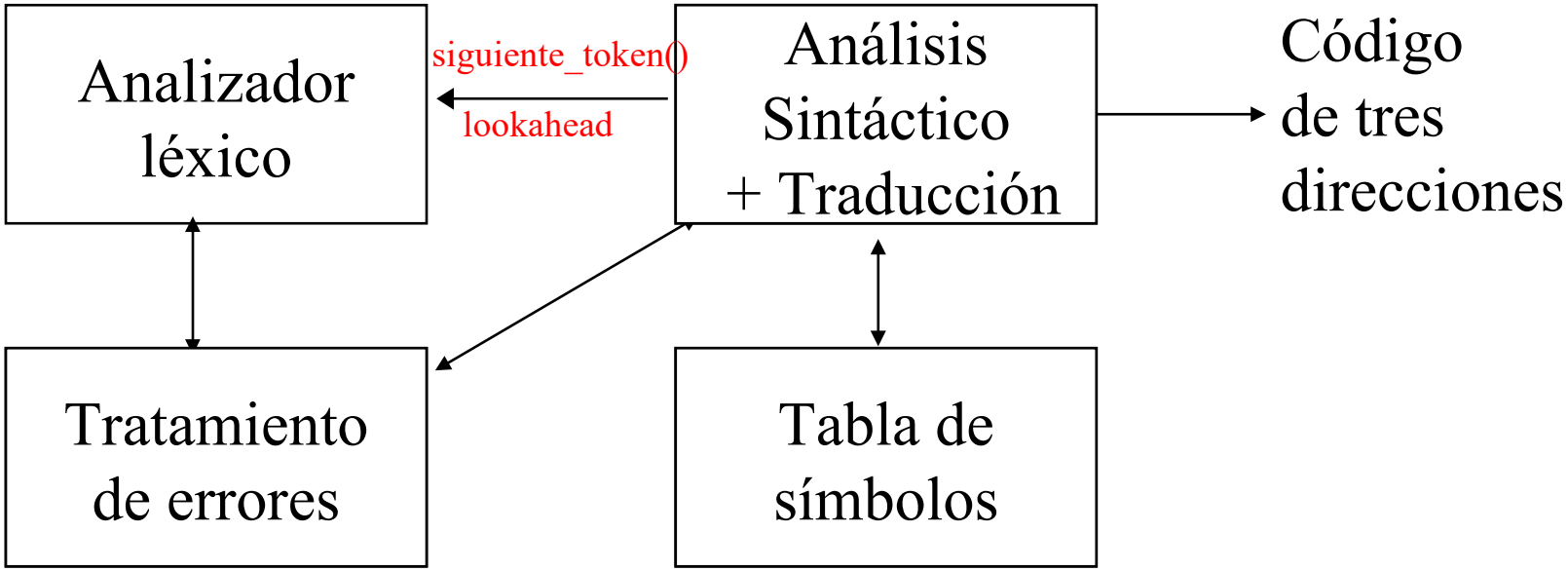
3.4 Análisis ascendente

3.5 YACC: generador de analizadores sintácticos

3.6 Tratamiento de errores sintácticos

3.1 El analizador sintáctico

Su relación con el resto del compilador



3.2 Definición sintáctica

Gramáticas Independientes del Contexto (Σ,N,P,S)

1. Un conjunto de símbolos terminales (tokens)
2. Un conjunto de símbolos no-terminales.
3. Un conjunto de producciones, donde cada producción consiste en un no-terminal y una secuencia, que puede ser vacía, de terminales y/o no-terminales.
4. Un no-terminal como símbolo inicial.

Gramáticas Independientes del Contexto

- G = (Σ, N, P, S)
- $P = \{A \rightarrow \alpha : A \in N \wedge \alpha \in (\Sigma \cup N)^*\}$
- Derivación:

  $\alpha_1 \Rightarrow \alpha_2$ sii $\alpha_1 = \beta_1 A \beta_2$ y $\alpha_2 = \beta_1 \alpha \beta_2$ con $A \in N$ y $(A \rightarrow \alpha) \in P$
- Lenguaje generado:

  $x \in \Sigma^*$ tal que $S \overset{*}{\Rightarrow} x$

Definición sintáctica

- Derivación a izquierdas:

  siempre se sustituye el símbolo no-terminal más a la izquierda

  - Análisis descendente

- Derivación a derechas:

  » siempre se sustituye el símbolo no-terminal más a la derecha

  - Análisis ascendente

Ejemplo

G = ({+,-,*,/,id,entero},{E},P,E)

P = { E$\rightarrow$E+E, E$\rightarrow$E-E,

E$\rightarrow$E*E, E$\rightarrow$E/E,

E$\rightarrow$id, E$\rightarrow$entero}

Ejemplo (notación abreviada)

$E \rightarrow$ E **+** E
| E **-** E
| E ***** E
| E **/** E
| **id**
| **entero**

Árboles de análisis (árboles de derivación)

1. La raíz está etiquetada con el símbolo inicial.
2. Cada hoja está etiquetada bien con un terminal, bien con la cadena vacía.
3. Cada nodo interior está etiquetado con un no-terminal.
4. Si un no-terminal A etiqueta un nodo interior y $X_1,X_2,...,X_n$ son etiquetas de los hijos de ese nodo, entonces $A \rightarrow X_1X_2...X_n$ es una producción.

   Como caso especial, si $A \rightarrow \xi$, entonces un nodo etiquetado A puede tener un solo hijo, etiquetado ξ.

Ambigüedad

- Árbol de derivación

  Está asociado a varias secuencias de derivación

  Admite una única secuencia de derivación a izquierdas

- Ambigüedad

  Para al menos un cadena hay varios árboles de derivación

  Para al menos una cadena hay más de una derivación a izquierdas

Ejemplo de ambigüedad

$E \rightarrow$ E **+** E

| E **-** E

| E ***** E

| E **/** E

| **id**

| **entero**

Ejemplo de ambigüedad

sent $\rightarrow$ if exp then sent

| if exp then sent else sent

| *otras*

if e1 then *if e2 then s1 else s2*

if e1 then *if e2 then s1* *else s2*

Gramáticas equivalentes

- Dos gramáticas son equivalentes si definen el mismo lenguaje
- Gramática equivalente a G, no ambigua:

$E \rightarrow E + T \mid E - T \mid T$

$T \rightarrow T * F \mid T / F \mid F$

$F \rightarrow \mathbf{id} \mid \mathbf{entero}$

El analizador sintáctico

- Dos técnicas de análisis sintáctico:
  - Análisis descendente:
    - Símbolo inicial $\rightarrow$ hojas (tokens-terminales)
    - Análisis LL
  - Análisis ascendente:
    - Cadena de tokens$\rightarrow$ Símbolo inicial
    - Análisis LR

3.3 Análisis descendente

- Análisis predictivo: condiciones LL(1)
  Cómo conseguir gramáticas predictivas
- Análisis descendente recursivo
- Análisis descendente no recursivo

Análisis descendente

- El árbol de análisis comienza a construirse por la raíz y se termina en las hojas
- Para cada nodo **n** etiquetado con un no-terminal **A**
  - » seleccionar una de las producciones para **A** y construir los hijos de **n**, que serán los símbolos de la parte derecha de la producción escogida

Ejemplo

tipo $\rightarrow$ tipo_simple

| $\uparrow$id

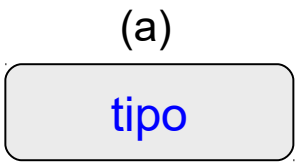
| array [tipo_simple] of tipo

tipo_simple $\rightarrow$ integer

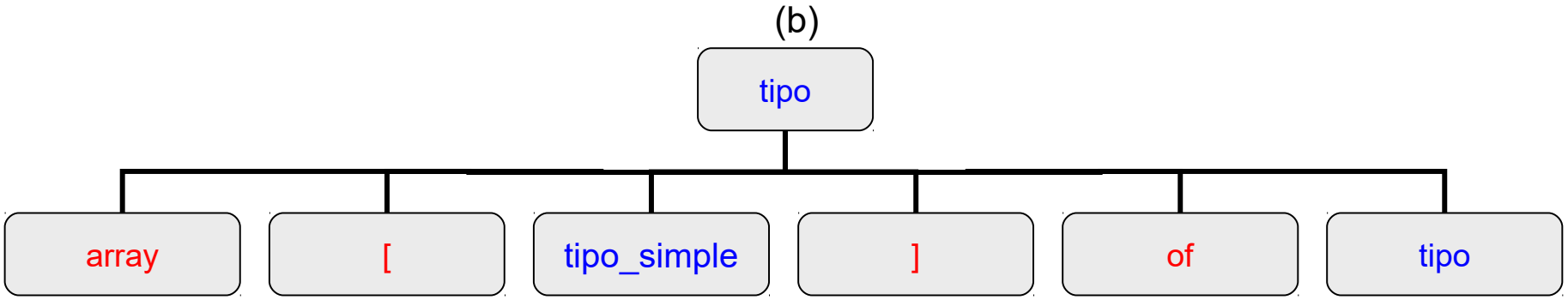
| char

| num punto punto num

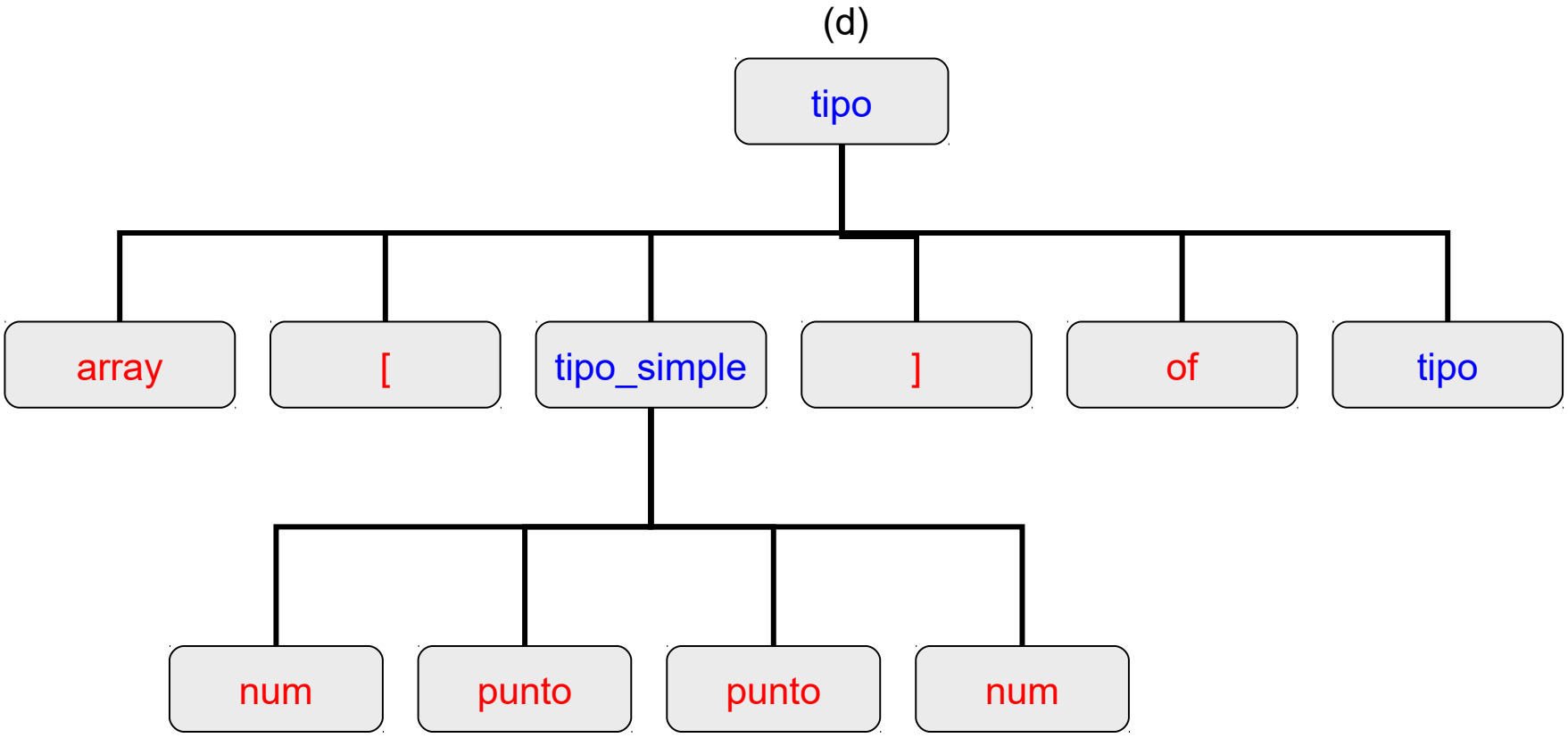
array [1..10] of ^ persona



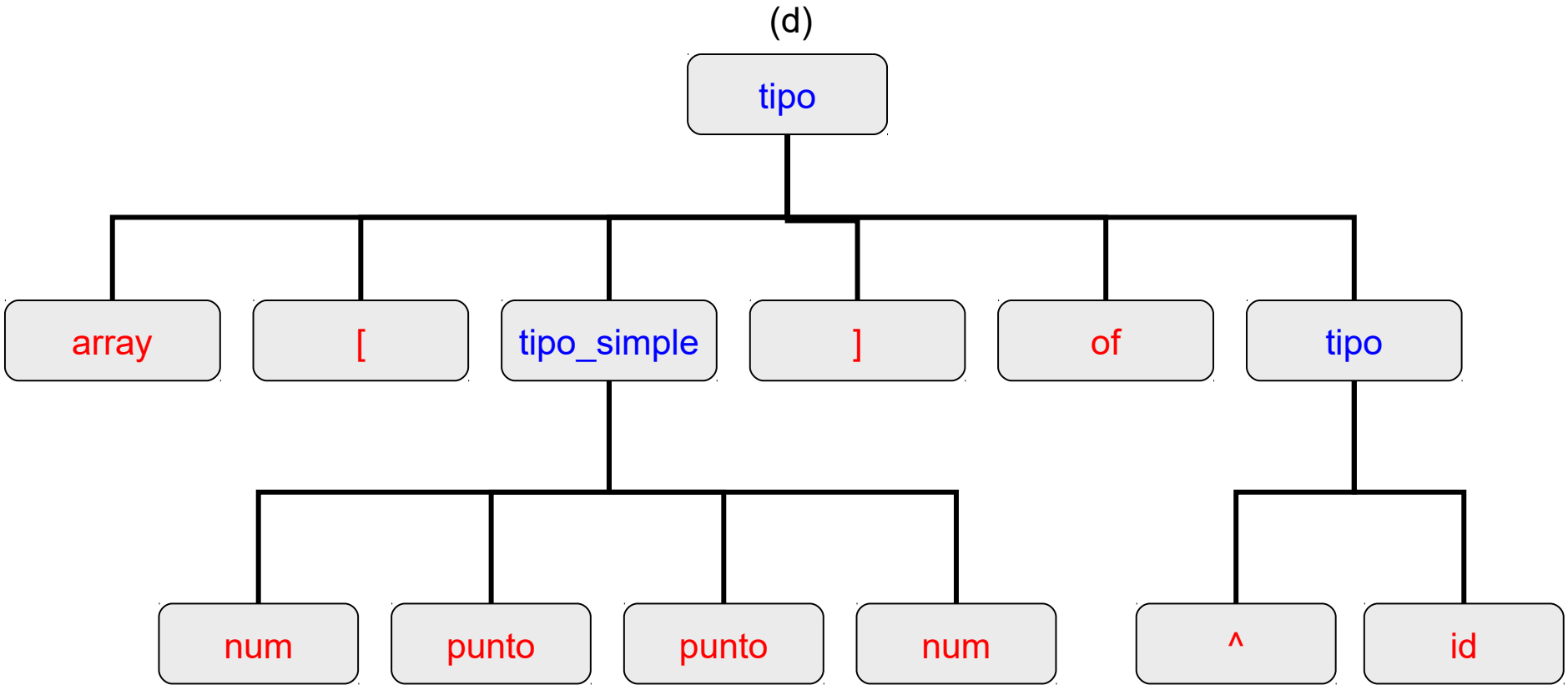
array [1..10] of ^ persona



array [1..10] of ^ persona



array [1..10] of ^ persona



Análisis predictivo (I)

- Para algunas gramáticas es posible implementar los pasos anteriores durante un recorrido de izquierda a derecha de la cadena de entrada.

- Examinando únicamente el *lookahead.*

Análisis predictivo (II)

- En el análisis predictivo, el *lookahead* determina sin ambigüedades cuál debe ser la producción elegida para cada no-terminal en un momento dado del proceso de análisis.
- **PRIMERO(α):** conjunto de tokens que aparecen como primer símbolo de las cadenas generadas por α.

Ejemplo (motivación)

tipo $\rightarrow$ tipo_simple

| $\uparrow$id

| array [tipo_simple] of tipo

tipo_simple $\rightarrow$ integer

| char

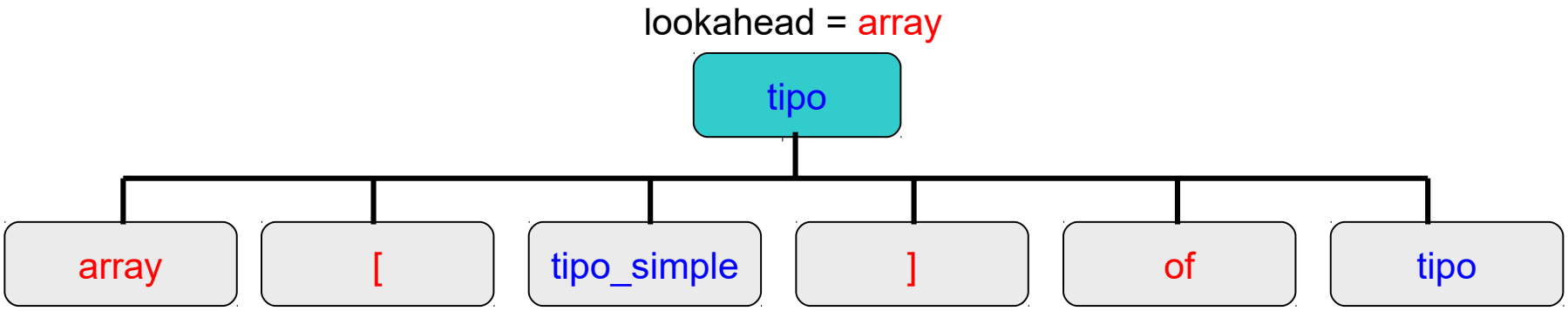
| num punto punto num

array [1..10] of ^ persona

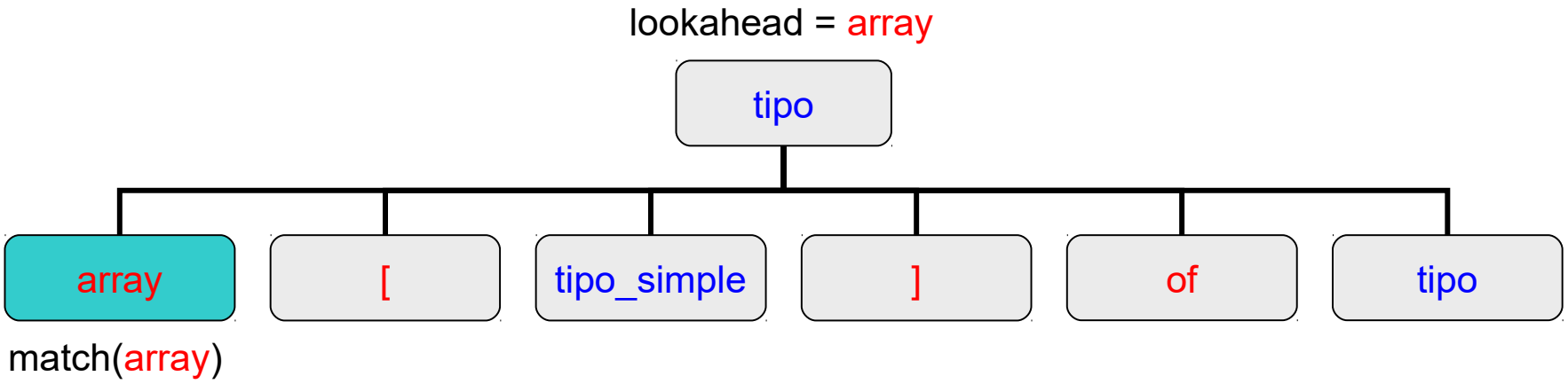
lookahead = array

tipo

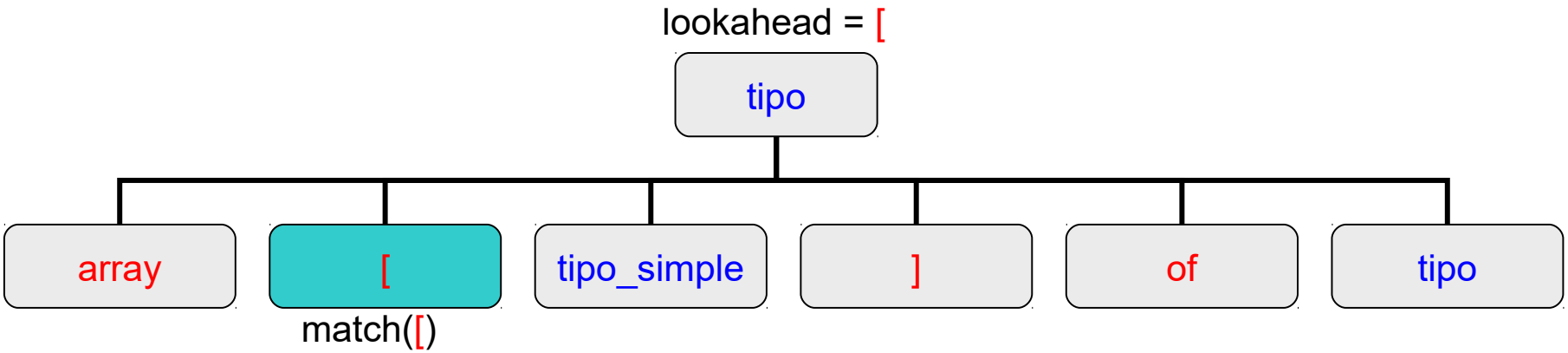
array [1..10] of ^ persona



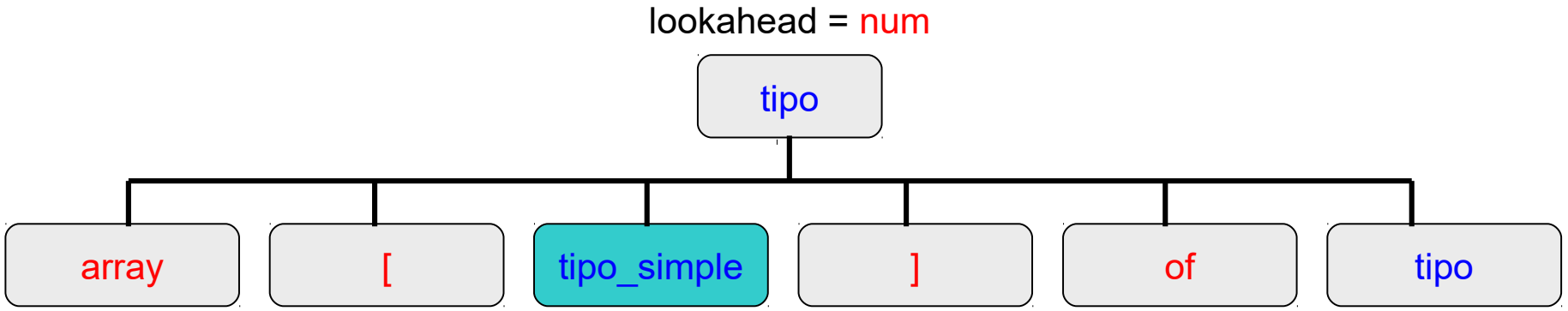
array [1..10] of ^ persona



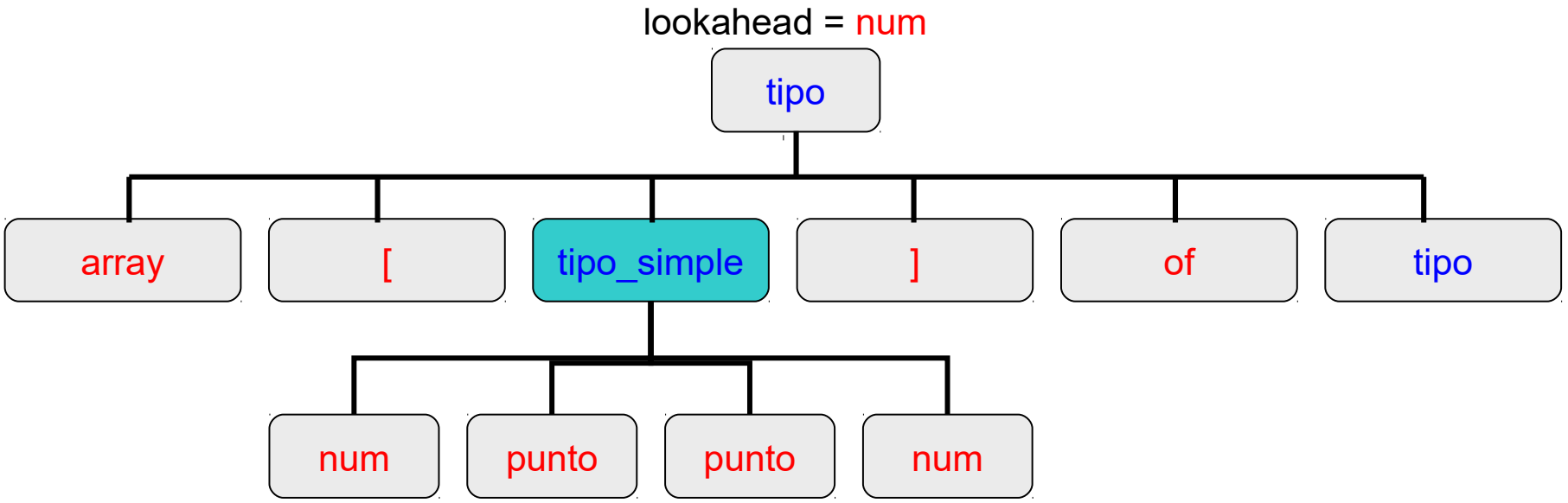
array [1..10] of ^ persona



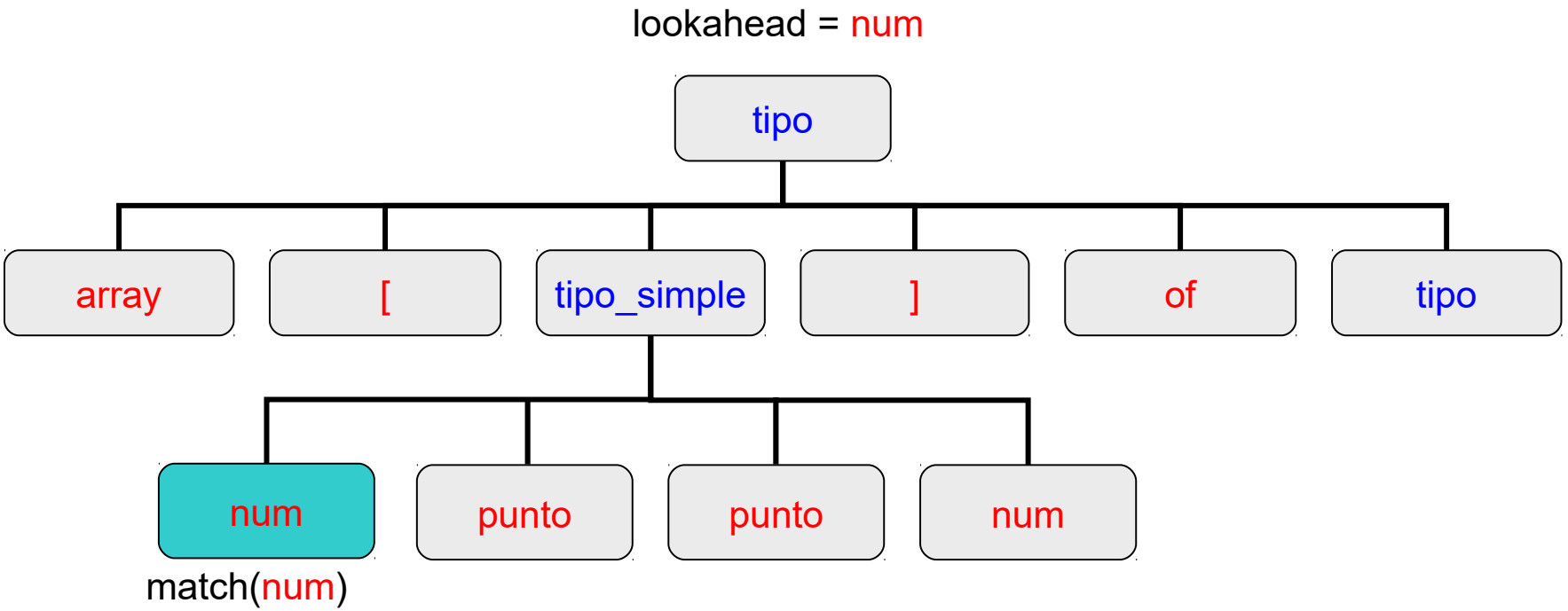
array [1..10] of ^ persona



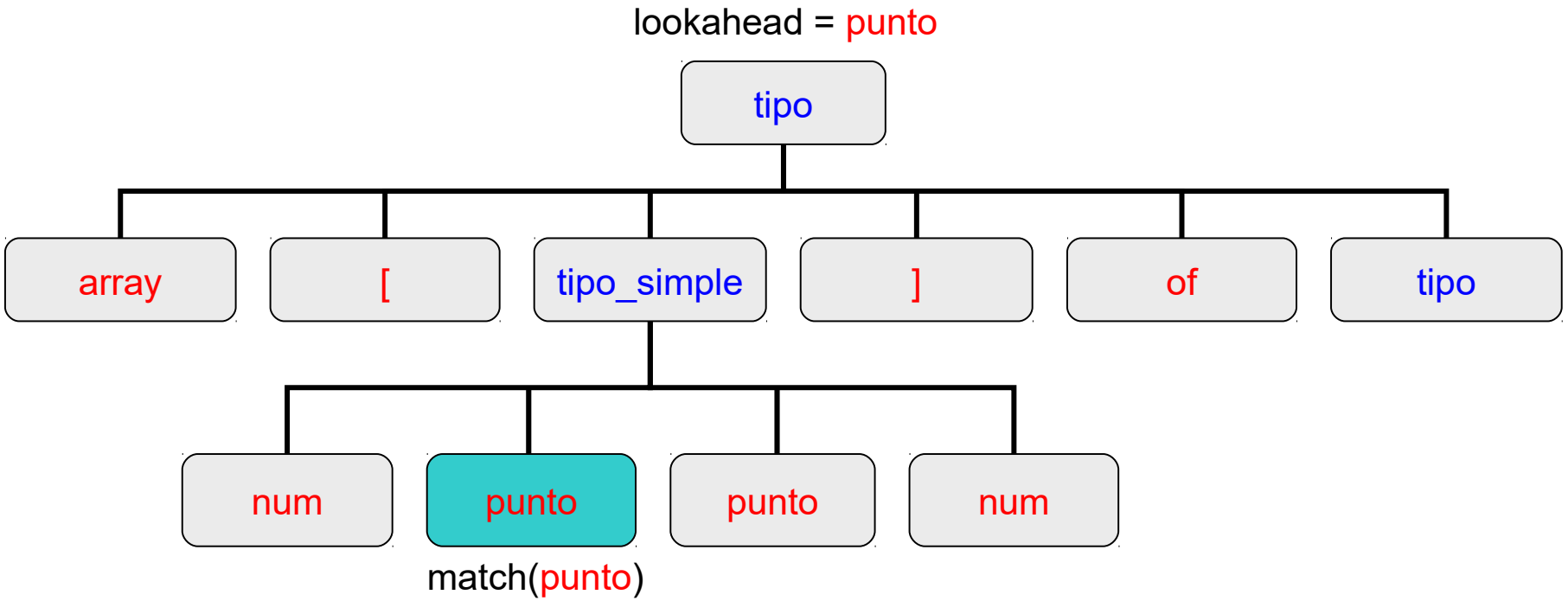
array [1..10] of ^ persona



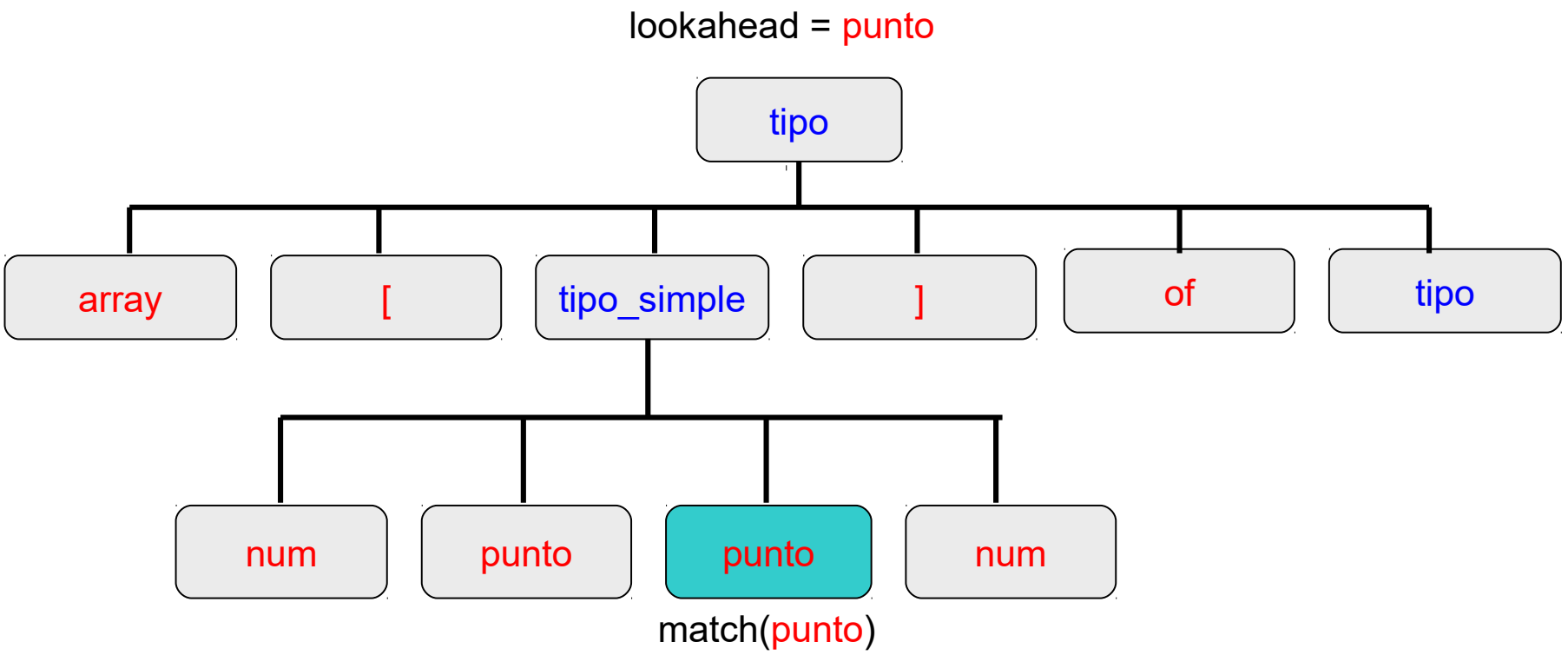
array [1..10] of ^ persona



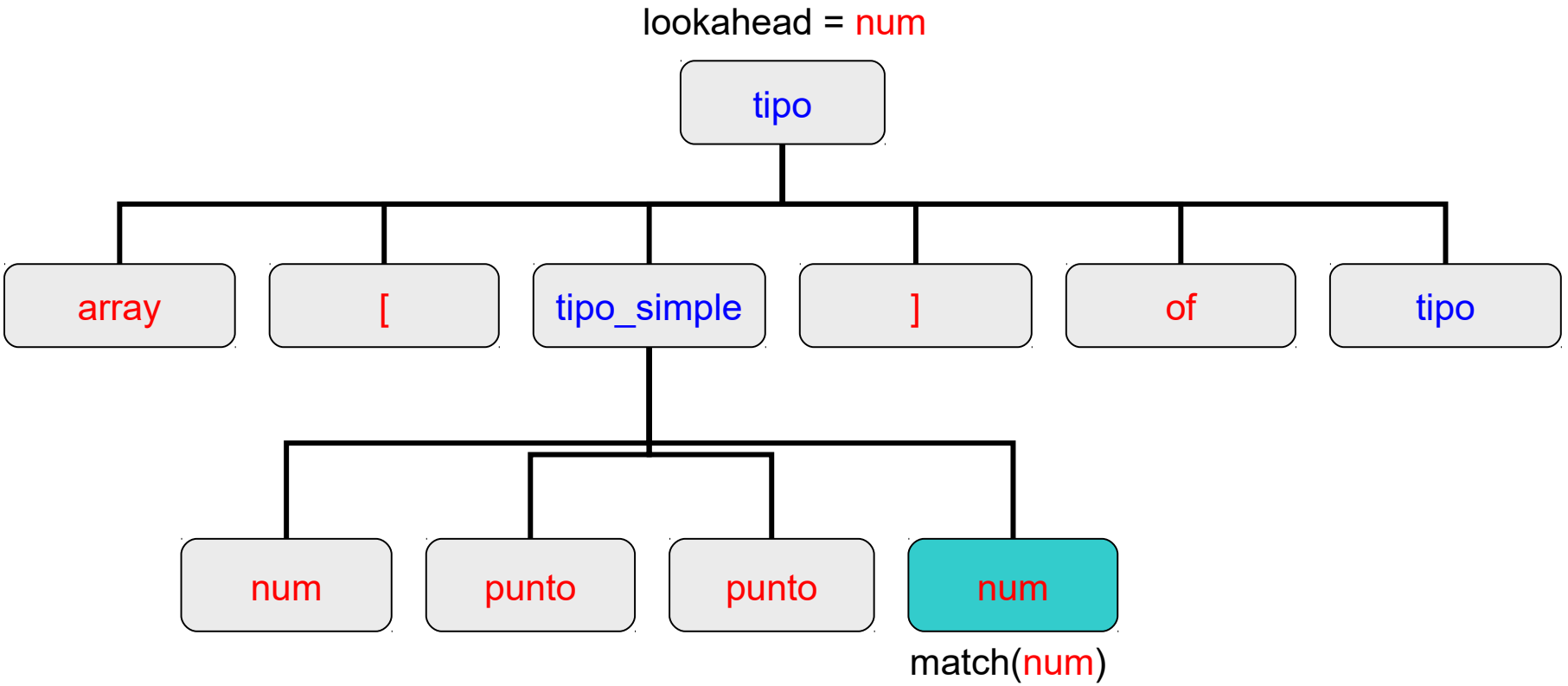
array [1..10] of ^ persona



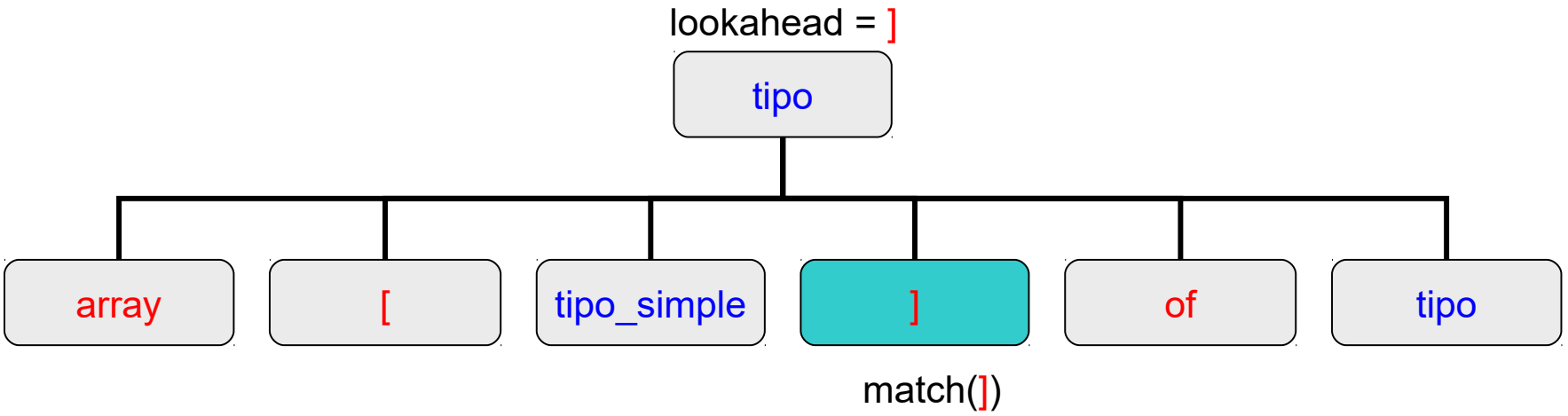
array [1..10] of ^ persona



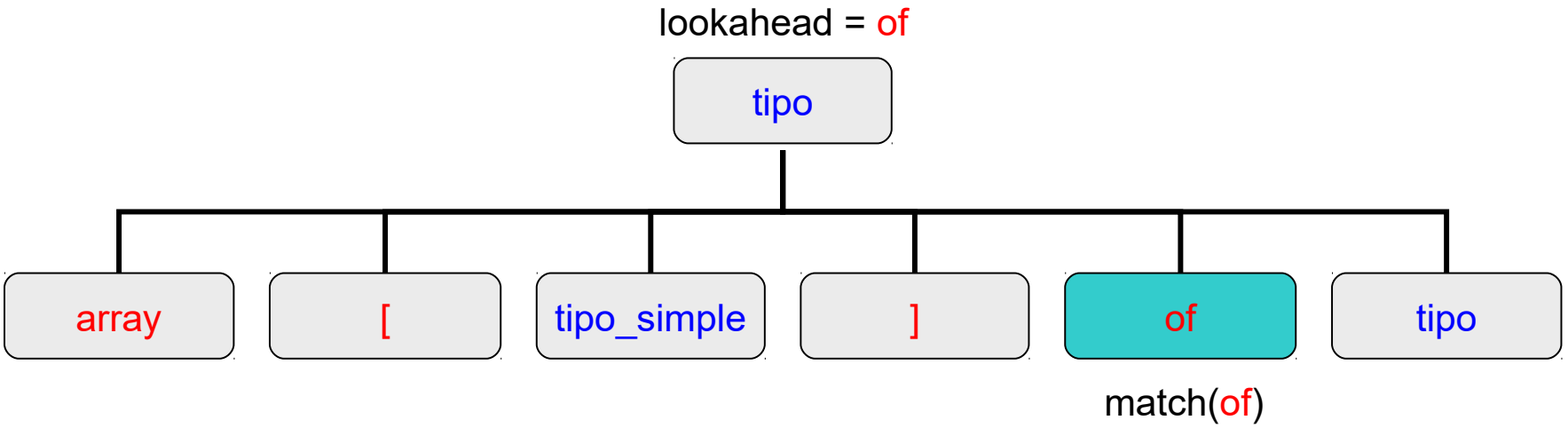
array [1..10] of ^ persona



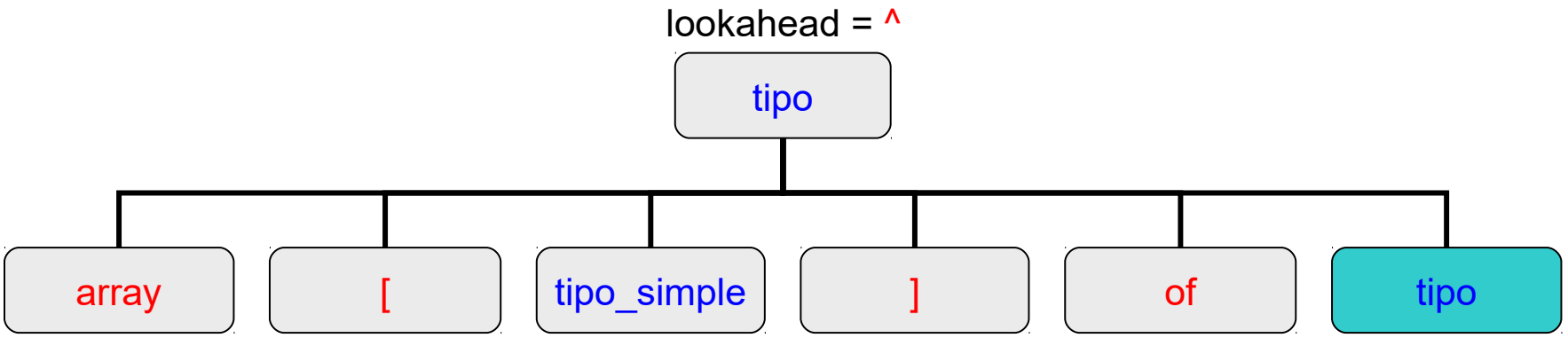
array [1..10] of ^ persona



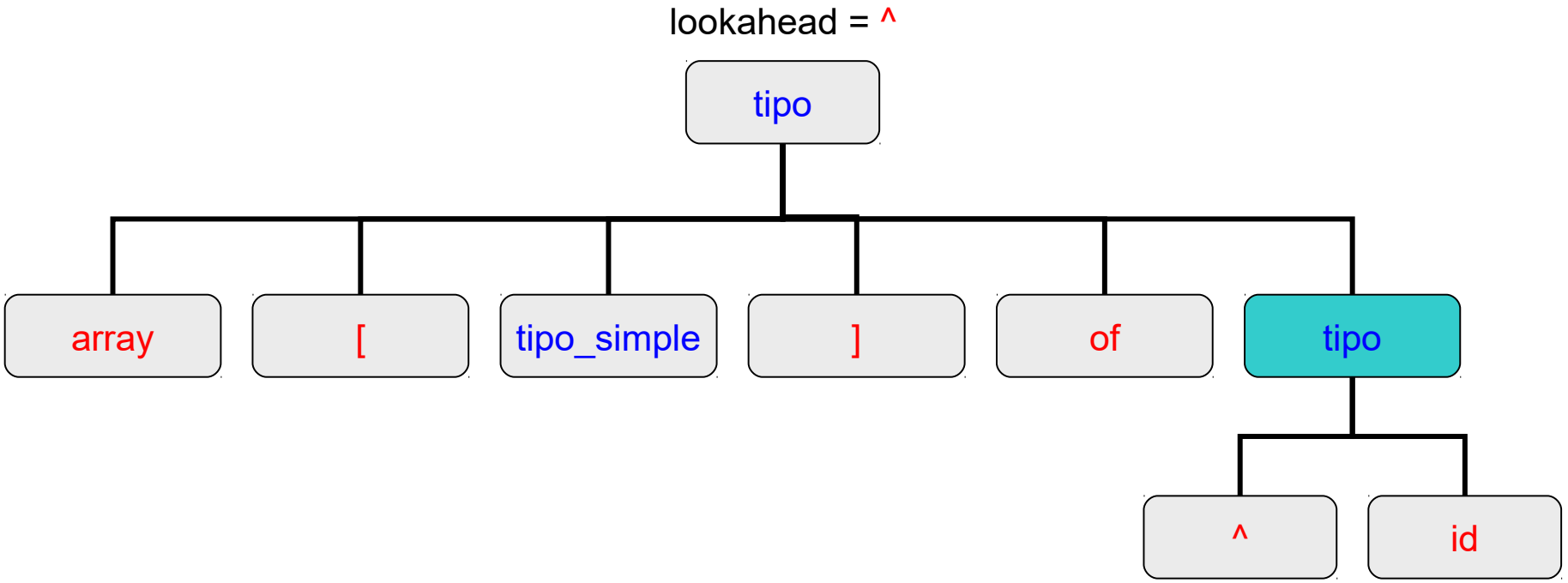
array [1..10] of ^ persona



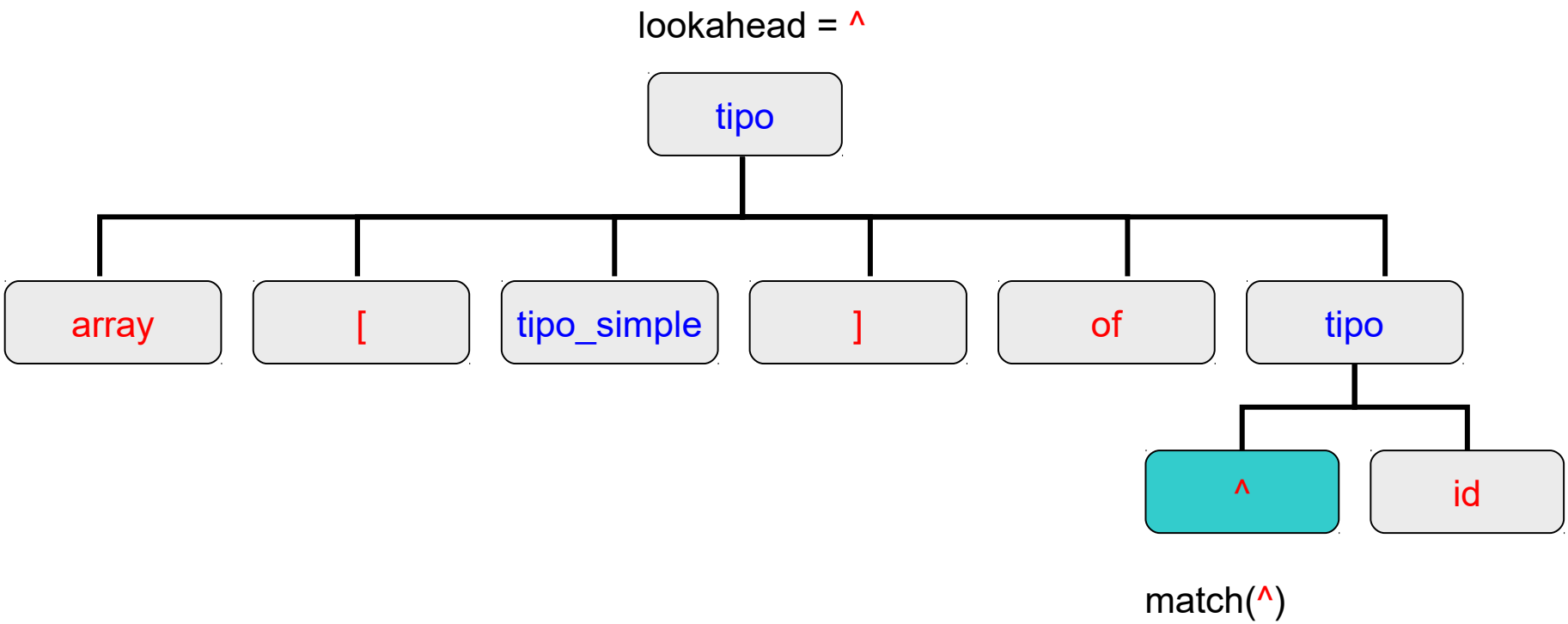
array [1..10] of ^ persona



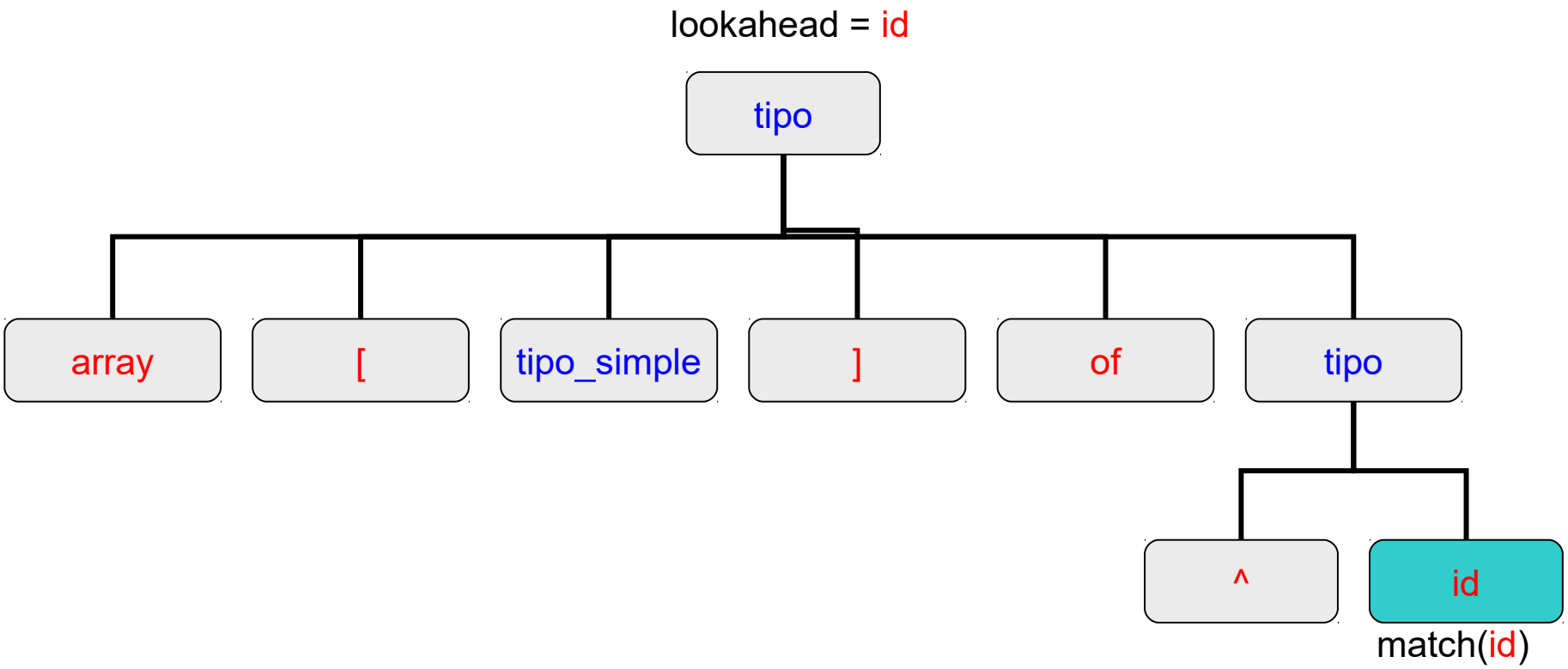
array [1..10] of ^ persona



array [1..10] of ^ persona



array [1..10] of ^ persona



Análisis predictivo
PRIMERO (I)

- Si X es un símbolo terminal, entonces PRIMERO(X) es {X}.

- Si X$\rightarrow$ ξ es una producción, entonces añadir ξ a PRIMERO(X).

Análisis predictivo PRIMERO (II)

- Si X es un no-terminal y $X \rightarrow Y_1 Y_2 \ldots Y_k$ es una producción,
  entonces añadir *b* a PRIMERO(X) si $b \neq \xi$ y existe algún i tal que:
  - *b* pertenece a PRIMERO(Y_i) y
  - ξ pertenece a PRIMERO(Y_1),..., PRIMERO(Y_{i-1}),
    - esto es, si a partir de $Y_1 Y_2 \ldots Y_{i-1}$ se puede derivar ξ
- Finalmente, habrá que añadir ξ a PRIMERO(X) si
  $\xi \in$ PRIMERO(Y_i) $\forall i \; 1 \leq i \leq k$

Análisis predictivo SIGUIENTE (I)

- Se define SIGUIENTE(A), **para un no terminal A**, como el conjunto de terminales *a* que pueden aparecer inmediatamente a la derecha de A en una forma sentencial.

Análisis predictivo SIGUIENTE (II)

1. Introducir # en SIGUIENTE(S), donde S es el símbolo inicial de la gramática y # representa el final de la cinta de entrada.

2. Si existe una producción de la forma $A \rightarrow \alpha B \beta$, entonces introducir todos los símbolos de PRIMERO(β), excepto ξ, en SIGUIENTE(B).

3. Si existe una producción de la forma $A \rightarrow \alpha B \beta$, entonces, si PRIMERO($\beta$) contiene la ξ, introducir los símbolos de SIGUIENTE(A) en SIGUIENTE(B).

Condiciones LL(1)

- Una gramática G es LL(1) si y sólo si para cualquier par de producciones de la forma $A \rightarrow \alpha \mid \beta$ se cumplen las siguientes condiciones:

  1. No existe ningún terminal *a* que pueda iniciar cadenas derivadas a partir de α y β.

  2. Como máximo sólo una parte derecha, α ó β pueden derivar la ξ.

  3. Si $\alpha \overset{*}{\Rightarrow} \xi$, entonces a partir de A no se deriva ninguna cadena que comience por algún terminal perteneciente a SIGUIENTE(A).

Expresión de las condiciones LL(1)

- Para cualquier par de producciones de la forma $A \rightarrow \alpha \mid \beta$ se cumplen las siguientes condiciones:
  1. $PRIMERO(\alpha) \cap PRIMERO(\beta) = \varnothing$
  2. Si $\alpha \overset{*}{\Rightarrow} \xi$ (ó $\beta \overset{*}{\Rightarrow} \xi$) entonces

     $SIGUIENTE(A) \cap PRIMERO(A) = \varnothing$

Transformación de una gramática en predictiva

Acciones a realizar en la gramática para conseguir un analizador predictivo:

1. Eliminar la ambigüedad
2. Eliminar la recursividad a izquierdas
3. Factorizar

Eliminación de la recursividad a izquierdas inmediata

- Gramática recursiva a izquierdas

  $A \overset{+}{\Rightarrow} A\alpha$, donde α pertenece a $(\Sigma \cup N)^*$

- Recursividad a izquierdas inmediata

  $A \rightarrow A\ \alpha_1 | A\ \alpha_2 | \ldots | A\ \alpha_m | \beta_1 | \beta_2 | \ldots | \beta_n$

  $\forall 1 \leq i \leq n\ \beta_i$ no comienza con A

- Estas reglas pueden transformarse en:

  $A \rightarrow \beta_1\ A' | \beta_2\ A' | \ldots | \beta_n\ A'$

  $A' \rightarrow \alpha_1\ A' | \alpha_2\ A' | \ldots | \alpha_m\ A' | \xi$

- $\forall$ i $1 \leq i \leq m\ \alpha_i \neq \xi$ (no hay producciones unitarias)
- $\forall$ j $1 \leq j \leq n\ \beta_j \neq \xi$ (no hay producciones vacías)

Eliminación de la recursividad a izquierdas-General

- **Enumerar los no-terminales.**
- **Aplicar el algoritmo de eliminación:**

para i **desde** 1 **hasta** n **hacer**

para j **desde** 1 **hasta** i -1 **hacer**

Remplazar cada $A_i \rightarrow A_j \gamma$

por las producciones $A_i \rightarrow \delta_1\gamma | \ldots | \delta_k\gamma$,

donde $A_j \rightarrow \delta_1 | \ldots | \delta_k$ son todas las A_j-prod.;

fin para

Eliminar la recursividad a izquierdas inmediata

sobre las A_i-producciones

fin para

Otra vez las expresiones

$E \rightarrow T\ E'$

$E' \rightarrow +\ T\ E'$

$\quad | -\ T\ E'$

$\quad | \xi$

$T \rightarrow F\ T'$

$T' \rightarrow *\ F\ T'$

$\quad | /\ F\ T'$

$\quad | \xi$

$F \rightarrow$ **entero**

$\quad |$ **id**

Factorización a izquierdas

Buscar el prefijo común más largo a las producciones de A, α

si $\alpha \neq \xi$,

sustituir $A \rightarrow \alpha\beta_1 \mid ... \mid \alpha\beta_n \mid \gamma$

por $A \rightarrow \alpha A' \mid \gamma$

donde $A' \rightarrow \beta_1 \mid ... \mid \beta_n$

y γ representa todas las producciones que no tienen como prefijo α

Análisis descendente predictivo **recursivo**

- Grámatica LL(1)
- Un procedimiento por cada no terminal
- Por cada terminal aplicar la función *match*
- Selección de la regla a aplicar: *lookahead*

Análisis descendente predictivo recursivo

```
procedure match(token: tipo_token);
begin
        if lookahead = token
                then  lookahead := siguiente_token()
        else  error
end;
```

Análisis descendente predictivo recursivo

- Un procedimiento para cada no-terminal

  Decidir qué producción usar.

  Ejecutar una serie de acciones construidas miméticamente con respecto a las partes derechas de las producciones.

  - No-terminales: Llamada a su procedimiento.
  - Terminales: Verificar que el token de entrada se empareja con el que corresponde (procedimiento match).

Análisis descendente predictivo recursivo $\xi \notin$ PRIMERO(A)

```
procedure A;
begin
   if lookahead pertenece a PRIMERO(α) then
         -- analizar α
   else if lookahead pertenece a PRIMERO(β) then
                                  -- analizar β
   else
               -- tratar error
   end if;
end A;
```

Ejemplo 1

tipo $\rightarrow$ tipo_simple

| $\uparrow$id

| array [tipo_simple] of tipo

tipo_simple $\rightarrow$ integer

| char

| num punto punto num

Ejemplo 1

```
procedure tipo;
begin
    if lookahead en { integer char num }
        then tipo_simple;
    else if lookahead en { ↑ }
        then match(↑); match(id);
    else if lookahead en { array }
        then match(array); match(‘[‘);
                tipo_simple; match(‘]’);
                match(of); tipo;
    else tratar error;
end;
```

Ejemplo 1

```
procedure tipo_simple;
begin
    if lookahead en { integer }
        then    match(integer);
    else if lookahead en { char }
        then    match(char);
    else if lookahead en { num }
        then    match(num);
                match(punto); match(punto);
                match(num);
    else tratar error;
end;
```

Análisis descendente predictivo recursivo $\xi \in$ PRIMERO(A)

```
procedure A;
begin
   if lookahead pertenece a PRIMERO(α) then
         -- analizar α
   else if lookahead pertenece a PRIMERO(β) then
                                  -- analizar β
   else if lookahead pertenece a SIGUIENTE(A) then
        -- no hacer nada (producción vacía)
   else
              -- tratar error
   end if;
end A;
```

Ejemplo 2

$E \rightarrow T\ E'$

$E' \rightarrow +\ T\ E'$

$\quad | -\ T\ E'$

$\quad | \xi$

$T \rightarrow F\ T'$

$T' \rightarrow *\ F\ T'$

$\quad | /\ F\ T'$

$\quad | \xi$

$F \rightarrow$ **entero**

$\quad |$ **id**

Ejemplo 2

```
procedure E is
begin
  if -- lookahead ∈ {entero, id} -- then
     T;
     E_PRIMA;
  else
     -- tratar error
  end if;
end E;
```

Ejemplo 2

```
procedure E_PRIMA is
begin
    if -- lookahead ∈ {+} -- then
        match(+);
        T; E_PRIMA;
    else if -- lookahead ∈ {-} -- then
        match(-);
        T; E_PRIMA;
    else if -- lookahead ∈ {#} -- then
        -- no hacer nada!!
    else -- tratar error
    end if;
end E_ PRIMA;
```

Análisis descendente predictivo **no recursivo**

- Pila de análisis: elementos por analizar
- Tabla de análisis sintáctico:

  matriz(no terminales, terminales)
- Entrada para un no-terminal A y un terminal *a*:

  regla a aplicar ó

  error

Construcción de tablas de análisis descendente no recursivo (II)

Entrada: Una gramática G

Salida: La tabla de análisis sintáctico M

para cada no-terminal A de la gramática G
  para cada regla de la forma $A \rightarrow \alpha$
    para cada terminal *a* (incluido la marca de final #)
      si a $\in$ PRIMERO(α) o
        $\xi \in$ PRIMERO(α) y a $\in$ SIGUIENTE(A)
      entonces M[A,a] := $A \rightarrow \alpha$
      fin si
    fin para
  fin para
fin para

Construcción de tablas de análisis descendente no recursivo (II)

- Las entradas no definidas corresponden a situaciones de error
- Si en el proceso de construcción de la tabla se intenta redefinir una entrada de la tabla, la gramática no cumple las condiciones LL(1)

Algoritmo de análisis sintáctico descendente no recursivo (I)

- **Entrada**: La tabla de análisis sintáctico M y la entrada a analizar

- **Salida**: Si la entrada pertenece al lenguaje definido por la gramática, la secuencia de derivación a izquierdas de la entrada; si no un aviso de error

Algoritmo de análisis sintáctico descendente no recursivo (II)

empezar
Lookahead apunta al primer símbolo de la entrada
La pila de análisis ha sido inicializada con S#
repetir
X := cima de la pila
si X es un terminal **entonces** -- emparejar (X)
si lookahead = X **entonces**
desempilar X; avanzar lookahead;
si no avisar del error
fin si
si no -- X es un no terminal

Algoritmo de análisis sintáctico descendente no recursivo (III)

si no -- X es un no terminal
    si M(X,lookahead) = X$\rightarrow\alpha$ **entonces**
        desempilar X;
        empilar los símbolos de α (de dcha. a izda.)
        -- ξ no es un símbolo
    si no Avisar del error y recuperar el control
    f**in si**
fin si
hasta (X = # y lookahead = #)
fin

3.4 Análisis ascendente

- **Analizadores LR**

  Análisis por reducción-desplazamiento (R/D)

  Construyen el árbol de análisis partiendo de las hojas hasta llegar a la raíz

  El token en curso que está siendo examinado en un momento dado se denomina *lookahead*

  Un único lookahead LR(1)

  Proceso similar a la inversa de la derivación a derechas de la cadena de entrada

Analizadores LR

- Un *handle* de una cadena es una subcadena que coincide con la parte derecha de una regla y cuya **reducción** por el no-terminal que se encuentra a la izquierda en dicha regla constituye un paso correcto en la **inversa de la derivación a derechas**

- Para obtener la inversa de la derivación a derechas bastará con ir localizando *los handles* en las formas sentenciales obtenidas sucesivamente

Ejemplos

- **Ejemplo 1**

  $S \rightarrow aABe$
  $A \rightarrow Abc \mid b$
  $B \rightarrow d$

$S \Rightarrow aABe \Rightarrow aAde \Rightarrow aAbcde \Rightarrow aAbcbcde \Rightarrow abbcbcde$

- **Ejemplo 2**

  $A \rightarrow BC$
  $B \rightarrow bb$
  $C \rightarrow cA \mid d$

$A \Rightarrow BC \Rightarrow BcA \Rightarrow BcBC \Rightarrow BcBd \Rightarrow Bcbbd \Rightarrow bbcbbd$

Implementación de un analizador por R/D usando una pila

- Inicialmente la pila está vacía (o contiene una marca) y en la entrada está la cadena a analizar. El objetivo es sintetizar en el tope de la pila el símbolo inicial de la gramática coincidiendo con el momento en que se alcanza el final de la cadena de entrada.

Pila	**Entrada**
$	w$
...	
$S	$

- El proceso de análisis se basa en el uso de cuatro operaciones: reducir, desplazar, aceptar y error.

Ejemplo 1

Pila	**Entrada**	**Acción**
$	abbcbcde$	desplazar
$a	bbcbcde$	desplazar
$a**b**	bcbcde$	reducir A $\rightarrow$ b
$aA	bcbcde$	desplazar
$aAb	cbcde$	desplazar
$a**Abc**	bcde$	reducir A $\rightarrow$ Abc
$aA	bcde$	desplazar
$aAb	cde$	desplazar
$a**Abc**	de$	reducir A $\rightarrow$ Abc
$aA	de$	desplazar
$aA**d**	e$	reducir B $\rightarrow$ d
$aAB	e$	desplazar
$aABe	$	reducir S $\rightarrow$ aABe
$S	$	aceptar

Ejemplo 2

Pila	**Entrada**	**Acción**
$	bbcbbd$	desplazar
$b	bcbbd$	desplazar
$bb	cbbd$	reducir B → bb
$B	cbbd$	desplazar
$Bc	bbd$	desplazar
$Bcb	bd$	desplazar
$Bc**bb**	d$	reducir B → bb
$BcB	d$	desplazar
$BcB**d**	$	reducir C → d
$Bc**BC**	$	reducir A → BC
$B**cA**	$	reducir C → cA
$BC	$	reducir A → BC
$A	$	aceptar

Ejemplo 3

1. S → **begin** I L_I **end**
2. L_I → **;** I L_I
3. | ξ
4. I → ID_ASIG **:=** REF_TABLA
5. ID_ASIG → **id**
6. REF_TABLA → ID_TABLA **[entero]**
7. ID_TABLA → **id**

Ejemplo 3

Pila	Entrada	Acción
$	begin id := id [entero] end$	empilar
$ begin	id := id [entero] end$	empilar
$ begin id	:= id [entero] end$	**reducir**
$ begin ID_ASIG	:= id [entero] end$	empilar
$ begin ID_ASIG:=	id [entero] end$	empilar
$ begin ID_ASIG:=id	[entero] end$	**reducir**
$ ... :=ID_TABLA	[entero] end$	empilar
$ ... :=ID_TABLA[	entero] end$	empilar
$ ...:=ID_TABLA[entero	] end$	empilar
$ ...:=ID_TABLA[entero]	end$	reducir
$ ...:=REF_TABLA	end$	reducir
$ begin I	end$	**reducir**
$ begin I L_I	end$	empilar
$ begin I L_I end	$	reducir
$ S $	aceptar	

Análisis LR

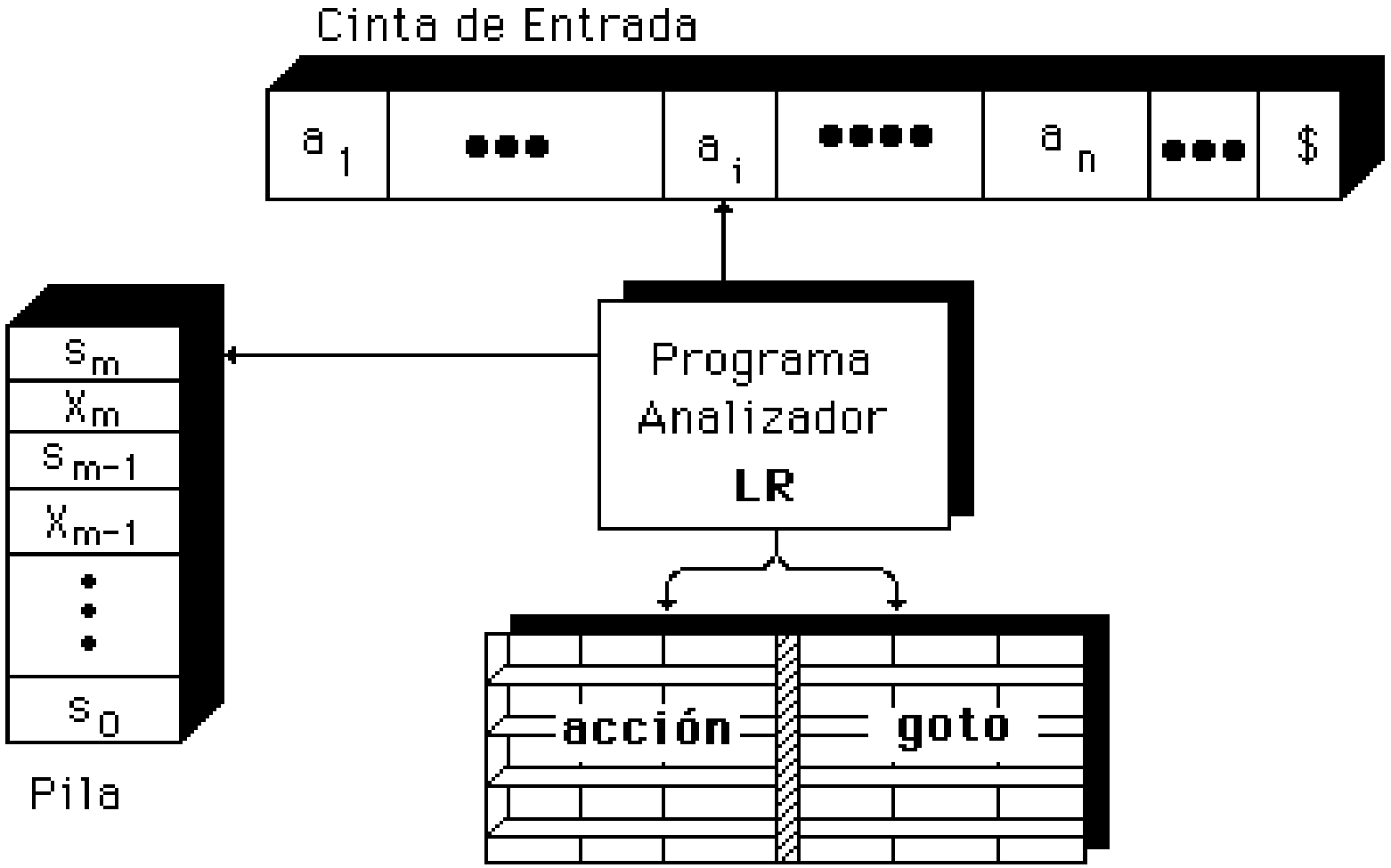
Permiten reconocer una amplia gama de los lenguajes descritos por medio de gramáticas independientes del contexto.

Es el método de análisis por R/D más general de los que no usan vuelta-atrás, además permite una implementación eficiente.

La clase de gramáticas a partir de las cuales es posible construir un analizador descendente predictivo es un subconjunto de la clase de gramáticas LR.

Existen tres técnicas de construcción de tablas de análisis LR: SLR, CLR y LALR.

Análisis LR


Cinta de Entrada
a 1
a i
a n
$
Programa
Analizador
LR
s m
X m
s m-1
X m-1
s 0
Pila
acción
goto

Análisis LR

estado pila tope estado entrada

configuración $(s_0 X_1 s_1 \ldots X_m s_m,\ a_i \ldots a_n\ \$)$

$X_1\ X_2\ \ldots\ X_m\ a_i\ a_{i+1}\ \ldots\ a_n$ **forma sentencial actual**

movimiento: símbolo de entrada actual $\mathbf{a_i}$

estado en el tope de la pila $\mathbf{s_m}$ **acción**$[s_m, a_i]$

1. Si **acción** $[s_m, a_i]$ **= desplazar s**
   $(s_0 X_1 s_1 \ldots X_m s_m\ a_i\ s,\ \ a_{i+1} \ldots a_n\ \$)$
2. Si **acción** $[s_m, a_i]$ **= reducir** $A \rightarrow \beta$ $(s_0 X_1 s_1 \ldots X_{m-r}\ s_{m-r}\ A\ s,\ \ a_i \ldots a_n\ \$)$
   donde **s = goto** $[s_{m-r}, A]$ y **r** es la longitud de β
3. Si **acción** $[s_m, a_i]$ **= aceptar** indica el final del análisis
4. Si **acción** $[s_m, a_i]$ **= error**, el analizador ha descubierto un error y llama a la rutina de recuperación de errores

Análisis LR

Entrada: Una cadena de entrada **w** y una tabla de análisis LR con las funciones *acción* y *goto* para la gramática **G.**

Salida: Si **w** pertenece a **L(G)** un *análisis ascendente*, en caso contrario un mensaje **de error.**

Método: Inicialmente, el analizador almacena s_o en la pila, donde s_o es el estado inicial, y *w#* es el contenido de la entrada. El analizador ejecuta el programa siguiente hasta que la cadena es reconocida, o hasta que se detecta un error.

Algoritmo de análisis LR

inicializar **ip** con el primer símbolo de **w#**;

repetir

s es el estado en el tope de la pila y

a el símbolo apuntado por **ip**;

si acción [s, a] = desplazar s' **entonces**

empilar **a** , **s'** en el tope de la pila;

avanzar **ip** al siguiente símbolo de la entrada

si no si acción [s, a] = reducir A $\rightarrow$ β **entonces**

desempilar 2 * | β | símbolos de la pila;

s' es ahora el estado del tope de la pila;

empilar **A**, y empilar *goto* [s', A];

si no si acción [s, a] = aceptar **entonces salir**

si no error ()

fin si

fin repetir

Algoritmo de análisis LR

ESTADO	acción						goto		
	id	+	*	(	)	$	E	T	F
0	s_5			s_4			1	2	3
1		s_6				acc			
2		r_2	s_7		r_2	r_2			
3		r_4	r_4		r_4	r_4			
4	s_5			s_4			8	2	3
5		r_6	r_6		r_6	r_6			
6	s_5			s_4				9	3
7	s_5			s_4					10
8		s_6			s_{11}				
9		r_1	s_7		r_1	r_1			
10		r_3	r_3		r_3	r_3			
11		r_5	r_5		r_5	r_5			

E	$\rightarrow$ E + T	(1)
	| T	(2)
T	$\rightarrow$ T * F	(3)
	| F	(4)
F	$\rightarrow$ (E)	(5)
	| **id**	(6)

Algoritmo de análisis LR

PILA	INPUT	ACCIÓN
(1) 0	id * id + id $	empilar
(2) 0 id 5	* id + id $	reducir por F → id
(3) 0 F 3	* id + id $	reducir por T → F
(4) 0 T 2	* id + id $	empilar
(5) 0 T 2 * 7	id + id $	empilar
(6) 0 T 2 * 7 id 5	+ id $	reducir por F → id
(7) 0 T 2 * 7 F 10	+ id $	reducir por T → T * F
(8) 0 T 2	+ id $	reducir por E → T
(9) 0 E 1	+ id $	empilar
(10) 0 E 1 + 6	id $	empilar
(11) 0 E 1 + 6 id 5	$	reducir por F → id
(12) 0 E 1 + 6 F 3	$	reducir por T → F
(13) 0 E 1 + 6 T 9	$	reducir por E → E + T
(14) 0 E 1	$	aceptar

Gramáticas LR

- Una gramática será LR si es posible construir para ella una tabla de análisis que dirija el proceso de análisis por reducción-desplazamiento.
- Un analizador LR **no** necesita recorrer toda la pila para saber si en el tope de ésta se encuentra un *handle.*
- EL ESTADO EN EL TOPE DE LA PILA CONTIENE LA INFORMACIÓN NECESARIA.
- La función GOTO de una tabla de análisis LR es básicamente un autómata finito, capaz de reconocer la aparición de un *handle* en el tope de la pila.
- Otra fuente de información es el *lookahead.*
- Las gramáticas LR pueden describir más lenguajes que las LL.

I_0: $E' \to \cdot E$
$E \to \cdot E + T$
$E \to \cdot T$
$T \to \cdot T * F$
$T \to \cdot F$
$F \to \cdot (E)$
$F \to \cdot \mathbf{id}$

I_1: $E' \to E \cdot$
$E \to E \cdot + T$

I_2: $E \to T \cdot$
$T \to T \cdot * F$

I_3: $T \to F \cdot$

I_4: $F \to (\cdot E)$
$E \to \cdot E + T$
$E \to \cdot T$
$T \to \cdot T * F$
$T \to \cdot F$
$F \to \cdot (E)$
$F \to \cdot \mathbf{id}$

I_5: $F \to \mathbf{id} \cdot$

I_6: $E \to E + \cdot T$
$T \to \cdot T * F$
$T \to \cdot F$
$F \to \cdot (E)$
$F \to \cdot \mathbf{id}$

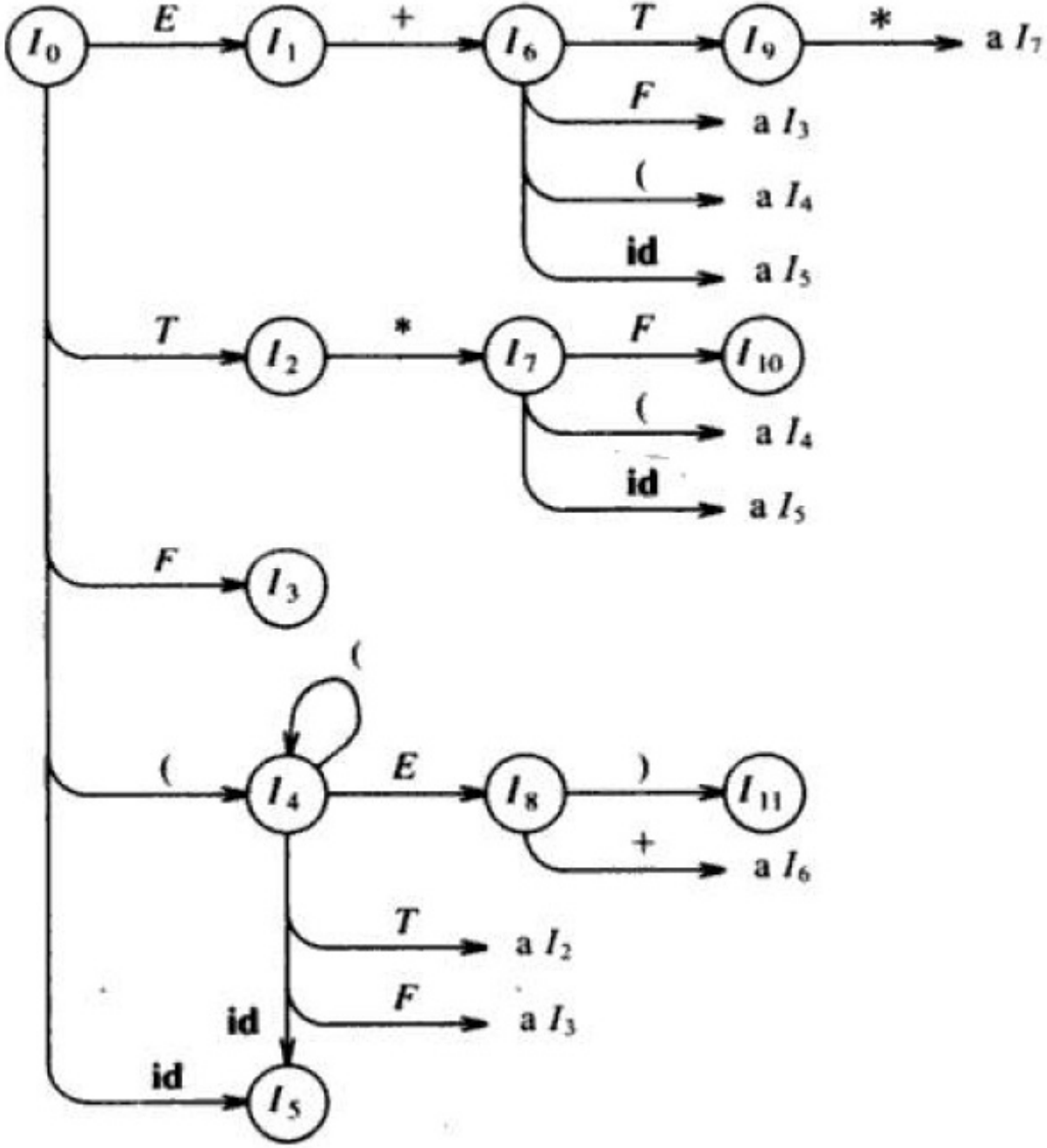
I_7: $T \to T * \cdot F$
$F \to \cdot (E)$
$F \to \cdot \mathbf{id}$

I_8: $F \to (E \cdot)$
$E \to E \cdot + T$

I_9: $E \to E + T \cdot$
$T \to T \cdot * F$

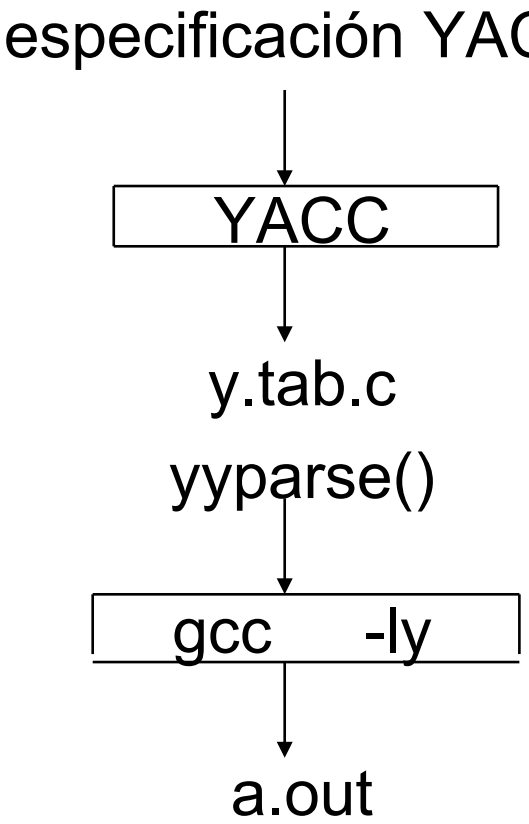
I_{10}: $T \to T * F \cdot$

I_{11}: $F \to (E) \cdot$


I_0
E
I_1
+
I_6
T
I_9
*
a I_7
F
a I_3
(
a I_4
id
a I_5
T
I_2
*
I_7
F
I_{10}
(
a I_4
id
a I_5
F
I_3
(
(
I_4
E
I_8
)
I_{11}
+
a I_6
T
a I_2
F
a I_3
id
id
I_5

3.5 YACC: generador de analizadores sintácticos

COMPILADOR DE COMPILADORES
GENERADOR DE ANALIZADORES
GENERADOR DE TRADUCTORES



Especificación YACC

DECLARACIONES

%%

REGLAS

%%

PROGRAMAS

declaraciones

Código C: Declaración de variables,...

Declaración de nombres de token: %token

Prioridades: %left,...

Símbolo inicial: %start

Especificación YACC

reglas

lista de:

regla de producción ACCION

a : PD1 {código C};
a : PD2;

a : PD1 {código C}
| PD2;

PDi Parte derecha de la regla

Acciones

- Pueden obtener valores y utilizar los obtenidos por otras acciones.
- Pueden aparecer en puntos intermedios de la parte derecha de una regla.
- { }
- Cálculo y uso de atributos a través de: $$, $1, $2...
- Acción por defecto: $$ = $1
- Manejo de objetos globales

Analizador léxico

- Trabajo en combinación con LEX
- yylex
- yylval (atributos)
- Referencias a nombres de tokens
- Definición previa en la sección de declaraciones o posterior en la sección de programas.
- Nombres de tokens ≠ Palabras reservadas de C

Funcionamiento del analizador generado

- Cuando se reduce se ejecuta la acción asociada a la regla, se desempilan los atributos asociados a los símbolos de la parte derecha y se empilan los atributos del símbolo de la parte izquierda.

Ambigüedad y conflictos

- YACC genera el analizador aunque la gramática no sea LALR.
- Reglas para la resolución de conflictos:

  Conflicto R/D $\rightarrow$ D

  Conflicto R/R $\rightarrow$ Reducción por medio de la primera regla en la secuencia de entrada
- YACC avisa del número de conflictos detectados y puede, opcionalmente, documentarlos.

Precedencia

expr : expr Op expr

expr : Op-unario expr

- Definición de prioridad de operadores (de menor a mayor y asociatividad):

```
%left  '+' '-'
%left '*' '/'
```

3.6 Tratamiento de errores sintácticos

- Detección del error
- Aviso del error
- Recuperación del error

Recuperación del error

- Modo de pánico
- Producciones de error

Modo pánico

- Saltar hasta encontrar un punto donde se pueda continuar el análisis.
- Sencillo
- Correcto, eligiendo conjuntos de sincronización adecuados.
- Desventaja: omite el análisis de parte de la entrada.

panic mode (bison)

- error (token reservado de bison)
- Sirve para marcar las posiciones de recuperación de un error

panic mode (bison)

- Los puntos de sincronización más habituales son los signos de puntuación entre sentencias.
- Para recuperarse de un error en una sentencia:

```
listasentencias : sentencia ';'
   |  listasentencias sentencia ';'
   | error ';'                      { yyerror("...");  }
   | listasentencias error ';'  { yyerror("...");  }
   ;
```

panic mode (bison)

- Cuando detecta un error:
  - » Dar un mensaje de error
  - » El analizador desecha tokens hasta poder emparejar con un token que aparezca después del token *error.*
  - » Empila *error*
  - » Continúa el análisis (reducirá por una regla que contenga el token *error*).

panic mode (bison)

- Cuando detecta un error:
  - » Dar un mensaje de error y empila *error*
  - » El analizador desecha tokens hasta poder emparejar con un token que aparezca después del error
    - – Siguiente(error) o conjunto de sincronización
  - » Continúa el análisis (reducirá por una regla que contenga el token *error*)
  - » No da otro error hasta analizar correctamente varios tokens

panic mode (bison)

¿Dónde se ponen los puntos de sincronización?

- En las reglas de niveles altos
  - » Ventaja: Siempre podrán sincronizarse
- En las reglas de niveles bajos
  - » Ventaja: Se desechan pocos tokens
- Suelen ser:
  - » Símbolos de puntuación , ; { } ...