EXAMEN DE COMPILADORES (2° Grado en Informática, final julio-2016)

FACULTAD DE INFORMÁTICA

SOLUCIONES

Parte I: PREGUNTAS TIPO TEST. 30 %.

1. a)	4. a)	7. b)	10. c)	13. b)
2. b)	5. b)	8. c)	11. a)	14. c)
3. a)	6. b)	9. c)	12. a)	15. c)

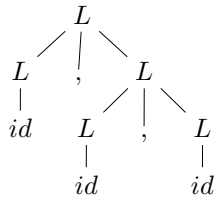
Parte II: PROBLEMA. 70 %.

Apartado 1.

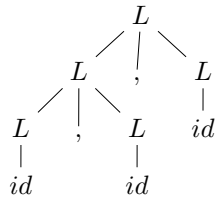
La gramática tiene no es LL(1) por las siguientes razones:

- Es ambigua debido a la regla $L \to L, L$ ya que no establece la asociatividad de la coma. Podemos comprobarlo con una cadena de símbolos como id, id, id para la cual existen dos derivaciones más a la derecha, y dos árboles distintos (centrando la derivación en las reglas de L):

$$\begin{aligned} L &\Rightarrow L\ ,\ L \\ &\Rightarrow L\ ,\ L\ ,\ L \\ &\Rightarrow L\ ,\ L\ ,\ id \\ &\Rightarrow L\ ,\ id\ ,\ id \\ &\Rightarrow id\ ,\ id\ ,\ id \end{aligned}$$



$$\begin{aligned} L &\Rightarrow L\ ,\ L \\ &\Rightarrow L\ ,\ id \\ &\Rightarrow L\ ,L\ ,\ id \\ &\Rightarrow L\ ,\ id\ ,\ id \\ &\Rightarrow id\ ,\ id\ ,\ id \end{aligned}$$



- Tiene recursividad inmediata por la izquierda en la regla $L \to L, L$.
- Tiene el factor común C en las tres reglas de E.

Modificamos la gramatica comenzando por la eliminación del problema de ambigüedad. Consideramos la coma como un operador, empleando una solución similar a la usada con las gramáticas de expresiones aritméticas. Teniendo en cuenta que la asociatividad es por la derecha, podemos llegar a esta gramática equivalente:

$$\begin{aligned} E &\to C \mid C \cup E \mid C \cap E \\ C &\to \{\ L\ \} \mid \emptyset \\ L &\to T\ ,\ L \mid T \\ T &\to id \end{aligned}$$

El no terminal T no tiene gran utilidad, porque en esta gramática sólo hay una derivación al terminal id. Por tanto, podemos llegar a esta gramática equivalente sin T:

$$\begin{aligned} E &\to C \mid C \cup E \mid C \cap E \\ C &\to \{\ L\ \} \mid \emptyset \\ L &\to id\ ,\ L \mid id \end{aligned}$$

La eliminación de la ambigüedad ha permitido, simultáneamente, eliminar la recursión inmediata de L, pero ha introducido factores comunes que pasamos ahora a eliminar:

$$\begin{aligned} E &\to C\ E' \\ E' &\to \lambda \mid \cup E \mid \cap E \\ C &\to \{\ L\ \} \mid \emptyset \\ L &\to id\ L' \\ L' &\to ,\ L \mid \lambda \end{aligned}$$

Para evitar confusión con la notación de los conjuntos PRIMERO, SIGUIENTE y PREDICT, se han sustituido los símbolos { y } por ⟨ y ⟩. Los conjuntos PRIMERO y SIGUIENTE de la gramática modificada son:

PRIMERO$(E) = \{ \langle \emptyset \}$
PRIMERO$(E') = \{ \lambda \cup \cap \}$
PRIMERO$(C) = \{ \langle \emptyset \}$
PRIMERO$(L) = \{ id \}$
PRIMERO$(L') = \{ , \lambda \}$

SIGUIENTE$(E) = \{ \$ \}$
SIGUIENTE$(E') = \{ \$ \}$
SIGUIENTE$(C) = \{ \cup \cap \$ \}$
SIGUIENTE$(L) = \{ \rangle \}$
SIGUIENTE$(L') = \{ \rangle \}$

Los conjuntos PREDICT de las reglas de producción de la gramática modificada son:

PREDICT$(E \to C\ E') = \{ \langle \emptyset \}$
PREDICT$(E' \to \cup\ E) = \{ \cup \}$
PREDICT$(E' \to \cap\ E) = \{ \cap \}$
PREDICT$(E' \to \lambda) = \{ \$ \}$
PREDICT$(C \to \langle\ L\ \rangle) = \{ \langle \}$

PREDICT$(C \to \emptyset) = \{ \emptyset \}$
PREDICT$(L \to id\ L') = \{ id \}$
PREDICT$(L' \to \lambda) = \{ \rangle \}$
PREDICT$(L' \to ,\ L) = \{ , \}$

Observando los PREDICT de las reglas con el mismo no terminal en la cabeza, podemos comprobar que todos tienen una intersección nula. Esto implica que no se va a producir ningún conflicto al generar la tabla LL(1). Por tanto, la gramática es LL(1).

Apartado 2.

En este apartado también se han sustituido los símbolos { y } por $\langle$ y $\rangle$ para evitar confusión con la notación del conjunto de ítems LR(1). Partiendo de la gramática inicial, calculamos la colección LR(1):

$I_0 = \{ [\ E' \to \cdot\ E\ ,\ \$\ ]$
$[\ E \to \cdot\ C\ ,\ \$\ ]$
$[\ E \to \cdot\ C\ \cup\ E\ ,\ \$\ ]$
$[\ E \to \cdot\ C\ \cap\ E\ ,\ \$\ ]$
$[\ C \to \cdot \langle\ L\ \rangle\ ,\ \$/\cup/\cap\ ]$
$[\ C \to \cdot\ \emptyset\ ,\ \$/\cup/\cap\ ]\ \}$

$I_1 = \text{GOTO}(I_0, E) = \{ [\ E' \to E \cdot\ ,\ \$\ ]\ \}$

$I_2 = \text{GOTO}(I_0, C) = \{ [\ E \to C \cdot\ ,\ \$\ ]$
$[\ E \to C\ \cdot \cup E\ ,\ \$\ ]$
$[\ E \to C\ \cdot \cap E\ ,\ \$\ ]\ \}$

$I_3 = \text{GOTO}(I_0, \langle\ ) = \{ [\ C \to \langle \cdot\ L\ \rangle\ ,\ \$/\cup/\cap\ ]$
$[\ L \to \cdot\ id\ ,\ \rangle/,\ ]$
$[\ L \to \cdot\ L\ ,\ L\ ,\ \rangle/,\ ]\ \}$

$I_4 = \text{GOTO}(I_0, \emptyset) = \{ [\ C \to \emptyset \cdot\ , \$/\cup/\cap\ ]\ \}$

$I_5 = \text{GOTO}(I_2, \cup) = \{ [\ E \to C\ \cup \cdot\ E\ ,\ \$\ ]$
$[\ E \to \cdot\ C\ ,\ \$\ ]$
$[\ E \to \cdot\ C\ \cup\ E\ ,\ \$\ ]$
$[\ E \to \cdot\ C\ \cap\ E\ ,\ \$\ ]$
$[\ C \to \cdot \langle\ L\ \rangle\ ,\ \$/\cup/\cap\ ]$
$[\ C \to \cdot\ \emptyset\ ,\ \$/\cup/\cap\ ]\ \}$

$I_6 = \text{GOTO}(I_2, \cap) = \{ [\ E \to C\ \cap \cdot\ E\ ,\ \$\ ]$
$[\ E \to \cdot\ C\ ,\ \$\ ]$
$[\ E \to \cdot\ C\ \cup\ E\ ,\ \$\ ]$
$[\ E \to \cdot\ C\ \cap\ E\ ,\ \$\ ]$
$[\ C \to \cdot \langle\ L\ \rangle\ ,\ \$/\cup/\cap\ ]$
$[\ C \to \cdot\ \emptyset\ ,\ \$/\cup/\cap\ ]\ \}$

$I_7 = \text{GOTO}(I_3, L) = \{ [\ C \to \langle\ L \cdot\ \rangle\ ,\ \$/\cup/\cap\ ]$
$[\ L \to L \cdot\ ,\ L\ ,\ \rangle/,\ ]\ \}$

$I_8 = \text{GOTO}(I_3, id) = \{ [\ L \to id \cdot\ ,\ \rangle/,\ ]\ \}$

$I_9 = \text{GOTO}(I_5, E) = \{ [\ E \to C\ \cup\ E \cdot\ ,\ \$\ ]\ \}$

$\text{GOTO}(I_5, C) = I_2$
$\text{GOTO}(I_5, \langle\ ) = I_3$
$\text{GOTO}(I_5, \emptyset) = I_4$

$I_{10} = \text{GOTO}(I_6, E) = \{ [\ E \to C\ \cap\ E \cdot\ ,\ \$\ ]\ \}$

$\text{GOTO}(I_6, C) = I_2$
$\text{GOTO}(I_6, \langle\ ) = I_3$
$\text{GOTO}(I_6, \emptyset) = I_4$

$I_{11} = \text{GOTO}(I_7, \rangle) = \{ [\ C \to \langle\ L\ \rangle \cdot\ ,\ \$/\cup/\cap\ ]\ \}$

$I_{12} = \text{GOTO}(I_7, \ ,) = \{ [\ L \to L\ ,\ \cdot\ L\ ,\ \rangle/,\ ]$
$[\ L \to \cdot\ id\ ,\ \rangle/,\ ]$
$[\ L \to \cdot\ L\ ,\ L\ ,\ \rangle/,\ ]\ \}$

$I_{13} = \text{GOTO}(I_{12}, L) = \{ [\ L \to L\ ,\ L \cdot\ ,\ \rangle/,\ ]$
$[\ L \to L \cdot\ ,\ L\ ,\ \rangle/,\ ]\}$

$\text{GOTO}(I_{12}, id) = I_8$
$\text{GOTO}(I_{13}, \ ,) = I_{12}$

Para determinar si la gramática es LALR, tenemos que comprobar si es posible unir conjuntos de ítems en la colección LR(1). Observamos que no hay conjuntos de ítems que contengan las mismas reglas de producción punteadas y sólo

difieran en los símbolos de anticipación. Por tanto, la tabla LALR coincide con la tabla LR(1). Por otra parte, si obtenemos los conjuntos SIGUIENTE de los no terminales de la gramática podemos comprobar si las reducciones se realizan con todos los símbolos de dichos conjuntos:

PRIMERO(E) = { $\langle$ $\emptyset$ }
PRIMERO(C) = { $\langle$ $\emptyset$ }
PRIMERO(L) = { id }

SIGUIENTE(E) = { \$ }
SIGUIENTE(C) = { $\cup$ $\cap$ \$ }
SIGUIENTE(L) = { $\rangle$, }

Observamos que todos los ítems de reducción usan la totalidad de los símbolos SIGUIENTE de los no terminales correspondientes. Por tanto, la tabla SLR será igual que la tabla LALR.

A partir de los cálculos anteriores, podemos construir la siguiente tabla de análisis:

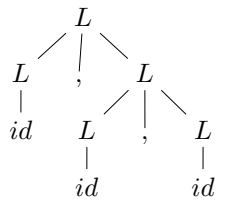
Estado	Acción								Ir-a		
	$\cup$	$\cap$	$\langle$	$\rangle$	$\emptyset$	id	,	\$	E	C	L
0			d3		d4				1	2	
1								acc.			
2	d5	d6						r1			
3						d8					7
4	r5	r5						r5			
5			d3		d4				9	2	
6			d3		d4				10	2	
7				d11			d12				
8				r6			r6				
9								r2			
10								r3			
11	r4	r4						r4			
12						d8					13
13				r7			d12/r7				

La numeración de las reglas de producción usada en la tabla es la siguiente:

1. $E \rightarrow C$
2. $E \rightarrow C \cup E$
3. $E \rightarrow C \cap E$
4. $C \rightarrow \langle\ L\ \rangle$
5. $C \rightarrow \emptyset$
6. $L \rightarrow id$
7. $L \rightarrow L\ ,\ L$

Al aparecer un conflicto desplaza/reduce en la tabla, podemos deducir que la gramática no es LALR. El conflicto se podía prever, porque en el apartado 1 hemos comprobado que la gramática es ambigua.

Para resolver el conflicto desplaza/reduce, observamos los ítems del estado 13. Teniendo en cuenta que la asociatividad de la coma es por la derecha, debemos desplazar para generar árboles similares al mostrado a la derecha cuando se analiza una entrada en la que aparece una lista de tres o más identificadores. Sólo se comenzará a reducir cuando aparezca un token $\rangle$ agrupando los identificadores empezando por el final.



Apartado 3.

Como hemos indicado en el apartado primero, la gramática es ambigua. Por tanto, no puede ser SLR ni LR-Canónica. Si no hubiésemos detectado este problema, podríamos deducir que no es SLR ni LR-Canónica porque las tres tablas son iguales, conforme a lo indicado en el apartado anterior. Por otra parte, también podíamos haber deducido que la gramática no es SLR por el hecho de no ser LALR, ya que el primer tipo de gramáticas están contenidas en el segundo tipo.

Apartado 4.

La simulación de la cadena $\{id, \emptyset\}$ con tratamiento de errores en modo pánico es la siguiente:

Pila	Entrada	Acción
0	$\langle id, \emptyset\rangle$ \$	$d3$
0 $\langle$ 3	$id, \emptyset\rangle$ \$	$d8$
0 $\langle$ 3 id 8	$, \emptyset\rangle$ \$	$r6$ $L \to id$
0 $\langle$ 3 L 7	$, \emptyset\rangle$ \$	$d12$
0 $\langle$ 3 L 7 , 12	$\emptyset\rangle$ \$	**Error**: desapilar hasta estado con Ir-a definido,
0 $\langle$ 3 L 7 , 12 L 13	$\emptyset\rangle$\$	apilar L y descartar símbolos hasta SIGUIENTE(L)
0 $\langle$ 3 L 7 , 12 L 13	$\rangle$\$	$r7$ $L \to L$, L
0 $\langle$ 3 L 7	$\rangle$\$	$d11$
0 $\langle$ 3 L 7 $\rangle$ 11	\$	$r4$ $C \to \langle$ L $\rangle$
0 C 2	\$	$r1$ $E \to C$
0 E 1	\$	**Fin**: no aceptar la entrada.

Apartado 5.

Para realizar la definición dirigida por la sintaxis que se solicita, emplearemos los siguientes atributos:

Símbolo	**Atributo**	**Descripción**
id	`lex`	Lexema del identificador
L	`tipo`	Tipo de la lista de elementos
C	`tipo`	Tipo de la lista de elementos del conjunto
E	`tipo`	Tipo con el que debe ser compatible la expresión
	`n`	Número del primer conjunto dentro de E
	`e`	Número del primer conjunto erróneo dentro de E

Los atributos `tipo` para los distintos símbolos pueden tomar los valores `char`, `int`, `float`, `undef` y `error`; el tipo `undef` representa que el tipo del conjunto no está definido por el uso de $\emptyset$; el tipo `error` indica que ha habido un error en la comprobación dentro del símbolo correspondiente. Se emplean las siguientes funciones auxiliares:

- `tipo getType(char *lex)`: recupera el tipo de un identificador de la tabla de símbolos.
- `tipo checkType(tipo t1, tipo t2)`: comprueba si dos tipos son compatibles, y devuelve el tipo del resultado.

Las reglas semánticas para la evaluación de los atributos de cada símbolo son:

Regla de producción	**Regla semántica**
$E' \to E$	E.`tipo = undef;` E.`n = 1;`

$E \to C$	`if (checkType(`E.`tipo,`C.`tipo) == error)`
	  E.`e =` E.`n;`
	`else`
	  E.`e = 0;`

$E \to C \cup E_1$
$E \to C \cap E_1$
	E_1.`n =` E.`n+1;`
	E_1.`tipo = checkType(`E.`tipo,`C.`tipo);`
	`if (`E_1.`tipo == error)`
	  E.`e =` E.`n;`
	`else`
	  E.`e =` E_1.`e;`

$C \to \{ L \}$	C.`tipo =` L.`tipo;`
$C \to \emptyset$	C.`tipo = undef;`
$L \to id$	L.`tipo = getType(`id.`lex);`
$L \to L_1$, L_2	L.`tipo = checkType(`L_1.`tipo,`L_2.`tipo);`

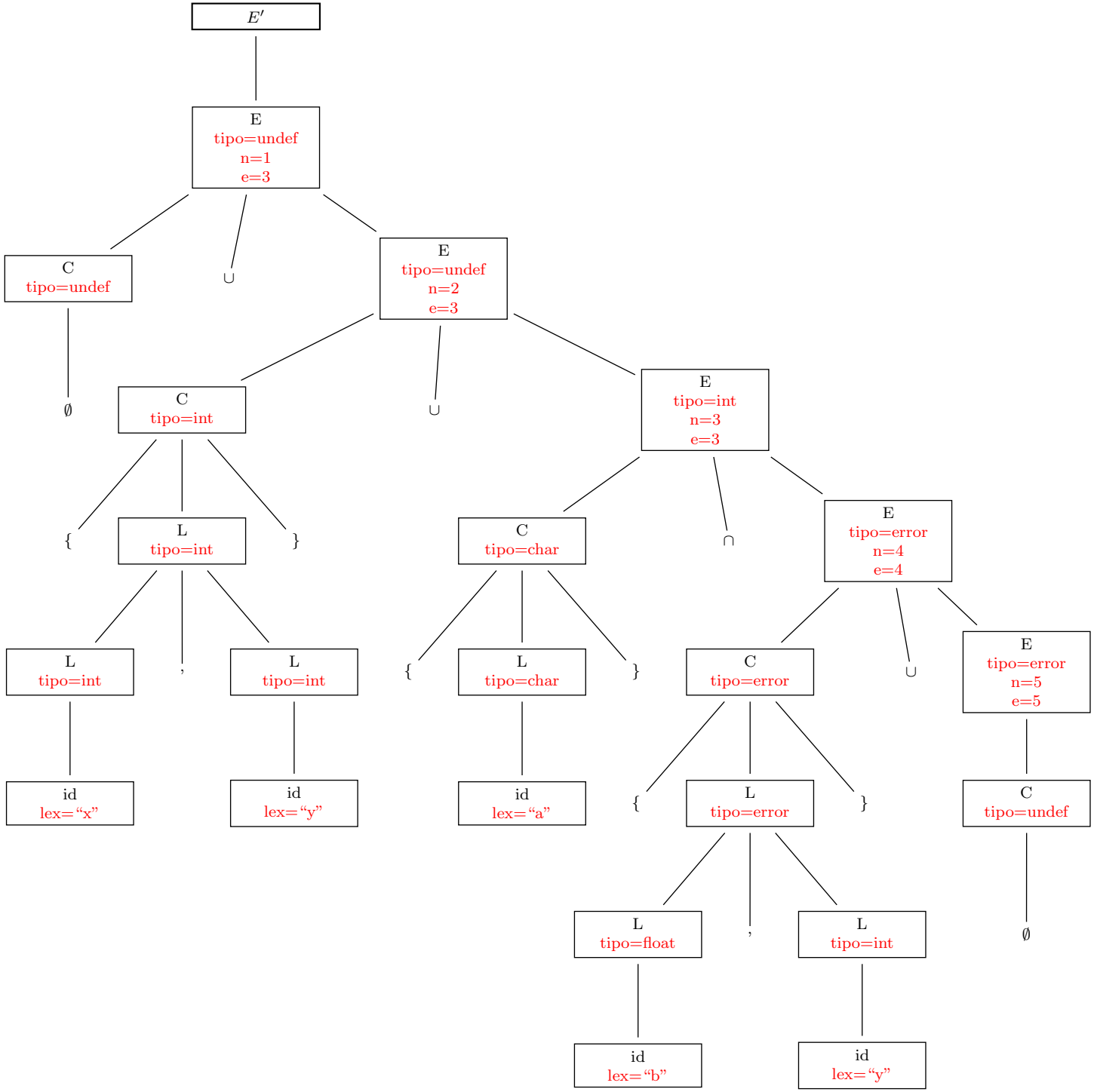
El pseudocódigo de la función `checkType` se muestra a continuación:

```
tipo checkType(tipo t1, tipo t2) {
        tipo res;
        if (t1 == error || t2 == error) res = error;
        else if (t1 == undef) res = t2;
        else if (t2 == undef) res = t1;
        else if (t1 != t2) res = error;
        else res = t1;
        return res;
}
```

Los atributos `tipo` de los símbolos C y L son sintetizados. Sin embargo, el atributo `tipo` de E es heredado, y toma un valor que es función del atributo `tipo` del padre y del hermano izquierdo en las reglas $E \to C \cup E_1$ y $E \to C \cap E_1$. El atributo `e` de E es sintetizado, mientras que el atributo `n` de E es heredado del padre. Por tanto, la gramática no es S-atribuida, pero sí es L-atribuida.

La regla semántica de $E' \to E$ permite inicializar la evaluación de los atributos. El resultado queda guardado en el primer nodo E, que se considera la raíz del árbol a efectos de almacenar el resultado.

Con esta DDS, para la cadena solicitada en el enunciado, podemos construir un árbol sintáctico decorado como el que se indica en la siguiente página. A partir de la cuarta expresión (la primera con $E.tipo = error$) ya no son significativos los valores de los atributos del subárbol, puesto que no influyen en el cálculo del valor del atributo e que llega a la raíz. Se propaga hacia abajo el tipo `error`. Se podría plantear una estrategia de recuperación de errores de tipos, pero queda fuera de lo solicitado en el apartado, ya que sólo se requiere indicar el primer conjunto erróneo.


E′
E
tipo=undef
n=1
e=3
C
tipo=undef
∪
E
tipo=undef
n=2
e=3
∅
C
tipo=int
∪
E
tipo=int
n=3
e=3
{
L
tipo=int
}
C
tipo=char
∩
E
tipo=error
n=4
e=4
L
tipo=int
,
L
tipo=int
{
L
tipo=char
}
C
tipo=error
∪
E
tipo=error
n=5
e=5
id
lex=“x”
id
lex=“y”
id
lex=“a”
{
L
tipo=error
}
C
tipo=undef
L
tipo=float
,
L
tipo=int
∅
id
lex=“b”
id
lex=“y”