EXAMEN DE COMPILADORES (2° **Grado en Informática, junio-2018)**

Facultad Informática Universidad Murcia

SOLUCIONES

Parte I: PREGUNTAS TIPO TEST. 30 %.

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

Parte III: PROBLEMA. 70 %.

Apartado 1.

La gramática es ambigua por la regla de producción $T \to T, T$. Además, tiene recursividad por la izquierda inmediata en esa misma regla. Estas dos características impiden que sea LL(1).

La ambigüedad se puede resolver teniendo en cuenta que la coma es asociativa por la izquierda. Esta asociatividad se puede forzar en la gramática con la recursividad por la izquierda de T. Se usa un nuevo no terminal F para evitar la ambigüedad:

$$
\begin{aligned}
&L \to [\ C\ ] \\
&C \to \lambda \mid T \\
&T \to T, F \mid F \\
&F \to id \mid L
\end{aligned}
$$

El algoritmo para eliminar la recursividad izquierda inmediata tiene como entrada una gramática λ-libre. Aunque en este caso la regla lambda no afecta a la eliminación de la recursividad izquierda, transformamos la gramática anterior para que sea λ-libre:

$$
\begin{aligned}
&L \to [\ ] \mid [\ C\ ] \\
&C \to T \\
&T \to T, F \mid F \\
&F \to id \mid L
\end{aligned}
$$

Podemos simplificar la gramática, eliminando la regla $C \to T$ y poniendo T en lugar de C:

$$
\begin{aligned}
&L \to [\ ] \mid [\ T\ ] \\
&T \to T, F \mid F \\
&F \to id \mid L
\end{aligned}
$$

Eliminamos ahora la recursividad izquierda:

$$
\begin{aligned}
&L \to [\ ] \mid [\ T\ ] \\
&T \to F \mid FT' \\
&T' \to, F \mid , FT' \\
&F \to id \mid L
\end{aligned}
$$

Quitamos ahora los factores comunes:

$$
\begin{aligned}
&L \to [\ L' \\
&L' \to] \mid T\ ] \\
&T \to FT'' \\
&T' \to, FT'' \\
&T'' \to \lambda \mid T' \\
&F \to id \mid L
\end{aligned}
$$

Y por último, simplificamos:

$$
\begin{array}{l}
L \to [\ L' \\
L' \to] \mid T\ ] \\
T \to FT' \\
T' \to \lambda \mid ,FT' \\
F \to id \mid L
\end{array}
$$

Apartado 2.

Obtenemos los conjuntos PRIMERO y SIGUIENTE sobre la gramática anterior:

PRIMERO$(L) = \{\ [\ \}$
PRIMERO$(L') = \{\ ]\ id\ [\ \}$
PRIMERO$(T) = \{\ id\ [\ \}$
PRIMERO$(T') = \{\ \lambda\ ,\ \}$
PRIMERO$(F) = \{\ id\ [\ \}$

SIGUIENTE$(L) = \{\ \$\ ,\ ]\ \}$
SIGUIENTE$(L') = \{\ \$\ ,\ ]\ \}$
SIGUIENTE$(T) = \{\ ]\ \}$
SIGUIENTE$(T') = \{\ ]\ \}$
SIGUIENTE$(F) = \{\ ,\ ]\ \}$

Para construir la tabla de análisis LL(1), calculamos los conjuntos PREDICT:

PREDICT$(L \to\ [\ L') = \{\ [\ \}$
PREDICT$(L' \to]) = \{\ ]\ \}$
PREDICT$(L' \to T\ ]) = \{\ id\ [\ \}$
PREDICT$(T \to FT') = \{\ id\ [\ \}$

PREDICT$(T' \to \lambda) = \{\ ]\ \}$
PREDICT$(T' \to ,FT') = \{\ ,\ \}$
PREDICT$(F \to id) = \{\ id\ \}$
PREDICT$(F \to L) = \{\ [\ \}$

De acuerdo con los conjuntos PREDICT anteriores, la tabla de análisis LL(1) es la siguiente:

NO TERM	**TERMINAL**				
	$[$	$]$	,	id	$\$$
L	$L \to\ [\ L'$				
L'	$L' \to T\ ]$	$L' \to\ ]$		$L' \to T\ ]$	
T	$T \to FT'$			$T \to FT'$	
T'		$T' \to \lambda$	$T' \to ,FT'$		
F	$F \to L$			$F \to id$	

Como se puede observar, no hay conflictos en la tabla y, por tanto, la gramática del final del apartado 1 es LL(1).

Apartado 3.

Para evitar confusiones entre los símbolos de la gramática y la notación de ítems LR(1), renombramos algunos símbolos como se indica a continuación: $<$ por [, $>$ por] y ; por ,.

$$
\begin{array}{l}
L \to < C > \\
C \to \lambda \mid T \\
T \to T;T \mid id \mid L
\end{array}
$$

La colección LR(1) de la gramática del enunciado es la siguiente:

$I_0 = \{\ [\ L' \to \cdot\ L\ ,\ \$\ ]$
$\qquad [\ L \to \cdot\ < C >\ ,\ \$\ ]\ \}$

$I_1 = \textsc{Goto}(I_0, L) = \{\ [\ L' \to L\ \cdot\ ,\ \$\ ]\ \}$

$I_2 = \textsc{Goto}(I_0, <) = \{\ [\ L \to < \cdot\ C >\ ,\ \$\ ]$
$\qquad [\ C \to \lambda\ \cdot\ ,\ >\ ]$
$\qquad [\ C \to \cdot\ T\ ,\ >\ ]$
$\qquad [\ T \to \cdot\ T;T\ ,\ > /;\ ]$
$\qquad [\ T \to \cdot\ id\ ,\ > /;\ ]$
$\qquad [\ T \to \cdot\ L\ ,\ > /;\ ]$
$\qquad [\ L \to \cdot\ < C >\ ,\ > /;\ ]\ \}$

$I_3 = \textsc{Goto}(I_2, C) = \{\ L \to < C \cdot >\ ,\ \$\ ]\ \}$

$I_4 = \text{GOTO}(I_2, T) = \{\ [\ C \to T \cdot\ ,\ >\ ]$
$[\ T \to T \cdot ;T\ ,\ >/;\ ]\ \}$

$I_5 = \text{GOTO}(I_2, id) = \{\ [\ T \to id \cdot\ ,\ >/;\ ]\ \}$

$I_6 = \text{GOTO}(I_2, L) = \{\ [\ T \to L \cdot\ ,\ >/;\ ]\ \}$

$I_7 = \text{GOTO}(I_2, <) = \{\ [\ L \to < \cdot\ C >\ ,\ >/;\ ]$
$[\ C \to \lambda \cdot\ ,\ >\ ]$
$[\ C \to \cdot\ T\ ,\ >\ ]$
$[\ T \to \cdot\ T;T\ ,\ >/;\ ]$
$[\ T \to \cdot\ id\ ,\ >/;\ ]$
$[\ T \to \cdot\ L\ ,\ >/;\ ]$
$[\ L \to \cdot\ <C>\ ,\ >/;\ ]\ \}$

$I_8 = \text{GOTO}(I_3, >) = \{\ L \to < C > \ \cdot\ ,\ \$\ ]\ \}$

$I_9 = \text{GOTO}(I_4, ;) = \{\ [\ T \to T\ ; \cdot\ T\ ,\ >/;\ ]$
$[\ T \to \cdot\ T;T\ ,\ >/;\ ]$
$[\ T \to \cdot\ id\ ,\ >/;\ ]$
$[\ T \to \cdot\ L\ ,\ >/;\ ]$
$[\ L \to \cdot\ <C>\ ,\ >/;\ ]\ \}$

$I_{10} = \text{GOTO}(I_7, C) = \{\ L \to < C \cdot >\ ,\ >/;\ ]\ \}$

$\text{GOTO}(I_7, T) = I_4$

$\text{GOTO}(I_7, id) = I_5$

$\text{GOTO}(I_7, L) = I_6$

$\text{GOTO}(I_7, <) = I_7$

$I_{11} = \text{GOTO}(I_9, T) = \{\ [\ T \to T;T \cdot\ ,\ >/;\ ]$
$[\ T \to T \cdot ;T\ ,\ >/;\ ]\ \}$

$\text{GOTO}(I_9, id) = I_5$

$\text{GOTO}(I_9, L) = I_6$

$\text{GOTO}(I_9, <) = I_7$

$I_{12} = \text{GOTO}(I_{10}, >) = \{\ L \to < C > \ \cdot\ ,\ >/;\ ]\ \}$

$\text{GOTO}(I_{11}, ;) = I_9$

Para formar la colección LALR(1) a partir de la colección LR(1), observamos que se pueden unir los estados 2-7, 3-10 y 8-12, quedando sustituidos por los siguientes tres estados:

$I_{2-7} = \{\ [\ L \to < \cdot\ C >\ ,\ \$/>/;\ ]$
$[\ C \to \lambda \cdot\ ,\ >\ ]$
$[\ C \to \cdot\ T\ ,\ >\ ]$
$[\ T \to \cdot\ T;T\ ,\ >/;\ ]$
$[\ T \to \cdot\ id\ ,\ >/;\ ]$
$[\ T \to \cdot\ L\ ,\ >/;\ ]$
$[\ L \to \cdot\ <C>\ ,\ >/;\ ]\ \}$

$I_{3-10} = \{\ [\ L \to < C \cdot >\ ,\ \$/>/;\ ]\ \}$

$I_{8-12} = \{\ [\ L \to < C > \ \cdot\ ,\ \$/>/;\ ]\ \}$

La definición de la función GOTO de la colección LR(1) queda modificada con la agrupación de estados anterior:

$\text{GOTO}(I_0, <) = I_{2-7}$ $\quad \text{GOTO}(I_{2-7}, T) = I_4$ $\quad \text{GOTO}(I_{2-7}, L) = I_6$ $\quad \text{GOTO}(I_{3-10}, >) = I_{8-12}$

$\text{GOTO}(I_{2-7}, C) = I_{3-10}$ $\quad \text{GOTO}(I_{2-7}, id) = I_5$ $\quad \text{GOTO}(I_{2-7}, <) = I_{2-7}$

A partir de la colección LALR(1), se construye la siguiente tabla de análisis, numerando las reglas por orden de aparición en el enunciado, usando sólo el primer número de los pares de estados que han quedado unidos (no son relevantes los huecos en la numeración), y empleando los símbolos de la gramática original:

ESTADO	ACCIÓN					IR-A		
	,	[	]	id	$	L	C	T
0		d2				1		
1					acep			
2		d2	r2	d5		6	3	4
3			d8					
4	d9		r3					
5	r5		r5					
6	r6		r6					
8	r1		r1		r1			
9		d2		d5		6		11
11	r4/d9		r4					

Hay un conflicto en la tabla, de modo que la gramática no es LALR(1). Tampoco es SLR(1) ni LR(1) puesto que la gramática es ambigua. Para resolver el conflicto, usamos la indicación de la asociatividad izquierda de la coma. Para aplicar esta asociatividad, seleccionamos la acción de reducción **r4**.

Apartado 4.

A continuación se muestra el análisis de la entrada `[id,[id[]]` con tratamiento de errores en modo pánico. Para poder tratar los errores, calculamos los conjuntos SIGUIENTE:

SIGUIENTE(L) = { \$,] } SIGUIENTE(C) = {] } SIGUIENTE(T) = { ,] }

El análisis de la entrada es el siguiente:

PILA	ENTRADA	ACCIÓN
0	`[id,[id[]]` \$	d2
0 `[` 2	`id,[id[]]` \$	d5
0 `[` 2 `id` 5	`,[id[]]` \$	r5 $T \rightarrow$ `id`
0 `[` 2 T 4	`,[id[]]` \$	d9
0 `[` 2 T 4 `,` 9	`[id[]]` \$	d2
0 `[` 2 T 4 `,` 9 `[` 2	`id[]]` \$	d5
0 `[` 2 T 4 `,` 9 `[` 2 `id` 5	`[]]` \$	ERROR: descartar en pila hasta estado 2 y apilar T descartar en entrada hasta] $\in$ SIGUIENTE(T)
0 `[` 2 T 4 `,` 9 `[` 2 T 4	`]]` \$	r3 $C \rightarrow T$
0 `[` 2 T 4 `,` 9 `[` 2 C 3	`]]` \$	d8
0 `[` 2 T 4 `,` 9 `[` 2 C 3 `]` 8	`]` \$	r1 $L \rightarrow$ `[` C `]`
0 `[` 2 T 4 `,` 9 L 6	`]` \$	r6 $T \rightarrow L$
0 `[` 2 T 4 `,` 9 T 11	`]` \$	r4 $T \rightarrow T,T$
0 `[` 2 T 4	`]` \$	r3 $C \rightarrow T$
0 `[` 2 C 3	`]` \$	d8
0 `[` 2 C 3`]` 8	\$	r1 $L \rightarrow$ `[` C `]`
0 L 1	\$	Parar: entrada no aceptada

Apartado 5.

La definición dirigida por la sintaxis puede emplear los siguientes atributos:

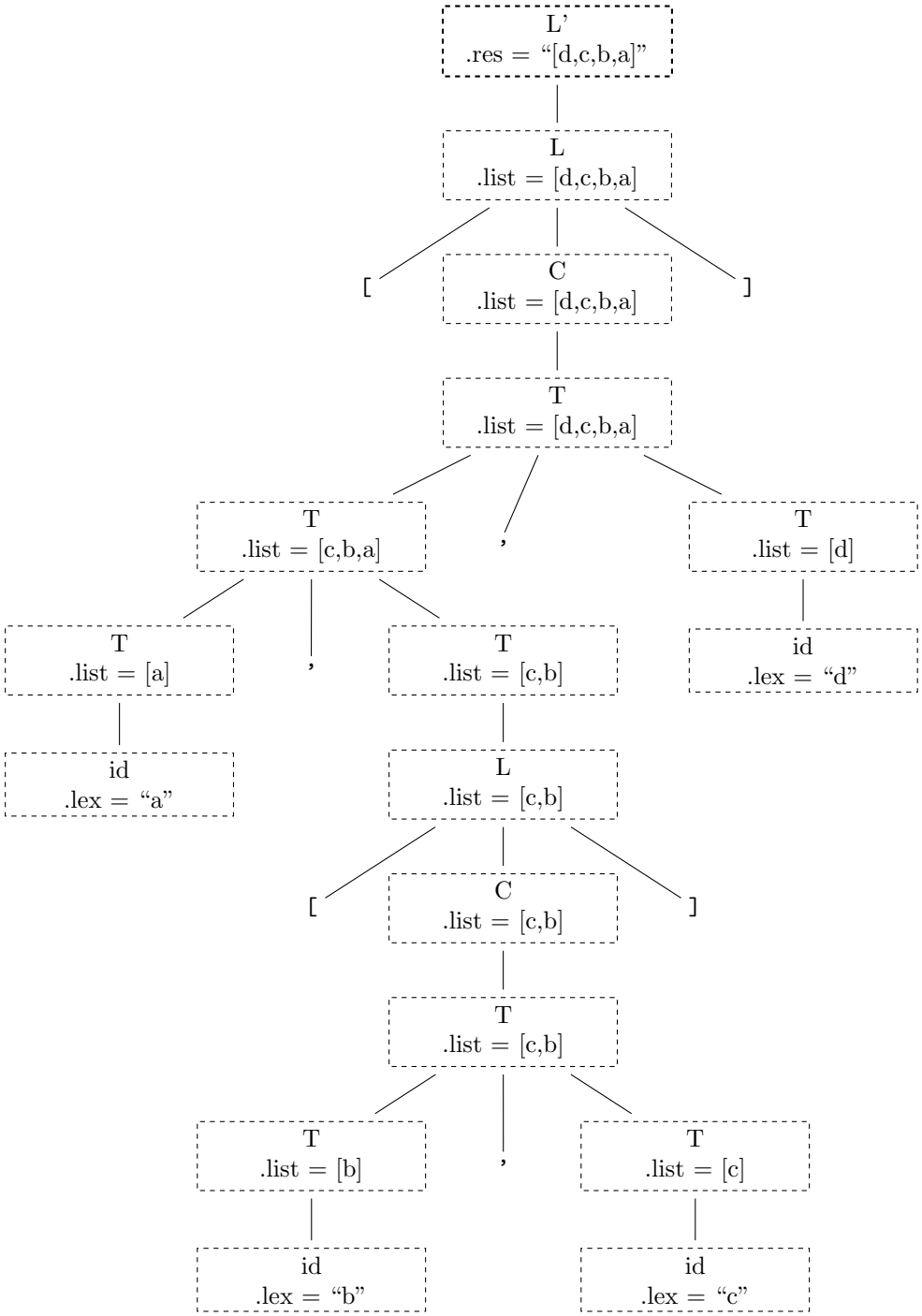
Símbolo	**Atributo**	**Dato**	**Tipo**	**Comentario**
L'	*res*	`Cadena de caracteres`	Sintetizado	Valor resultado con la lista en orden inverso.
L	*list*	`Lista de cadenas`	Sintetizado	Lista de identificadores en orden inverso.
C	*list*	`Lista de cadenas`	Sintetizado	Lista de identificadores en orden inverso.
T	*list*	`Lista de cadenas`	Sintetizado	Lista de identificadores en orden inverso.

En cuanto a las reglas semánticas que evalúan los atributos, podemos plantear las siguientes:

Regla de producción	**Regla semántica**
$L' \rightarrow L$	$L'.res = toString(L.list)$;
$L \rightarrow$ `[` C `]`	$L.list = C.list$;
$C \rightarrow \lambda$	$C.list =$ `[ ]`;
$C \rightarrow T$	$C.list = T.list$;
$T \rightarrow T_1$ `,` T_2	$T.list = T_2.list + T_1.list$;
$T \rightarrow id$	$T.list =$ `[` $id.lex$ `]`;
$T \rightarrow L$	$T.list = L.list$;

En las reglas semánticas se ha considerado que [ ] permite construir una lista vacía, [x] construye una lista con el parámetro x, y + es un operador que permite concatenar dos listas. Además, la función *toString*(*list*) recorre los elementos de la lista y genera una cadena de caracteres de salida con el formato solicitado en el enunciado:

```
string toString(list) {
        string res;
        if (list == []) res = "[]";
        else {
                string res = "[" + list[0];
                for (int i = 1; i < list.length(); i++) res = res + "," + list[i];
                res = res + "]";
        }
        return res;
}
```



El árbol anotado para la entrada `[a,[b,c],d]` se indica en la figura anterior. La DDS define una gramática S-atribuida, porque todos los atributos son sintetizados. Además también es L-atribuida, puesto que por definición las gramáticas L-atribuidas pueden usar atributos sintetizados.