EXAMEN DE COMPILADORES (2° Grado en Informática, junio-2014)

FACULTAD DE INFORMÁTICA

SOLUCIONES

Parte I: PREGUNTAS TIPO TEST. 30 %.

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

Parte II: PROBLEMA. 70 %.

Apartado 1.

Para demostrar que la gramática es LALR(1), se debe aplicar dicho método a la gramática para comprobar que la tabla de análisis no tiene conflictos. Numeramos las reglas de producción de la siguiente forma:

(1) $S \rightarrow$ `if` C `then` P E
(2) $C \rightarrow$ `id == num`
(3) $P \rightarrow$ `print str ;`
(4) $E \rightarrow$ `else` P
(5) $E \rightarrow \lambda$

Obtenemos ahora los conjuntos PRIMERO y SIGUIENTE de los no terminales:

PRIMERO(S) = { `if` }
PRIMERO(C) = { `id` }
PRIMERO(P) = { `print` }
PRIMERO(E) = { `else` , λ }

SIGUIENTE(S) = { $ }
SIGUIENTE(C) = { `then` }
SIGUIENTE(P) = { `else` , $ }
SIGUIENTE(E) = { $ }

Los conjuntos de ítems de la colección LR-canónica son los siguientes:

$I_0 = \{ [S' \rightarrow \cdot\ S\ ,\ \$]$
$[S \rightarrow \cdot$ `if` C `then` $P\ E\ ,\ \$] \}$

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

$I_2 = \text{GOTO}(I_0, \texttt{if}) = \{ [S \rightarrow$ `if` $\cdot\ C$ `then` $P\ E\ ,\ \$]$
$[C \rightarrow \cdot$ `id == num` , `then` $] \}$

$I_3 = \text{GOTO}(I_2, C) = \{ [S \rightarrow$ `if` $C\ \cdot$ `then` $P\ E\ ,\ \$] \}$

$I_4 = \text{GOTO}(I_2, \texttt{id}) = \{ [C \rightarrow$ `id` $\cdot$ `== num` , `then` $] \}$

$I_5 = \text{GOTO}(I_3, \texttt{then}) = \{ [S \rightarrow$ `if` C `then` $\cdot\ P\ E\ ,\ \$]$
$[P \rightarrow \cdot$ `print str ;` , `else`/$ $] \}$

$I_6 = \text{GOTO}(I_4, \texttt{==}) = \{ [C \rightarrow$ `id ==` $\cdot$ `num` , `then` $] \}$

$I_7 = \text{GOTO}(I_5, P) = \{ [S \rightarrow$ `if` C `then` $P\ \cdot\ E\ ,\ \$]$
$[E \rightarrow \cdot$ `else` $P\ ,\ \$]$
$[E \rightarrow \cdot\ ,\ \$] \}$

$I_8 = \text{GOTO}(I_5, \texttt{print}) = \{ [P \rightarrow$ `print` $\cdot$ `str ;` , `else`/$ $] \}$

$I_9 = \text{GOTO}(I_6, \texttt{num}) = \{ [C \rightarrow$ `id == num` $\cdot$, `then` $] \}$

$I_{10} = \text{GOTO}(I_7, E) = \{ [S \rightarrow$ `if` C `then` $P\ E\ \cdot\ ,\ \$] \}$

$I_{11} = \text{GOTO}(I_7, \texttt{else}) = \{ [E \rightarrow$ `else` $\cdot\ P\ ,\ \$]$
$[P \rightarrow \cdot$ `print str ;` , $ $] \}$

$I_{12} = \text{GOTO}(I_8, \texttt{str}) = \{ [P \rightarrow$ `print str` $\cdot$ `;` , `else`/$ $] \}$

$I_{13} = \text{GOTO}(I_{11}, P) = \{ [E \rightarrow$ `else` $P\ \cdot\ ,\ \$] \}$

$I_{14} = \text{GOTO}(I_{11}, \texttt{print}) = \{ [P \rightarrow$ `print` $\cdot$ `str ;` , $ $] \}$

$I_{15} = \text{GOTO}(I_{12}, ;) = \{ [P \rightarrow$ `print str ;` $\cdot$, `else`/$ $] \}$

$I_{16} = \text{GOTO}(I_{14}, \texttt{str}) = \{ [P \rightarrow$ `print str` $\cdot$ `;` , $ $] \}$

$I_{17} = \text{GOTO}(I_{16}, ;) = \{ [P \rightarrow$ `print str ;` $\cdot$, $ $] \}$

Comprobamos ahora que se pueden unir estados de la colección LR-canónica. Los estados a unir deben cumplir la condición de que todos sus ítems sean iguales, salvo en los símbolos de anticipación. Al unir, se combinan los símbolos de anticipación. Denominamos I_{i_j} al conjunto de ítems que se forma por la unión de los ítems de I_i e I_j respectivamente. Se forman los siguientes conjuntos:

- $I_{8\_14} = I_8 \cup I_{14} = \{ [P \rightarrow \texttt{print} \cdot \texttt{str} \texttt{ ;} , \texttt{else}/\$] \}$
- $I_{12\_16} = I_{12} \cup I_{16} = \{ [P \rightarrow \texttt{print str} \cdot \texttt{ ;} , \texttt{else}/\$] \}$
- $I_{15\_17} = I_{15} \cup I_{17} = \{ [P \rightarrow \texttt{print str ;} \cdot , \texttt{else}/\$] \}$

Hay que tener en cuenta que, en el método LALR, si se forma un conjunto I_{i_j}, cualquier transición de la colección LR-canónica desde I_i e I_j, o hacia I_i e I_j, pasará a ser una transición desde o hacia I_{i_j}. Por tanto, la tabla de análisis LR-canónica es la siguiente:

Estado	Acción										Ir-a			
	`if`	`then`	`id`	`==`	`num`	`print`	`str`	`;`	`else`	$	S	C	P	E
0	d2										1			
1										acc				
2			d4									3		
3		d5												
4				d6										
5						d8_14							7	
6					d9									
7									d11	r5				10
8_14							d12_16							
9		r2												
10										r1				
11						d8_14							13	
12_16								d15_17						
13										r4				
15_17									r3	r3				

La gramática es LALR, ya que la tabla no contiene ningún conflicto.

Apartado 2.

La simulación de la cadena de entrada `if id == ; then print str` usando tratamiento de errores en modo pánico es la siguiente::

Pila	Entrada	Acción
0	`if id == ; then print str`$	*d*2
0 `if` 2	`id == ; then print str`$	*d*4
0 `if` 2 `id` 4	`== ; then print str`$	*d*6
0 `if` 2 `id` 4 `==` 6	`; then print str`$	Error: desapila hasta estado 2 y apila C
0 `if` 2 *C* 3	`; then print str`$	Eliminar tokens hasta `then`$\in$ SIG(C)
0 `if` 2 *C* 3	`then print str`$	*d*5
0 `if` 2 *C* 3 `then` 5	`print str`$	*d*8_14
0 `if` 2 *C* 3 `then` 5 `print` 8_14	`str`$	*d*12_16
0 `if` 2 *C* 3 `then` 5 `print` 8_14 `str` 12_16	$	Error: desapila hasta estado 5 y apila P
0 `if` 2 *C* 3 `then` 5 *P* 7	$	$ $\in$ SIG(P). Sigue el análisis.
0 `if` 2 *C* 3 `then` 5 *P* 7	$	*r*5 $E \rightarrow \lambda$
0 `if` 2 *C* 3 `then` 5 *P* 7 *E* 10	$	*r*1 $S \rightarrow$ `if` *C* `then` *P* *E*
0 *S* 1	$	Fin: entrada con errores

Apartado 3.

La gramática es LR(1) porque toda gramática LALR(1) es también LR(1). Para comprobar si es también SLR(1), debemos analizar los ítems que indican reducción, y comprobar si todos los símbolos de SIGUIENTE del no terminal que se reduce no provocan conflictos con otras acciones de los mismos estados:

- En I_7 se encuentra el ítem $[E \rightarrow \cdot , \$]$ y SIGUIENTE(E) = { $ }.
- En I_9 se encuentra el ítem $[C \rightarrow \texttt{id == num} \cdot , \texttt{then}]$ y SIGUIENTE(C) = { `then` }.
- En I_{10} se encuentra el ítem $[S \rightarrow \texttt{if } C \texttt{ then } P \ E \cdot , \$]$ y SIGUIENTE(S) = { $ }.
- En I_{13} se encuentra el ítem $[E \rightarrow \texttt{else } P \cdot , \$]$ y SIGUIENTE(E) = { $ }.
- En I_{15_17} se encuentra el ítem $[P \rightarrow \texttt{print str ;} \cdot , \texttt{else}/\$]$ y SIGUIENTE(P) = { `else` , $ }.

Luego las reducciones se realizan en los mismos casos que indican los símbolos de anticipación, de modo que la tabla LALR coincide con la SLR. Al no producirse conflictos, la gramática es también SLR(1).

Apartado 4.

La gramática no es ambigua porque no permite la construcción de sentencias `if-then-else` anidadas, como:

```
if id1 == 1 then if id2 == 2 then print "1 y 2"; else print "1 y no 2";
```

Esta otra gramática sí lo permite (y sí que es ambigua):

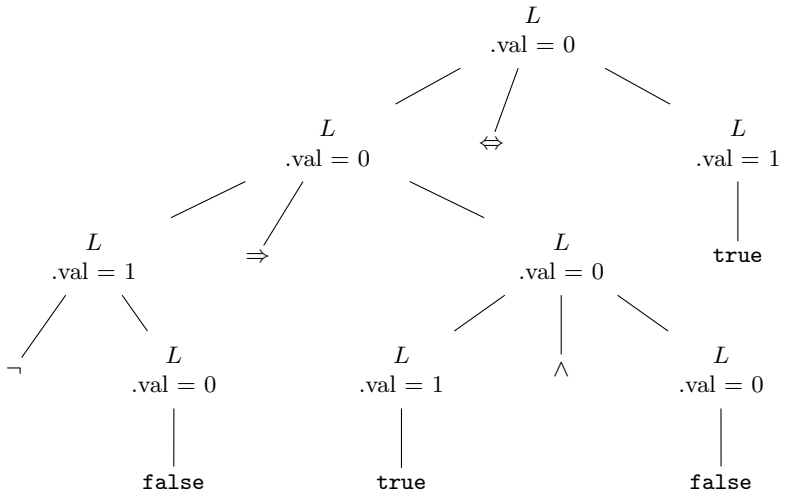
$S \rightarrow$ `if` C `then` S E | `print str ;`
$C \rightarrow$ `id == num`
$E \rightarrow$ `else` S | λ

Apartado 5.

Para realizar la definición dirigida por la sintaxis de la gramática de expresiones de lógica proposicional, sólo es necesario un atributo asociado al no terminal L. Podemos denominar al atributo `val` y hacer que tome un valor 0 en caso de que el valor lógico asociado al no terminal L sea falso, y 1 en caso de que sea verdadero[1]. Empleando los operadores de C, podemos definir del siguiente modo las reglas semánticas:

Regla de producción	**Regla semántica**
$L \rightarrow L_1 \wedge L_2$	L`.val = `L_1`.val && `L_2`.val;`
$L \rightarrow L_1 \vee L_2$	L`.val = `L_1`.val || `L_2`.val;`
$L \rightarrow \neg L_1$	L`.val = !`L_1`.val;`
$L \rightarrow L_1 \Rightarrow L_2$	L`.val = (!`L_1`.val || `L_2`.val);`
$L \rightarrow L_1 \Leftrightarrow L_2$	L`.val = (!`L_1`.val || `L_2`.val) && (!`L_2`.val || `L_1`.val);`
$L \rightarrow (L_1)$	L`.val = `L_1`.val;`
$L \rightarrow$ `true`	L`.val = 1;`
$L \rightarrow$ `false`	L`.val = 0;`

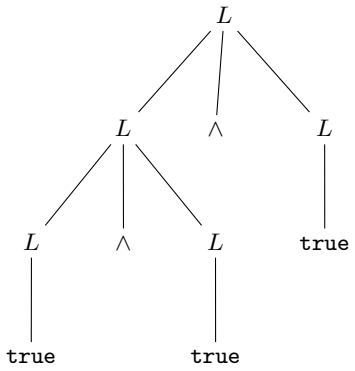
La DDS anterior define una gramática S-atribuida, porque el único atributo que emplea es sintetizado. Y por definición, toda gramática S-atribuida también es L-atribuida. Además, al ser S-atribuida, puede ser evaluada simultáneamente al análisis sintáctico ascendente (considerando que la ambigüedad de la gramática se elimina mediante reglas de precedencia y asociatividad de los operadores). El árbol anotado de la entrada $\neg$`false` $\Rightarrow$ `true` $\wedge$ `false` $\Leftrightarrow$ `true` es el siguiente:



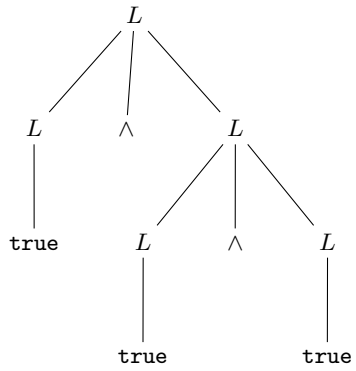
Apartado 6.

Un ejemplo de entrada ambigua es `true` $\wedge$ `true` $\wedge$ `true`. Para demostrar que es ambigua se puede comprobar que tiene dos derivaciones más a la derecha (o izquierda), o bien que tiene dos árboles de derivación distintos. A continuación se muestran los dos árboles de dicha entrada, el primero con asociatividad izquierda y el segundo con asociatividad derecha:

[1] Si se prefiere, se pueden emplear directamente valores boolean como TRUE y FALSE.



$L \Rightarrow L \wedge L \Rightarrow L \wedge \texttt{true} \Rightarrow L \wedge L \wedge \texttt{true}$
$\Rightarrow L \wedge \texttt{true} \wedge \texttt{true} \Rightarrow \texttt{true} \wedge \texttt{true} \wedge \texttt{true}$



$L \Rightarrow L \wedge L \Rightarrow L \wedge L \wedge L \Rightarrow L \wedge L \wedge \texttt{true}$
$\Rightarrow L \wedge \texttt{true} \wedge \texttt{true} \Rightarrow \texttt{true} \wedge \texttt{true} \wedge \texttt{true}$

Para proponer una gramática no ambigua equivalente, podemos aprovechar la similitud de la gramática G_2 con la que hemos empleado repetidas veces durante el curso para representar expresiones aritméticas de forma no ambigua. Basta con considerar que $\Rightarrow$ y $\Leftrightarrow$ tienen prioridad similar a $+$ y $-$, que $\wedge$ y $\vee$ son operadores de prioridad similar a $*$ y $/$, y que $\neg$ tiene la misma prioridad que el $-$ unario. A continuación se muestra la gramática de expresiones aritméticas no ambigua, y la gramática construida de forma similar para las expresiones lógicas:

$E \to E + T \mid E - T \mid T$
$T \to T * F \mid T/F \mid F$
$F \to \texttt{num} \mid (E) \mid -F$

$L \to L \Rightarrow T \mid L \Leftrightarrow T \mid T$
$T \to T \wedge F \mid T \vee F \mid F$
$F \to \texttt{true} \mid \texttt{false} \mid (L) \mid \neg F$

Apartado 7.

La gramática obtenida en el apartado anterior es recursiva por la izquierda. Podemos aplicar el algoritmo de eliminación de la recursividad inmediata en los no terminales L y F, obteniendo:

$L \to T \mid T\ L'$
$L' \to \Rightarrow T \mid \Leftrightarrow T \mid \Rightarrow T\ L' \mid \Leftrightarrow T\ L'$
$T \to F \mid F\ T'$
$T' \to \wedge F \mid \vee F \mid \wedge F\ T' \mid \vee F\ T'$
$F \to \texttt{true} \mid \texttt{false} \mid (L) \mid \neg F$

Ahora se deben eliminar los factores comunes:

$L \to T\ L''$
$L' \to \Rightarrow T\ L'' \mid \Leftrightarrow T\ L''$
$L'' \to \lambda \mid L'$
$T \to F\ T''$
$T' \to \wedge F\ T'' \mid \vee F\ T''$
$T'' \to \lambda \mid T'$
$F \to \texttt{true} \mid \texttt{false} \mid (L) \mid \neg F$

Un último paso podría simplificarla, eliminando las variables L' y T':

$L \to T\ L'$
$L' \to \lambda \mid \Rightarrow T\ L' \mid \Leftrightarrow T\ L'$
$T \to F\ T'$
$T' \to \lambda \mid \wedge F\ T' \mid \Leftrightarrow F\ T'$
$F \to \texttt{true} \mid \texttt{false} \mid (L) \mid \neg F$