Universidad del Valle de Guatemala
Diseño de Lenguajes de Programación
Catedrático: Tomás Gálvez
Semestre 2, 2014

Tarea #3: pre-examen 2

1. Explique en sus palabras cuál es la relación entre derivaciones por la izquierda y por la derecha con los métodos de *parseo top-down* y *bottom-up.* ¿Qué son "forma sentencial" y "sentencia"?
   R/ *Top-down* busca construir una derivación por la izquierda, *bottom-up* construye una derivación por la derecha. Una forma sentencial es un "paso" en una derivación, y una sentencia es la última forma sentencial de una derivación, compuesta únicamente por los terminales *parseados*.

2. ¿Por qué las gramáticas LR(0) no pueden tener producciones nulas (o sea, con la forma $A \to \varepsilon$)?
   R/ Al construir el autómata LR(0), una producción nula $A \to \varepsilon$ se convertiría en un *item* $A \to \cdot$, que indica una reducción. El problema es que un no-terminal *A* no puede puede estar únicamente a la cabeza de una producción nula, porque entonces sería inútil (siempre que aparezca *A* en otra producción lo tendríamos que ignorar porque simplemente es ε). A consecuencia, la producción nula forzaría siempre a un conflicto *shift/reduce*, ya que nos llenaría toda una fila de la tabla de *parseo* con reducciones, incluso la celda donde cualquier otro *item* no nulo con *A* a la cabeza nos indique que tenemos que hacer *shift*.

3. Considere la siguiente gramática:

$$
\begin{aligned}
S &\to A && (1)\\
A &\to \mathtt{a}BE && (2)\\
B &\to \mathtt{b}CD && (3)\\
C &\to \mathtt{c} && (4)\\
D &\to \mathtt{d} && (5)\\
E &\to \mathtt{e}\,FG && (6)\\
F &\to \mathtt{f} && (7)\\
G &\to \mathtt{g} && (8)
\end{aligned}
$$

   Suponga que se *parsea* el *string* "abcdefg". Si el *parser* es LL(1), ¿en qué orden se han de utilizar las producciones durante el *parseo*? ¿En qué orden se utilizarán si el *parser* es [Canonical] LR(1)?
   R/ LL(1): 1 2 3 4 5 6 7 8; LR(1): 4 5 3 7 8 6 2 1

4. La siguiente gramática no es LL(1):

$$Addr \to Name\ \mathtt{@}\ Name\ \mathtt{.}\ \mathtt{id}$$

$$Name \to \mathtt{id} \mid \mathtt{id}\ \mathtt{.}\ Name$$

   Explique por qué, y modifíquela para que sea LL(1).
   R/ No es LL(1) porque los conjuntos FIRST de los cuerpos de *Name* no son mutuamente exclusivos. Empezamos factorizando *Name:*

$$Name \to \mathbf{id}\ Name'$$
$$Name' \to \varepsilon \mid .\,Name$$

   Ahora el problema es que el FIRST de "*.Name*" y el FOLLOW de *Name'* comparten '.', pero lo podemos arreglar fácilmente intercambiando las posiciones de *Name* y **id** en *Addr*, así:

$$Addr \rightarrow Name \ @ \ \mathbf{id} \ . \ Name$$

Esto se comprende al observar que esta gramática produce cadenas de **id** separadas por puntos, luego una arroba, y luego más **id** separados por puntos. Y ya. Quizás querramos reemplazar la ocurrencia de *Name* por su cuerpo en *Name'* sólo para evitar dar una vuelta por gusto. Al final, la gramática quedaría así:

$$Addr \rightarrow Name \ @ \ \mathbf{id} \ . \ Name$$
$$Name \rightarrow \mathbf{id} \ Name'$$
$$Name' \rightarrow \varepsilon \mid . Name$$

con la opción de que la última producción sea $Name' \rightarrow \varepsilon \mid . \mathbf{id} \ Name'$.

5. La siguiente gramática no es LL(1):

$$\begin{aligned} \mathrm{S} &\rightarrow \mathbf{noun} \mid \mathbf{noun \ and \ noun} \mid \mathrm{M}, \ \mathbf{noun}, \ \mathbf{and \ noun} \\ \mathrm{M} &\rightarrow \mathrm{M}, \ \mathbf{noun} \mid \mathbf{noun} \end{aligned}$$

Explique por qué, modifíquela para que sea LL(1) y construya los conjuntos FIRST, FOLLOW y la tabla de *parseo* predictivo. Luego, *parsee* el *string* "noun, noun, and noun".

R/ No es LL(1) por varias razones: las diferentes producciones para *S* tienen elementos en común en sus conjuntos FIRST. Además, es recursiva por la izquierda por culpa de *M*. Para hacerla LL(1) empezamos por factorizar *S* de la siguiente forma:

$$S \rightarrow \mathbf{noun} \ Z$$
$$Z \rightarrow \varepsilon \mid \mathbf{and \ noun}$$

Dejemos la última producción de *S* pendiente por el momento. Si eliminamos la recursión inmediata en *M* obtenemos:

$$M \rightarrow \mathbf{noun} \ M'$$
$$M' \rightarrow, \mathbf{noun} \ M' \mid \varepsilon$$

Ahora podríamos reemplazar *M* por su nuevo cuerpo en la producción pendiente de *S*, lo cual nos llevaría a la factorización

$$Z \rightarrow \varepsilon \mid \mathbf{and \ noun} \mid M', \mathbf{noun}, \mathbf{and \ noun}$$

Con estos cambios introdujimos un problema: como una de las formas de *M'* puede generar ε, es necesario que FIRST(, **noun** *M'*) y FOLLOW(*M'*) sean mutuamente exclusivos para que la gramática sea LL(1), y notaremos que este no es el caso por culpa de la tercera forma de *Z*. Lo primero que podemos hacer es observar que en $Z \rightarrow M', \mathbf{noun}, \mathbf{and \ noun}$ podríamos eliminar un "**, noun**", ya que es redundante con *M'*. Queda entonces $Z \rightarrow M', \mathbf{and \ noun.}$ Sin embargo, el problema persiste pues FIRST(, **noun** *M'*) y FOLLOW(*M'*) todavía comparten la coma. Como eso no sirvió de nada, otra forma de aprovechar dicha redundancia sería cambiar el orden de estos factores en la producción, resultando $Z \rightarrow, \mathbf{noun} \ M', \mathbf{and \ noun}$. La coma sigue dando problema, entonces en lugar de ese cambio buscaremos quitarnos la necesidad de cumplir con esa condición. Es decir que si logramos que *M'* no pueda generar ε eliminamos la necesidad de que FIRST(, **noun** *M'*) y FOLLOW(*M'*) sean mutuamente exclusivos. Tomando $Z \rightarrow, \mathbf{noun} \ M', \mathbf{and \ noun}$; como el problema es la coma podríamos remover todo lo que sigue a *M'* y, viendo que ahora *M'* debe "terminar" *Z*, trasladamos ", **and noun**" hacia *M'*, logrando

$$Z \rightarrow, \mathbf{noun} \ M'$$
$$M' \rightarrow, \mathbf{noun} \ M' \mid , \mathbf{and \ noun}$$

Casi funciona, pero ahora los FIRSTs de los cuerpos de *M'* no son mutuamente exclusivos porque ambos son ",". Si movemos la coma de la siguiente forma $M' \rightarrow \mathbf{noun}, M'$ eliminamos su necesidad en la otra forma de *M'*, quedando

$$M' \rightarrow \mathbf{noun}, M' \mid \mathbf{and\ noun}$$

Lo último que hay que ajustar es que como ahora *M'* ya no inicia con coma, debemos agregarla en *Z* para evitar la posibilidad de un *string* que incluya ", noun noun" (nótese que hay dos "noun" seguidos). La gramática, al final, queda así:

$$S \rightarrow \mathbf{noun}\ Z$$
$$Z \rightarrow \varepsilon \mid \mathbf{and\ noun} \mid , \mathbf{noun}, M'$$
$$M' \rightarrow \mathbf{noun}, M' \mid \mathbf{and\ noun}$$

Ahora podemos construir la tabla de *parseo*, en el proceso computando los correspondientes conjuntos FIRST y FOLLOW:

FIRST(**noun** *Z*) = $\{\mathbf{noun}\}$, entonces

	,	and	noun	$
S			$S \rightarrow \mathbf{noun}\ Z$	

FIRST(**and noun**) = $\{\mathbf{and}\}$
FIRST(**, noun,** *M'*) = {, }, entonces

	,	and	noun	$
S			$S \rightarrow \mathbf{noun}\ Z$	
Z	$Z \rightarrow , \mathbf{noun}, M'$	$Z \rightarrow \mathbf{and\ noun}$		

Z puede generar ε entonces debemos considerar FOLLOW(*Z*) = FOLLOW(*S*) = {$}

	,	and	noun	$
S			$S \rightarrow \mathbf{noun}\ Z$	
Z	$Z \rightarrow , \mathbf{noun}, M'$	$Z \rightarrow \mathbf{and\ noun}$		$Z \rightarrow \varepsilon$

Finalmente, FIRST(**noun,** *M'*) = $\{\mathbf{noun}\}$, entonces

	,	and	noun	$
S			$S \rightarrow \mathbf{noun}\ Z$	
Z	$Z \rightarrow , \mathbf{noun}, M'$	$Z \rightarrow \mathbf{and\ noun}$		$Z \rightarrow \varepsilon$
M'		$M' \rightarrow \mathbf{and\ noun}$	$M' \rightarrow \mathbf{noun}, M'$	

Ahora sólo queda *parsear* "noun, noun, and noun".

Matched	Stack	Input	Action
	$S\$$	noun, noun, and noun	
	noun $Z\$$	noun, noun, and noun	$S \rightarrow$ **noun** Z
noun	$Z\$$	, noun, and noun	Match(noun)
noun	, **noun**, $M'\$$	, noun, and noun	$Z \rightarrow$, **noun**, M'
noun,	**noun**, $M'\$$	noun, and noun	Match(,)
noun, noun	, $M'\$$	, and noun	Match(noun)
noun, noun,	$M'\$$	and noun	Match(,)
noun, noun,	**and noun**$\$$	and noun	$M' \rightarrow$ **and noun**
noun, noun, and	**noun**$\$$	noun	Match(and)
noun, noun, and noun	$\$$		Match(noun)

6. ¿Qué tipo de los *parsers* vistos en clase podría *parsear* la siguiente gramática?

1. S → R
2. R → RR
3. R → R | R
4. R → R*
5. R → ε
6. R → a
7. R → (R)

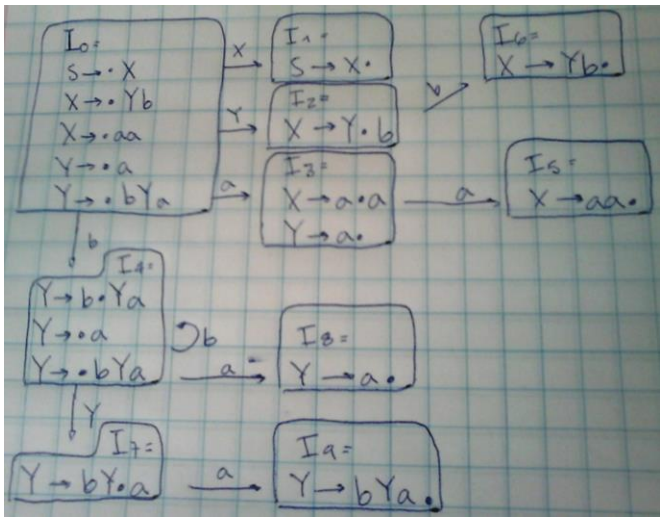
R/ Ninguno, porque la gramática es ambigua (prueben hacer una derivación para "a").

7. Construya el autómata LR(0) para la siguiente gramática:

S → X
X → Yb | aa
Y → a | bYa

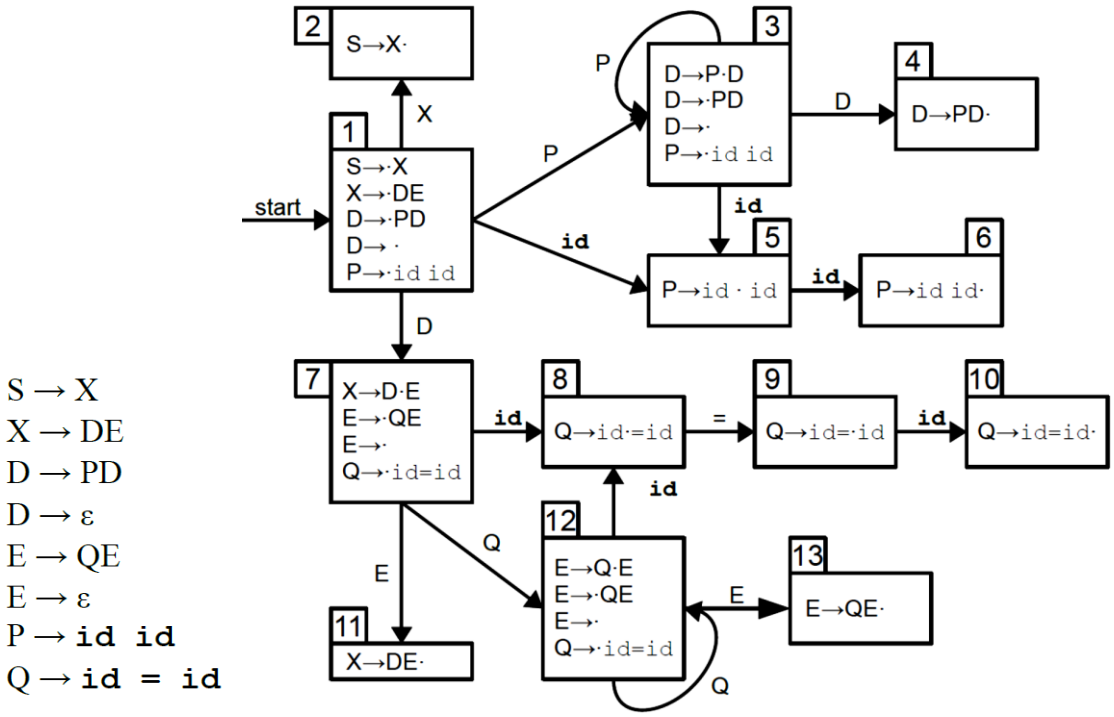
A partir del autómata, pruebe que la gramática no es SLR.

R/



En el estado 3 hay un conflicto *shift/reduce* que SLR no puede manejar, ya que en ese estado se realizará la reducción $Y \rightarrow a$ únicamente en las columnas de la tabla cuyos terminales estén en FOLLOW(*Y*). Como FOLLOW(*Y*) contiene a '*a*' (culpa de la producción $Y \rightarrow bYa$), en la celda $[I_3, a]$ de la tabla de *parseo* habrá un *shift* hacia el estado 5, y un *reduce* con la producción $Y \rightarrow a$.

8. Observe la siguiente gramática y su correspondiente autómata LR(0):

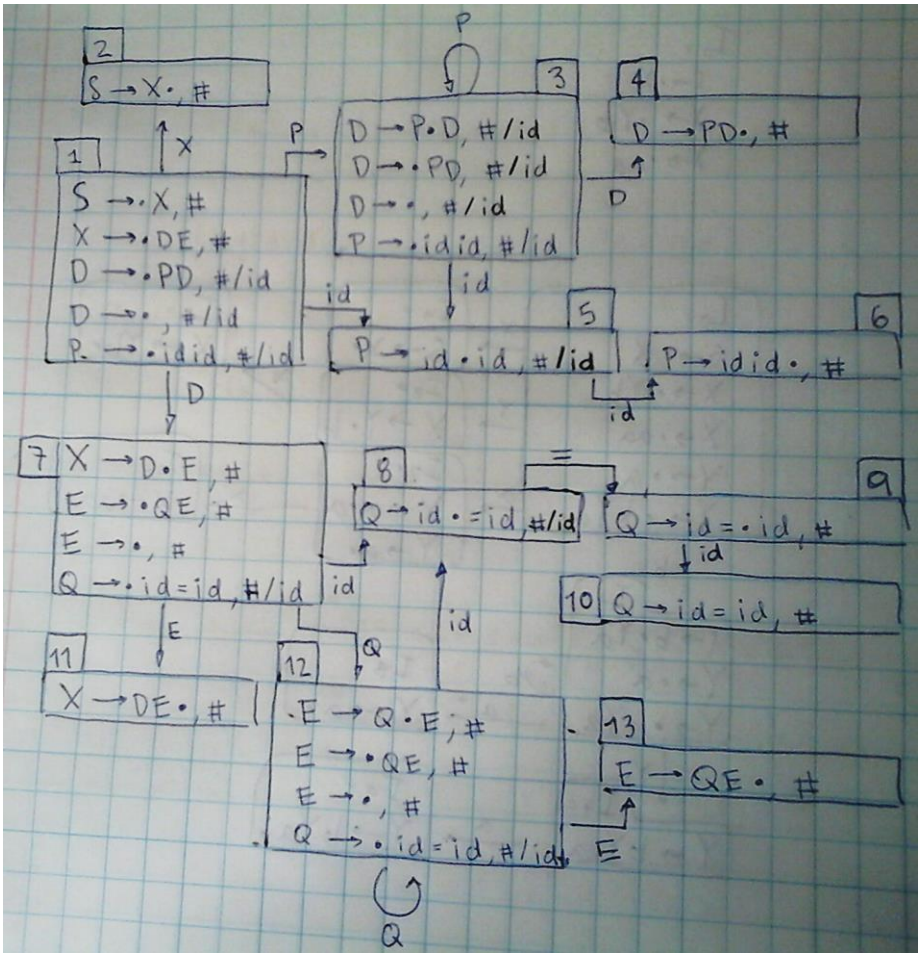


$S \rightarrow X$
$X \rightarrow DE$
$D \rightarrow PD$
$D \rightarrow \varepsilon$
$E \rightarrow QE$
$E \rightarrow \varepsilon$
$P \rightarrow \texttt{id id}$
$Q \rightarrow \texttt{id = id}$

Construya el autómata de *items* LALR determinando los *lookaheads* a partir del autómata LR(0). Determine si la gramática es o no *parseable* por un LALR *parser*. Luego, determine si es *parseable* por un [Canonical] LR(1) *parser*.

R/ (siguiente página)

Universidad del Valle de Guatemala
Diseño de Lenguajes de Programación
Catedrático: Tomás Gálvez
Semestre 2, 2014



Los *lookaheads* marcados en negro son resultado de generación espontánea al transicionar entre estados. Luego de la generación espontánea, construimos la siguiente tabla de propagación:

Set	Item	Lookaheads				
		Init	Pass 1	Pass 2	Pass 3	Pass 4
1	$S \rightarrow \cdot X$	$	$	$	$	$
2	$S \rightarrow X \cdot$		$	$	$	$
3	$D \rightarrow P \cdot D$	id	$/id	$/id	$/id	$/id
4	$D \rightarrow PD \cdot$		id	$/id	$/id	$/id
5	$P \rightarrow \mathbf{id} \cdot \mathbf{id}$	id	$/id	$/id	$/id	$/id
6	$P \rightarrow \mathbf{id\ id} \cdot$		id	$/id	$/id	$/id
7	$X \rightarrow D \cdot E$		$	$	$	$
8	$Q \rightarrow \mathbf{id} \cdot = \mathbf{id}$	id	id	$/id	$/id	$/id
9	$Q \rightarrow \mathbf{id} = \cdot \mathbf{id}$		id	id	$/id	$/id
10	$Q \rightarrow \mathbf{id} = \mathbf{id} \cdot$			id	id	$/id
11	$X \rightarrow DE \cdot$			$	$	$
12	$E \rightarrow Q \cdot E$			$	$	$
13	$E \rightarrow QE \cdot$				$	$

Con esto procedemos a construir la tabla de *parseo:*

	=	id	$	X	D	P	E	Q
1		S5/R4	R4	G2	G7	G3		
2			acc					
3		S5/R4	R4		G4	G3		
4		R3	R3					
5		S6						
6		R7	R7					
7		S8					G11	G12
8	S9							
9		S10						
10		R8	R8					
11			R2					
12		S8	R6				G13	G12
13		R5						

Observemos que hay conflictos de *shift/reduce* en la tabla, y por lo tanto la gramática no es LALR. Más aún, comprimir un autómata LR(1) a uno LALR sólo puede producir conflictos *reduce/reduce*, por lo que estos conflictos *shift/reduce* deben haber sido acarreados desde el autómata LR(1). Es decir, la gramática tampoco es LR(1).

9. Construya la tabla de *parseo* SLR para la siguiente gramática:

$$
\begin{aligned}
&(1) \quad S \rightarrow P \\
&(2) \quad P \rightarrow (P)\,P \\
&(3) \quad P \rightarrow \varepsilon
\end{aligned}
$$

La colección canónica LR(0) que le corresponde es la siguiente:

(1)	(2)	(3)	(4)	(5)	(6)
$S \rightarrow \cdot P$ $P \rightarrow \cdot(P)P$ $P \rightarrow \cdot$	$S \rightarrow P\cdot$	$P \rightarrow (\cdot P)P$ $P \rightarrow \cdot(P)P$ $P \rightarrow \cdot$	$P \rightarrow (P\cdot)P$	$P \rightarrow (P)\cdot P$ $P \rightarrow \cdot(P)P$ $P \rightarrow \cdot$	$P \rightarrow (P)P\cdot$

A la tabla que construya márquele los puntos donde ocurrirían conflictos si la tabla fuera LR(0). Luego, *parsee* el *string* "(() ()) ()" (los espacios en blanco se introducen sólo para claridad).

R/

	(	)	$	P
1	S3	R3	R3	G2
2			acc	
3	S3	R3	R3	G4
4		S5		
5	S3	R3	R3	G6
6		R2	R2	

Stack	*Input*	Acción
1	(() ()) () $	S3
1 3	() ()) () $	S3
1 3 3	) ()) () $	R3
1 3 3	) ()) () $	G4
1 3 3 4	) ()) () $	S5
1 3 3 4 5	()) () $	S3
1 3 3 4 5 3	)) () $	R3
1 3 3 4 5 3	)) () $	G4
1 3 3 4 5 3 4	)) () $	S5
1 3 3 4 5 3 4 5	) () $	R3
1 3 3 4 5 3 4 5	) () $	G6
1 3 3 4 5 3 4 5 6	) () $	R2
1 3 3 4 5	) () $	G6
1 3 3 4 5 6	) () $	R2
1 3	) () $	G4
1 3 4	) () $	S5
1 3 4 5	() $	S3
1 3 4 5 3	) $	R3
1 3 4 5 3	) $	G4
1 3 4 5 3 4	) $	S5
1 3 4 5 3 4 5	$	R3
1 3 4 5 3 4 5	$	G6
1 3 4 5 3 4 5 6	$	R2
1 3 4 5	$	R3
1 3 4 5	$	G6
1 3 4 5 6	$	R2
1	$	G2
1 2	$	acc

10. ¿Cómo se determina dónde van las reducciones en una tabla de *parseo* LALR o [Canonical] LR(1)?
    R/ Los *items* LR(1) que tengan el punto al final de su núcleo colocan reducciones, por su producción correspondiente, en las columnas de la tabla de *parseo* correspondientes a sus *lookaheads*.

11. ¿Pueden existir gramáticas LL(1) pero no LR(0)? ¿Y gramáticas LR(1) que no sean LR(0)? ¿Qué tal gramáticas que sean LL(1) y a la vez LR(0)?
    R/ Sí. Sí. Sí.

Fuentes:

Los problemas de este examen fueron tomados (en su mayoría) de tareas y exámenes del curso de compiladores del 2012 de Stanford University (CS143).