

Teoría de Lenguajes
2dos Parciales (y recuperatorios) Resueltos

Sebastián Taboh

4 de junio de 2018



Este apunte contiene soluciones a segundos parciales y recuperatorios de la materia Teoría de Lenguajes.

Este documento fue creado cuando cursé en el 2017, en caso de encontrar errores, por favor comunicarlo por mail a sebi_282@hotmail.com.

Índice

2015 C2 Recuperatorio	3
Ejercicio 1	3
Ejercicio 2	6
2017 C1 Parcial	8
Ejercicio 1	9
Ejercicio 3	14

2015 C2 Recuperatorio

Ejercicio 1

(30 pts) La siguiente gramática representa un fragmento de las expresiones de la lógica proposicional: $G_1 = \langle \{E\}, \{p, \rightarrow, \vee, \neg, (,)\}, E, P \rangle$, con P :

$$E \longrightarrow E \rightarrow E \mid E \vee E \mid \neg E \mid p \mid (E)$$

(a) Dar su tabla SLR, señalando todos los conflictos que tenga.

Primero aumentamos la gramática. Queda $G'_1 = \langle \{E', E\}, \{p, \rightarrow, \vee, \neg, (,)\}, E', P' \rangle$, con P' :

$$\begin{aligned} E' &\longrightarrow E \\ E &\longrightarrow E \rightarrow E \mid E \vee E \mid \neg E \mid p \mid (E) \end{aligned}$$

Ahora armamos el autómata con los conjuntos de ítems LR(0).

Estado 0:

$$\begin{aligned} E' &\longrightarrow \cdot E \\ E &\longrightarrow \cdot E \rightarrow E \\ E &\longrightarrow \cdot E \vee E \\ E &\longrightarrow \cdot \neg E \\ E &\longrightarrow \cdot p \\ E &\longrightarrow \cdot (E) \end{aligned}$$

Estado 1:

$$\begin{aligned} E' &\longrightarrow E \cdot \\ E &\longrightarrow E \cdot \rightarrow E \\ E &\longrightarrow E \cdot \vee E \end{aligned}$$

Estado 2:

$$\begin{aligned} E &\longrightarrow \neg \cdot E \\ &\text{---} \\ E &\longrightarrow \cdot E \rightarrow E \\ E &\longrightarrow \cdot E \vee E \\ E &\longrightarrow \cdot \neg E \\ E &\longrightarrow \cdot p \\ E &\longrightarrow \cdot (E) \end{aligned}$$

Estado 3:

$$E \longrightarrow p \cdot$$

Estado 4:

$$\begin{array}{l}
 E \longrightarrow (\cdot E) \\
 \text{-----} \\
 E \longrightarrow \cdot E \rightarrow E \\
 E \longrightarrow \cdot E \vee E \\
 E \longrightarrow \cdot \neg E \\
 E \longrightarrow \cdot p \\
 E \longrightarrow \cdot (E)
 \end{array}$$

Estado 5:

$$\begin{array}{l}
 E \longrightarrow E \rightarrow \cdot E \\
 \text{-----} \\
 E \longrightarrow \cdot E \rightarrow E \\
 E \longrightarrow \cdot E \vee E \\
 E \longrightarrow \cdot \neg E \\
 E \longrightarrow \cdot p \\
 E \longrightarrow \cdot (E)
 \end{array}$$

Estado 6:

$$\begin{array}{l}
 E \longrightarrow E \vee \cdot E \\
 \text{-----} \\
 E \longrightarrow \cdot E \rightarrow E \\
 E \longrightarrow \cdot E \vee E \\
 E \longrightarrow \cdot \neg E \\
 E \longrightarrow \cdot p \\
 E \longrightarrow \cdot (E)
 \end{array}$$

Estado 7:

$$\begin{array}{l}
 E \longrightarrow \neg E \cdot \\
 E \longrightarrow E \cdot \rightarrow E \\
 E \longrightarrow E \cdot \vee E
 \end{array}$$

Estado 8:

$$\begin{array}{l}
 E \longrightarrow (E \cdot) \\
 E \longrightarrow E \cdot \rightarrow E \\
 E \longrightarrow E \cdot \vee E
 \end{array}$$

Estado 9:

$$\begin{array}{l}
 E \longrightarrow E \rightarrow E \cdot \\
 E \longrightarrow E \cdot \rightarrow E \\
 E \longrightarrow E \cdot \vee E
 \end{array}$$

Estado 10:

$$E \longrightarrow E \vee E \cdot$$

$$E \longrightarrow E \cdot \rightarrow E$$

$$E \longrightarrow E \cdot \vee E$$

Estado 11:

$$E \longrightarrow (E) \cdot$$

Recordemos que SLR(1) sólo reduce para los símbolos en SIGUIENTES de ese no terminal. Así, hay que calcular SIGUIENTES para cada no terminal.

NT	SIGUIENTES
E'	{ $\$$ }
E	{ $\rightarrow, \$, \vee,)$ }

ESTADO	ACTION							GoTo
	p	\rightarrow	\vee	\neg	()	$\$$	E
0	s3			s2	s4			1
1		s5	s6				accept	
2	s3			s2	s4			7
3		r(4)	r(4)			r(4)	r(4)	
4	s3			s2	s4			8
5	s3			s2	s4			9
6	s3			s2	s4			10
7		s5/r(3)	s6/r(3)			r(3)	r(3)	
8		s5	s6			s11		
9		s5/r(1)	s6/r(1)			r(1)	r(1)	
10		s5/r(2)	s6/r(2)			r(2)	r(2)	
11		r(5)	r(5)			r(5)	r(5)	

$$r(1) = r(E \longrightarrow E \rightarrow E)$$

$$r(2) = r(E \longrightarrow E \vee E)$$

$$r(3) = r(E \longrightarrow \neg E)$$

$$r(4) = r(E \longrightarrow p)$$

$$r(5) = r(E \longrightarrow (E))$$

- (b) Resolver los conflictos eligiendo en cada caso una de las entradas de la gramática de manera que el árbol de derivación resultante respete las siguientes reglas de precedencia y asociatividad: la negación tiene mayor precedencia que la disyunción, que es asociativa a izquierda y tiene mayor precedencia que la implicación, que es asociativa a derecha.

Si la negación tiene mayor precedencia que la disyunción significa que si se puede reducir $\neq E$ a E o shiftear por un \vee , la reducción de la negación se tiene que dar antes que el shifteo. Es decir, tiene que ocurrir la primera. Esto resuelve el conflicto de (7, \vee).

El de $(10, \vee)$ se resuelve porque la disyunción es asociativa a izquierda, o sea que tiene más prioridad reducir $E \vee E$ a E que shiftear por leer otro \vee .

La mayor precedencia de la disyunción sobre la implicación indica que si se puede reducir $E \vee E$ a E o shiftear por leer \rightarrow , debe darse la primera: resuelve el conflicto $(10, \rightarrow)$. También establece que si se puede reducir $E \rightarrow E$ a E o shiftear por leer \vee debe darse la segunda, lo que define la entrada de $(9, \vee)$.

La precedencia a derecha de \rightarrow nos dice cómo resolver el conflicto de $(9, \rightarrow)$.

Además, por transitividad la negación tiene mayor precedencia que la implicación, por lo que se resuelve el conflicto $(7, \rightarrow)$.

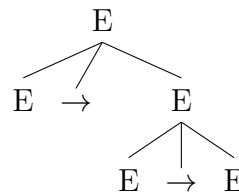
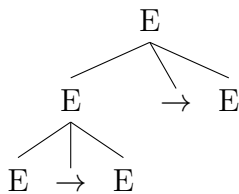
ESTADO	ACTION							GoTo
	p	\rightarrow	\vee	\neg	()	$\$$	E
0	s3			s2	s4			1
1		s5	s6				accept	
2	s3			s2	s4			7
3		r(4)	r(4)			r(4)	r(4)	
4	s3			s2	s4			8
5	s3			s2	s4			9
6	s3			s2	s4			10
7		r(3)	r(3)			r(3)	r(3)	
8		s5	s6			s11		
9		s5	s6			r(1)	r(1)	
10		r(2)	r(2)			r(2)	r(2)	
11		r(5)	r(5)			r(5)	r(5)	

Ejercicio 2

(35 pts) Dar una gramática extendida que genere $L(G_1)$ y sea ELL(1).

Dar su parser iterativo-recursivo.

Primero notemos que esta gramática es ambigua dado que existe una cadena con más de un árbol de derivación. Por ejemplo, $p \rightarrow p \rightarrow p$.



Es importante recordar que eliminar la recursión o factorizar NUNCA van a desambiguar una gramática.

Por otra parte, dado que lo que piden es una gramática extendida, tampoco es útil usar la técnica de eliminación de la recursión a izquierda. Lo que hay que hacer es remplazar la recursión por iteración.

Entonces, lo primero es desambiguar la gramática. Podemos aprovechar que en el Ejercicio 1 nos dieron reglas de precedencia y asociatividad, aunque no es obligatorio seguirlas para este ejercicio.

Siguiendo esas reglas una gramática no ambigua posible es:

$$\begin{aligned} E &\longrightarrow D \rightarrow E \mid D \\ D &\longrightarrow D \vee A \mid A \\ A &\longrightarrow \neg A \mid p \mid (E) \end{aligned}$$

Ahora que tenemos una gramática no ambigua podemos convertirla en extendida ELL(1).
Para E podemos factorizar:

$$E \longrightarrow D(\rightarrow E)?$$

y para D convertir recursión en iteración:

$$D \longrightarrow A(\vee A)^*$$

A queda como está.

Finalmente obtenemos

```

Proc Main()
  E();
  match('$');
  accept();
End

Proc E()
  D();
  if(tc == '->'){
    match('->');
    E();
  }
End

Proc D()
  A();
  while(ct == 'v') {
    match('v');
    A();
  }
End

Proc A()
  A();
  if(ct == '!') {
    match('!');
    A();
  }
  elseif(ct == 'p'){
    match('p');
  }
  elseif(ct == '('){
    match('(');
    E();
    match(')');
  }
  else{
    error();
  }
End

```

2017 C1 Parcial

Teoría de Lenguajes - Segundo Parcial

Primer cuatrimestre de 2017

Hacer cada ejercicio en hojas separadas.

Poner nombre, número de orden y número de página en cada ejercicio.

Justificar todas las respuestas.

El examen es a libro (no electrónico) abierto.

Se aprueba con 65 puntos.

1. (30 pts) Sea G_1 una gramática libre de contexto extendida,

$$G_1 = \langle \{S, C, D\}, \{a, b, c, d, \cdot\}, S, P_1 \rangle, \text{ con } P_1:$$

$$\begin{aligned} S &\rightarrow (abC \mid acD)^* a \\ C &\rightarrow c(\cdot, c)^* \\ D &\rightarrow d(d?)^* \end{aligned}$$

- a) ¿Es G_1 ELL(1)? Si no, dar una gramática G'_1 que sí lo sea, tal que $\mathcal{L}(G'_1) = \mathcal{L}(G_1)$.
b) Dar el parser recursivo-iterativo para la gramática ELL(1).

2. (40 pts) Dada la gramática G_2 , en la que se definen algunas operaciones con listas, se desea realizar una traducción dirigida por sintaxis que imprima la evaluación de la expresión dada.

$$G_2 = \langle \{S, L, L', L'', C, C', P, V\}, \{\text{function}, (\cdot), [,], \{, \}, \cdot, \text{var}, \text{num}\}, S, P_2 \rangle, \text{ con } P_2:$$

$$\begin{aligned} S &\rightarrow \text{function}(L) \mid L \\ L &\rightarrow C [L'] \\ L' &\rightarrow L'' \mid \lambda \\ L'' &\rightarrow V, L'' \mid V \\ C &\rightarrow \{C'\} \mid \lambda \\ C' &\rightarrow P, C' \mid P \\ P &\rightarrow (\text{var}, \text{num}) \\ V &\rightarrow \text{num} \mid \text{var} \mid (\text{num}, \text{num}) \end{aligned}$$

El token `function` tiene un atributo `name`, que indica la función a evaluar:

- `compress`: que, dada una lista, obtiene la versión comprimida de la misma.
Por ejemplo, `compress([1,3,3,5,5,5,3])` \rightsquigarrow `[1,(3,2),(5,3),3]`,
- `expand`: que, dada una lista, obtiene la versión expandida de la misma.
Por ejemplo, `expand([(2,8),3,10])` \rightsquigarrow `[2,2,2,2,2,2,2,2,3,10]`,
- `max`: que obtiene el máximo valor¹ de una lista, comprimida o no.
Por ejemplo, `max([9,10,-1])` \rightsquigarrow `10`, `max([9, 13, (14,10)])` \rightsquigarrow `14`

Las listas pueden contener variables, por lo que hay un contexto de variables, opcional, donde constan variables y su valor. Todas las variables deben tener un valor asignado para que se pueda evaluar una expresión.

Por ejemplo: `{(x,3),(y,1)}[3,x,34,x,5]` \rightsquigarrow `[3,3,34,3,5]`, `{(z,3)}[]` \rightsquigarrow `[]`,
`compress({(y,8),(x,8),(z,3)}[x,y,z])` \rightsquigarrow `[(8,2),3]`,
`{(x,3)}[x,x,y]` \rightsquigarrow `ERROR ('y' sin definir)`,
`max({(x,1)}[8,z])` \rightsquigarrow `ERROR ('z' sin definir)`.

3. (30 pts) Sea $G_3 = \langle \{D, R, T, V\}, \{\text{vname}, \cdot, =, \text{int}, \text{string}, \text{bool}, \text{literal}\}, D, P_3 \rangle$, con P_3 :

$$\begin{aligned} D &\rightarrow T \text{ vname } R; \\ R &\rightarrow = V \mid R R \mid \lambda \\ T &\rightarrow \text{int} \mid \text{string} \mid \text{bool} \\ V &\rightarrow \text{vname} \mid \text{literal} \end{aligned}$$

- a) Dar la tabla SLR correspondiente.
b) ¿Es G_3 SLR? Si no, indicar cómo podría/n resolverse el/los conflictos existentes sin modificar la gramática, en caso de ser posible, y justificar.

¹Si la lista está vacía, el máximo es null

Ejercicio 1

$G_1 = \text{GLC extendida.}$

$$G_1 = \langle \{S, C, D\}, \{a, b, c, d\}, S, P_1 \rangle$$

$$P_1: S \rightarrow (abC \mid acD)^* a$$

$$C \rightarrow c \mid c^*$$

$$D \rightarrow d \mid d^*$$

a) Para ver si G_1 es ELL(1) vamos a ver si su derivada es LL(1).

Su derivada es:

$$S \rightarrow A_1 a$$

$$A_1 \rightarrow A_2 A_1 \mid \lambda$$

$$A_2 \rightarrow A_3 \mid A_4$$

$$A_3 \rightarrow abC$$

$$C \rightarrow c A_5$$

$$A_5 \rightarrow c A_5 \mid \lambda$$

$$A_4 \rightarrow acD$$

$$D \rightarrow d A_6$$

$$A_6 \rightarrow A_7 A_6 \mid \lambda$$

$$A_7 \rightarrow d \mid \lambda$$

$$\begin{aligned} \text{SD}(A_7 \rightarrow \lambda) &= \text{Sigvientes}(A_7) \\ &= \text{Primeros}(A_6) \cup \text{Sigvientes}(A_6) \\ &= \text{Primeros}(A_7) \cup \text{Sigvientes}(D) \\ &= \{d\} \cup \text{Sigvientes}(A_4) \\ &= \{d\} \cup \text{sigvientes}(A_2) \\ &= \{d\} \cup (\text{Primeros}(A_1) \cup \text{Sigvientes}(A_1)) \\ &= \{d\} \cup (\text{Primeros}(A_2) \cup \{a\}) \\ &= \{d\} \cup (\text{Primeros}(A_3) \cup \text{Primeros}(A_4)) \cup \{a\} \\ &= \{d\} \cup \{a\}. \end{aligned}$$

$$SD(A_7 \rightarrow d) = \{d\}.$$

No es LL(1) ya que

$$SD(A_7 \rightarrow \lambda) \cap SD(A_7 \rightarrow d) = \{d\}.$$

Así, G_1 no es ELL(1).

Vamos a probar con la gramática

$$G_1' = \langle V_N(G_1), V_T(G_1), S, P_1' \rangle.$$

con P_1' :

$$\begin{aligned} S &\rightarrow (a(bc|cd))^* a. \\ C &\rightarrow c|c^* \\ D &\rightarrow d|d^* \end{aligned}$$

Es claro que $L(G_1) = L(G_1')$

En la producción de S sólo se sacó a de factor común y en D se usó que concatenar con λ no cambia los cadenas.

$$\begin{aligned} S &\rightarrow A_1 a \\ A_1 &\rightarrow A_2 A_1 | \lambda \\ A_2 &\rightarrow a(bc|cd) \end{aligned}$$

Esto ya parece ser conflictivo:

$$\begin{aligned} SD(A_1 \rightarrow \lambda) &= \text{Siguientes}(A_1) \ni \{a\} \\ SD(A_1 \rightarrow A_2 A_1) &= \text{Primeros}(A_2) \ni \{a\}, \end{aligned}$$

Repensemos G_1' cambiando P_1' sin cambiar el lenguaje generado.

$$\Rightarrow S \rightarrow (a(bc|c^*|cd^*))^* a.$$

$$\Rightarrow S \rightarrow (a(b|a)c|(c|c^*|dd^*))^* a.$$

Usando un "truco" parecido al que se podía usar en el Ej. 12) b) de la P.8 llego a

$$\Rightarrow S \rightarrow a \left((b|a)c (c|d)^* | dd^* \right) a$$

Ahora, su derivada es

$$\begin{aligned} S &\rightarrow a A_1 \\ A_1 &\rightarrow A_2 A_1 | \lambda \\ A_2 &\rightarrow A_3 c A_4 \\ A_3 &\rightarrow b | \lambda \\ A_4 &\rightarrow A_5 | A_6 \\ A_5 &\rightarrow c A_5 | \lambda \\ A_6 &\rightarrow d A_7 \\ A_7 &\rightarrow d A_7 | \lambda \end{aligned}$$

$$\begin{aligned} SD(A_7 \rightarrow \lambda) &= \text{Sigvientes}(A_7) \\ &= \text{Sigvientes}(A_6) \\ &= \text{Sigvientes}(A_4) \\ &= \text{Sigvientes}(A_2) \\ &= (\text{Primeros}(A_1) \cup \text{Sigvientes}(A_1)) \\ &= \text{Primeros}(A_2) \cup \text{Sigvientes}(S) \\ &= (\text{Primeros}(A_3) \cup \text{Sigvientes}(A_3)) \cup \{\$\} \\ &= \{b\} \cup \{c\} \cup \{\$\}. \end{aligned}$$

$$SD(A_7 \rightarrow d A_7) = \{d\}.$$

$$\begin{aligned} SD(A_5 \rightarrow \lambda) &= \text{Sigvientes}(A_5) \\ &= \text{Sigvientes}(A_4) \\ &= \text{Sigvientes}(A_2) \\ &= \{b\} \cup \{c\} \cup \{\$\}. \end{aligned}$$

$$SD(A_5 \rightarrow c A_5) = \{c\}.$$

$$SD(A_4 \rightarrow A_6) = \text{Primeros}(A_6) = \{d\}.$$

$$SD(A_4 \rightarrow A_5) = \{, b, c, \$\}.$$

Es primeros(A_5) \cup siguientes(A_4).

$$SD(A_3 \rightarrow \lambda) = \text{Siguietes}(A_3) \\ = \{c\}.$$

$$SD(A_3 \rightarrow b) = \{b\}.$$

$$SD(A_1 \rightarrow \lambda) = \text{Siguietes}(A_1) \\ = \{\$\}.$$

$$SD(A_1 \rightarrow A_2 A_1) = \text{Primeros}(A_2) \\ = \text{Primeros}(A_3) \cup \text{Siguietes}(A_3) \\ = \{b\} \cup \{c\}.$$

La derivada es LL(1).

Entonces

$$G_1' = \langle \{S\}, \{a, b, c, d, \lambda\}, S, P_1' \rangle$$

$$\text{con } P_1' : S \rightarrow a((b\lambda)c(c\lambda)^*|dd^*)a)^*$$

es ELL(1), y vale $L(G_1) = L(G_1')$.

```

b) S() {
    match('a');
    while (tc in {b,c}) {
        if (tc == 'b') {
            match('b');
        }
        match('c');
        if (tc == ',') {
            while (tc == ',') {
                match(',');
                match('c');
            }
        }
        elseif (tc == 'd') {
            match('d');
            while (tc == 'd') {
                match('d');
            }
        }
        else error;
        match('a');
    }
    match('$');
    accept;
}

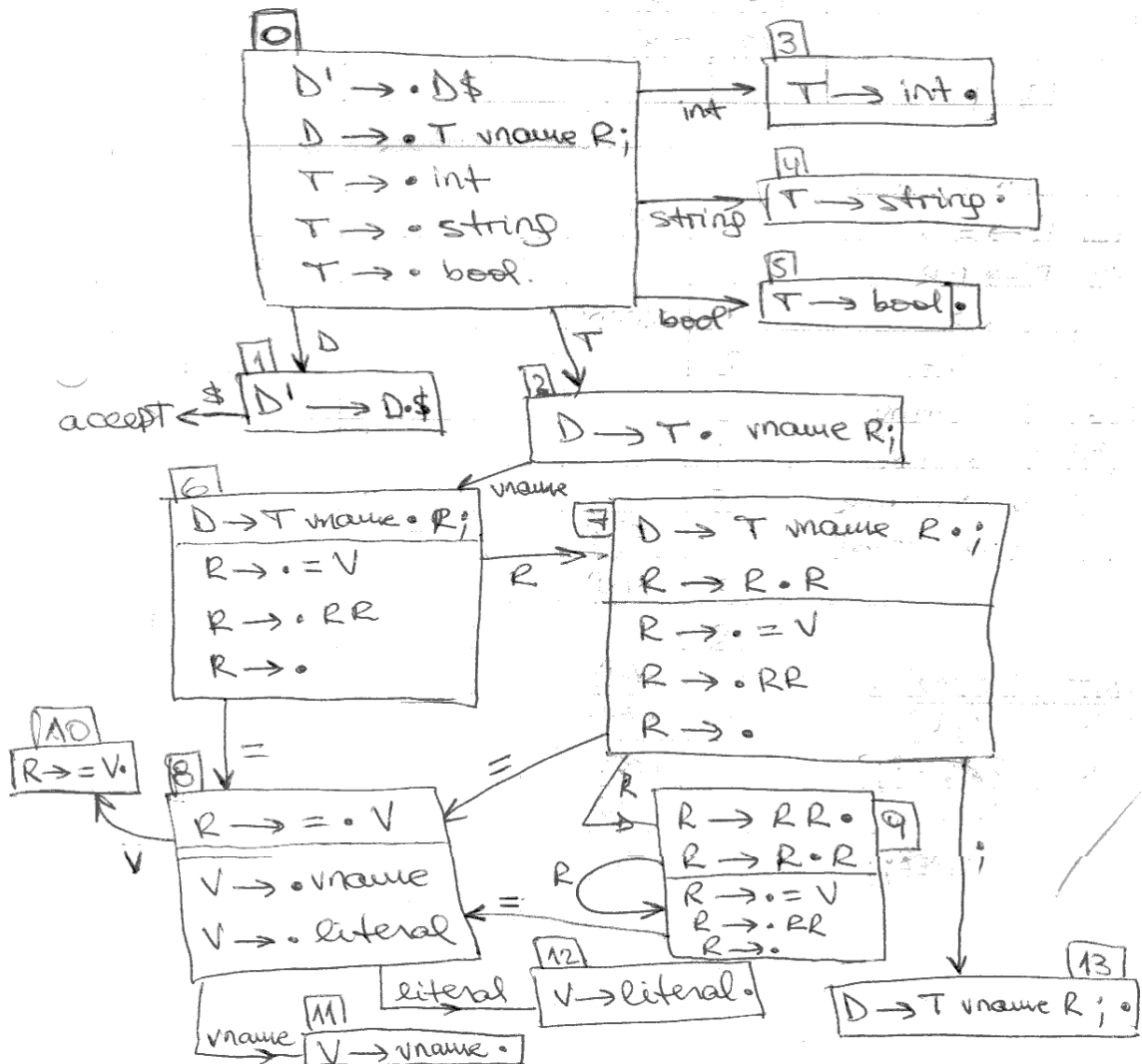
```

Ejercicio 3

$$G_3 = \langle \{D, R, T, V\}, \{vname, ;, =, int, string, bool, literal\}, D, P_3 \rangle$$

$$P_3: \begin{aligned} D &\rightarrow T \ vname \ R; \\ R &\rightarrow = V \mid R \ R \mid \lambda \\ T &\rightarrow int \mid string \mid bool \\ V &\rightarrow vname \mid literal. \end{aligned}$$

a) Aumento la gramática con $D' \rightarrow D$ y hago a D' el $\$$ inicial. Ahora armo el autómata.



La tabla queda:

Estado	vname	Action					
		;	=	int	string	bool	literal
0				s3	s4	s5	
1							
2	s6						
3	r(5)						
4	r(6)						
5	r(7)						
6		r(4)	r(4)/s8				
7		s13/r(4)	s8/r(4)				
8	s11						s12
9		r(3)	s8/r(4)				
10		r(2)	r(2)				
11		r(8)	r(8)				
12		r(9)	r(9)				
13							

(1) $D \rightarrow T \text{ vname } R ;$
 (2) $R \rightarrow = V$
 (3) $R \rightarrow RR$
 (4) $R \rightarrow \lambda$
 (5) $T \rightarrow \text{int}$
 (6) $T \rightarrow \text{string}$
 (7) $T \rightarrow \text{bool}$
 (8) $V \rightarrow \text{vname}$
 (9) $V \rightarrow \text{literal}$

NT	siguientes	0	1	2	3	4	5	6	7	8	9	10	11	12	13
D	{ \$ }														
R	{ =, ; }														
T	{ vname }														
V	{ =, ; }														r(1)

GOTO

b)

G_3 **no** es SLR dado que tiene conflictos.

Con las siguientes decisiones logramos parsear todas las cadenas posibles sin conflictos

- (6, =): s8
- (7, =): s8
- (7, ;): s13
- (9, =): r3
- (9, ;): r3

La más clara es que en el conflicto s13/r(4) de (7, ;) necesito que esté s13 porque si no nunca proceso ';', que siempre está en las cadenas.

Para las otras decisiones, prestemos atención a la gramática.

$$\begin{aligned}
 D &\rightarrow T \text{ vname } R; \\
 R &\rightarrow = V \mid RR \mid \lambda \\
 T &\rightarrow \text{int} \mid \text{string} \mid \text{bool} \\
 V &\rightarrow \text{vname} \mid \text{literal}
 \end{aligned}$$

Podríamos decir que el “lenguaje” generado es el mismo que el de la “expresión regular”

$$(\text{int} \mid \text{string} \mid \text{bool}) \text{ vname } (= V)^* ;$$

(Las comillas se sacarían si V se reemplazara por $(\text{vname} \mid \text{literal})$, pero para los fines de la explicación es mejor que quede esa expresión, pensemos que es un símbolo terminal.)

Vamos a separar en 3 casos:

- I. $(\text{int} \mid \text{string} \mid \text{bool}) \text{ vname} ;$
- II. $(\text{int} \mid \text{string} \mid \text{bool}) \text{ vname} = V ;$
- III. $(\text{int} \mid \text{string} \mid \text{bool}) \text{ vname} (= V)^k ;$ considerando $k \geq 2$

Sin pérdida de generalidad, vamos a ver que parsea las cadenas con `int`, separando en 3 casos:

I.

Pila	Símbolos	Cadena	Acción
0	\$	<code>int vname ; \$</code>	s3
0 3	<code>\$ int</code>	<code>vname ; \$</code>	r5
0 2	<code>\$ T</code>	<code>vname ; \$</code>	s6
0 2 6	<code>\$ T vname</code>	<code>; \$</code>	r4
0 2 6 7	<code>\$ T vname R</code>	<code>; \$</code>	s13
0 2 6 7 13	<code>\$ T vname R ;</code>	<code>\$</code>	r1
0 1	<code>\$ D</code>	<code>\$</code>	accept

II.

Pila	Símbolos	Cadena	Acción
0	\$	int vname = V ; \$	s3
0 3	\$ int	vname = V ; \$	r5
0 2	\$ T	vname = V ; \$	s6
0 2 6	\$ T vname	= V ; \$	s8
0 2 6 8	\$ T vname =	V ; \$	s10
0 2 6 8 10	\$ T vname = V	; \$	r2
0 2 6 7	\$ T vname R	; \$	s13
0 2 6 7 13	\$ T vname R ;	\$	r1
0 1	\$ D	\$	accept

III.

Pila	Símbolos	Cadena	Acción
0	\$	int vname = V = V ; \$	s3
0 3	\$ int	vname = V = V ; \$	r5
0 2	\$ T	vname = V = V ; \$	s6
0 2 6	\$ T vname	= V = V ; \$	s8
0 2 6 8	\$ T vname =	V = V ; \$	s10
0 2 6 8 10	\$ T vname = V	= V ; \$	r2
0 2 6 7	\$ T vname R	= V ; \$	s8
0 2 6 7 8	\$ T vname R =	V ; \$	s10
0 2 6 7 8 10	\$ T vname R = V	; \$	r2
0 2 6 7 9	\$ T vname R R	; \$	r3
0 2 6 7	\$ T vname R	; \$	s13
0 2 6 7 13	\$ T vname R ;	\$	r1
0 1	\$ D	\$	accept

Pila	Símbolos	Cadena	Acción
0	\$	int vname = V = V = V ; \$	s3
0 3	\$ int	vname = V = V = V ; \$	r5
0 2	\$ T	vname = V = V = V ; \$	s6
0 2 6	\$ T vname	= V = V = V ; \$	s8
0 2 6 8	\$ T vname =	V = V = V ; \$	s10
0 2 6 8 10	\$ T vname = V	= V = V ; \$	r2
0 2 6 7	\$ T vname R	= V = V ; \$	s8
0 2 6 7 8	\$ T vname R =	V = V ; \$	s10
0 2 6 7 8 10	\$ T vname R = V	= V ; \$	r2
0 2 6 7 9	\$ T vname R R	= V ; \$	r3
0 2 6 7	\$ T vname R	= V ; \$	s8
...
0 1	\$ D	\$	accept

Cada = V se consume y se vuelve a esta línea roja con un = V menos que leer en la cadena, así como de la verde se llegó a esta roja. Va a seguir así hasta llegar a lo azul de la tabla anterior.