Universidad Simón Bolívar
Departamento de Computación y Tecnología de la Información
CI3725 - Traductores e Interpretadores
Septiembre-Diciembre 2012

Examen Parcial III

(25 puntos)

Carnet:

Nombre:

1. Considere la gramática

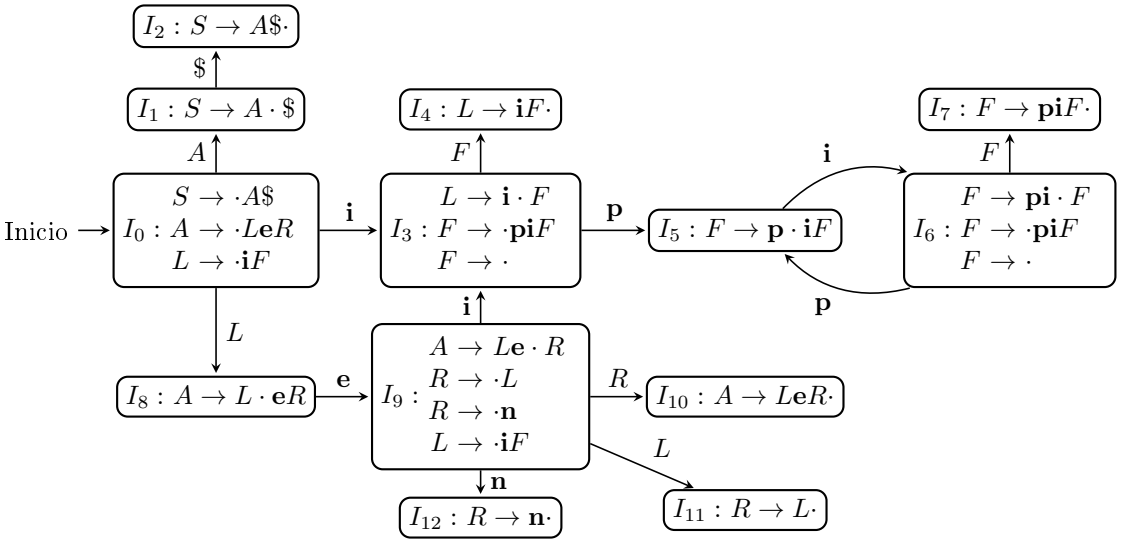
$$\begin{array}{lcl} A & \to & L\mathbf{e}R \\ L & \to & \mathbf{i}F \\ F & \to & \mathbf{pi}F \\ F & \to & \lambda \\ R & \to & L \\ R & \to & \mathbf{n} \end{array}$$

a) **(3 puntos)** Calcule el Autómata de Prefijos Viables para la gramática extendida y determine si la gramática es LR(0).

Aumentamos la Gramática con un nuevo símbolo inicial y enumeramos cada producción

$$\begin{array}{lcl} \text{Regla 0:} & S & \to & A\$ \\ \text{Regla 1:} & A & \to & L\,\mathbf{e}\,R \\ \text{Regla 2:} & L & \to & \mathbf{i}\,F \\ \text{Regla 3:} & F & \to & \mathbf{p}\,\mathbf{i}\,F \\ \text{Regla 4:} & F & \to & \lambda \\ \text{Regla 5:} & R & \to & L \\ \text{Regla 6:} & R & \to & \mathbf{n} \end{array}$$

El Autómata de Prefijos Viables nos queda como



La Gramática **no** es LR(0) pues presenta conflictos *shift-reduce* en los conjuntos I_3 e I_6.

b) **(2 puntos)** Construya la Tabla de Parsing SLR usando el algoritmo descrito en clase.
Para construir la Tabla de Parsing SLR es necesario calcular el $FOLLOW$ de todos los símbolos no terminales. Así tenemos

$$\begin{aligned} FIRST(S) = FIRST(A) = FIRST(L) &= \{\mathbf{i}\} \\ FIRST(P) &= \{\lambda, \mathbf{p}\} \\ FIRST(R) &= \{\mathbf{i}, \mathbf{n}\} \\ FOLLOW(S) = FOLLOW(A) = FOLLOW(R) &= \{\$\} \\ FOLLOW(L) = FOLLOW(F) &= \{\$, \mathbf{e}\} \end{aligned}$$

y la Tabla de Parsing nos queda

Estado	**e**	**i**	**n**	**p**	$	A	L	F	R
0		s3				1	8		
1					s2				
2					accept				
3				s5				4	
4	r2				r2				
5		s6							
6	r4			s5	r4			7	
7	r3				r3				
8	s9								
9		s3	s12				11		10
10					r1				
11					r5				
12					r6				

c) **(2 puntos)** Utilice el parser SLR construido para obtener la derivación más derecha para la palabra **ipipien**.

Entrada	Pila	Acción
ipipien$	0$	shift 3
pipien$	30$	shift 5
ipien$	530$	shift 6
pien$	6530$	shift 5
ien$	56530$	shift 6
en$	656530$	reduce 4; pop 0; goto(6,F)
en$	7656530$	reduce 3; pop 3; goto(6,F)
en$	76530$	reduce 3; pop 3; goto(3,F)
en$	430$	reduce 2; pop 2; goto(0,L)
en$	80$	shift 9
n$	980$	shift 12
$	12980$	reduce 6; pop 1; goto(9,R)
$	10980$	reduce 1; pop 3; goto(0,A)
$	10$	shift 2
$	210$	accept

y la derivación más derecha para la palabra **ipipien** se construye usando en orden inverso las reglas indicadas por las reducciones, por tanto

$$\begin{aligned} A &\Rightarrow^1 L\mathbf{e}\underline{R} \\ &\Rightarrow^6 \underline{L}\mathbf{en} \\ &\Rightarrow^2 \mathbf{i}\underline{F}\mathbf{en} \\ &\Rightarrow^3 \mathbf{ipi}\underline{F}\mathbf{en} \\ &\Rightarrow^3 \mathbf{ipipi}\underline{F}\mathbf{en} \\ &\Rightarrow^4 \mathbf{ipipien} \end{aligned}$$

2. **(7 puntos)** Se cuenta con un robot autónomo programable capaz de desplazarse sobre una cuadrícula bidimensional y con capacidad de transportar M muestras de terreno. Se le indican las tareas enviando una cadena no vacía de hasta N símbolos que constituye la secuencia de comandos. El robot recibe la secuencia de comandos y las ejecuta inmediatamente, por lo que es necesario que dicha secuencia sea verificada antes de ser enviada para asegurar que puede ser ejecutada por el robot. La secuencia de comandos puede contener cualquier combinación de los símbolos

   - **n**, **s**, **e**, **o** – para moverse un paso en la dirección correspondiente al punto cardinal indicado.
   - **p** – para tomar una muestra de terreno.

   El robot siempre sale de su base, ubicada en las coordenadas (0,0), con la batería al 100 % de capacidad. Tomar una muestra de terreno consume una unidad de batería para accionar el brazo mecánico. Cada movimiento consume una unidad de batería por la acción propiamente dicha, más una unidad de batería por cada muestra de terreno que esté siendo transportada.

   Proponga una Gramática de Atributos *formal* que solamente acepte secuencias de movimientos en las cuales el robot pueda cumplir todas sus tareas regresando al punto de origen.

   Suponga que los *tokens* del lenguaje **no** tienen ningún valor intrínseco asociado excepto por el *lexema*. Para la construcción de la Gramática de Atributos puede utilizar atributos heredados o sintetizados según le resulte conveniente.

   Defino la Gramática de Atributos

   $$G \quad = \quad (\{S, L, I\}, \{\mathbf{n}, \mathbf{s}, \mathbf{e}, \mathbf{o}, \mathbf{p}\}, P, S)$$

   con las producciones que se indican más abajo.

   a) El no-terminal I modela una instrucción simple.

      - Tiene los atributos sintetizados dx y dy de tipo entero para indicar el desplazamiento horizontales o vertical, respectivamente, consecuencia de la instrucción, con valores posibles -1, 0 o 1.
      - Tiene el atributo sintetizado c de tipo entero para indicar la cantidad de carga agregada al robot como consecuencia del comando ejecutado, con valores posibles 0 o 1.
      - Tiene el atributo sintetizado m de tipo entero para indicar si el comando es un movimiento o no, con valores posibles 0 o 1. Este será utilizado como coeficiente para determinar el consumo de batería durante los movimientos.

   b) El no-terminal L modela la lista de instrucciones, la cual se construye de izquierda a derecha.

      - Tiene los atributos sintetizados ox y oy de tipo entero para indicar la posición *después* de procesar la instrucción. En el caso base, toman su valor en base a los atributos dx y dy de la instrucción inicial, pues el robot parte de la coordenada (0,0). En el caso recursivo, como la lista se construye de izquierda a derecha, la posición final de la lista de instrucciones se calcula tomando la posición final de la lista parcial y agregando los atributos dx y dy de la última instrucción ejecutada.
      - Tiene el atributo sintetizado oc de tipo entero para indicar la carga *después* de procesar la lista. En el caso base, toma su valor en base al atributo c de la instrucción inicial, pues el robot parte con carga 0. En el caso recursivo, como la lista se construye de izquierda a derecha, la carga final de la lista de instrucciones se calcula tomando la carga de la lista parcial y agregando la carga c de la última instrucción ejecutada.
      - Tiene el atributo sintetizado n de tipo entero para contar el número de instrucciones en la lista. En el caso base, toma su valor inicial 1. En el caso recursivo, simplemente se incrementa en uno el conteo de instrucciones.
      - Tiene el atributo sintetizado b de tipo entero para acumular el consumo de batería requerido para completar los movimientos. En el caso base, como no hay ninguna carga todavía, el consumo de batería siempre será 1 independientemente de la instrucción que se ejecute. En el caso recursivo, el consumo de batería de la lista parcial se incrementa en 1 en virtud de la instrucción que se está ejecutando, y se agregan tantas unidades de consumo como carga tenga el robot, sólo si la instrucción es un movimiento

c) El no-terminal S se define exactamente como lo especifica el enunciado. Utiliza el atributo sintetizado ok de tipo booleano para verificar la viabilidad del camino verificando la posición final, que la carga sea no mayor a M, que la longitud sea no mayor a N y que no se exceda la capacidad de la batería.

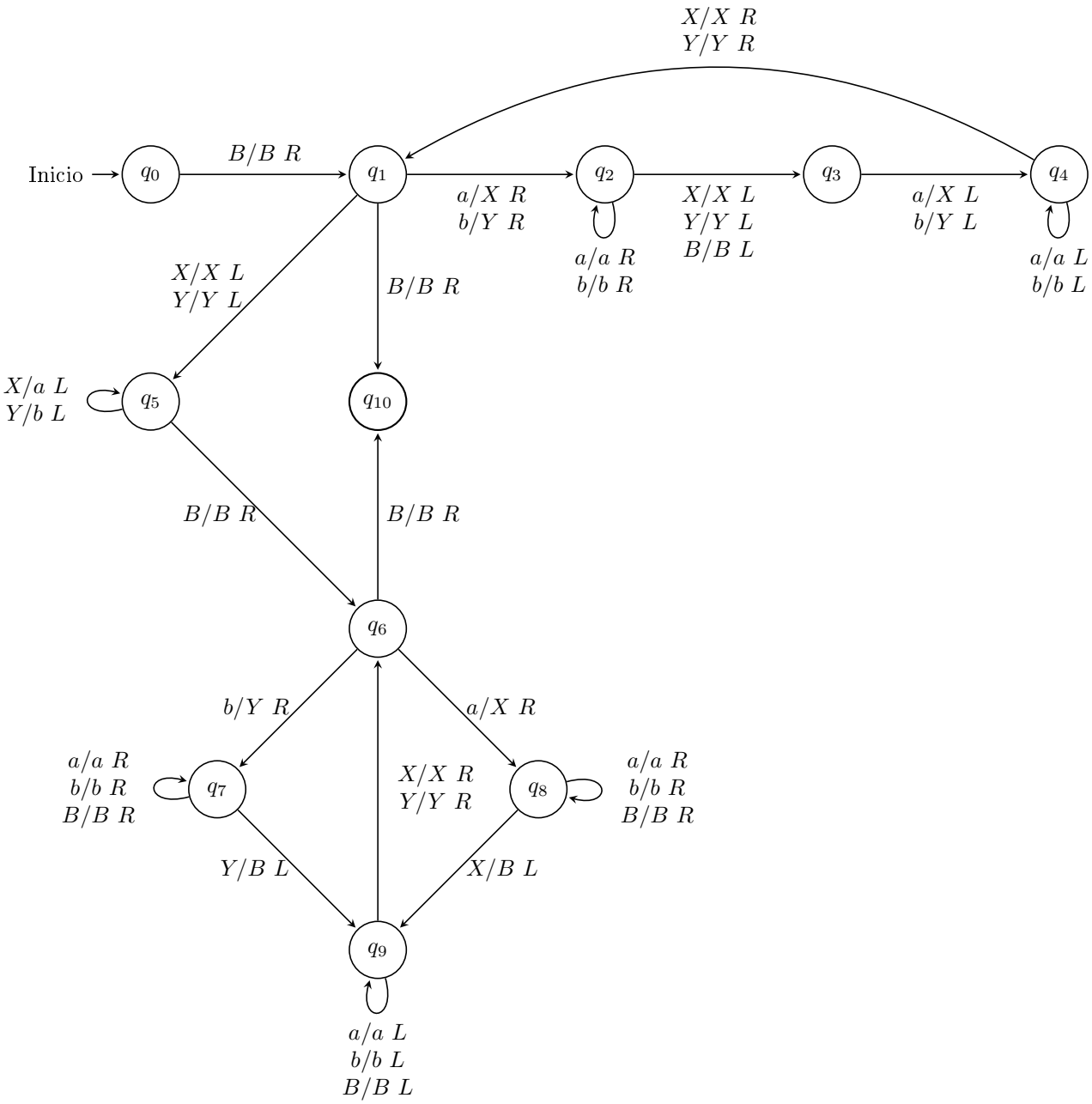
Las producciones de la gramática junto con sus acciones semánticas quedan como sigue

$$
\begin{array}{lcll}
S & \to & L & \{\ S.ok \leftarrow L.ox = 0 \land L.oy = 0 \land L.oc \leq M \land L.n \leq N \land L.b \leq 100 \\
L & \to & I & \left\{ \begin{array}{rcl} L.ox & \leftarrow & I.dx \\ L.oy & \leftarrow & I.dy \\ L.oc & \leftarrow & I.c \\ L.n & \leftarrow & 1 \\ L.b & \leftarrow & 1 \end{array} \right. \\
L & \to & L_1\, I & \left\{ \begin{array}{rcl} L.ox & \leftarrow & L_1.ox + I.dx \\ L.oy & \leftarrow & L_1.oy + I.dy \\ L.oc & \leftarrow & L_1.oc + I.c \\ L.n & \leftarrow & L_1.n + 1 \\ L.b & \leftarrow & L_1.b + L_1.oc * I.m + 1 \end{array} \right. \\
I & \to & \mathbf{n} & \left\{ \begin{array}{rcl} I.dx & \leftarrow & 0 \\ I.dy & \leftarrow & 1 \\ I.c & \leftarrow & 0 \\ I.m & \leftarrow & 1 \end{array} \right. \\
I & \to & \mathbf{s} & \left\{ \begin{array}{rcl} I.dx & \leftarrow & 0 \\ I.dy & \leftarrow & -1 \\ I.c & \leftarrow & 0 \\ I.m & \leftarrow & 1 \end{array} \right. \\
I & \to & \mathbf{e} & \left\{ \begin{array}{rcl} I.dx & \leftarrow & 1 \\ I.dy & \leftarrow & 0 \\ I.c & \leftarrow & 0 \\ I.m & \leftarrow & 1 \end{array} \right. \\
I & \to & \mathbf{o} & \left\{ \begin{array}{rcl} I.dx & \leftarrow & -1 \\ I.dy & \leftarrow & 0 \\ I.c & \leftarrow & 0 \\ I.m & \leftarrow & 1 \end{array} \right. \\
I & \to & \mathbf{p} & \left\{ \begin{array}{rcl} I.dx & \leftarrow & 0 \\ I.dy & \leftarrow & 0 \\ I.c & \leftarrow & 1 \\ I.m & \leftarrow & 0 \end{array} \right.
\end{array}
$$

3. **(7 puntos)** Construya una Máquina de Turing *determinística* que acepte el lenguaje $L = \{ww \,|\, w \in \{a,b\}^*\}$.
La Máquina de Turing $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$ puede ser como sigue

$$\begin{aligned} Q &= \{q_0, q_1, q_2, q_3, q_4, q_5, q_6, q_7, q_8, q_9, q_{10}\} \\ \Sigma &= \{\mathbf{a}, \mathbf{b}\} \\ \Gamma &= \{\mathbf{a}, \mathbf{b}, \mathbf{X}, \mathbf{Y}, \mathbf{B}\} \\ F &= \{q_{10}\} \end{aligned}$$

cuya función de transición es explícita en el diagrama



4. **(4 puntos)** Demuestre que no existe un **algoritmo** tal que reciba la codificación de una Máquina de Turing M, la codificación de un número de estado y la palabra de entrada w, que pueda **decidir** si M pasa al menos una vez por el estado particular mientras procesa w.

   Asumamos que existe la Máquina de Turing $L_{reaches}$ definiendo un **algoritmo** que recibe como entrada la representación de una Máquina de Turing M, la codificación de un número de estado r y la palabra de entrada w, aceptando si al procesar w con la máquina M esta pasa al menos una vez por el estado r.

   Consideremos ahora la Máquina de Turing Pre-procesadora P definida de la siguiente forma:

   a) La entrada P es la Representación de una Máquina de Turing M y una cadena $w \in \{0,1\}^*$, i.e. $R(M)w$.

   b) La máquina P verifica que la entrada corresponda a una representación válida para una máquina. En caso contrario, se detiene rechazando la entrada.

   c) La máquina P construye la representación de una nueva Máquina de Turing M' tal que:

   - Se agrega un estado a la máquina M. Sea p la codificación para dicho estado.
   - Para todo estado q en M y todo símbolo $a \in \Sigma$ que **no** tenga transición definida, se define una nueva transición hacia el estado adicional p de la máquina M'. De este modo, en cualquier estado q en donde M se detendría, ahora pasa al estado p de M'.

   d) La máquina P emite por su salida $R(M')pw$

   Ahora, podemos construir la Máquina de Turing H que recibe como entrada la Representación de una Máquina de Turing M y una cadena $w \in \Sigma^*$. Esta máquina:

   a) Recibe la entrada $R(M)w$ y la pasa al pre-procesador P que construye la máquina M'.

   b) La máquina P emite $R(M')pw$ por su salida, la cual es pasada a la máquina $L_{reaches}$.

   Es claro que H aceptará $R(M)w$ si y sólo si $L(M')$ acepta $R(M')pw$. La construcción de M' nos asegura que esta alcanzará el estado p siempre y cuando M se detenga con w, y que nunca alcanzará el estado p si M no se detiene con w. Esto quiere decir que H es equivalente al Problema de la Parada para Máquinas de Turing, y lo hemos reducido a $L_{reaches}$ utilizando P como pre-procesador. Sabemos que H no es decidible, por tanto $L_{reaches}$ no puede ser decidible como habíamos supuesto; en consecuencia **no existe** un **algoritmo** que decida si M pasa al menos una vez por un estado particular mientras procesa w.

5. **(3 puntos extra)** Suponga que L es un lenguaje recursivamente enumerable pero **no** es recursivo. Demuestre que si M es una Máquina de Turing que acepta precisamente las palabras de L entonces existen *infinitas* palabras de entrada para las cuales M se cuelga.

   Nota: si la sumatoria de los puntos obtenidos en las preguntas 1 a 4 **no** son suficientes para aprobar el examen, esta pregunta **no** será tomada en cuenta.

   Como L es recursivamente enumerable pero no es recursivo, entonces para *cualquier* Máquina de Turing M que reconozca palabras de L siempre existe al menos una entrada $w \notin L$ tal que M se cuelga al procesarla. Supongamos que el conjunto $C = \{w | w \notin L \text{ que hace colgar a } M\}$ es finito.

   Como C es un lenguaje finito, podemos construir una nueva Máquina de Turing M' que lo decida. Esto es:

   a) Si M' recibe $x \in C$ como entrada, se detiene y acepta.

   b) Si M' recibe $x \notin C$ como entrada, se detiene y rechaza.

   Pero entonces, podríamos construir otra Máquina de Turing M'' que aproveche a M' para verificar su entrada de manera tal que:

   a) Si M' acepta la entrada x de M'', entonces M'' se detiene y rechaza, porque x haría colgar a M.

   b) Si M' rechaza la entrada x de M'', entonces M'' ejecuta M con la entrada x. Sabemos que x no puede colgar a M, de modo que lo único que se espera es que M la acepte o rechace.

   Pero la máquina M'' que acabamos de construir efectivamente **decide** a L lo cual contradice la premisa de que L **no es** recursivo. Esta contradicción sólo puede ser consecuencia de haber supuesto que el conjunto C es finito, de manera que el conjunto de palabras $x \notin L$ que hacen colgar a M tiene que ser infinito.