Teoría de Lenguajes

2do. Parcial – Curso 2023
Soluciones

Ejercicio 1 [Evaluación individual del obligatorio]

a) Responda las siguientes preguntas respecto a NLTK y la entrega realizada:

i) ¿Cómo es la sintaxis para definir gramáticas libre de contexto de NLTK? Responda concentrándose en variables, terminales y reglas. Dé un ejemplo sencillo que lo ayude a mostrar la sintaxis descrita.

En NLTK las GLC mediante un string multilinea (""" ... """) se definen de la siguiente manera:

- Variables: se escriben directamente (ej. A)
- Terminales: se escriben entre comillas simples (ej. 'a')
- Reglas: se utiliza la secuencia -> para separar el lado izquierdo del derecho. Las variables y terminales del lado derecho de las reglas se separan con espacios. Varias reglas con el mismo lado izquierdo pueden escribirse de forma compacta con el carácter | (ej. A -> 'a' | B C 'd')
- Símbolo inicial: es la variable del lado izquierdo de la primera regla.

A continuación se presenta un ejemplo de gramática
""" S → 'a' S 'b' | 'a' 'b' """

ii) Enumere las funcionalidades de NLTK que fueron utilizadas para resolver el laboratorio 2.

ii) Las funcionalidades de NLTK utilizadas en el laboratorio son:

- La representación de la gramática. La lectura se realizó a partir de un string con la notación descripta en la parte anterior.
- Tokenización de la entrada. Opcionalmente podría realizarse sin NLTK, por ejemplo con una expresión regular.
- Construcción del árbol sintáctico (en caso de existir) con alguno de los parsers disponibles en NLTK.

iii) Describa el retorno de la función *parse* de los *parsers* para gramáticas libre de contexto de NLTK.

El retorno de la función parse para GLC de NLTK es un generador de los árboles para la entrada y la gramática en cuestión.

También se considerará correcta la siguiente respuesta:

El retorno de la función parse para GLC de NLTK es una lista con los árboles para la entrada y la gramática en cuestión, o la lista vacía en caso que la entrada no sea reconocida por la gramática.

b) ¿Cómo resolvió la *tokenización* de los programas entregados? En caso de haber utilizado diferentes formas dependiendo el programa, indique cada caso.

Esta respuesta depende de cada entrega.
Podía realizarse mediante un tokenizador de NLTK, mediante una expresión regular o el método split de Python, entre otras.

c) En el programa 3 (reconocimiento de sintaxis inspirada en Python), ¿de qué forma reutilizó las gramáticas de las partes anteriores?

Está respuesta depende de cada entrega.

Una posible forma de hacerlo es concatenando la gramática reutilizada al final de la gramática que la reutiliza, usando diferentes símbolos iniciales en cada una.

Ejercicio 2

a) $L_2 = \{a^p b^j a^{p \bmod 2} \text{ con } p \geq 0, j \geq 0\}$

Notar que si p es par, entonces la tira tiene cantidad par de **a**' y puede o no seguir con **b**'s. Si p es impar, puede o no seguir con **b**'s pero terminan en **a** (p mod 2). Hay que discriminar esos 2 casos según cantidad de **a**'s al comienzo

El lenguaje es regular. Se construye una gramática lineal derecha con las siguientes reglas de producción:

S –> aI | ε | bY |b
I –> aS | a | bX
X –> bX | a
Y –> bY | b

Esta gramática está simplificada por tener todos símbolos útiles, no tiene unitarias y la única producción con épsilon es S –> ε

b) ¿Las siguientes afirmaciones son Verdaderas o Falsas?. Justifique.

i) Si L_a es un lenguaje recursivamente enumerable pero no libre de contexto y L_b es un lenguaje libre de contexto pero no regular (ambos definidos sobre el mismo alfabeto),
entonces $L_a \cap L_b = \varnothing$

Falso. Sea $L_a = \{a^k b^k c^k \,/\, k \geq 0, \}$ que es recursivamente enumerable y no libre de contexto y $L_b = \{a^k b^k \,/\, k \geq 0 \}$ que es libre de contexto pero no regular. Se puede observar que si se hace su intersección $L_a \cap L_b = \{ \varepsilon \} \neq \varnothing$

ii) Si L_c y L_d son lenguajes recursivamente enumerables cualquiera, entonces $L_c.L_d$ puede ser un lenguaje libre de contexto no regular.

Verdadero. Como dice *puede ser* alcanza con encontrar un ejemplo que se cumpla.
Sean $L_c = \{0^k 1^k, k>0\}$ y $L_d = \{1\}$ Ambos son libres de contexto; en particular L_d es regular (está formado por un único string de un símbolo). Además son por Jerarquía de Chomsky, recursivamente enumerables.
La concatenación de esos lenguajes es $\{0^k 1^{k+1}, k>0\}$ que es libres de contexto. Para ver que no es regular, se realiza un PL similar al realizado en el teórico con $\{0^k 1^k, k>0\}$.

Ejercicio 3

L_3= { b^{k+s} #b^k#a $^{t+s}$ con s ≥ 0, k,t ≥ 1 }

a) Como dice la letra que no es regular, se verá en la parte b) que se puede construir una Gramática Libre de Contexto, con lo cual justamente se probaría entonces que es un Lenguaje Libre de Contexto.

b) El lenguaje L_3 se puede reescribir de la siguiente forma

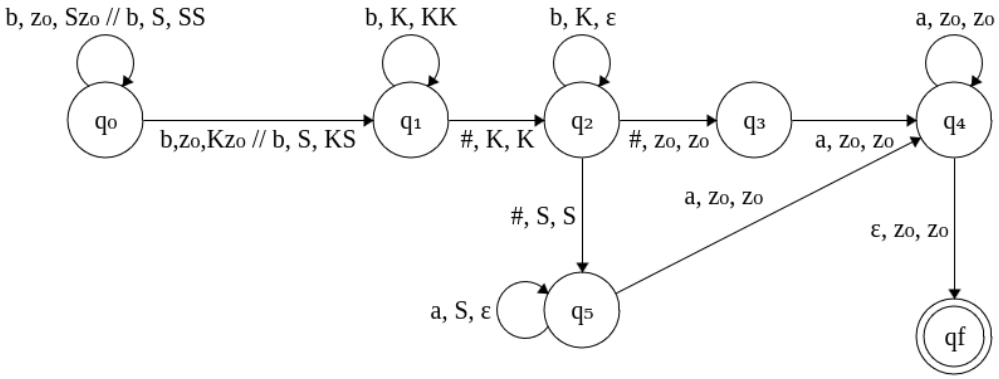
L_3 = {b^{k+s}#b^k#a^{t+s}: s≥0, k,t ≥ 1} = {b^{s+k}#b^k#a^{s+t}: s≥0, k,t ≥ 1} = {$b^s b^k$#b^k#$a^s a^t$: s≥0, k,t ≥ 1}
= {$b^s b^k$#b^k#a^s: s≥0, k ≥ 1}.{a^t: t ≥ 1}

A partir de la concatenación de lenguajes se nota que en el conjunto de la izquierda las dependencias de los símbolos con cantidad *s* están en los extremos y en el centro aquellos con cantidad *k*. Este es el lenguaje no regular, y además es libre de contexto, lo que se probará dando un autómata que lo reconozca y una gramática que lo genere.

La gramática surge de la reescritura del lenguaje. Primero se resuelven la cantidad *s* de los extremos y luego la cantidad *k* del centro. Por último se concatena la cantidad *t* de símbolos "a".

S → XT
X → bXa | K#
K → bKb | b#b
T → aT | a

c) Se construye entonces un APD (M_3) que lo reconozca



El APD es No Determinista porque existen: $\delta(q_4,a,Zo)$ y $\delta(q_4,\varepsilon,Zo)$

Ejercicio 4

L_4={ $0^a 1^b 0^c 1^d$ con c=min(a,b) ; d=max(a,b) ; a,b ≥ 1}

a) El lenguaje L_4 es recursivamente enumerable como se verá en la parte b) al construir una G.I. que lo genera. Se demostrará que NO es libre de contexto utilizando el CR del Pumping Lema (para Libres de Contexto).

Para eso se elige la tira z = $0^N 1^{N+1} 0^N 1^{N+1}$, siendo N la constante del PL.

Nota: Las gramáticas y los autómatas deben corresponderse con el tipo del lenguaje considerado en cada caso, según la Jerarquía de Chomsky. Se valora positivamente la simplicidad de las soluciones propuestas así como una breve explicación de éstas. Todas las respuestas deben estar debidamente justificadas.

El siguiente paso es considerar todas las descomposiciones posibles de la tira z en uvwxy que cumplen $|vwx| \leq N$ y $|vx|>0$, y se prueba que existe un i tal que uv^iwx^iy no pertenece a L_4

Familia	0^N	1^{N+1}	0^N	1^{N+1}
1	v x			
2		v x		
3			v x	
4				v x
5	v	x		
6			v	x
7		v	x	
8	v x	x		
9	v	v x		
10			v x	x
11			v	v x
12		v x	x	
13		v	v x	

Familia 1
$u = 0^j$
$v = 0^k$
$w = 0^l$
$x = 0^m$
$y = 0^{N-j-k-l-m}1^{N+1}0^N1^{N+1}$ con $k+m>0$

$z_i = 0^{N+(i-1)*(k+m)}1^{N+1}0^N1^{N+1}$

Tomando i=3, $z_3 = 0^{N+2(k+m)}1^{N+1}0^N1^{N+1}$ que no pertenece al lenguaje pues al ser $k+m>0$, la cantidad de 0's del comienzo es mayor que la cantidad de 1's que le siguen, con lo cual **no** se cumple la propiedad de c=min(a,b) que corresponde al 2do bloque de 0's, que es N y permanece fija.

Familia 2
Es análoga, pero razonando con los 1's. El primer bloque de 1's crece y el 2do permanece fijo.

Familia 3
Es también con un razonamiento similar a la Familia 1, sólo que la cantidad de 0's del 2do bloque al aumentar, no coincide con el menor de los los 0's o 1's anteriores (falla propiedad de c=min(a,b))

Familia 4
Idem Familia 3 pero razonando con los 1's que son los que varían.

Familias 5
$u = 0^{N-k-p}$
$v = 0^k$ $k+r>0$
$w = 0^p1^l$ $k+p+l+r\leq N$
$x = 1^r$
$y = 1^{N+1-l-r}0^N1^{N+1}$

Nota: Las gramáticas y los autómatas deben corresponderse con el tipo del lenguaje considerado en cada caso, según la Jerarquía de Chomsky. Se valora positivamente la simplicidad de las soluciones propuestas así como una breve explicación de éstas. Todas las respuestas deben estar debidamente justificadas.

$z_i = 0^{N+(i-1)k}1^{N+1+(i-1)r}0^N1^{N+1}$

Tomando i=0, $z_0 = 0^{N-k}1^{N+1-r}0^N1^{N+1}$ donde:

- si k=0, r>0 entonces la cantidad de 1's del primer bloque es menor que la cantidad de 0's del primer bloque y por tanto fallan las condiciones de c=min(a,b) y d=max(a,b), las cuales permanecen fijas.
- si r=0, k>0 y el razonamiento es análogo pero con los 0's
- si ambos (k,r) >0 tampoco se cumple ninguna de las condiciones

Siempre pasa lo mismo, al permanecer fijos las cantidades del 2do bloque de 0's y 1's

Familias 6 y 7

Idem Familia 5 pero las que varían son los 2dos bloques de 0's y 1's (Familia 6) o cambian los primeros 1's y/o los segundos 0's (Familia 7) con lo cual tampoco va a cumplir las condiciones de la estructura del lenguaje L_4.

Familia 8

$u = 0^{N-k-l-p}$
$v = 0^k$ $k+p+m > 0$
$w = 0^l$ $k+l+m+p \leq N$
$x = 0^p1^m$
$y = 1^{N+1-m}0^N1^{N+1}$

$z_i = 0^{N+(i-1)k}(0^p1^m)^i 1^{N+1-m}0^N1^{N+1}$

Asumimos:

- p>0 porque sino estamos en Familia 5
- m>0 porque sino estamos en Familia 1

Tomando i=2, $z_2 = 0^{N+k+p}1^m0^p1^{N+1}0^N1^{N+1}$ de donde como ya se dijo que p>0 se mezclan 0's con 1's, con lo cual z_2 no pertenece a L_4

Familias 9, 10, 11, 12 y 13

Son análogas a la Familia 8, haciendo en cada caso la salvedad de no caer en los casos ya estudiados, el argumento es el mismo ya que se mezclan 0's con 1's.

Como estas son todas las descomposiciones posibles de z en uvwxy que cumplen |vx|>0 y $|vwx| \leq N$, por el CR del PL el lenguaje L_4 **NO** es libre de contexto.

b) Se construye una Gramática Irrestricta para mostrar que L_4 es recursivamente enumerable

La idea es: genero por cada 0 una A y por cada 1 una B, esos pares AB los junto y ahí voy a saber el numero mínimo (genero los 0^c y la primer parte de los 1^d). Luego, las A's o B's que me sobren las sumo para el max, es decir que se agregan al 1^d.
Acepta que a y b sean iguales

S → IXYF

X → 0XA | 1A // genero 0^a y una A por cada 0

Y → 1YB | 1B // genero 1^b y una B por cada 1

A1 → 1A // muevo los 1^b a la izquierda (al lado de 0^b)

AB → CU // junto los AB generados, para ver cuanto es el mínimo, acá genero una C (cero) de 0^c y una U (uno) 1^d por cada par

AC → CA // muevo los C generados a la izquierda para dejar AB's juntos

AU → UA // muevo los U generados a la izquierda

UC → CU // ordeno los UC generados, yo se que quiero generar $0^c\,1^d$
obs: si no se ordenan todos los CU y se ejecuta la regla CU → R01T queda trancada por tener variables CU al lado de R y T

I0 → 0I // muevo la marca inicial hasta donde termina 0^a1^b en un principio, luego me salteo también todos los 0,1's que voy generando

I1 → 1I // muevo marca inicial

CU → R01T // en el CU central (voy a tener algo como CCCUUU) genero el 01 central y genero las marcas R y T que van a cambiar las otras C's y U's por 0's y 1's

CR → R0 // la marca R se encuentra con una C, lo cambia por un 0 y muevo R

TU → 1T // la marca T se encuentra con una U, lo cambia por un 1 y muevo T

IR → I // significa que la marca R ya cambió todas la C's por 0's, entonces desaparece

IT → I // como la marca I avanza hacia la derecha al cruzar 0's o 1's, cuando se cambian todas las C's por 0's I avanza a la derecha, y cuando llega hacia T, significa que se cambiaron también todas las U's por 1's

AF → F1 // cuando queda una A al lado de la F, significa que consumí todos los pares AB que habían (el numero min) y el numero max es a (0^a), por lo que debo agregar al final tantos 1's como A's haya, moviendo la marca final

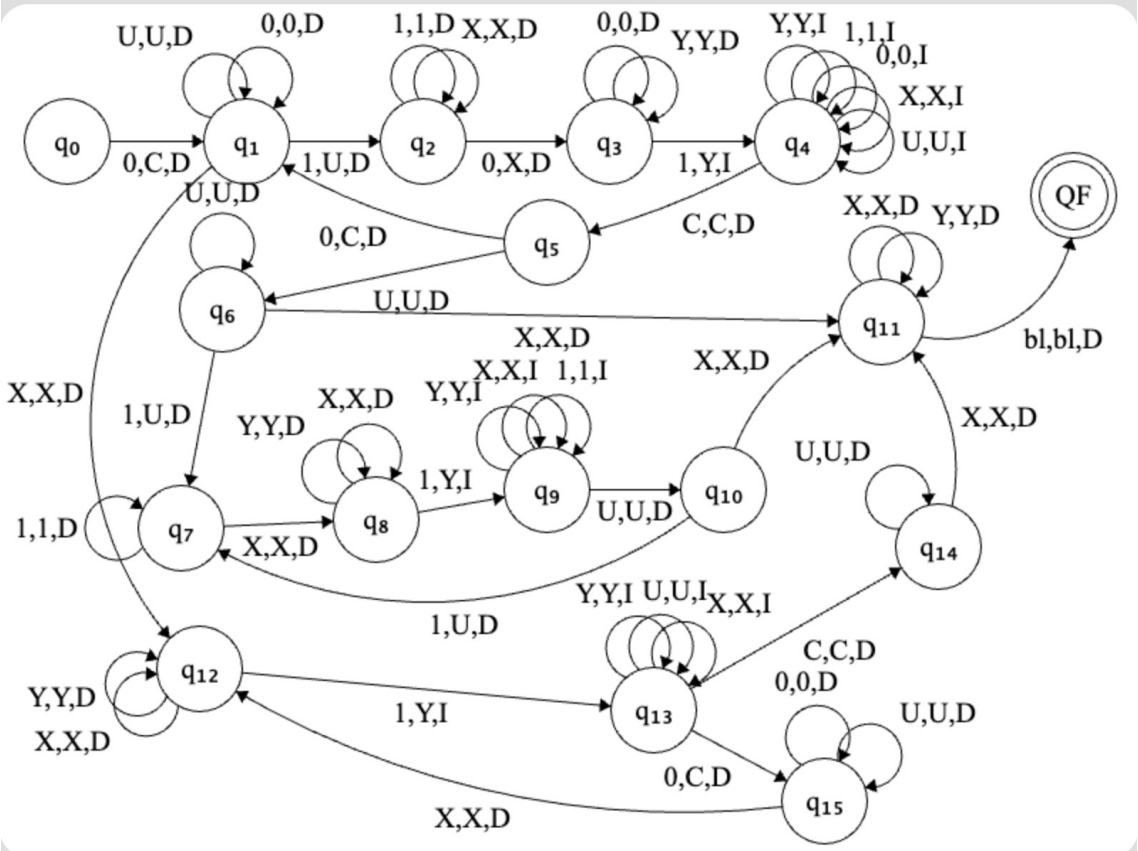
IB -> 1I // cuando queda una B al lado de la I, significa que consumí todos los pares AB y el numero max es b (1^b), entonces agrego tantos 1's como B's haya

IF → ε // cuando se juntan las marcas inicial y final significa que la marca I paso por todos 0's y 1's, y las variables que pudieran quedar entre I y F fueron convertidas en las reglas anteriores. Se eliminan las marcas

Otra idea:
Generar primero al mismo tiempo todos los 0´s y 1´s mínimos que son comunes a las dos subtiras de 0's y a las dos subtiras de 1's, y luego por cada 0 inicial mayor que los 1's iniciales (a>b) generar un 1 adicional en la subtira final de 1's, y lo mismo para cada 1 inicial mayor que los ceros iniciales (b>a) se genera también un 1 adicional al final.

S → I0CX1UF
X → 0CX1U|0UY|1UZ | ε
Y → 0UY | ε
Z → 1UZ | ε
C0 → 0C
C1 → 1C
CF → F0
U0 → 0U
U1 → 1U
UF → F1
I0 → 0I
I1 → 1I
IF → ε

c) Como es un lenguaje que no es libre de contexto por parte a) y es recursivamente enumerable, parte b), se construye una máquina de Turing que lo reconozca.



El autómata dibujado es determinista.
Cualquier transición δ(q,a) siendo q cualquiera y a un símbolo de la cinta, siempre hay una sola opción de qué hacer.