MINERVA SCHOOLS AT K.G.I.

Assignment 1

Quang Tran

CS142 Fall 2019

Part 1. Warm-up

Question 1.

Give an example of a homomorphism, using the same alphabet, , for both languages A and B .

Let $\sum = \{0, 1\}$. Let us define a homomorphism f as follows:

$f : \sum \rightarrow \sum *$

$f(0) = 00$
$f(1) = 11$

For any $w = w_1 w_2 ... w_n$ where $w_i \in \sum$

$f(w) = f(w_1)f(w_2)...f(w_n)$.
Then, let A be the language defined as: $A = \{bin(x) \mid x \in \{5, 6, 7, 8\}\}$
Then,
$A = \{101, 110, 111, 1000\}$
and $B = f(B) = \{f(w) \mid w \in A\}$
$= \{f(101), f(110), f(111), f(1000)\}$
$= \{f(1)f(0)f(1),\ f(1)f(1)f(0),\ f(1)f(1)f(1),\ f(1)f(0)f(0)f(0)\}$
$= \{110011, 111100, 111111, 11000000\}$

Question 2.

Now, give a second example of a homomorphism but this time using two diferent alphabets for languages A and B , respectively.

Let $\sum = \{0, 1\}$ and $\Gamma = \{a, b\}$. Let $A = \{bin(x) \mid x \in \{1, 2, 3, 4\}\}$

Then, $A = \{1, 10, 11, 100\}$
Let us define a homomorphism f as follows:

$f : \sum \to \sum *$

$f(0) = aba$
$f(1) = a$

For any $w = w_1 w_2 ... w_n$ where $w_i \in \sum$

$f(w) = f(w_1)f(w_2)..f(w_n).$
$B = f(B) = \{f(w) \mid w \in A\}$
$= \{f(1), f(10), f(11), f(100)\}$
$= \{a, aaba, aa, aabaaba\}$

Question 3.

Prove that the class of regular languages is closed under a homomorphism.

Let A be any regular language over alphabet $\sum$ and $f : \sum \to \Gamma^*$ be a homomorphism.

Let A' be the regular expression for regular language A (this is always possible, because if A is regular, then A can be recognized by a DFA, and there is an equivalent between regular expressions and DFAs.) So we have $A = L(A')$.

We will prove that $B = f(A)$ with alphabet Γ is also regular

We assume that $f(\varepsilon) = \varepsilon$. This will be used as a base case for the proof by induction below.

Observe that f can operate on a regular expression as well, by applying f on every symbol in the expression. For example, $f((01 \cup 10)^*) = (f(0)f(1) \cup f(1)f(0))^*$.
In this sense, because A' is a regular expression, $f(A')$ is fully defined.

1. Prove that $L(f(A')) = f(L(A'))$.

We will prove this by induction.

- Base case:
  - If $A' = \varepsilon$: $f(A') = \varepsilon$. Then $L(f(A')) = \{\varepsilon\}$.

On the other hand, $f(L(A')) = f(L(\varepsilon)) = f(\{\varepsilon\}) = \{\varepsilon\}$
Therefore, $L(f(A')) = f(L(A'))$ (both are equal to $\{\varepsilon\}$).

- If A' is a single symbol. That is, $A' = x \in \sum$. Then

$L(f(A')) = L(f(x)) = \{f(x)\}$
$f(L(A')) = f(\{x\}) = \{f(x)\}$
Therefore, $L(f(A')) = f(L(A'))$ (both are equal to {f(x)})

- Inductive hypothesis: Suppose $L(f(M')) = f(L(M'))$ and $L(f(N')) = f(L(N'))$.
- Induction step: We will prove all three: 1) $f(L(M' \cup N')) = L(f(M' \cup N'))$, 2) $f(L(M'N')) = L(f(M'N'))$, and 3) $f(L(M'^*)) = L(f(M'^*))$. We will prove 1), the other two cases are similar

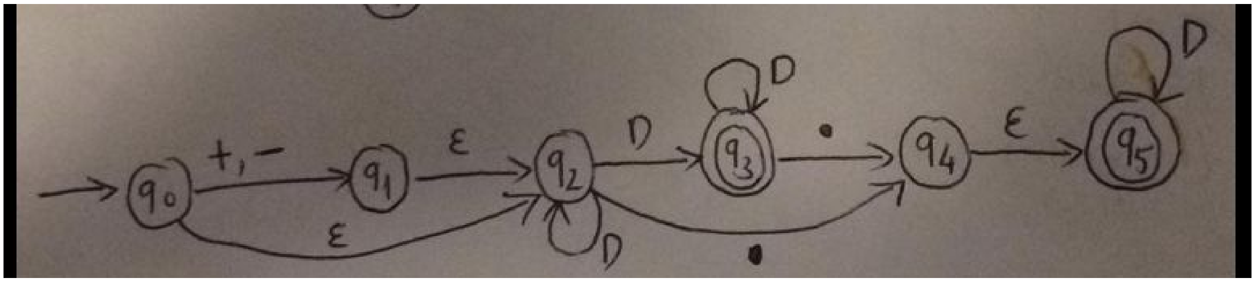
We have $f(M' \cup N') = f(M') \cup f(N')$ (this is just the definition of f operating on a regular expression.)
$f(L(M' \cup N')) = f(L(M') \cup L(N'))$
We admit without proving that $f(X \cup Y) = f(X) \cup f(Y)$, with X and Y being any two languages. Then the above equation becomes

$$
\begin{aligned}
f(L(M' \cup N')) &= f(L(M') \cup L(N')) \\
&= f(L(M')) \cup f(L(N')) \\
&= L(f(M')) \cup L(f(N')) \quad \text{(according to inductive hypotheses)} \\
&= L(f(M') \cup f(N')) \\
&= L(f(M' \cup N'))
\end{aligned}
$$

Therefore, we have proven that, regardless of whether A' contains how many start, union, or concatenation operations, $L(f(A')) = f(L(A'))$.

2. Prove that $B = f(A)$ is regular.

We have $A = L(A')$,
then $B = f(A) = f(L(A')) = L(f(A'))$.
We see that $f(A')$ is a regular expression, and B is $L(f(A'))$ which means B is a language that corresponds to a regular expression. This means B is a regular language.
Part 2. Automata and CFG
Question 1.
(a) Draw the state diagram of Mb1 .



(b) Define Mb1 using the formal mathematical notation.

This is an NFA M_{b1} defined by $(Q, \sum, \delta, q_0, F)$ where:

1. $Q = \{q_0, q_1, q_2, q_3, q_4, q_5\}$
2. $\sum = D \cup \{\bullet, +, -\}$
3. q_0 is the start state
4. $F = \{q_3, q_5\}$ the set of accept states
5. δ is defined below:

	$x \in D$	•	+	-	ε
q0	$\emptyset$	$\emptyset$	{q1}	{q1}	{q2}
q1	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	{q2}
q2	{q2,q3}	{q4}	$\emptyset$	$\emptyset$	$\emptyset$
q3	{q3}	{q4}	$\emptyset$	$\emptyset$	$\emptyset$
q4	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	{q5}
q5	{q5}	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$

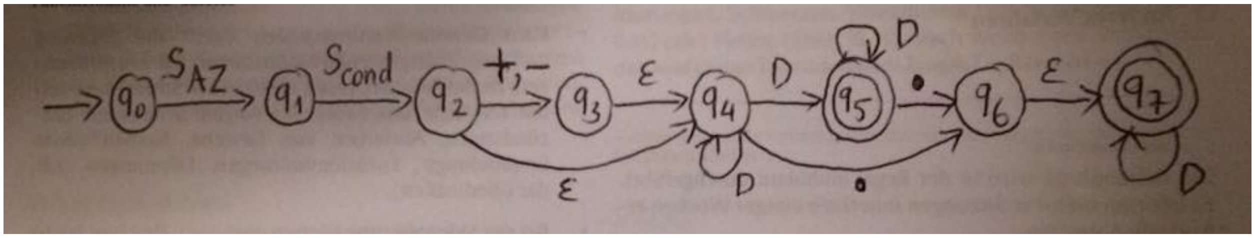
Question 2.

(a) Draw the state diagram

First let us defined:

S_{AZ} : the set of single letters (i.e., A,a,B,b,...,Z,z)

$S_{cond} = \{<, =, >, <=, >=\}$



(b) Write the equivalent regular expression for Mb2

$$S_{AZ}S_{cond}(+\cup-\cup\varepsilon)(D^{+}\cup D^{+}.D^{*}\cup D^{*}.D^{+})$$

(c) Define Mb2 formally

M_{b2} is defined by $(Q, \sum, \delta, q_0, F)$ where:

1. $Q = \{q_i \mid i = 0, 1, 2, .., 7\}$
2. $\sum = S_{AZ} \cup S_{cond} \cup D \cup \{\bullet, +, -\}$
3. q_0 is the initial state
4. $F = \{q_5, q_7\}$ the set of accept states
5. δ is given by:

	$x \in S_{AZ}$	$x \in S_{con}$	$x \in D$	•	+	-	ε
q0	{q1}	⊘	⊘	⊘	⊘	⊘	⊘
q1	⊘	{q2}	⊘	⊘	⊘	⊘	⊘
q2	⊘	⊘	⊘	⊘	{q3}	{q3}	{q4}
q3	⊘	⊘	⊘	⊘	⊘	⊘	{q4}
q4	⊘	⊘	{q4,q5}	{q6}	⊘	⊘	⊘
q5	⊘	⊘	{q5}	{q6}	⊘	⊘	⊘
q6	⊘	⊘	⊘	⊘	⊘	⊘	{q7}
q7	⊘	⊘	{q7}	⊘	⊘	⊘	⊘

Question 3.

In this problem, I have assumed:

1. **If** is a concatenation of letters **I** and **f,** both of which is from S_{AZ}
2. Similarly, **then** is a concatenation of **t, h, e,** and **n.**
3. I believe what is meant by ε in the below expression

$$\textbf{If} \;\; \epsilon < condition\_expression > \epsilon \;\; \textbf{then} \;\; \epsilon < expression > \;\; \epsilon \;\; \textbf{endif} \;\; \epsilon,$$

is a special character of a white space (‘ ‘), not really an epsilon as in a regular expression and I will call it # to distinguish it from the empty string. The reason I make this distinction is I intend my automata to accept the string

“If A>0 then B=10.5 endif”

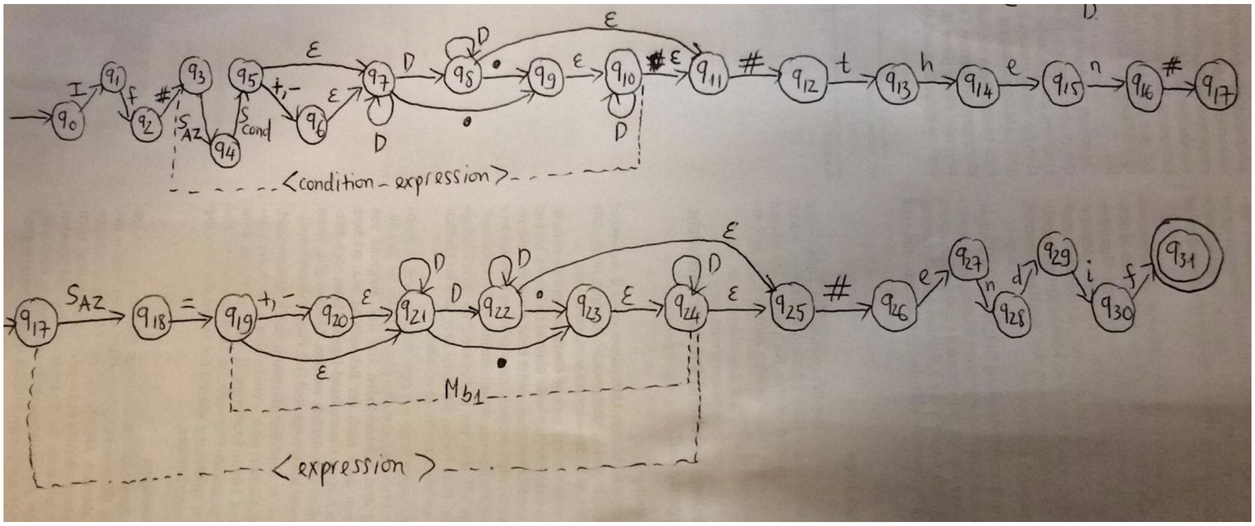
but not the string

“IfA>0thenB=10.5endif"

nor the string

“I f A>0 t h en B=10.5 e n dif”

This expression is a lot of concatenation of the previously built automata. In concatenating automata A and automata B, we change all the accept states in A to reject states and add connections from each of those accept states in A to the start state of B (with ε on the transition arrows.) For example, look at the block <condition expression> in the state diagram below, q8 and q10 are accept states in the original Mb2, but are changed to reject states, and the two arrows with the ε going from q8 and q10 to q11 are added.



(b) Regular expression

If#[$S_{AZ}S_{cond}(+\cup-\cup\varepsilon)(D^+\cup D^+.\,D^*\cup D^*.\,D^+)$]#then# S_{AZ} = $(+\cup-\cup\varepsilon)(D^+\cup D^+.\,D^*\cup D^*.\,D^+)$

c) **Define M_B formally.**

M_B is defined by $(Q,\sum,\delta,q_0,F)$ where:

1. $Q=\{q_i \mid i=0,1,2,..,31\}$
2. $\sum = S_{AZ}\cup S_{cond}\cup D\cup\{\bullet,+,-\}\cup\{\#\}$ (# is a whitespace)
3. q_0 is the initial state
4. $F=\{q_{31}\}$ the set of accept state
5. δ : the transition function. Because there are so many states, I would want to refrain from describing all transitions here and hope you can refer to the diagram states for this.

Define $L(M_B)$ using formal mathematical notation.
$L(M_B)=\{If\#A\_\#then\#B\_=C\_\#endif \mid A\_\in L(M_{b2}), B\_\in S_{AZ},\ C\_\in L(M_{b1}\ \}$
where
$L(M_{b2})=\{A\_B\_C\_ \mid A\_\in S_{AZ},\ B\_\in S_{cond},\ C\_\in L(M_{b1})\}$

Question 4.

See the notebook attached.

Question 5.

(a) + (b) Define the CFG and explain why the CFG is equivalent to M_B

CFG is a way of performing substitution. The original form of the strings accepted by M_B is:

$$\textbf{If} \ \epsilon < condition\_expression > \epsilon \ \textbf{then} \ \epsilon < expression > \ \epsilon \ \textbf{endif} \ \epsilon,$$

Then we can have a start variable A and the first rule:

A -> If#B#then#C#endif (B and C are variables)

B is our <condition_expression>, C is <expression>

Therefore,

B -> DEF (D: <variable>, E: <condition >, F:<token>)

C -> D=F (because expression is <variable>=<token>)

D -> A | a | B | b | C | c | … | Z | z

E -> > | < | = | >= | <=

F -> GH (G is either +,-, or ε and H is our $(D^+ \cup D^+ . D^* \cup D^* . D^+)$)

G -> + | - | ε

H -> I | J | K (I is D^+ , J is $D^+ . D^*$, K is $D^* . D^+$. However, because I is the same as capital i in "If", in the rewriting below, we will change all variables I to P)

I -> L | LM (M is D^*)

J -> I $\cdot M$

K -> M $\cdot$ I

L -> 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7|8|9

M -> ML | ε

In conclusion, the context free grammar is as follows:

$A \rightarrow If\#B\#then\#C\#endif$

$B \rightarrow DEF$

$C \rightarrow D = F$

$D \rightarrow A \mid a \mid B \mid b \mid C \mid c \mid \ldots \mid Z \mid z$

$E \rightarrow \ > \mid < \mid = \mid >= \mid <=$

$F \rightarrow GH$

$G \rightarrow \ + \mid - \mid \varepsilon$

$H \rightarrow P \mid J \mid K$

$P \rightarrow L \mid LM$

$J \rightarrow \ P \cdot M$

$K \rightarrow M \cdot P$

$L \rightarrow \ 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9$

$M \rightarrow ML \mid \varepsilon$

(c) Explain why in general a finite automaton has an equivalent CFG but not the other way around. You may use an example to explain this.

We can always construct a CFG from a finite automaton because:

1. A finite automaton is equivalent to a regular expression
2. A regular expression is operations on characters. These operations include star, concatenation, and union.
3. A CFG can describe star (by having a recursive symbol, see, for example, how D^* was constructed in question (a) and (b) above.), can describe concatenation (just put characters/variables next to each other on the right hand side of a rule), and union (just use “ | “ on the right hand side of a rule).
4. The fact from (3) is a stepping stone (base cases) for inductively building the entire CFG, just like we did in part (a) and (b) above.

The other way round is generally not true: CFG needs pushdown automata. Moreover, CFG is able to represent non-regular languages (besides regular ones), which a finite automaton certainly cannot recognize.