23/01/2020

CS6013: JAN – MAY 2020

WRITTEN ASSIGNMENT-1

ARJUN BHARAT

CS17B006

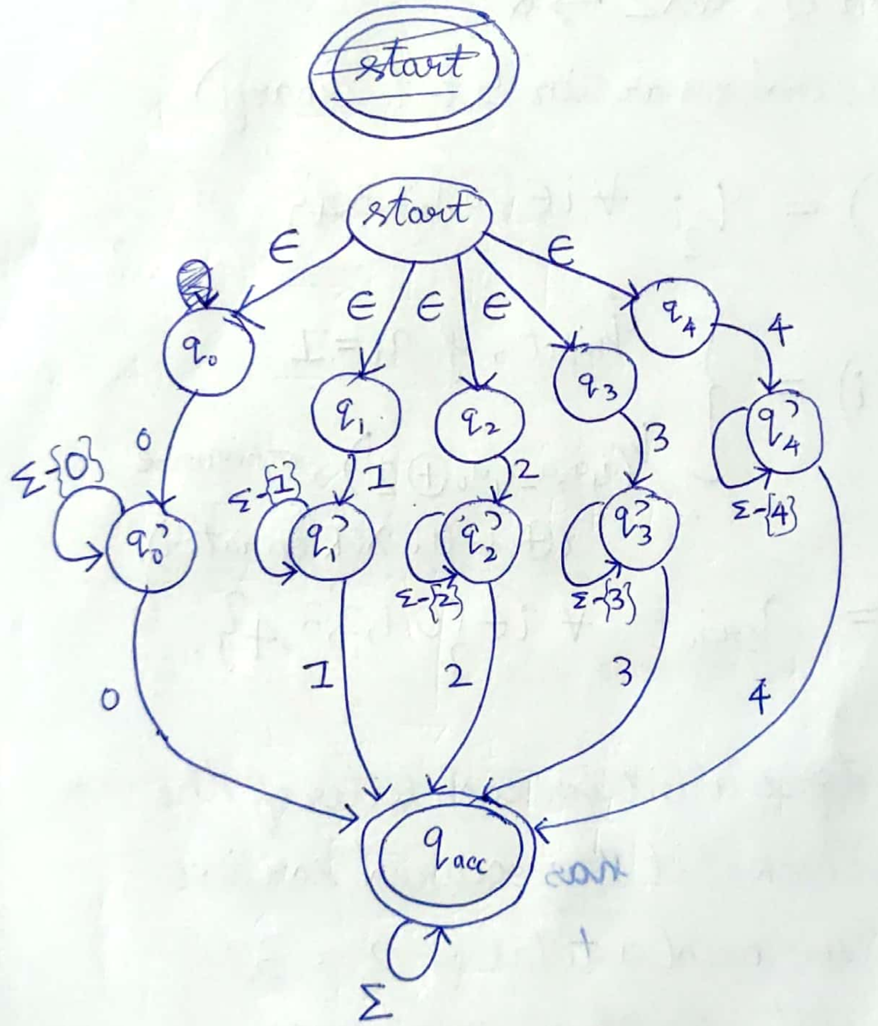
64 + 10 Bonus

Question 1:

a) $L = \{w \in \{0,1,2,3,4\}^*$ such that w has no repeated digits$\}$

5

→ We draw an NFA for ~~this~~ ~~grammar~~ the complement of the language, as follows:



→ To obtain a DFA for the original language, we would have to use subset construction and then swap the accept and reject states.

→ The resulting DFA is too big to draw, so we describe only the states and the transition function.

Set of states $Q = \{q_{start}, q_{reject}\} \cup \{q_{abcde}\ \forall\ a,b,c,d,e \in \{0,1\}\}$

Start state $= q_{start}$

Alphabet $\Sigma = \{0,1,2,3,4\}$

Accept states $= \{q_{start}\} \cup \{q_{abcde}\ \forall\ a,b,c,d,e \in \{0,1\}\}$

Transition function $\delta : Q \times \Sigma \to Q$:

(assume all state indexes are in 5 bit binary)

$$\delta(q_{start}, i) = q_{2^i}\ \forall\ i \in \{0,1,2,3,4\}$$

$$\delta(q_{\underbrace{a_4a_3a_2a_1a_0}_{\text{(5 bit binary)}}}, i) = \begin{cases} q_{reject}, & \text{if } a_i = 1 \\ q_{(a_4a_3a_2a_1a_0 \oplus 2^i)}, & \text{otherwise} \end{cases}$$

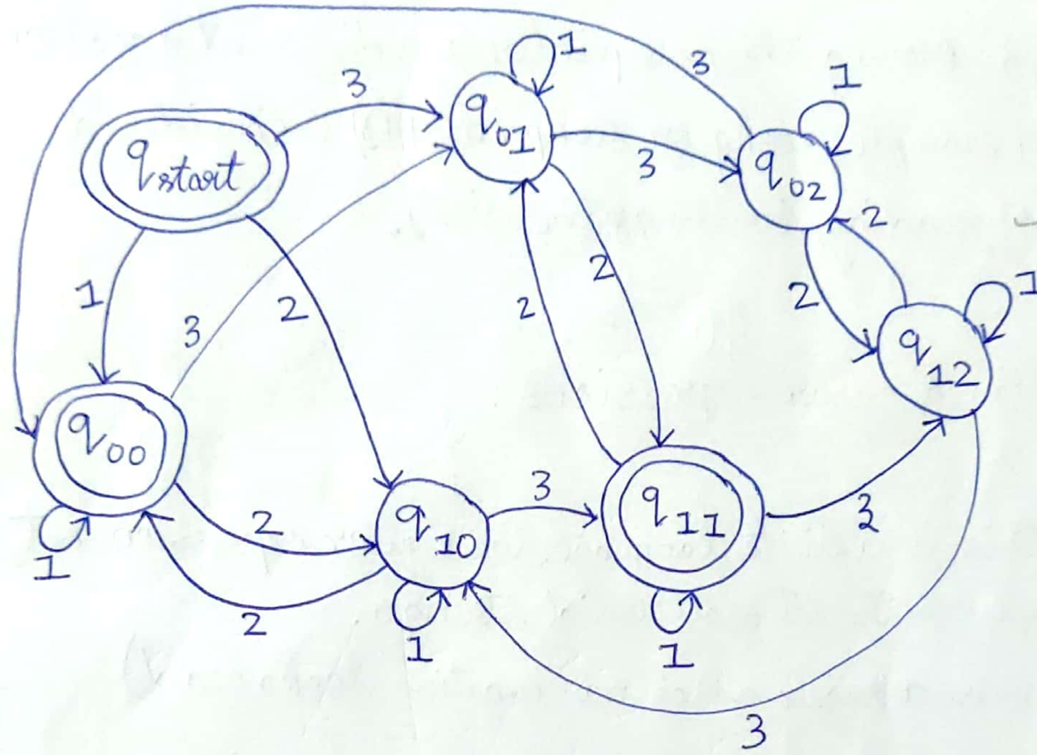
($\oplus$ is the XOR operator).

$$\delta(q_{reject}, i) = q_{reject}\ \forall\ i \in \{0,1,2,3,4\}.$$

* Idea: Each state stores a bit for each letter of the alphabet to check if it has occured earlier or not. Thus, we need a total of $2^5 = 32$ intermediate states to check what digits have already occured, and one start and one reject state.

b) $L = \{ w \in \{1,2,3\}^* \mid$ number of 2's modulo 2 = number of 3's modulo 3 $\}$

The DFA is as follows:

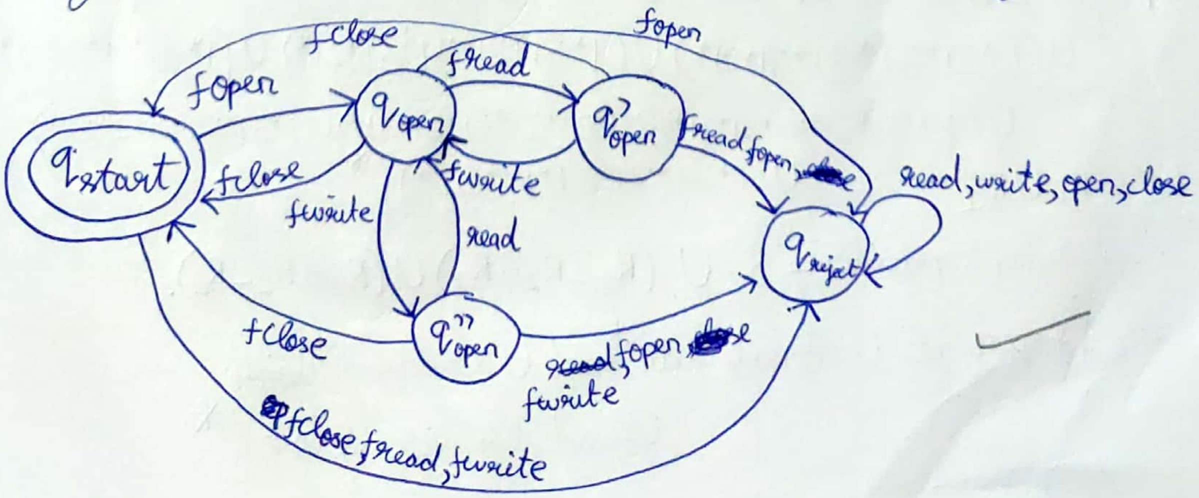


5

c) $L = \{ w \in \{fopen, fclose, fread, fwrite\}^* \mid$ w denotes a sequence of valid file operations $\}$

5

* Assumption: One can read and also write into an open file until closed. This is allowed in C++, for example.



* Assumption 2: Write operations are consolidated, that is, writing 5 and then 4 is treated as a single write operation. Same for reading. (This is just a convention, not necessarily how it must be interpreted.)

* Assumption 3: Moving the file pointer to a different location (for example, seekg or seekp in C++) is considered part of reading/writing to the file.

10

BONUS: Regular expressions: ?

Only, + (or ∪)
· (concatenation)
Kleen / * are operations

a) Since this is a finite language, the regular expression set can simply be the set of all valid strings.
(We can use a regex which has negative lookahead)

$$\cancel{R = \wedge \{0} \quad R = \underbrace{\wedge}_{\text{(not operator)}} \{(\Sigma^*0\Sigma^*0\Sigma^*) \cup (\Sigma^*1\Sigma^*1\Sigma^*) \cup (\Sigma^*2\Sigma^*2\Sigma^*) \cup (\Sigma^*3\Sigma^*3\Sigma^*) \cup (\Sigma^*4\Sigma^*4\Sigma^*)\}$$

b) Let $R_0 = 1^* \cup ((1^*21^*21^*) \cup (1^*31^*31^*31^*))^*$

$$R_1 = \{(1^*21^*21^*31^*31^*31^*) \cup (1^*21^*31^*21^*31^*3) \cup (1^*21^*31^*31^*21^*31^*) \cup (1^*21^*31^*31^*31^*21^*) \cup (1^*31^*21^*31^*31^*21^*) \cup (1^*31^*31^*21^*31^*21^*) \cup (1^*31^*31^*31^*21^*21^*) \cup (1^*31^*21^*31^*21^*31^*) \cup (1^*31^*31^*21^*21^*31^*) \cup (1^*31^*21^*21^*31^*31^*)\}^*$$

$R_2 = R_0 \cup R_1$

∴ Regular Expression $= R_2 \cup (R_2 2 R_2 3 R_2) \cup (R_2 3 R_2 2 R_2)$.

c) $R = \{\text{fopen}\ \{\epsilon \cup (\text{fread}\,\text{fwrite})^* \cancel{\text{fread}} \cup (\text{fwrite}\,\text{fread})^* \cancel{\text{fwrite}} \cup \text{fread} \cup \text{fwrite}\}\ \text{fclose}\}^*$

Question 2:

Context Free Grammars for Given Languages:

a) $L = \{ \omega \in \{0,1\}^* \mid \omega \text{ has equal number of 0s and 1s} \}$

5

Terminals: $0, 1, \epsilon$

Non Terminals: S_0, S

Start Symbol: S_0

Rules:

1) $S_0 \rightarrow S$
2) $S \rightarrow 0S1$
3) $S \rightarrow 1S0$
4) $S \rightarrow SS$
5) $S \rightarrow \epsilon$

b) $L = \{ w \in \{0,1\}^* \mid w \text{ has unequal number of 0s and 1s} \}$

* The idea is to use the "balanced strings" (those having an equal number of 0^s and 1^s) as subroutines and then create an imbalance.

5

Terminals: $0, 1, \epsilon$

Non-Terminals: S_0, S, U_0, U_1, B

Start Symbol: S_0

Scanned with CamScanner

Rules:

1) $S_0 \rightarrow S$
2) $S \rightarrow U_0$
3) $S \rightarrow U_1$
4) $U_1 \rightarrow 1B$
5) $U_1 \rightarrow B1$
6) $U_0 \rightarrow 0B$
7) $U_0 \rightarrow B0$
8) $B \rightarrow 0B1$
9) $B \rightarrow 1B0$
10) $B \rightarrow BB$
11) $B \rightarrow \epsilon$

c) $L = \{ w \in \{push, pop, top\}^* \mid w$ denotes a sequence of valid stack operations$\}$.

4

Terminals: push, pop, top, ϵ

Non Terminals: S_0, S_1, P, B, T, S_2

Start Symbol: S_0

Rules:

1) $S_0 \rightarrow S_1$
2) $S_2 \rightarrow PBT$
3) $B \rightarrow$ push BT pop
4) $B \rightarrow BB$
5) $B \rightarrow \epsilon$
6) $T \rightarrow$ top T
7) $T \rightarrow \epsilon$
8) $P \rightarrow$ push P
9) $P \rightarrow \epsilon$
10) $S_1 \rightarrow S_2 S_1$
11) $S_1 \rightarrow \epsilon$

* We assume that the stack is initially empty.
* We assume that the stack has infinite capacity.

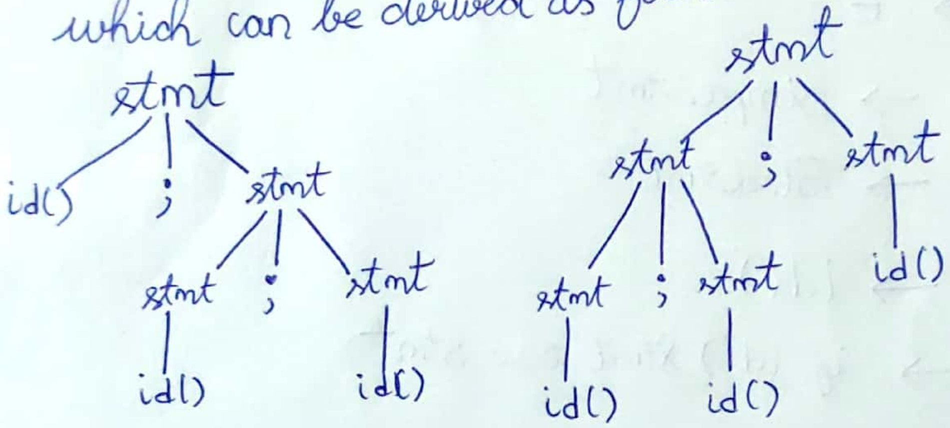
Question 3:

The given grammar is as follows:

stmt → id()
    | stmt ; stmt
    | { stmt }
    | if id() stmt else stmt

The given grammar is not LL(1) because:

a) It is ambigious. Consider a statement id(); id(); id(), which can be derived as follows:



b) It is left recursive (the 2nd rule uses left recursion on the non terminal stmt).

→ We apply Factoring and use right recursion to end up with an LL(1) grammar as follows:

List of Non Terminal Symbols:

stmt, BasicStmt, SimpleStmt, BlockStmt, RepStmt.

List of Terminal Symbols:

id() ; if else { }

Start symbol: Stmt

Grammar Rules:

35

1) Stmt → BasicStmt RepStmt
2) RepStmt → ; BasicStmt RepStmt
3) RepStmt → ∈
4) BasicStmt → SimpleStmt
5) BasicStmt → BlockStmt
6) SimpleStmt → id()
7) SimpleStmt → if id() Stmt else Stmt
8) BlockStmt → { stmt }

Grammar is not LL(1)

First and Follow Sets:

Follows are wrong

Non-Terminal	First	Follow
Stmt	id(), if, {	else, ;, }, $
BasicStmt	id(), if, {	;, $
SimpleStmt	id(), if	;, $
BlockStmt	{	;, $
RepStmt	;, ∈	else, }, $

We now proceed to draw the parsing table.

Parsing Table:

	id()	if	else	{	}	;	$
Stmt	1	1		1			
BasicStmt	4	4		5			
SimpleStmt	6	7					
BlockStmt				8			
RepStmt			3		3	2	3

Conflicts in Table

→ The numbers indicate which grammar rule to reduce by.

→ The empty entries are the error states of the parser.

Argument that the grammar is LL(1):

→ The parsing tables has no conflicts.

→ Considering rules 4 and 5, we see that
first(SimpleStmt) $\cap$ first(Block Stmt) = ϕ
A similar argument holds if we consider rules 6 and 7.

→ Considering rules 2 and 3 together, we see that
first(; BasicStmt RepStmt) $\cap$ follow(RepStmt) = ϕ

→ There is no left recursion being used anywhere.

→ The ambiguity caused by the rule
stmt → stmt; stmt has been removed, by using right recursion alone.

→ As argued earlier, factoring the grammar has eliminated reduce conflicts.

→ The language represented by the original grammar and the given grammar are identical.

Therefore, the new grammar is indeed LL(1).

× ———— × ———— × ————