CS6013 - Modern Compilers - Assignment - 1

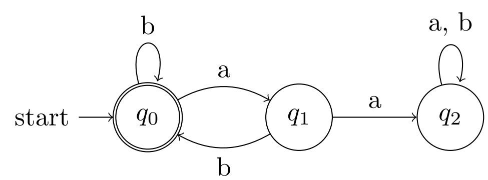
Bersilin C CS20B013

February 2, 2024

1. **Regular Expressions and DFA**
   Draw DFAs for the following languages
   (a) The language of all strings over the alphabet {a, b} where every 'a' is immediately followed by at least one 'b'.

Solution: The DFA for the language of all strings over the alphabet {a, b} where every 'a' is immediately followed by at least one 'b' is the 5-tuple (Q, Σ, δ, s, F).

$$Q = \{q_0, q_1, q_2\} \qquad \Sigma = \{a, b\} \qquad s = q_0 \qquad F = \{q_0\}$$

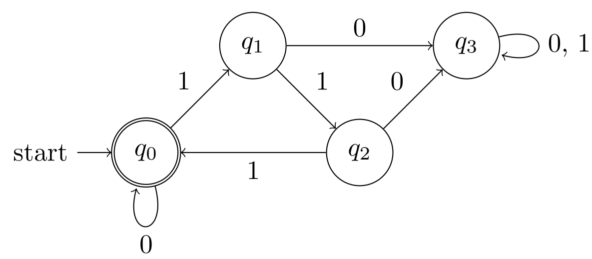


The regular expression for the above DFA is `(ab | b)`*

   (b) The language of all strings over the alphabet {0, 1} in which the number of consecutive 1s is divisible by 3.

Solution: The DFA for the language of all strings over the alphabet {0, 1} in which the number of consecutive 1s is divisible by 3 is the 5-tuple (Q, Σ, δ, s, F).

$$Q = \{q_0, q_1, q_2, q_3\} \qquad \Sigma = \{0, 1\} \qquad s = q_0 \qquad F = \{q_0\}$$

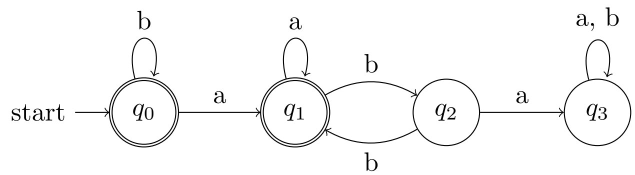


The regular expression for the above DFA is `(111 | 0)`*

(c) The language of all strings over the alphabet {a, b} where each 'a' is followed by an even number of 'b's.

Solution: The DFA for the language of all strings over the alphabet {a, b} where each 'a' is followed by an even number of 'b's is the 5-tuple (Q, Σ, δ, s, F).

$$Q = \{q_0, q_1, q_2, q_3\} \quad \Sigma = \{a, b\} \quad s = q_0 \quad F = \{q_0, q_1\}$$



The regular expression for the above DFA is `b`*`(a(bb)`*`)`*

2. **CFG**

Write the CFG for the following language:

(a) L=$\{w \in \{0,1\}^* | \ w$ contains double the number of 0s than 1s$\}$.

Solution: The CFG for the language of {0, 1} which contains double the number of 0s than 1s is the 4-tuple (V_n, V_t, P, S)

$$V_n = \{S, T, V\} \quad V_t = \{0, 1\} \quad S(goal) = S$$

The productions (P):

```
S -> 1T | T1 | 0V0 | ϵ
T -> 00S | 0S0 | S00
V -> 1S | S1
```

(b) L=$\{w \in \{0,1\}^* | \ w$ contains unequal number of 0s and 1s$\}$.

Solution: The CFG for the language of {0, 1} which contains unequal number of 0s and 1s is the 4-tuple (V_n, V_t, P, S)

$$V_n = \{S, T, V, E, U\} \quad V_t = \{0, 1\} \quad S(goal) = S$$

The productions (P):

```
S -> T | V
T -> 1E | E1 | TT
V -> 0E | E0 | VV
E -> 0U | U0 | ϵ
U -> E1 | 1E
```

(c) L=$\{w \in \{lock_x, unlock_x, access_x\}* \mid w$ denotes a sequence of valid accesses over a shared location and x is any integer.

Solution: The CFG for the language: $\{w \in \{lock_x, unlock_x, access_x\}* \mid w$ denotes a sequence of valid accesses over a shared location and x is any integer, is the 4-tuple (V_n, V_t, P, S)

$$V_n = \{S, T_x\} \quad V_t = \{lock_x, unlock_x, access_x\} \quad S(goal) = S$$

The productions (P):

S -> $lock_x$ T_x $unlock_x$ S | ϵ
T_x -> $access_x$ T_x | ϵ

3. **Parsing**

Consider the grammar

`stmt ...= id(); stmt stmt | { stmt } | if (id) stmt`

where `stmt` is the only non-terminal symbol, `stmt` is the start symbol, and

`{id, (, ), ;, {, }, if}`

is the list of terminal symbols. The terminal symbol id is defined using the regular expression (letter+) where letter is an ascii character in the interval a. . . z. The grammar generates a subset of the Java statements. Rewrite the grammar into a grammar which is LL(1), and use the rewritten grammar as the basis for implementing a recursive descent parser.

(a) Write the LL(1) grammar

Solution: The given grammar is ambiguous. We can eliminate ambiguity and left recursion by introducing two new non-terminals. The rewritten grammar is:

```
stmt -> stmt1 stmt2
stmt1 -> id(); | if(id) stmt1 | { stmt }
stmt2 -> stmt | ϵ
```

(b) Write the FIRST and FOLLOW sets for each non-terminal symbol

Solution: The FIRST and FOLLOW sets for the rewritten grammar are:

	FIRST	FOLLOW
`stmt`	id, if, {	\$, }
`stmt1`	id, if, {	id, if, {, \$, }
`stmt2`	id, if, {, ϵ	\$, }
id	id	-
(	(	-
)	)	-
;	;	-
{	}	-
}	{	-
if	if	-

(c) Write the predictive parsing table for the grammar

Solution: The predictive parsing table for the rewritten grammar:

NT	id	(	)	;	if	{	}	$
(s)	s -> s1 s2				s -> s1 s2	s -> s1 s2		
(s1)	s1 -> id();				s1 -> if(id) s1	s1 -> {s}		
(s2)	s2 -> s				s2 -> s	s2 -> s	s2 -> ϵ	s2 -> ϵ

(d) Argue that the rewritten grammar is LL(1)

Solution: A grammar G is LL(1) iff. for each set of productions `A ->` α_1 | α_2 | . . . | α_n:

1. FIRST(α_1), FIRST(α_2), ... FIRST(α_n) are all pairwise disjoint.

   Here,

   (a) `stmt` has only one production.

   (b) `stmt1` has three productions whose FIRSTs are `id, if,` { which are all pairwise disjoint.

   (c) `stmt2` has 2 productions where one of them is `stmt1` and the other, ϵ. FIRST(`stmt1`) doesn't have an ϵ, so it satisfies the condition.

2. If any of the $\alpha_i \to *\epsilon$, then FIRST$(\alpha_j) \cap$ FOLLOW(A) $= \phi, \forall 1 \leq j \leq n, i \neq j$.

   Here, `stmt2` goes to ϵ and the other production has `stmt1` whose FIRST = {id, if, { }, none of which match with the FOLLOW of `stmt2` which is { }, $ }.

Thus the rewritten grammar is LL(1).