CS 5000 Midterm

Sally Devitry (A01980316)

Problem 1

Convert the following grammar $G = (V, T, S, P)$ into CNF, where $V = \{S\}$, $T = \{a, b, c, d\}$, and P contains the following productions:
1. $S \to aSbb$;
2. $S \to aSa$;
3. $S \to bSaa$;
4. $S \to bSb$;
5. $S \to cd$

Solution

Looking at production 1, S appears in the right hand side, so we introduce new start state, S', and add the following production.
$S' \to S$

CNF does not allow unit productions, so we will replace the R.H.S. of $S' \to S$ with the R.H.S. of S.

Continuing the conversion of production 1 ($S \to aSbb$), we must eliminate the combination of terminals and non terminals. If we add the productions $A \to a$ and $B \to b$, we can rewrite S as $S \to ASBB$

CNF allows for only two nonterminals on the R.H.S., so we will add the productions, $C \to AS$ and $D \to BB$. This allows us to rewrite S as $S \to CD$, which follows CNF.

Following a similar pattern, we can convert the rest of the production rules to CNF. For production 2, we add the following production:
$S \to CA$
To convert production 3 to CNF, we add the following productions:
$S \to EF$
$E \to BS$
$F \to AA$
to convert production 4 to CNF, we add the following production:
$S \to EB$
To convert production 5 to CNF, we add the following productions:
$S \to GH$
$G \to c$
$H \to d$

The final CNF grammar for G is:

$S \to CD \mid CA \mid EF \mid EB \mid GH$
$S' \to CD \mid CA \mid EF \mid EB \mid GH$
$C \to AS$
$D \to BB$
$E \to BS$
$F \to AA$
$A \to a$
$B \to b$
$G \to c$
$H \to d$

Problem 2

Consider the following CNF grammar.
1. $S \to AB \mid BC$;
2. $A \to BA \mid a$;
3. $B \to CC \mid b$;
4. $C \to AB \mid a$.

Use the CYK algorithm to decide if $aaab \in L(G)$.

Solution

To begin, we construct a 4 by 4 table as the length of the string is 4. One side of the table represents the 'starts at' position, and one represents the length of the substring. Each cell in the following table was calculated by looking at the grammar to see what could be derived from a string of length L and starting at position S.

l \ S	a 1	a 2	a 3	b 4
1	{A,C}	{A,C}	{A,C}	{B}
2	{B}	{B}	{S}	
3	{A,S}	{ }		
4	{S}			

Because cell [1, 4] has the start state, S, we can say that the string, $aaab$, is accepted by the language, $aaab \in L(G)$

Problem 3

Let M1 and M2 be two DFAs. Outline an algorithm to decide if $L(M1) = L(M2)$.

Solution

A DFA M1 will be equivalent to M2 if and only if L(M1) is contained in L(M2), and L(M2) is contained in L(M1). In other words, to decide if $L(M1) = L(M2)$, we can check if $\overline{L(M1)} \cap L(M2)$ gives a language that accepts nothing. To do this, we follow these steps:

1. Take the complement of the DFA $M1$. (Swap the start and final states. Swap the direction of the arrows.)
2. Create a DFA that recognizes the language of the DFA from part 1, (the complement of L(M1)) and intersect it with L(M2). This new DFA accepts $\overline{L(M1)} \cap L(M2)$
3. Check the previously formed DFA, $\overline{L(M1)} \cap L(M2)$, for emptiness. If there are no strings that are accepted by the DFA, then the two languages given by M1 and M2 are equivalent. If the language given by the new DFA is not empty, then L(M1) and L(M2) are not equivalent.

Problem 4

Construct a stack machine for $L = a^n b^{3n}, n \in N$.

Solution

The grammar for the above language, L, is, $G = (\{S\}, \{a, b\}, S, P)$, where the set of productions P includes:
$S \to aSbbb$;

The stack machine for this languages is as follows:

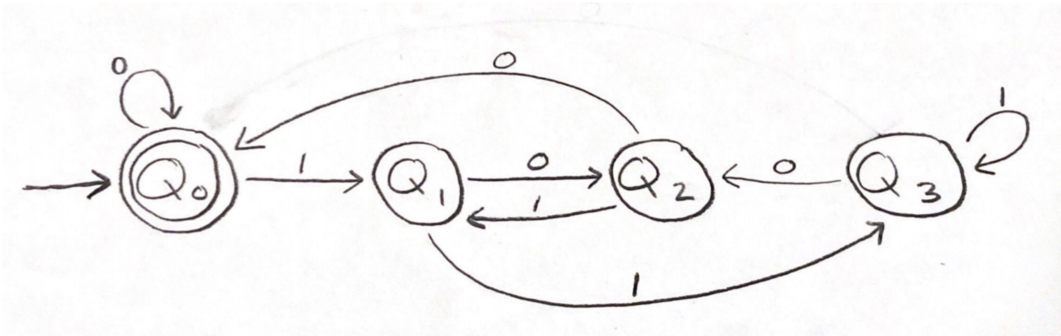
read	pop	push
ϵ	S	$aSbbb$
a	a	ϵ
b	b	ϵ

Problem 5

Construct a regular grammar for $L = \{x \in \{0, 1\}^* \mid$ x is the binary encoding of a number divisible by $4\}$.

Solution

The automata for this language is as follows:



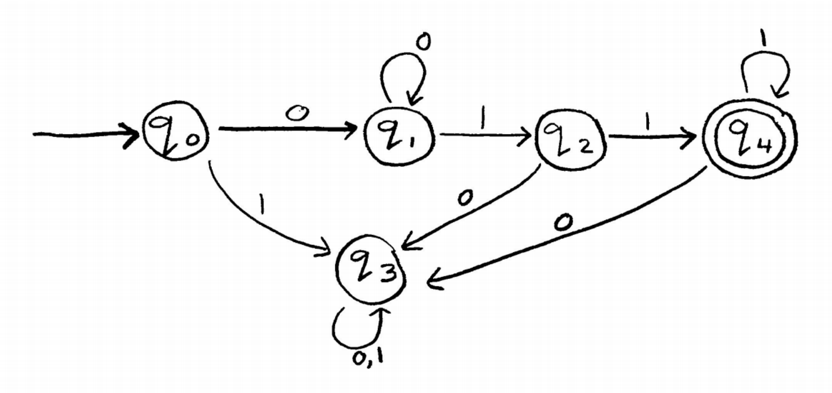
The regular grammar for L is $G = (\{Q_0, Q_1, Q_2, Q_3\}, \{0, 1\}, Q_0, P)$, where the set of productions P includes:
$Q_0 \to \epsilon$
$Q_0 \to 0Q_0$
$Q_0 \to 1Q_1$
$Q_1 \to 0Q_2$
$Q_1 \to 1Q_3$
$Q_2 \to 0Q_0$
$Q_2 \to 1Q_1$
$Q_3 \to 0Q_0$
$Q_3 \to 1Q_3$

Problem 6

Construct a minimal DFA for $L = \{0^n 1^m \mid n \geq 2, m \geq 2\}$.

Solution

A DFA for this language looks as follows, where q_3 is the sink.



To begin minimizing the DFA using Myhill-Nerode Theorem, we will create a table of all states. Half of the table is ignored (denoted with black squares) because the pairs are repeated.

A checkmark in the below table means that in that pair, one is a final state and the other is a nonfinal state.

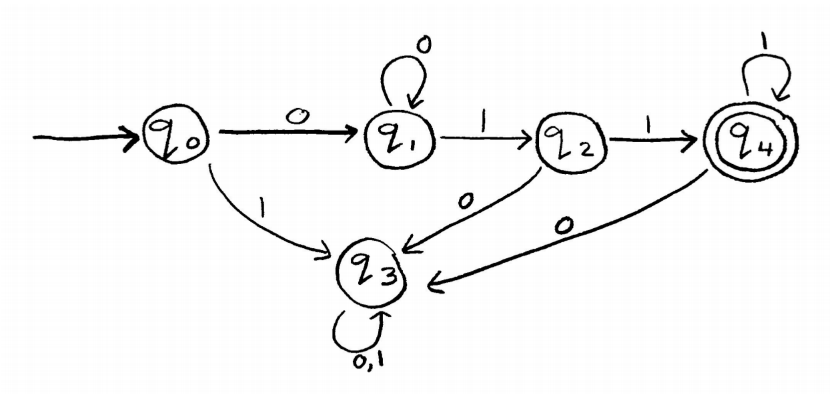
	q_0	q_1	q_2	q_3	q_4
q_0	■	■	■	■	■
q_1		■	■	■	■
q_2			■	■	■
q_3				■	■
q_4	✓	✓	✓	✓	■

Next, we must look at all unmarked pairs (P, Q) to see if $[\delta(P, x), \delta(Q, x)]$ is marked (x is any transition). If so, (P, Q) should also be marked.

All unmarked states in the above table become marked after finding that a final/nonfinal pair can be reached on a transition, x. The new table looks as follows:

	q_0	q_1	q_2	q_3	q_4
q_0	■	■	■	■	■
q_1	✓	■	■	■	■
q_2	✓	✓	■	■	■
q_3	✓	✓	✓	■	■
q_4	✓	✓	✓	✓	■

No pair is unmarked, meaning there are no states that can be combined. The original DFA is already minimized. The minimal DFA for the language, L, is:



Problem 7

Use the subset construction algorithm to convert the following NFA $M = (Q, \Sigma, q_0, \delta, F)$ into a DFA, where $Q = \{q_0, q_1, q_2, q_3\}, \Sigma = \{a, b\}$, $F = \{q4\}$, and δ is defined as 1. $\delta(q_0, a) = \{q_1\}$;
2. $\delta(q_1, a) = \{q_2\}$;
3. $\delta(q_2, a) = \{q_2\}$;
4. $\delta(q_2, b) = \{q_3\}$;
5. $\delta(q_3, b) = \{q_4\}$;
6. $\delta(q_4, b) = \{q_4\}$;

Solution

I began doing this problem by hand, only to discover that Q has 32 subsets. This would take a long time to do by hand. Luckily, I coded an algorithm to do subset construction on homework 3.
I created the given DFA and ran it in my Python file as:

```
    def test_midterm(self):
       print('\n***** MIDTERM NFA to DFA!************')
       NFA_DELTA_03 = {}
       NFA_DELTA_03[('q0', 'a')] = set(['q1'])
       NFA_DELTA_03[('q1', 'a')] = set(['q2'])
       NFA_DELTA_03[('q2', 'a')] = set(['q2'])
       NFA_DELTA_03[('q2', 'b')] = set(['q3'])
       NFA_DELTA_03[('q3', 'b')] = set(['q4'])
       NFA_DELTA_03[('q4', 'b')] = set(['q4'])
       NFA_03 = (set(['q0', 'q1', 'q2', 'q3', 'q4']),
                set(['a', 'b']),
                NFA_DELTA_03,
                'q0',
                set(['q4']))
       dfa_03 = nfa_to_dfa(NFA_03)
       display_dfa(dfa_03)
}
```

The output(below) shows that only a sink state needs to be added to make this a DFA.

```
***** MIDTERM NFA to DFA!************
STATES: {('q3',), ('q2',), ('q1',), ('q0',), (), ('q4',)}
SIGMA: {'a', 'b'}
START STATE: ('q0',)
DELTA:
0) d(('q0',), 'a')), = ('q1',)
1) d(('q0',), 'b')), = ()
2) d((), 'a')), = ()
3) d((), 'b')), = ()
4) d(('q1',), 'a')), = ('q2',)
5) d(('q1',), 'b')), = ()
6) d(('q2',), 'a')), = ('q2',)
7) d(('q2',), 'b')), = ('q3',)
8) d(('q3',), 'a')), = ()
9) d(('q3',), 'b')), = ('q4',)
10) d(('q4',), 'a')), = ()
11) d(('q4',), 'b')), = ('q4',)
FINAL STATES: [('q4',)]
```

The resulting DFA looks as follows, where q_5 is the sink and q_0 is the start state.