COMP3610 Problem Set 1

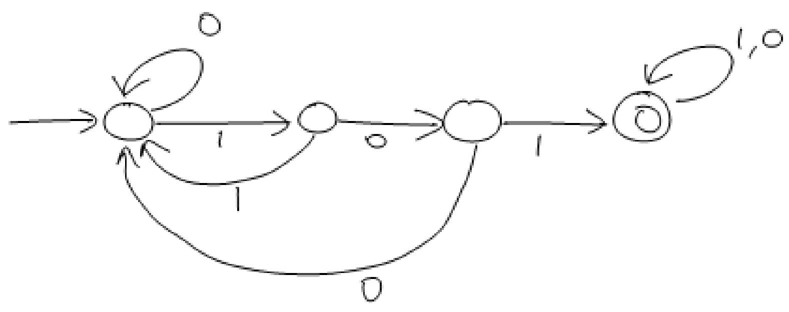
Enoch Lau

September 12, 2006

Question 1

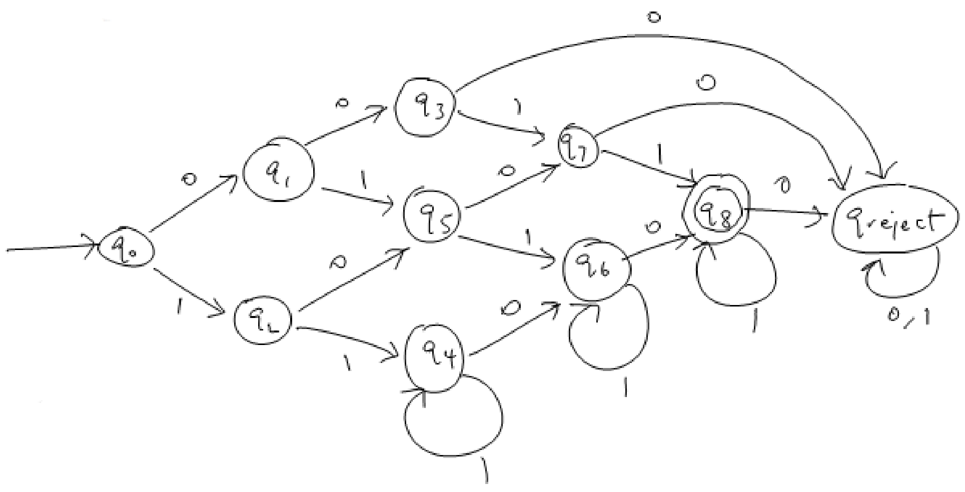
a)

L_1 = { all strings that contain 0101 }



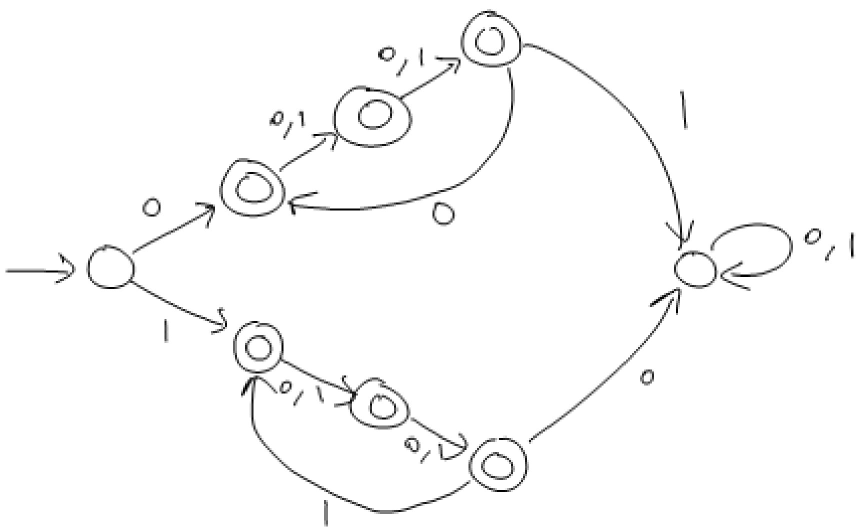
b)

L_2 = { all strings with exactly two 0s and at least two 1s }



c)

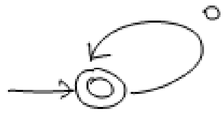
L_3 = { all strings such that every third symbol in the string is the same as the first symbol in the string }
(Note: this question was interpreted to mean that the 1st, 4th, 7th, 10th, . . . symbols in the string are identical.)



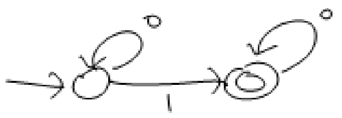
Question 2

a)

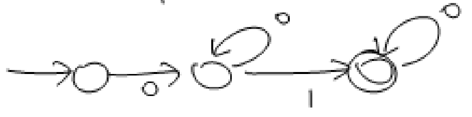
0^*



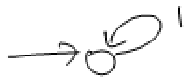
$0^* 1 0^*$



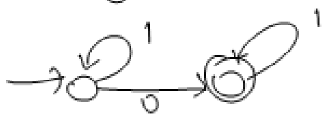
$0(0^* 1 0^*)$



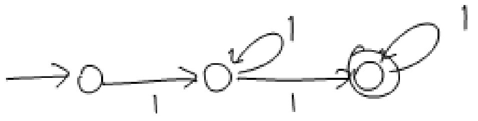
1^*



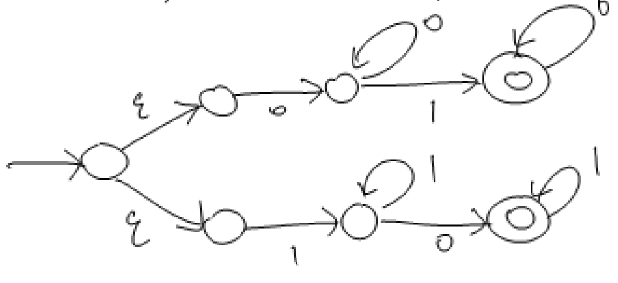
$1^* 0 1^*$



$1(1^* 0 1^*)$

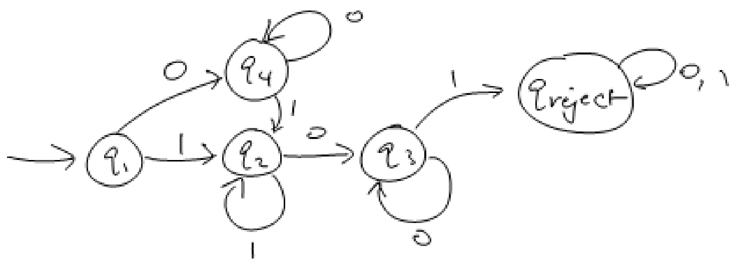


$(0(0^* 1 0^*)) \cup (1(1^* 0 1^*))$



b)

	state	0	1
q_1	A	$\{A,C\}$	$\{B\}$
q_2	B	$\{C\}$	$\{B\}$
q_3	C	$\{C\}$	ϕ
q_4	$\{A,C\}$	$\{A,C\}$	$\{B\}$



Question 3

a)

Proof by contradiction. Assume $L_4 = \{1^m 0^n 1^{m+n} | m, n \geq 1\}$ is regular. Let p be the pumping length given in the pumping lemma. Choose s to be $10^p 1^{p+1}$. Because $s \in L_4$ and $|s| > p$, the pumping lemma guarantees that s can be split into 3 pieces, namely $s = xyz$, where for any $i \geq 0$, $xy^i z \in L_4$.
Let us proceed by way of a few cases:

1. Suppose y consists of a mixture of 0s and 1s. When s is pumped to create $xyyz$, the 0s and 1s will be out of order. The resulting string cannot be in L_4, and thus y must contain either 0s or 1s.

2. Suppose y consists of 0s only. Since $|y| > 0$, when s is pumped to create $xyyz$, the resulting string will be of the form $10^q 1^{p+1}$ for some arbitrary $q > p$. This is not in L_4.

3. Suppose y consists of 1s only. There are two subcases:

   - y consists of the single 1 before the p 0s. When s is pumped to create $xyyz$, the resulting string will be of the form $110^p 1^{p+1} \notin L_4$.
   - y consists of 1s after the p 0s. This is not allowed, because this will result in $|xy| > p$.

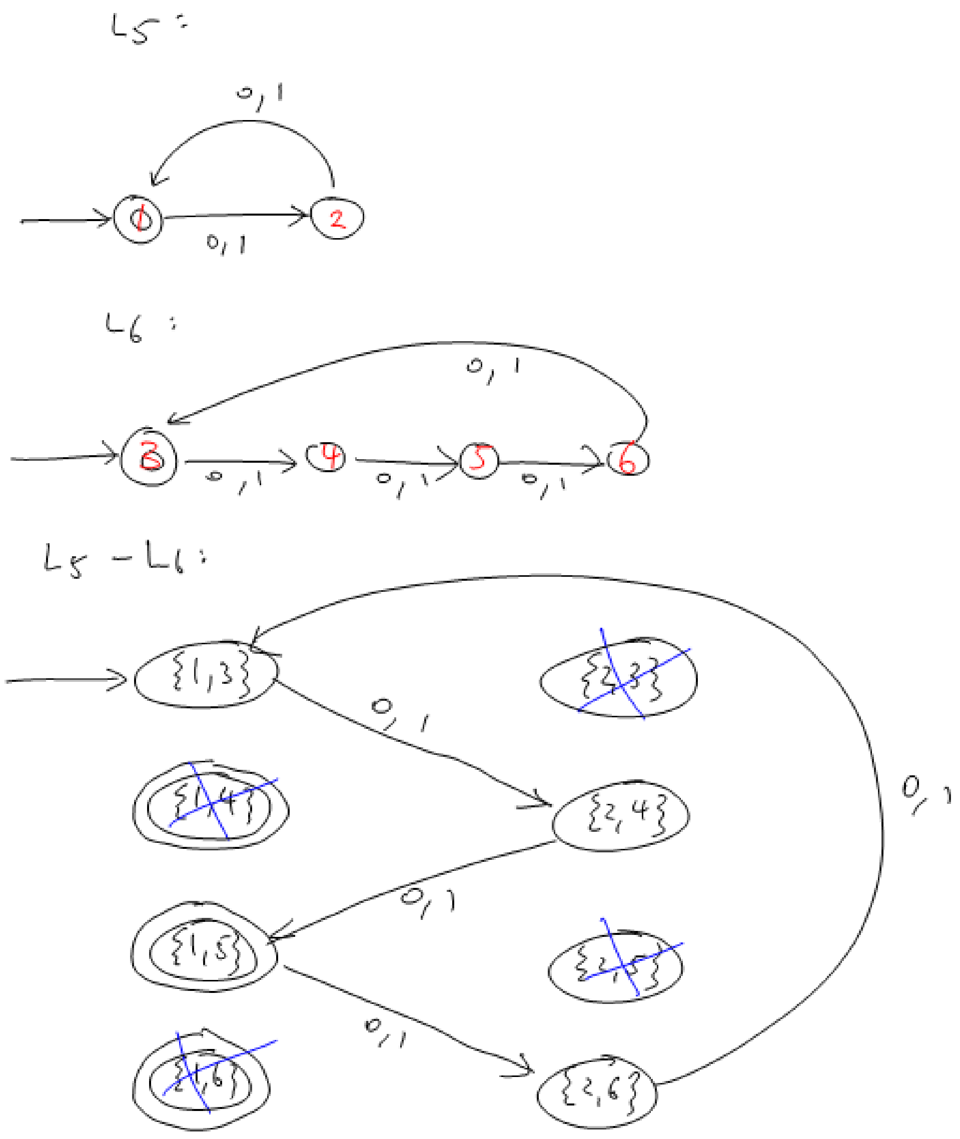
We conclude that a contradiction is unavoidable if L_4 is regular, so L_4 is not regular.

b)

Suppose that DFA $D_i = (Q_i, \Sigma_i, \delta_i, q_i, F_i)$ recognises R_i, $i = 1, 2$. We will prove that $R_1 - R_2$ is regular by constructing a DFA D that recognises this language.
We simulate D_1 and D_2 simultaneously by creating states in D that encapsulate states from both of the original DFAs. We are in an accepting state if the state from D_1 is an accepting state in D_1, and if the state from D_2 is not an accepting state in D_2.
Hence, construct $D = (Q, \Sigma, \delta, q_0, F)$ where:

- $Q = \big\{\{q_i, r_i\} | q_i \in Q_1, r_i \in Q_2\big\}$
- $\Sigma = \{0, 1\}$ (on the assumption that $\Sigma_i = \{0, 1\}$ as per assignment instructions)
- $\delta\big(\{q_i, r_i\}, a\big) = \big\{\delta_1(q_i, a), \delta_2(r_i, a)\big\}$
- $q_0 = \{q_1, q_2\}$
- $F = \big\{\{q_i, r_i\} | q_i \in F_1, r_i \in Q_2 \backslash F_2\big\}$

c)



Question 4

a)

$L = \{\langle M \rangle | M$ is a Turing machine such that, when started on a blank tape, it will eventually write a non-blank symbol onto its tape $\}$ is decidable. We prove this by giving a description of a machine R that will decide L.
On input $\langle M \rangle$ where m is a Turing machine, $R =$

1. Set $q = q_0$ (starting state of M).
2. Do the following n times (where n is the number of states of M):

   (a) Examine $\delta(q, \sqcup)$. If the transition function specifies that a non-blank symbol be written, *accept*.
   (b) Otherwise, set q to be the state specified by $\delta(q, \sqcup)$.
   (c) If $q = q_{\text{accept}}$ (the accepting state of M), then *reject*.

3. *Reject*.

This is equivalent to simulating M for up to n turns. Note that any non-blank symbol must be written in the first n turns. This is because M's tape is initially blank and remains blank (otherwise we terminate the simulation, as specified in the description of R). Thus, the position of the head and the symbol underneath the head conveys no useful information; we only ever consider transitions of the form $\delta(q, \sqcup)$, for some state q. Hence, after n steps, we will have exhausted all distinct configurations, and any further steps will repeat previous configurations (that is, it will have entered into a loop where it will never print a non-blank symbol). Hence, R decides L in this manner.

b)

We prove that PCP is decidable over the unary alphabet by constructing M that decides this problem.
On input $\langle P \rangle$, where P is a list of dominoes, $M =$

1. Go through the list of dominoes and see if any have the same number of 1s on top as on the bottom. If such a domino exists, *accept*.
2. Otherwise, go through the list of dominoes, and:
   - If the top has more 1s than the bottom, then remember that we have seen such a tile.
   - If the bottom has more 1s than the top, then remember that we have seen such a tile.
3. If we have seen both a tile with more 1s at the top, as well as a tile with more 1s at the bottom, then *accept*. Otherwise, *reject*.

If we *accept* in the first stage, then it is trivially true. If we *accept* in the second stage, then we will have one tile with i excess 1s on the top (say a), and another tile with j excess 1s on the bottom (say b). We can take j lots of a and i lots of b, and the excess 1s on the top and the bottom balance each other out.

c)

The problem is whether the following language is decidable: $L = \{\langle M \rangle | M$ has states that are never entered in any input $\}$.
We will show that it is undecidable by a reduction from E_{TM}. We construct a machine S that decides E_{TM} by using a machine U that decides the language L.
$S =$

1. Construct a Turing machine R from M. Take M and make sure every state apart from the accept state is not useless by adding a transition $\delta(q, a)$ for every $q \in Q$, $q \neq q_{\text{accept}}$, $a \notin \Gamma$. Add a to the tape alphabet of R. The resultant machine behaves in the same way as M given the same input.
2. Use U to decide whether R has any useless states.
3. If it accepts, *accept*.
4. Otherwise, *reject*.

U accepts iff q_{accept} is a useless state, and it is a useless state iff $L(M) = \emptyset$. However, E_{TM} is undecidable, so we have a contradiction. Hence, L is undecidable.

Question 5

a)

SPATH

We use a modified version of BFS to show that *SPATH* $\in P$.
$S =$

```
1. Set $i = 0$.
2. Mark $a$ as visited.
3. $\text{queue}_0 = \{a\}$.
4. While $i \leq k$:
   (a) For each vertex $v$ in $\text{queue}_i$:
       i. For each vertex adjacent to $v$, if it has not been marked as visited,
          mark it as visited, and place it into $\text{queue}_{i+1}$.
   (b) Increment $i$ by 1.
5. If $b$ has been marked as visited, accept.
6. Otherwise, reject.
```

This runs in polynomial time. A (very loose) bound on its complexity is as follows. 4(a)(i) runs in $O(n)$ time since there may be at most $n - 1$ adjacent vertices. Hence, 4(a) runs in $O(n^2)$ time. Since $k < n$, step 4 runs in $O(n^3)$. Step 2 is at worst $O(n)$, and hence the procedure runs in $O(n^3)$ time which is polynomial.

LPATH

- *LPATH* $\in NP$, as we can verify a certificate in polynomial time. The certificate is simply the path, and we need to check that we start a a and end up at b, that each vertex is distinct, that the edges taken are in the graph, and that we use at most k edges.

  On input $\langle G, a, b, k \rangle$, $V =$

```
1. Check that the first vertex is $a$, and start there.
2. $i = 0$.
3. For each successive edge in the path:
    (a) Check that the edge is adjacent to the previous vertex and that we
        have not visited the next vertex before.
    (b) Follow the edge to the next vertex.
    (c) Increment $i$.
4. Check that we are at $b$ and that $i \leq k$. If so, *accept*.
5. Otherwise, *reject*.
```

A (very loose) bound is as follows. Step 3(a) is at worst $O(n^2)$ and hence step 3 is at worst $O(n^3)$. Hence, the verifier runs in polynomial time.

- *LPATH* $\in$ *NP*-complete, as we will reduce *HAMPATH* to *LPATH*.
  - Let $f(\langle G, a, b\rangle) = \langle G, a, b, |V| - 1\rangle$. This is obviously computable in polynomial time.
  - If we have a solution to *HAMPATH*, we obviously have a path of length $|V| - 1$, which is the longest simple path you can have in a graph with $|V|$ vertices. Hence, it is a solution to *LPATH* for any value of k.
  - If we have a solution to *LPATH*, setting $k = |V| - 1$, we have a solution to *HAMPATH*, since a hamiltonian path visits each vertex once, using $|V| - 1$ edges.
  - Hence, *HAMPATH* is reducible to *LPATH*, and because *HAMPATH* is *NP*-complete, we conclude that *LPATH* must also be *NP*-complete.

b)

For a DNF to be satisfiable, at least one of the AND of ORs must be satisfied. For that to occur, all the terms in the AND of terms must evaluate to true. The only requirement for that to occur is that there are no contradicitions in the AND of terms; that is, x_i and $\bar{x}_i$ cannot appear in the same AND of terms.
We give a machine M that decides $L = \{\langle\phi\rangle | \phi$ is an OR of AND of terms $\}$.
$M =$

```
1. For each AND of terms in the sequence of ORs of ANDs:
    (a) For each term in the AND of terms:
        i. If we have seen the negative of the current term previously in this
           AND of terms, then break out of the inner loop.
        ii. Otherwise, remember that we have seen the current term.
    (b) *Accept* if inner loop completes.
2. *Reject* if outer loop completes.
```

1(a) runs in $O(n^2)$ time, and step 1 runs in $O(n^3)$ time. M runs in polynomial time, and hence deciding whether a formula given in DNF is satisfiable is in P.