Solutions for Assignment 1

October 22, 2003

1. Answers

a) False

We will show by contradiction that $f \notin O(g)$. Suppose $f \in O(g)$, then there exist constants $c > 0$ and $n_0 > 0$ such that $\frac{f(n)}{g(n)} \leq c$ for all $n \geq n_0$. But

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} n = \infty$$

which contradicts the previous statement.

b) False

Take $f(n) = 1$ and $g(n) = n$. Then $f_{min}(n) = 1$, and

$$(f + g)(n) = f(n) + g(n) = n + 1$$

$f+g \notin \Theta(f_{min})$ since $f+g \notin O(f_{min})$ (because $\lim_{n\to\infty} \frac{f(n)+g(n)}{f_{min}(n)} = \lim_{n\to\infty} \frac{n+1}{1} = \infty$).

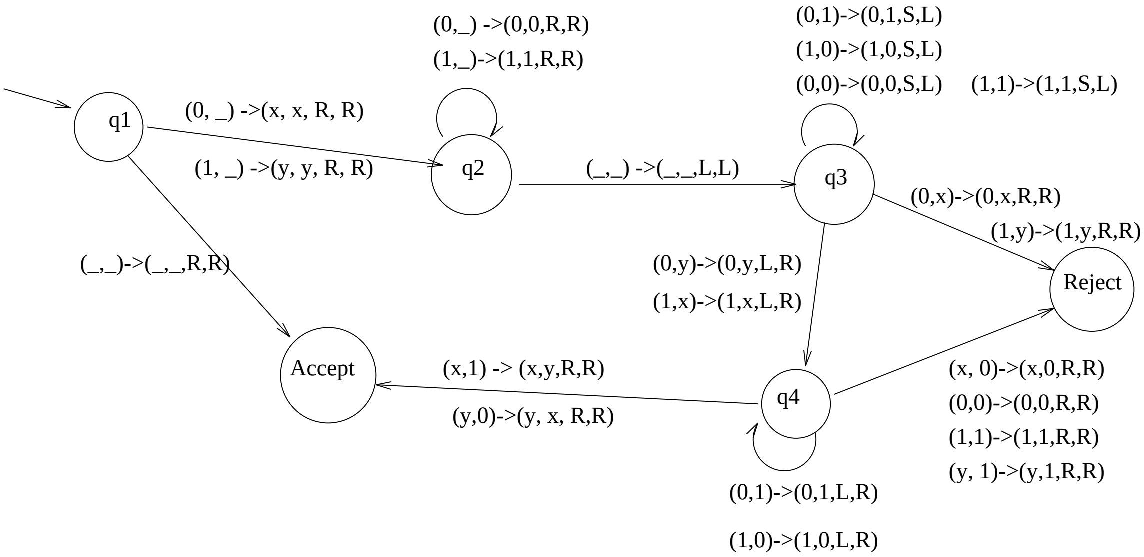
c) True

Suppose $f \in O(g)$. Then we know there exist postive constants c and n_0 such that $f(n) \leq cg(n)$ for all $n \geq n_0$. Thus $\frac{1}{c}f(n) \leq g(n)$ for all $n \geq n_0$; so there exists a constant $d = \frac{1}{c} > 0$ such that $df(n) \leq g(n)$ for all $n \geq n_0$, which implies $g \in \Omega(f)$.

d) False

Counterexample: $f(n) = 1$, $g(n) = n$. In this case $f \in O(g)$ since for $c = 1$ and $n_0 = 1$, $f(n) = 1 \leq n = 1 \cdot g(n)$ for all $n \geq 1$. However $g \notin O(f)$ since $\lim_{n\to\infty} \frac{g(n)}{f(n)} = \lim n = \infty$, which is not below any constant.

2.1

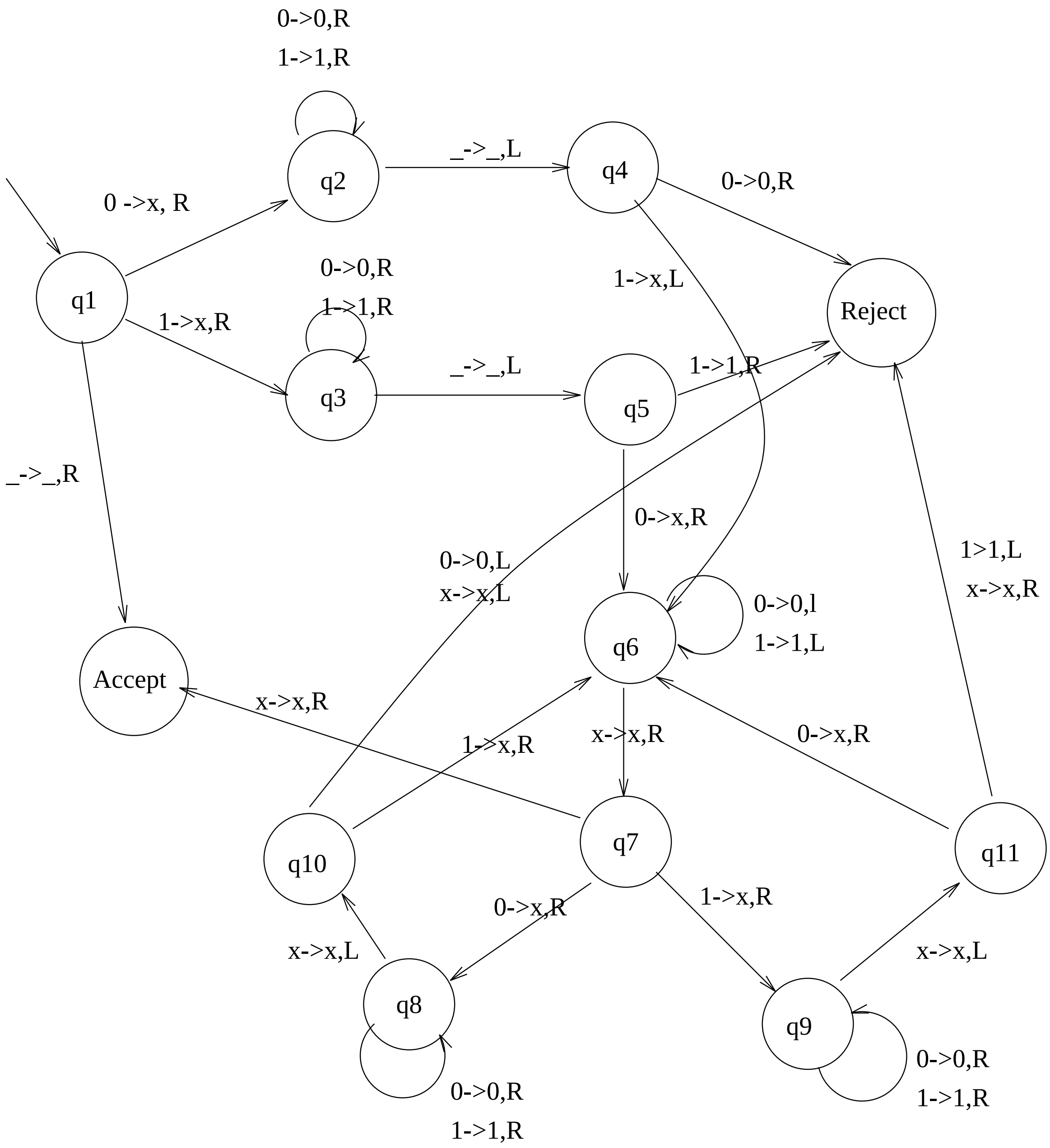


2.2

- Step1 (states q1 and q2): Go over tape 1 writing its conents to tape 2.
- Step2 (state q3): Rewind tape 2 while keeping tape 1 in its last input symbol.
- Step 3 (state q4): Go over both tapes, tape 1 from right to left and tape 2 from left to right, comparing symbols. If symbols are equal at any point, rejest. Stop when reaching leftmost point of tape 1.

Let $T(n)$ be the worst case running time of M_2. Steps 1, 2, and 3 each consists of a single scanning of the tape so $T(n) \in \Theta(n)$

2.3



2.4. Description and analysis

- Step 1: Scan the tape crossing the first and last symbols. If symbols were equal then reject.

- Step 2: Scan the tape again crossing the first symbol that is not an X and the last symbol that is not an X. If original symbols were equal then reject. If no symbol other than X has been found, accept.

- Step 3: GO to step 2.

The first step can be done with a single scan of the tape, so it takes $O(n)$ steps. Similarly step 2 takes $O(n)$ steps. Step 2 is repeated at most $\frac{n}{2}$ times, since at each step two symbols are transformed into X. Therefore the total running time is in $O(n^2)$

2.5 RAM Program

We will use registers:

r_0 = accumulator

r_1 = store index of leftmost symbol to be examined

r_2 = store index of rightmost symbol to be examined

1 LOAD = 1
2 STORE 1
3 READ ↑ 1
4 STORE 2
5 LOAD 1
6 ADD = 2
7 STORE 1
8 LOAD 2
9 STORE ↑ 1
10 ADD 1
11 JZERO 16
12 LOAD 1
13 SUB 1
14 STORE 1
15 JUMP 3 // loop 3-15 reads $i_1, \ldots, i_n$ into $r_3, \ldots, r_{n+2}$
16 LOAD 1
17 SUB =1
18 STORE 2 // $r_2 \leftarrow n+2$
19 LOAD =3
20 STORE 1 // $r_1 \leftarrow 3$
21 LOAD 2
22 SUB 1

23 JNEG 34 // if $r_2 < r_1$ then go to accept
24 LOAD $\uparrow$ 1
25 SUB $\uparrow$ 2
26 JZR 36 // if leftmost symbol = rightmost symbol then reject
27 LOAD 1
28 ADD =1
29 STORE 1 // $r_1 \leftarrow r_1 + 1$
30 LOAD 2
31 SUB =1
32 STORE 2 // $r_2 \leftarrow r_2 - 1$
33 JMP 21
34 LOAD =1 // accept
35 HALT
36 LOAD =0 // reject
37 HALT

2.6 Running Time

The first loop (line 3 to line 15) simply reads the input which takes time $O(n)$. The second loop (line 21 to line 33) runs for $\left\lceil \frac{n}{2} \right\rceil$ iterations, so it takes time in $O(n)$. The whole program takes time in $O(n)$.

3. Let $A \in P$, $A \neq \Sigma^*$ and $A \neq \phi$. Since $A \in P$, we know $A \in NP$. It remains to show A is NP-hard.

We need to show that $L \leq_p A$ for all $L \in NP$.

Let $L \in NP$ be an arbitrary language in NP. Since $P = NP$, we conclude $L \in P$ and so there exists a polynominal time algorithm D that decides L. We will build a reduction algorithm F (reduction from L to A) in the following way:

Algorithm $F(x)$
{ Let a be a string in A, Let b be a string in $\Sigma^* \backslash A$
if $D(x) = 1$ then return a
else return b
}

if $x \in L$ then $D(x) = 1$ and $F(x) = a \in A$.
if $x \notin L$, then $D(x) = 0$ and $F(x) = b \notin A$.

Moreover, since D runs in polynominal time and F simply calls D (polynomial time) and does a constant number of steps, F runs in polynominal time.

4. Since HAMPATH $\in P$, there exists a polynomial time algorithm A that decides HAMPATH.

Below we describe the polynomial time algorithm B that finds a hamiltonian path from u to v, if one exists.

```
Algorithm B (G = (V, E), u, v)
{
1       if A(G, u, v) = 0 then
2           output "no hamiltonian path from u to v exists "
3       else
4       { n = |V|
5           w = u
6           for i = 1 to n do
7           { PATH[i] = w;
8               G' ← G\{w};
9               for each edge (w, t) in G coming out of w do
10                {if (A(G', t, v) = 1) then
11                    { w = t;
12                      break this loop;
13                    }
14                }
15              G ← G';
16            }
17            return (PATH);
18      }
19 }
```

The correctness of the algorithm comes from the idea that if there exists a hamiltonian path from u to v, having second vertex t, when we remove u from G, there must be hamiltonian path from t to v in the reduced graph. Iterating this principle, we can determine the sequence of the vertices in the hamiltonian path.

The algorithm runs in polynomial time, for the following reasons:

Let $T(n, m)$ be the worst case running time of algorithm A for a graph with n vertices and m edges.

step1 takes time $T(n, m)$

step 2-5 takes constant time

loop in line 6 runs n times

loop in line 9 runs at most m times

therefore line 10 over all iteration run in $n \times m \times T(n,m)$

line 11- 12 run in time $n \times m$

line 8 and 15 are repeated n times and each time it may take $O(n \times m)$, so totally, it is $O(n^2 \times m)$

Thus the running time for B is in $O(n^2 \times m + n \times m \times T(n,m))$, since $T(n,m)$ is a polynomial, B runs in polynomial time.

5.Proof:

Step 1 DoubleSAT $\in NP$

Certificate : Two truth assignments

Verification Algorithm:

$A(<\phi>, y_1, y_2)$

- Evaluate formula ϕ using truth assignment given in y_1. If (y_1 is not a satifying assignment) then return 0;

- Evaluate formula ϕ using truth assignment given in y_2. If (y_2 is not a satifying assignment) then return 0;

- If($y_1 \neq y_2$) then return 1; else return 0;

ϕ is satisfiable $\Leftrightarrow$ there exist two distinct truth assignments that satisfy ϕ $\Leftrightarrow$ there exists inputs y_1, y_2 for A that causes A to return 1.

Algorithm A runs in polynomial time since the formula evaluation can be done in linear time with the size of ϕ. Also y_1 and y_2 have size n, the number of variables in ϕ, so these comparison also take linear time.

Step 2 We will prove SAT $\leq_p$ DoubleSAT

Step 3 We will describe the reduction algorithm F.

Algorithm F$(<\phi>)$

{

Let $x_1, x_2, ..., x_n$ be the variable in ϕ

Build another formula ϕ_2 equivalent to ϕ but substituting variables $x_1, x_2, ..., x_n$ by $y_1, y_2, ..., y_n$ respectively.

Create a formula ϕ' as the disjunction of ϕ and ϕ_2: $\phi' = \phi \vee \phi_2$;

return ϕ'

}

Step 4

- If $\phi \in$ SAT, then there exists a satisfying assignment $\sigma = (\sigma_1, \sigma_2, ..., \sigma_n)$, where $\sigma_i = 0$ or 1 for ϕ. Thus, σ satisfies ϕ and ϕ_2. Let $\tau = (\tau_1, \tau_2, ..., \tau_n)$, $\tau \neq \sigma$.Therefore $\sigma^1 = (\sigma_1, \sigma_2, ..., \sigma_n, \tau_1, \tau_2, ..., \tau_n)$ and $\sigma^2 = (\tau_1, \tau_2, ..., \tau_n, \sigma_1, \sigma_2, ..., \sigma_n)$ satisfy $\phi' = \phi \vee \phi_2$. So $\phi' \in$ DoubleSAT

- If $\phi \notin$SAT, then for all truth assignments $\sigma = (\sigma_1, \sigma_2, ..., \sigma_n)$, ϕ evaluates to 0. Therefore, for any truth assignment τ, ϕ_2 evaluates to 0. So $\phi' = \phi \vee \phi_2$ evaluates to 0 for any truth assignment $(\sigma_1, \sigma_2, ..., \sigma_n, \tau_1, \tau_2, ..., \tau_n)$. So $\phi' \notin$ DoubleSAT

Step 5

ϕ can be copied to ϕ_2 in linear time and both can be combined with an or operation in constant time. So F runs in linear time on the size of ϕ