Homework 1 for Intro to Computational Complextiy

Shenlong Gu

917-544-8927, sg3301@columbia.edu, 14, September 2015

Part I

We make some assumptions, turning machine M_1 decides language L and turning machine M_2 decides language $L^{'}$

a) **union**
We give a turning machine $M^{'}$, which works as follows.
For a given string l,
Simulate l on M_1,
if M_1 accepts l:
    $M^{'}$ accepts l.
else:
    simulate l on M_2.
    if M_2 accepts l:
        $M^{'}$ accepts l.
    else
        $M^{'}$ rejects l.
We can find $M^{'}$ decides the language union of L and $L^{'}$, so union operation is closed for two decidable languages.

b) **concatenation**
We give a turning machine $M^{'}$, which works as follows.
For a give string l with length n.
We enumerates i from 0 -> n to generate $n+1$ pairs $(i, n-i)$
For each pair$(i, n-i)$:
    we simulate $l[0:i]$ on machine M_1 and $l[i:n]$ on machine M_2
    if M_1 accepts $l[0:i]$ and M_2 accepts $l[i:n]$
        $M^{'}$ accepts l
    else
        continue
End
$M^{'}$ rejects l. (because after enumerating each pair, we can not find a solution)
We can find $M^{'}$ decides the language concatenation of L and $L^{'}$, so concatenation operation is closed for two decidable languages.

c) **complementation**
We give a turning machine $M^{'}$, which works as follows:

For a given string l,
simulate l on M_1.
if M_1 accepts l:
    $M^{'}$ rejects l.
else:
    M' accepts l.

d) **intersection**
We give a turning machine $M^{'}$, which works as follows:
for a given string l,
simulate l on M_1,
if M_1 rejects l:
    $M^{'}$ rejects l.
else:
    simulate l on M_2,
    if M_2 rejects l:
        $M^{'}$ rejects l.
    else:
        $M^{'}$ accepts l.
We can find $M^{'}$ decides the language intersection of L and $L^{'}$, and intersection operation is closed for two decidable languages.

Part II

We will prove for any lanauge which can be decided by an original turning machine can be decided by a new designed turning machine and for any language which can be decided by the new designed turning machine can be decided by an original turning machine.

a) **First**
For a language which can be decided by an orignal turning machine M_1, we can invents a new designed turning machine M_2 which works as follows:
We add one more character into the character sets to set it to the left of the start point of input(just like a left-end), then we use the same state change function as M_1 does, (just the left end is replaced by the new character), obviously this new designed turning machine M_2 decides the same language as M_1 does.

b) **Second**
For a language which can be decided by a new designed turning machine M_1, we can design an original turning machine M_2 as follows:
Three are conditions below.
1.M_1 scans in the range[left, right]
M_2 has the same state change as M_1 does including state change and character write
2.M_1 moves from rightmost to the right of rightmost(which means go outside rightmost)
M_2 moves right to the right of rightmost of the input
3.M_1 moves left/right and in the range[right + 1, ++]
everytime M_1 moves one place left/right, M_2 moves two places left/right, and do similar state change.
4.M_1 moves from the right + 1 position left to the rightmost of the input

M_2 moves left one place to the rightmost of the input
5.M_1 moves from letmost to the left of leftmost(which means go outside leftmost)
M_2 moves to the rightmost of the input and moves two places right.
6.M_1 moves left/right and in the range[–, left - 1]
everytime M_1 moves one place left/right, M_2 moves two places right/left, and do similar state change.
7.M_1 moves from the left - 1 position right to the leftmost of the input
M_2 moves to the leftmost of the input

All steps above are just cursors change when an input runs in the M_1 and M_2.
In the actual operation, we can set one more characters to tag the rightmost, and the core idea is the following: First, We can see the difference is that the new-designed turning machine has two direction infinite tapes, however, original turning machine has only one. But we can extend the original turning machine to change it to "have two direction tapes", we use rightmost 1, 3, 5, 7, 9 ...2n + 1 positions to correspond to (count) the right 1,2,3...n positions, use rightmost 2, 4, 6, 8, ... 2n positions to correspond to the left, 1, 2, 3....n positions. And we do similar state changes in the M_2 as M_1 does, so we can see For a language which can be decided by a new designed turning machine M_1, we can design an original turning machine M_2. So we can see the power of these two turning machines is the same, can decide the same class of languages.

Part III

First, we will prove that a write-twice turning machine has the same power as the origin turning machine. For each of the write operation in the origin turning machine, we copy the entire tape used right to a new unused portion and marked the tape copied(then we know the head of new tape because the left of it has been marked, and the whole used tape has been copied), so a cell in the tape needs only to be marked and writen (write-twice most)), for the position to be updated, we marked it first, and when copying, if it is already marked, we update the target cell, othervise copy the cell and mark this cell. so a write-twice turning machine has the same power as the origin turning machine.
Second, we will prove write-once turning machine can work the same. To marked a cell, we split this cell into two parts, which means we use two cells, one to record character (can be update at most once), and a flag cell(marked or unmarked, can be updated only once), so this write-once turning machine works as the same logic as the write-twice turning machine (but each cell will be written at most once).
So in total, write-once turning machines can decide the same class of languages as the original turning machines.

Part IV

First, we can find this turning machine can do state-change and move left/right, no less powerful than DFA, so this turning machines can decide regular languages.
Next, we will prove this kind of modified turning machines can only decide regular languages. First, even thought we can write on the rest blank position, we can not simply copy the whole string to the rest and do as original turning machines does. Because we can not mark

input string which means, we can not remember which position we are copying now. So extra space is useless. Next we will prove this 2-way DFA can not bring more power to this kind machine than DFA, can noly decide regular languages.
We will prove this by using Myhill-Nerode Theorem. First, we will see how this 2-way DFAs work. we can split the input to x, z. At the beginning, the head is in the left of$x$, and within a series of operation, the machine will ends or the heads goes to z with a state q, and within z this machine will do a series of operation, and the machine will end or the head will go back to x with a state p. This loop will continue until the machine ends.
We can see x as a component, it will accept a state p from right or start from left, and do a series of operation finally the machine will stop or will emit a state q to the right.
So we can construct a state-change function: $\mathrm{T}(p) = q$ specified by input x, which means x will accept from right by a state p and emit a state q to right, but there are spcial output, $\mathrm{T}(p) = accept$, $\mathrm{T}(p) = reject$ which means input a state p, the machine will reject or accept, there are also special input $\mathrm{T}(start) = q$, which means when the machine starts, if the head emits x, it will emit a state q.
Then we can easily verify that if two inputs x, y with same function T will be equivalent to each other, because for a input z if $xz \in L$, yz must $\in$ to L, because with the same state from z, x, y will emit same state to z, all the machine will end with same results for x or y. And it is clear, there are just finite function T. If the state number is k, the number of function T is almost k^k, a finite number. That's to say, the number of equivalent classes for language L is finite, and according to Myhill-Nerode Theorem, language L is regular languages, so this modified turning machine can only decide regular languages.

Part V

1. Assume, we can construct a machine D, given an encoding of a turning machine M, it accepts when M halts on empty string, and it rejects when M doesn't halt on empty string.
   Now we can construct a turning machine H, given an encoding of a turning machine M_1, input w, we constructs a turning machine $T(M_1, w)$, when input is empty, this turning machine T will simulate w in M_1, Now, we can use D to judge if $T(M_1, w)$ will halt on empty string, if it halts, H accepts, otherwises, rejects. We see H can solve the halting problem which is undecidable, so we get a contradiction, so the problem above can not be decided.

2. Assume, we can construct a machine D, given an encoding of a turning machine M, it accepts when M will halt on some string, rejects if M won't halt on any string. Now we can construct a turning machine H, given an encoding of a turning machine M_1, input w, we constructs a turning machine $T(M_1, w)$, given any input, this turning machine T will simulate w in M_1, Now, we can use D to judge if $T(M_1, w)$ will halt some string, if it halts, H accepts, otherwises, rejects. Now we can see H accepts iff D accepts $T(M_1, w)$ iff $T(M_1, w)$ will halt on some string iff W_1 on w, so H solves the halting problem which is undecidable, so we get a contradiction, so the problem above can not be decided.

3. Assume, we can construct a machine D, given an encoding of a turning machine M, it accepts when M will contains finite number of strings that make M accepts. Now we can construct a turning machine H, given an encoding of a turning machine M_1, input

w, we constructs a turning machine $T(M_1, w)$, this turning machine T will simulate w in M_1, if M_1 halts, accepts. Now, we can use D to judge if T will contains finite number of strings that make T accepts, if it D accepts, H rejects, otherwises, accepts. Now we can see H accepts iff D rejects T iff T contains infinite number of strings that make T accepts iff W_1 halts on w, so H solves the halting problem which is undecidable, so we get a contradiction, so the problem above can not be decided.

4. Assume, we can construct a machine D, given an encoding of a turning machine M_1, M_2, it accepts if $L(M_1)$ equals $L(M_2)$, otherwise rejects.
   We construct a machine F, accepts given any input.
   Now we can construct a turning machine H, given an encoding of a turning machine M_1, input w, we constructs a turning machine $T(M_1, w)$, this turning machine T will simulate w in M_1, if M_1 halts, accepts. Now, we can use D to judge if $L(T)$ equals $L(F)$, if it D accepts, H accpets, otherwises, rejects. Now we can see H accepts iff D accepts T iff T decides langauge L containing all strings constructed from the alphabet iff W_1 halts on w, so H solves the halting problem which is undecidable, so we get a contradiction, so the problem above can not be decided.

Part VI

First, assume L is recursive, then we generates an enumerator, we enumerates strings in length increase and, in the same length, in lexicographical order, for each string w, we use L to judge if L accepts w, if L accepts w, print w. Then we gets an enumerator to enumerate L with length-increasing.
Second, assume that we can enumerates L in lenght-increasing fashion. Then we constructs a turning machine M, for each input w with length n, we starts the enumerator, if the output string equals w, then stop the enumerator, M accepts w, if the enumerator bgoes to output string with length $> n$, we stops enumerator, M rejects w.