Problem Set 6

Spring 10

Due: Thursday 2pm, 18th March, in class before the lecture begins.
Please follow the homework format guidelines posted on the class web page:
`http://www.cs.uiuc.edu/class/sp10/cs373/`

1. [**Category**: Closure properties, **Points**: 40]

   Prove that the following languages are not regular using only closure properties. You may assume that regular languages are closed under union, intersection, concatenation, complement, reverse and homomorphism. You may also assume that $\{0^n1^n \mid n \geq 0\}$ is not regular.

   Closure under homomorphisms: We define homomorphisms as follows. Let Σ and Π be two finite alphabets. A homomorphism is a function $h : \Sigma \to \Pi^*$, that maps a symbol from the alphabet Σ to a word in Π^*. We can extend this function to strings: $h(x_1x_2 \ldots x_n) = h(x_1) \cdot h(x_2) \cdot \ldots \cdot h(x_n)$, where $x_1 \ldots x_n \in \Sigma^*$. Intuitively, we just transform each symbol of the string and then concatenate everything together. We can now extend this function to languages: $h(L) = \{h(w) \mid w \in L\}$. It now turns out that regular languages are closed under homomorphisms.

   Theorem. Closure under homomorphisms: If L is a regular language over Σ and $h : \Sigma \to \Pi^*$ is a homomorphism, then $h(L)$ is also regular.

   For example, if $\Sigma = \{0, 1, 2\}$ and $\Pi = \{a, b\}$, and h is a homomorphism that maps $h(0) = ab$, $h(1) = \epsilon$ and $h(2) = bb$, then since $L = (01)^*12^*$ is regular, $h(L) = (ab)^*(bb)^*$ is also regular. You may assume the above theorem to solve the problems below.

   Prove that these languages are not regular:

   (a) $L_k = \{0^{kn}1^{kn} \mid n \geq 0\}$ for any fixed k (Hint: do this one first $\{0^{2n}1^{2n} \mid n \geq 0\}$)

Solution:

In all of these problems we assume that the set in question is regular. Then by using operations that preserve regularity we obtain a set that we know is not regular (for example $\{0^n1^n \mid n \geq 0\}$). We conclude that our assumption is wrong and that the set in question is irregular.

As a warm-up we first prove it just for L_2: Assume L_2 is regular. Then by closure properties $A = L_2 \cup (\{0\}L_2\{1\})$ is regular too. Now note that all the strings of the form 0^n1^n are in A (when n is even $0^n1^n \in L_2$ and when n is odd $0^n1^n \in (\{0\}L_2\{1\})$), therefore $A = \{0^n1^n \mid n \geq 0\}$. But we know that this last set is not regular and therefore we have got a contradiction. So L_2 is not regular.

Note for arbitrary k: Assume L_k is regular. Note that $\bigcup_{i=0}^{k-1}\{0^i\}L_k\{1^i\} = \{0^n1^n \mid n \geq 0\}$ (why? categorize strings in 0^n1^n by the remainder of n on k), but we

know that this last set is not regular (while it should be by closure properties). Therefore L_k is non-regular.

(b) $L = \{0^{2n}1^{3n} \mid n \geq 0\}$ (Hint: use homomorphism)

Solution:

Assume L is regular. Define this homomorphism: $h(0) = 000$ and $h(1) = 11$. Then observe that $h(L) = \{0^{6n}1^{6n} \mid n \geq 0\} = L_6$, which we know is not regular (previous part) while by closure properties it should be. Therefore L is non-regular.

(c) $L = \{0^n\$1^n\$\$^* \mid n \geq 0\}$

Solution:

Assume L is regular. Let $A = L \cap L(0^*\$1^*\$) = \{0^n\$1^n\$ \mid n \geq 0\}$. Now consider this homomorphism: $h(0) = 0$, $h(1) = 1$ and $h(\$) = \epsilon$. Observe that $h(A) = \{0^n1^n \mid n \geq 0\}$. This last set is not regular while by closure properties it should be. Therefore L is not regular.

(d) $L = \{w_1\$w_2\$w_3 \mid w_1, w_2, w_3 \in \{0,1\}^*$ number of 0s in w_1 is equal to the number of 1s in $w_2\}$

Solution:

Assume L is regular. Let $A = L \cap L(0^*\$1^*\$) = \{0^n\$1^n\$ \mid n \geq 0\}$. Using h from the previous part: $h(A) = \{0^n1^n \mid n \geq 0\}$. Again the last set is not regular while closure properties imply that it should be. Therefore L is non-regular.

(e) $L = \{0^n\$0^m \mid 0 \leq n \leq m\}$

Solution:

Assume L is regular. Observe that $L \cap L^R = \{0^n\$0^n : n \geq 0\}$. We know that this last set is non-regular (from MNT), while it should be regular by closure properties. Thus we get a contradiction and L can't be regular.

2. 2D Tape TM [**Category**: Simulation., **Points**: 20]

A *two-dimensional Turing machine* is like a Turing machine except that instead of a one-dimensional tape it has a two-dimensional tape that is like the upper right quadrant of the plane, infinite in *up* and *right* directions. It has a finite input alphabet Σ and a finite tape alphabet Γ. If $x \in \Sigma^*$ is the input, $|x| = n$, the machine starts in its start state s with x written in tape cells $(0,0), (0,1), \ldots, (0,n-1)$. It has a read/write head initially pointing to the origin $(0,0)$. In each step, it reads the symbol of Γ

currently occupying the cell it is scanning. The transition of the Turing machine tells the TM, when reading a symbol and when at the current state of the finite control, to rewrite the symbol to another one in Γ, and moves the head either *up, down, left* or *right*, and move to a new state. Prove that two-dimensional TMs and normal TMs are equivalent. Describe the simulations *informally* but in sufficient detail that the simulation is understandable, clear and easy to see it's correct.

Solution:

We first simulate the 2D tape with a 1D tape as follows:

1 Introduce a fresh new symbol \$ into the input alphabet Γ. The row of the 2D tape starting from the "bottom-most" row are written in the 1D tape next to each other, as in the figure below. Note that this is indeed possible, at any point of time, the TM has looked at only a finite number of rows, and a finite number of tape cells in each row.

2 When the machine tries to move right from a cell in the 2D tape, we can simulate it in the 1D tape trivially, except when it tries to move right from the rightmost cell in a row. In that case, we "make room" for the new tape cell by shifting all the symbols to the right. For instance, when the machine tries to move right after reading x_5 in the figure, we simulate the effect in the 1D tape as follows: We move all the symbols starting from the \$ right after x_5 in the 1D tape, one cell to the right. In the newly vacant cell, we write a blank, and continue simulating.

3 When the machine tries to move left from the leftmost cell in a row, we do nothing(The machine hits the wall). Otherwise, we can trivially simulate it in the 1D tape.

4 When the machine tries to move up, we move left until we reach a \$ symbol to determine which column we are in. Then we move right beyond the next \$ and into the corresponding column one row up(To count, we might use an auxiliary tape and later use the 2-tape TM to 1-tape TM simulation). If there is no enough room in the next row, we make room by moving all the tape cells to the right, just as in step 2.

5 When the machine tries to move down, we proceed exactly as in step 4, except that now, we move one row down. In case we are in the bottommost row, we do nothing(Again, the machine hits the wall).

⋮	⋮	⋮	⋮		
⊔	⊔	⊔	⊔	$\cdots$	Row 3
x_6	⊔	x_7	⊔	$\cdots$	Row 2
x_4	x_5	⊔	⊔	$\cdots$	Row 1
x_1	x_2	x_3	⊔	$\cdots$	Row 0

x_1	x_2	x_3	\$	x_4	x_5	⊔	\$	x_6	⊔	x_7	\$	$\cdots$

In the other direction, to simulate a 1D tape TM with a 2D tape TM, we simply use the bottommost row of the 2D tape, then an 1D tape TM is automatically a 2D tape TM.

3. Enumeration [**Category**: Comprehension, **Points**: 20]

Fix an alphabet Σ. Assume we fix some encoding to represent regular expressions over Σ and DFAs as binary strings. (so as usual $\langle R \rangle$ represents the binary encoding of the regular expression X). Assume also that there is some encoding of DFAs as binary strings.

Consider the following language:

$L = \{\langle R \rangle \mid R$ is a regular expression such that there is some DFA D with language $L(R)$ and $\langle D \rangle$ is a prime number written in binary.$\}$

In other words, L is the set of all encodings of regular expressions R such that some DFA D, whose binary representation represents a prime number, accepts the same language as that of R.

Show that L is TM-recognizable by constructing a TM recognizing L. Your construction could be pseudo-code or an exact explanation of how it works. You don't need to do a formal low-level construction, but just intuitively describe a high-level construction.

Solution:

The idea is that once we have the regular expression R, we start checking all the possible DFA's (we know that DFA's are enumerable, in fact lines 2-5 in pseudo-code below enumerate all DFA's). Once we get a DFA $\langle D \rangle$ that we want to check, first we check that $\langle D \rangle$ is a prime number (just by dividing it to all the smaller numbers to see whether it has a nontrivial factor) and second we check that $L(D) = L(R)$ (by converting R to a DFA, building a new DFA D' for $L(R) \Delta L(D)$ (symmetric difference: $A \Delta B = (A - B) \cup (B - A)$) and checking whether this DFA accepts something or its language is empty). We accept if both checks succeed otherwise, we check the next DFA.

Now note that once the regular expression $\langle R \rangle$ is a member of L, such a DFA D exists for it (by the definition of L) and since we are checking all the DFA's (and each check finishes in finite amount of time) we will finally find such a witness DFA. Therefore by the definition, this construction is a recognizer for language L.

Here is pseudo-code for this recognizer:

Algorithm $isInL(r)$

1. Check that r is a valid encoding of some regular expression, say $r = \langle R \rangle$.
2. $\mathcal{N} \leftarrow$some enumerator of all binary strings
3. **for** $i \leftarrow 1$ **to** ∞
4. $\quad B \leftarrow \mathcal{N}(i)$
5. $\quad$ **if** B is a valid encoding of some DFA D and $isPrime(B)$

(∗ To check that B is prime, we just divide it on all naturals smaller than B to ensure ... ∗)

(∗ ... it has exactly two divisors ∗)

6. $\quad\quad$ **then if** $L(D) = L(R)$

(∗ for this check, we can convert R into a DFA, build a DFA D' for ... ∗)

(∗ ... $L(R)\Delta L(D)$, and check that D' has no path from its initial state to some final state. ∗)

7. $\quad\quad\quad$ **then return** "yes".

Algorithm $isPrime(x)$

1. **if** $x = 1$
2. $\quad$ **then return false**
3. **for** $i \leftarrow 2$ **to** $x - 1$
4. $\quad$ **do if** x/i is integer
5. $\quad\quad$ **then return false**
6. **return true**