Harvard University
Computer Science 121
Problem Set 5

Due October 27, 2015 11:59pm

Problem set by Kevin Zhang

Collaboration Statement: I collaborated with Tomoya Hasegawa and Mandela Patrick, got help from no one else, and used no other resources.

PART A (Graded by Cecilia and Varun)

PROBLEM 1 (4+2+1 points, suggested length of 1/2)

Consider the Turing machine $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$ where

- $Q = \{q_0, q_1, q_2, q_{\text{accept}}, q_{\text{reject}}\}$,
- $\Sigma = \{a, b\}$ and $\Gamma = \{a, b, \sqcup\}$,
- The start, accept and reject states are q_0, q_{accept}, and q_{reject} respectively.
- The function δ is given by:

q	σ	$\delta(q,\sigma)$
q_0	a	(q_0, b, R)
q_0	b	(q_1, a, R)
q_0	$\sqcup$	$(q_{\text{reject}}, \sqcup, R)$
q_1	a	(q_2, b, R)
q_1	b	(q_1, a, R)
q_1	$\sqcup$	$(q_{\text{reject}}, \sqcup, R)$
q_2	a	(q_2, b, R)
q_2	b	(q_2, a, R)
q_2	$\sqcup$	$(q_{\text{accept}}, \sqcup, R)$

(A) Give the sequence of configurations describing M's computation on the string $aabba$

(B) Describe $L(M)$.

Solution.

(A) The sequence of configurations is given by q_0aabba, bq_0abba, bbq_0bba, $bbaq_1ba$, $bbaaq_1a$, $bbaabq_2\sqcup$, $bbaab \sqcup q_{accept}$. $aabba$ is accepted.
(B) The only accept state results when a configuration contains $q_2\sqcup$. q_2 is only reachable if a configuration contains q_1a, and q_1 is only reachable if a configuration contains q_0b. This sequence of events tells us that $L(M) = \{w : w \text{ has a } b \text{ before an } a \text{ somewhere inside it}\}$.

PROBLEM 2 (4 points, suggested length of 1/4)

A Modern Turing Machine (MTM), instead of $\{L, R\}$, has $\{L_k, R_k\}$ (move the head left or right k spaces on the tape, for any integer k encoded in the rule in the MTM). Prove that the MTMs are equivalent in power to the TMs. In other words, any MTM can be converted to a TM with an equivalent language, and vice versa.

Solution.
Every TM is an MTM with $k = 1$, so converting from a TM to an MTM is easy. It remains to figure out some construction from an MTM to a TM. This is easy to do if we insert some (a lot) transition states. Let's take every rule in the MTM and convert it as follows. If the rule is defined as $\delta(q, \sigma) = (q', \sigma', L_k)$ (something similar follows for R_k), we insert $k-1$ new states and define the rules as follows:

- Define new states $q_{q\sigma i}, 1 \leq i < k$
- $\delta(q, \sigma) = (q', \sigma', L_k)$, the original rule in the MTM, becomes $\delta(q, \sigma) = (q_{q\sigma 1}, \sigma', L)$
- $\delta(q_{q\sigma i}, x) = (q_{q\sigma(i+1)}, x, L)$ for all $x \in \Gamma$, where $1 \leq i < k - 1$
- $\delta(q_{q\sigma(k-1)}, x) = (q', x, L)$ for all $x \in \Gamma$

What this does is insert $k - 1$ states that do essentially nothing except move one place in the correct direction and transition to the next state. The transition $q \mapsto q'$ becomes $q \mapsto q_{q\sigma 1} \mapsto q_{q\sigma 2} \mapsto \cdots \mapsto q_{q\sigma(k-1)} \mapsto q'$. The result is a TM that has an large number of "useless" states that do not convey much information but that still recognizes the same language as the MTM.

*N.B. The case $k = 0$ is handled by simply having the TM move one space to the right then one space to the left.

PROBLEM 3 (2 + 2 + 1 points, suggested length of 1/2 page)

A Democratic rule of a CFG (V, Σ, R, S) is one whose right-hand side is a member of $V\Sigma^* \cup \Sigma^*$, that is, only the leftmost symbol can be a nonterminal. Republican rules are defined analogously where only the rightmost symbol can be a nonterminal. Show that a grammar with only Democratic rules, or only Republican rules, generates a regular language, but a grammar with a mixture of Democratic and Republican rules may generate a non-regular language.

Solution.
We'll examine the Democratic case first. Let the CFG be (V, Σ, R, S). We can construct an NFA for this grammar as follows:

- $Q = V$
- $\Sigma = \Sigma$
- $S = S'$, where S' is a new state we define
- Transitions in the DFA correspond to productions between nonterminals; if $A \mapsto Bs$, then a transition in the DFA exists from B to A on s; i.e. $\delta(B, s) = A$. This takes care of all rules with non-terminals in them.
- Rules of the form $X \mapsto s$ are taken care of by the transition $\delta(S', s) = X$ where F is the start state we defined above.

- $F = S$; i.e. the original start state becomes the final state.

For example, if we had the rules $S \mapsto Ab, A \mapsto Aa|B, B \mapsto b$, the NFA's transition function would be given by $\delta(S', b) = B, \delta(B, \varepsilon) = A, \delta(A, a) = A, \delta(A, b) = S$.

Similarly, we can define such an NFA for Republican grammars. Let the CFG be (V, Σ, R, S). We can construct an NFA for this grammar as follows:

- $Q = V$
- $\Sigma = \Sigma$
- $S = S$
- All rules $A \mapsto sB$ are converted to $\delta(A, s) = B$
- All rules $A \mapsto s$ are converted to $\delta(A, s) = F$, where F is a single state that we define
- $F = F$.

For example, if we had the rules $S \mapsto A, A \mapsto aA|B, B \mapsto b$, the NFA's transition function would be given by $\delta(S, \varepsilon) = A, \delta(A, a) = A, \delta(A, \varepsilon) = B, \delta(B, b) = F$.

For the last part of this proof, we just need to find a grammar with a mixture of Democratic and Republican rules can generate a non-regular language. Consider the rules $S \mapsto aA, A \mapsto Bb, B \mapsto aA|\varepsilon$. With a bit of analysis, one can actually show that this generates the language $a^n b^n, n \geq 0$, which is the canonical non-regular language we have seen so much in this class.

PROBLEM 4 (7 points, suggested length of 3/4 page)

Let us define EvenPalindrome$(L) = \{ww^R : w \in L\}$. Show that if L is regular, then EvenPalindome(L) is context-free. Note that this is not the same as the language of all even palindromes, which is actually just EvenPalindrome(Σ^*). Explain briefly how your solution works. You need not provide a formal proof of correctness.

Solution.
Since L is regular, there exists a CFG that produces exactly L; in fact, from the reading in Sipser, it is actually possible to construct a right-linear CFG (a "Republican" one, if you will). Let this CFG be (V, Σ, R, S). This tells us that it is possible to express L purely in terms of rules of the form $A \mapsto sB$ or $A \mapsto s$. We can construct a CFG that accepts EvenPalindrome(L) as follows:

- $V' = V$
- $\Sigma' = \Sigma$
- $S' = S$
- Rules in R of the form $A \mapsto s$ are transformed into $A \mapsto ss^R$
- Rules in R of the form $A \mapsto sB$ are transformed into $A \mapsto sBs^R$.

This works by taking every production rule that adds some characters to the current string and transforming it into a rule that adds the same characters both in order and in reverse order around the center of the string. For example, if we had the set of rules $S \mapsto aA, A \mapsto aA|B, B \mapsto b$, then we would have $S \mapsto aAa, A \mapsto aAa|B, B \mapsto bb$. The original set of rules generates a^*b, and the second set of rules generates $a^n bba^n, n \geq 0$, as required.

PART B (Graded by Madhu and Zhengyu)

PROBLEM 5 (CHALLENGE!! (3) points, suggested length of 1/2 page)

A Boring Turing Machine (BTM) can only write # on the tape (assume $\# \notin \Sigma$). Prove that the BTMs are equivalent in power to the TMs.

Solution.

PROBLEM 6 (6 points, suggested length of 1/3 page)

Show that every infinite Turing-recognizable language has an infinite decidable sublanguage.

Solution.

The idea here is to determine some property of a language that makes it easy to construct a decisive TM for it. We look at enumerators for this; since they output all of the strings of a language, we can use the strings outputted by the enumerator to determine whether an input is in the language or not. To be more exact, we can compare an input string against every output of the enumerator to see if there's a match; if there is, we accept.

There is one nuance, which is that we need some way to deciding to reject an input string; if we don't, the TM will continue comparing against the enumerator's output strings until the enumerator stops (or, if it doesn't, the TM will never terminate). To solve this, we need to figure out some way to determining when we should stop reading from the enumerator's output. In other words, at some string s_i from the enumerator, we need some way to compare an input string s to s_i and decide to reject. One way we can do this is require that the enumerator output the strings in order of nondecreasing length; in this way, if we start reading strings from the enumerator that have longer length than s, we can stop and reject.

Using this, we give the following construction for an infinite, decidable sublanguage. Suppose we have an infinite Turing-recognizable language L with enumerator E. We construct E' as follows:

- Run E.
- For each string that comes out of E,
  - If the string has equal or greater length than the last string printed out by E', have E' print out the string.
  - Otherwise, ignore and continue.

This process constructs an enumerator that outputs strictly nondecreasing length strings, which, by the above argument, results in an outputted language that is decidable. Moreover, note that this language must have infinite length, because E is infinite and therefore, if it has already outputted a string of length n, eventually must output a string of length $n+1$ (because the starting language is infinite and there are only finitely many strings of length at most n). Hence, the above construction gives us an infinite decidable sublanguage, and we are done.

PROBLEM 7 (4+2+4 points, suggested length of 1 page)

Let $G = (V, \Sigma, R, S)$ where $V = \{S, V\}$, $\Sigma = \{a, b\}$, and R is the set of rules:

$$S \to bSS \mid aS \mid aV$$
$$V \to aVb \mid bVa \mid VV \mid \varepsilon$$

(A) Transform G into an equivalent grammar G' in Chomsky normal form. Show your work for each step of this conversion, but long justifications for each step are not necessary.

(B) Explain in English what language the grammar G generates. (One clearly worded sentence is fine; thorough justification is not necessary, but it might get you some partial credit if your answer is wrong.)

(C) Check whether the strings *abaab* and *bbabaa* are generated by G, using the recognition algorithm for grammars in Chomsky normal form given in class. Show the complete filled-in matrix for each of the two strings.

Solution.

(A) First, we add the new start variable S_0 and the rule $S_0 \to S$. Next, we remove ε-rules; $V \to \varepsilon$ gets removed and $S \to aV$ becomes $S \to aV|a$. Now we remove unit rules; $S_0 \to S$ becomes $S_0 \to bSS|aS|aV|a$ instead. The last step is converting all remaining rules into the form $A \to BC$ by adding new nonterminals. Hence, $S_0 \to bSS$ becomes $S_0 \to BC$, $B \to b$, and $C \to SS$; $V \to aVb$ becomes $V \to AD$, $A \to a$, $D \to VE$, and $E \to b$; $V \to bVa$ becomes $V \to BF$, $F \to VA$. In total, we have the rules,

$$\begin{gathered} S_0 \to BC|AS|AV|a \\ S \to BC|AS|AV|a \\ A \to a \\ B \to b \\ C \to SS \\ V \to AD|BE|AB|BA|VV \\ D \to VB \\ E \to VA \end{gathered}$$

(B) The given first set of rules for S, if we replace the V with a ε, results in strings that always have at least one more a's than b's. We can see this because S's are eliminated by replacing them with aV's, and b's can only be inserted by increasing the number of S's by 1 as well. The second set of rules for V describes all strings with equal numbers of a's and b's. V can be inserted almost anywhere because of the rule $S \to aV$. This language is the language of strings that have at least one more a's than b's. We can thoroughly prove this if we wish by reverse-constructing the production rules for any string.

(c)

S, S_0				
C, S, S_0	V			
E	E, S, S_0	S, S_0		
V	V	S, S_0, C	V	
A, S, S_0	B	A, S, S_0	A, S, S_0	B
a	b	a	a	b

abaab is accepted because S_0 is in the topmost left cell.

V					
$-$	E, S, S_0				
$-$	V	C			
$-$	D	E, S, S_0	E		
$-$	V	V	V	C, S, S_0	
B	B	A, S, S_0	B	A, S, S_0	A, S, S_0
b	b	a	b	a	a

bbabaa is not accepted because S_0 is not in the topmost left cell.