CSE 555: Homework 1

February 8, 2015

(1)

(a) ALLLEN\textsubscript{NFA} is decidable. The following Turing machine \( U \) decides the language ALLLEN\textsubscript{NFA}:

\( U \) = "On input \(<M>\), where \( M = (Q, \Sigma, \delta, q_0, F) \) is an NFA:

1. Build a new NFA \( M' \) that has the same set of states, start state and set of accept states as \( M \), but uses the unary alphabet \( \{1\} \). The transitions in \( M' \) are are same as that of \( M \) but all of them are now labeled by 1, i.e. the transition function \( \delta' \) of \( M' \) is defined as follows: for all \( q \in Q \), \( \delta'(q, 1) = \cup_{a \in \Sigma} \delta(q, a) \). Convert \( M' \) into an equivalent DFA \( M'' \). (\( M'' \) recognizes the language \( length(L(M)) \)).

2. Build a DFA \( N \) that recognizes the language \( 1^* \).

3. Let \( T \) be the Turing machine that decides EQ\textsubscript{DFA} (it exists since EQ\textsubscript{DFA} is decidable). Run \( T \) on \(<M'', N>\).

4. If \( T \) accepts, then accept, if \( T \) rejects then reject."

(b) REGLEN\textsubscript{NFA} is decidable. In step 1 of the Turing machine \( U \) in the previous question we have shown how to build an NFA to recognize \( length(L(M)) \), given the NFA \( M \). So for every NFA \( M \), \( length(L(M)) \) is regular.

Therefore the trivial Turing machine \( S \) that decides REGLEN\textsubscript{NFA} just checks if \(<M>\) is a proper encoding of an NFA. It accepts if it is, and rejects if it is not.

(2)

Suppose that \( L' \) is a CFL with CFG \( G' \) in Chomsky normal form. Replace every terminal symbol by a 1 to get a CFG \( G \) for the language \( L = length(L') \). Then \( G \) is a CFG in CNF with \(|\Sigma| = 1\) for \( L \).

Claim: Every context-free language \( L \subseteq \Sigma^* \) with \(|\Sigma| = 1\) is regular.

Proof: Let \( G \) be a CFG in CNF for \( L \). Let \( p \) be the pumping length in the pumping lemma for CFLs. Given \( G \), we can compute the value of \( p \) (and it is fixed once \( G \) is fixed). Every string \( w \in L \) with \(|w| > p \) can be written as \( wxyz \) so that \( S \xrightarrow{*} u.Az \), \( A \xrightarrow{*} vAy \), and \( A \xrightarrow{*} x \), with \( 1 \leq |vy| \) and \(|vxy| \leq p \). From \( G \) we will form a new grammar \( R \) with the same variables as \( G \), and a set of rules \( \mathcal{R} \), which is initially empty. First for every \( w \) with \(|w| \leq p \) and every variable \( A \) for which \( A \xrightarrow{*} w \) (this is easily determined because \( G \) is in CNF), add rule \( A \rightarrow 1^{|w|} \) to \( \mathcal{R} \). Then for every two variables \( A, B \), and strings \( f, g \) with \( 1 \leq |fg| \leq p \), whenever \( A \xrightarrow{*} fBg \) in \( G \), add the rule \( A \rightarrow 1^{|f|+|g|}B \) to \( \mathcal{R} \).

Because each rule in \( \mathcal{R} \) has at most one variable on the right hand side of each rule, and when there is one it appears at the end, it is easy to convert \( R \) into an equivalent regular grammar.

So we need only check that \( L(G) = L(R) \). If \(|w| \leq p \), we have the rule \( S \rightarrow w \) if and only if \( w \in L(G) \), so either \( w \) is in both \( L(R) \) and \( L(G) \) or neither.
If \(|w| > p\), whenever \(w \in L(R)\) we have \(w \in L(G)\) because we can replace the rules of \(R\) by the corresponding derivations in \(G\). On the other hand, if \(w \in L(G)\), the pumping lemma ensures that it has a decomposition of the required type and now a structural induction establishes that \(w \in L(R)\). End of claim.

We note that the proof of the claim actually produces an equivalent regular grammar from the CFG given. So we use the proof to convert the CFG to a regular grammar. This can in turn be converted to an equivalent NFA.

At this point, we have reduced the answers to 2(a) and 2(b) to the corresponding answers to questions 1(a) and 1(b). So both are decidable (indeed the second is trivial, because if \(L\) is a CFL, \(\text{length}(L)\) is always regular).

(3)

(a)

This statement is true. Let \(S\) be the Turing machine that decides \(L\). We build a Turing machine \(T\) that decides \(\text{length}(L)\) as follows:

\[
T = \text{"On input } 1^n, n \geq 0:\n\]

1. Enumerate all the strings \(w\) in \(\Sigma^n\) (there are only finite number of such strings \(w\)). Run \(S\) on such strings one by one.

2. If \(S\) accepts any such \(w\) then accept, otherwise reject.”

(b)

This statement is false. \(L = \overline{\text{A}_{\text{TM}}}\) is a counter example. \(\text{length}(\overline{\text{A}_{\text{TM}}}) = 1^*\), since for any \(n\), one can construct a string \(w\) with \(|w| = n\), such that \(w\) is not a valid encoding of any Turing machine, so \(w \in \overline{\text{A}_{\text{TM}}}\). So \(\text{length}(\overline{\text{A}_{\text{TM}}})\) is decidable, but \(\overline{\text{A}_{\text{TM}}}\) is not recognizable.

(4)

By Rice’s theorem \(\text{REGLEN}_{\text{TM}}\) is undecidable. To apply Rice’s theorem we need to show that \(\text{REGLEN}_{\text{TM}}\) is a non-trivial property of the language of a Turing machine. It is non trivial – we have already seen examples where \(\text{length}(L(M))\) is regular (since NFAs can be simulated by Turing machines). Here is an example of a Turing machine \(S\) that does not belong to \(\text{REGLEN}_{\text{TM}}\) – let \(S\) be the Turing machine that decides the language \(\{w : w\text{ is the unary encoding of a prime number}\}\). \(\text{length}(L(S))\) is not regular. Moreover, \(\text{REGLEN}_{\text{TM}}\) is a property of the language of a Turing machine, i.e. whenever \(L(M_1) = L(M_2)\), we have \(\text{length}(L(M_1))\) is regular iff \(\text{length}(L(M_2))\) is regular.

(5)

Here is a Turing machine \(U\) that recognizes \(\overline{D}\):

\[
U = \text{"On input } < G_1, G_2 >, \text{ where } G_1 \text{ and } G_2 \text{ are CFGs:}\n\]

1. Convert \(G_1\) and \(G_2\) to CNF grammars \(G'_1\) and \(G'_2\) respectively.

2. For \(n = 1, 2, \ldots\) list all derivations of length \(2n - 1\) of \(G'_1\) and \(G'_2\) and compare them.

3. If there is any string that is present in both the lists (i.e. it can be derived in derivations of length \(2n - 1\) in both \(G'_1\) and \(G'_2\)) then accept.

4. Otherwise increase the value of \(n\) by one and go to step 2.”
We will show that \( D \) is Turing undecidable. For contradiction, assume that \( D \) is decidable and \( S \) is a Turing machine that decides \( A_{TM} \). Given input \( < M, w > \) where \( M \) is a Turing machine and \( w \) is a string, there will be an accepting computation history of \( M \) on \( w \) iff \( M \) accepts \( w \). Let us encode an accepting computation history as: 
\[
\#C_1\#C_{rev}^2\#C_3\#C_{rev}^3\# \ldots \#C_{rev}^l\#
\]
where \( C_1 \) is the starting configuration, \( C_l \) is an accepting configuration and where each \( C_i \) follows \( C_{i-1} \) according to the transitions function of the Turing machine \( M \).

\( T \) constructs two CFGs \( G_1 \) and \( G_2 \) in such a way that the only string that can be derived in both the grammars is the accepting computation history. So \( L(G_1) \cap L(G_2) \) is non-empty if \( M \) accepts \( w \), otherwise this intersection is empty. Next we show the construction of \( G_1 \).

\[
\begin{align*}
G_1 & : S \to \#C_1\#C_{rev}^2\#T, \\
T & \to X\#T | Y\#, \\
X & \to aXa \text{ for } a \in \Gamma \\
X & \to qZq \text{ for } q \in Q \\
X & \to qaZq' b \text{ when } \delta(q, a) = (q', b, R) \\
X & \to cqaZbcq' \text{ when } \delta(q, a) = (q', b, L) \\
Z & \to aZa \text{ for } a \in \Gamma \\
Z & \to \# \\
Y & \to aYa \text{ for } a \in \Gamma \\
Y & \to qaWq_{\text{accept}} b \text{ when } \delta(q, a) = (q_{\text{accept}}, b, R) \\
X & \to cqaWbcq_{\text{accept}} \text{ when } \delta(q, a) = (q_{\text{accept}}, b, L) \\
W & \to aWa \text{ for } a \in \Gamma \\
W & \to \#
\end{align*}
\]

Note that because we can generate the same configuration twice (or more) consecutively, we can assume that the computation history has an even number of configurations.

Similarly \( T \) constructs the grammar \( G_2 \) that generates the start configuration, followed by pairs of matching configurations that follow one another via the transition function (as generated by \( X \) above, but the R and L are reversed). Then it generates any configuration that contains \( q_{\text{accept}} \).

Next \( T \) runs \( S \) on \( \langle G_1, G_2 \rangle \) and accept \( \langle M, w \rangle \) if \( S \) rejects. If \( S \) accepts then \( T \) builds similar grammars \( G_1' \) and \( G_2' \) assuming that \( l \) is odd. Next \( T \) runs \( S \) on \( \langle G_1', G_2' \rangle \) and accepts \( \langle M, w \rangle \) if \( S \) rejects. Otherwise \( T \) rejects \( \langle M, w \rangle \).

What is going on? The first grammar ensures that the configurations with even index are indeed obtained from the prior configuration, and that we start with the initial configuration and end with an accepting one. The second grammar ensures that the configurations with odd index are indeed obtained from the prior configuration. So any string is generated by both grammars if and only if it is an accepting computation history of \( M \) on \( w \).