

CS273: Theory of Computation, Summer 2008

Homework 7

Due 3:00PM Monday, July 28, 2008 at SC 3303

Revision due 3:00PM Thursday, July 31, 2008 at SC 3303

1. [60 points; 10 per part] Show that the following languages are decidable:

(a) $\{\langle R, S \rangle \mid R, S \text{ are regular expressions and } L(R) \subseteq L(S)\}$

Solution: The important observation is that $A \subseteq B$ if and only if $A \cap \overline{B}$ is empty. Intersection and complementation are closure properties of regular languages, so if A and B are regular, so is this new language. Thus we can reduce this problem to E_{DFA} .

M_a . On input $\langle R, S \rangle$:

1. Convert R and S to equivalent DFAs M_1 and M_2
2. Obtain a DFA M_{test} where $L(M_{\text{test}}) = L(M_1) \cap \overline{L(M_2)}$, using the closure operations of DFAs (swap accept/reject states, and product construction).
3. Run the decider for E_{DFA} on $\langle M_{\text{test}} \rangle$ and agree with its output. ■

(b) $\{\langle M, w \rangle \mid M \text{ is a DFA that accepts some string } x \text{ that has } w \text{ as a substring}\}$

Solution: For a string w , let R_w be the regular expression $\Sigma^*w\Sigma^*$. Thus $L(R_w)$ is the set of strings that contain w as a substring. We want to know whether $L(M) \cap L(R_w)$ is empty (i.e., whether there is a string that is both accepted by M and also contains w as a substring).

M_b . On input $\langle M, w \rangle$:

1. Given w , construct the regular expression R_w , and convert it to a DFA M_w .
2. Obtain a DFA M_{test} where $L(M_{\text{test}}) = L(M) \cap L(M_w)$, using the product construction.
3. Run the decider for E_{DFA} on $\langle M_{\text{test}} \rangle$ and disagree with its output. ■

(c) $\{\langle M \rangle \mid M \text{ is a DFA that accepts both } w \text{ and } w^R \text{ for some string } w\}$

Solution: Equivalently, we want to know whether there is any w in both $L(M)$ and $L(M)^R$. In terms of a set emptiness question, we want to know whether $L(M) \cap L(M)^R$ is empty. Recall that set-reversal is a closure property of regular languages.

M_c . On input $\langle M \rangle$:

1. Construct a DFA M_R where $L(M_R) = L(M)^R$. This requires making an NFA and converting back to a DFA.
2. Obtain a DFA M_{test} where $L(M_{\text{test}}) = L(M) \cap L(M_R)$, using the product construction.
3. Run the decider for E_{DFA} on $\langle M_{\text{test}} \rangle$ and disagree with its output. ■

- (d) $\{\langle N \rangle \mid N \text{ is an NFA and } N \text{ accepts some string that has more 0s than 1s}\}$

Solution: Similar to the previous parts, we want to know whether there is any w in both $L(N)$ and $\{x \mid \#0(x) > \#1(x)\}$. However, the second language is not regular but context-free, so we have to do something slightly different. The machine will have a PDA for this CFL hard-coded.

M_d . On input $\langle N \rangle$:

1. Convert N to a DFA M_N .
2. Using the product construction, obtain a PDA P where $L(P) = L(M_N) \cap \{x \mid \#0(x) > \#1(x)\}$, using the hard-coded PDA for the latter language.
3. Convert P to an equivalent CFG G_{test} .
4. Run the decider for E_{CFG} on $\langle G_{\text{test}} \rangle$ and disagree with its output. ■

- (e) $\{\langle M \rangle \mid M \text{ is a DFA that accepts some palindrome}\}$

Solution: The solution this part is almost identical to the previous part, except we replace the hard-coded PDA for $\{x \mid \#0(x) > \#1(x)\}$ with a hard-coded PDA for the set of palindromes. ■

- (f) $\{a^n \mid \text{the decimal expansion of } \pi \text{ contains } n \text{ consecutive zeros}\}$

Solution: There are two cases here. Either there is no limit to the length of these runs of 0s in the expansion of π , or there is some limit. It is an open question in number theory which case is actually true. Nevertheless, we can say that in either case the above language is decidable (in fact, regular).

- Case 1: There is no limit to the length of runs of 0s in π . That is, every long sequence of 0s does eventually occur somewhere in π 's digits. Then the language in this problem is just $L(a^*)$.
- Case 2: There is some limit to the length of runs of 0s in π . That is, there is a maximum length to the runs of 0s that occur in π . Call that length N . So π contains a run of N 0s, but no longer runs. Runs of length less than N also occur in π , as substrings of the run of length N .
So in this case, the language in this problem is $\{\varepsilon, a, aa, \dots, a^N\}$, which is a finite set and therefore regular. ■

Hint: In parts (a) through (e), you might try to reduce these languages to E_{DFA} and/or E_{CFG} , which were defined in lecture and in the textbook.

$$E_{\text{DFA}} = \{\langle M \rangle \mid M \text{ is a DFA and } L(M) = \emptyset\}$$

$$E_{\text{CFG}} = \{\langle G \rangle \mid G \text{ is a CFG and } L(G) = \emptyset\}$$

2. [30 points; 10 per part] A *useless state* of an automaton (DFA, PDA, TM) is a state that is never entered on any input.

- (a) Show that $\{\langle D, q \rangle \mid D \text{ is a DFA containing a useless state } q\}$ is decidable.

Solution: Let $A = \{\langle D, q \rangle \mid D \text{ is a DFA containing a useless state } q\}$. We reduce A to E_{DFA} . Because E_{DFA} is decidable, there is a Turing machine (call it N) that decides E_{DFA} . We define a Turing machine M that decides A as follows.

M. On input $\langle D, q \rangle$:

1. Construct a new DFA D' by modifying D so that q is the only accept state.
2. Simulate N on $\langle D' \rangle$. If N accepts, then accept. Otherwise reject.

If q is useless in D , then there is no string w on which D enters q , and therefore there is no string accepted by D' . Therefore $L(D') = \emptyset$, so N accepts $\langle D' \rangle$ and therefore M also accepts. Conversely, if q is not useless in D , then there is a string w on which D enters q , and therefore D' accepts w . It follows that $L(D') \neq \emptyset$, so N rejects $\langle D' \rangle$ and therefore M also rejects. Hence, M decides A .

Note: many students did this problem by marking the states in the input DFA. If you look carefully at these solutions, what they are really doing is replacing step (2) of M with the algorithm for E_{DFA} given in Sipser. The power of expressing M in the way that we have above — that is, giving a reduction from A to E_{DFA} — is that the reduction technique generalizes to give a solution to part (b) and provides insight into part (c). The same cannot be said for the technique of marking states, which does not help in part (b) because simply marking states does not take into account the PDA's stack. ■

(b) Show that $\{\langle P, q \rangle \mid P \text{ is a PDA containing a useless state } q\}$ is decidable.

Solution: Let $B = \{\langle P, q \rangle \mid P \text{ is a PDA containing a useless state } q\}$. We reduce B to E_{CFG} . Because E_{CFG} is decidable, there is a Turing machine (call it N) that decides E_{CFG} . We define a Turing machine M that decides B as follows.

M. On input $\langle P, q \rangle$:

1. Construct a new PDA P' by modifying P so that q is the only accept state.
2. Using the algorithms we have seen in class, convert P' to a CFG G .
3. Simulate N on $\langle G \rangle$. If N accepts, then accept. Otherwise reject.

Note that if q is useless in P , then there is no string w on which P enters q , and therefore there is no string accepted by P' . Therefore $L(P') = \emptyset$ and $L(G) = \emptyset$ also. Therefore N accepts $\langle G \rangle$ and so M also accepts. Conversely, if q is not useless in P , then there is a string w on which P enters q , and therefore P' accepts w and G generates w . It follows that $L(G) \neq \emptyset$, so N rejects $\langle G \rangle$ and therefore M also rejects. Hence, M decides B . ■

(c) Show that $\{\langle M, q \rangle \mid M \text{ is a TM containing a useless state } q\}$ is **undecidable**.

Solution: We give two solutions. The first solution is a reduction from E_{TM} ; the second solution is a reduction from A_{TM} . The first solution is easier to understand, but the second solution presents a technique that is more powerful and more useful in proving other languages are undecidable. I recommend you read and understand them both.

Let $C = \{\langle M, q \rangle \mid M \text{ is a TM containing a useless state } q\}$ and suppose for a contradiction that some Turing machine M_C decides C . We show how to use M_C to build a Turing machine M that decides E_{TM} ¹ The existence of M yields a contradiction because E_{TM} is undecidable. The contradiction means that no such Turing machine M_C exists, and so C is undecidable.

¹Recall that $E_{\text{TM}} = \{\langle M \rangle \mid M \text{ is a TM and } L(M) = \emptyset\}$.

Given access to a black box M_C that can answer questions of the form “Is this particular state useless in this particular Turing machine?”, we must be able to answer questions of the form “Does this particular Turing machine (call it M_{in}) not accept any strings?”. We translate questions in this form to questions that our black box can answer by asking the black box whether or not the accept state of M_{in} is useless.

M. On input $\langle M_{in} \rangle$:

1. Set q to be equal to the accept state of M_{in} .
2. Simulate M_C on $\langle M_{in}, q \rangle$. If M_C accepts, then accept. Otherwise reject.

Note that if $L(M_{in})$ is empty, then the accept state of M_{in} is useless, and so M_C accepts $\langle M_{in}, q \rangle$, which in turn implies that M accepts $\langle M_{in} \rangle$. Conversely, if $L(M_{in})$ is not empty, then the accept state of M_{in} is not useless, and so M_C rejects $\langle M_{in}, q \rangle$, which in turn implies that M rejects $\langle M_{in} \rangle$. Because M decides E_{TM} and E_{TM} is undecidable, we have a contradiction. The contradiction implies that C is undecidable.

For our second solution, we reduce from A_{TM} , as suggested by the hint. Again, let $C = \{ \langle M, q \rangle \mid M \text{ is a TM containing a useless state } q \}$ and suppose for a contradiction that some Turing machine M_C decides C . We show how to use M_C to build a Turing machine M that decides A_{TM} ². The existence of M yields a contradiction because A_{TM} is undecidable. The contradiction means that no such Turing machine M_C exists, and so C is undecidable.

Given access to a black box M_C that can answer questions of the form “Is this particular state useless in this particular Turing machine?”, we must be able to answer questions of the form “Does this particular Turing machine (called M_{in}) accept this particular string (call it w_{in})?”. This time translating from the question we’d like to answer to the questions that our black box can answer is more difficult. Our strategy is to create a new Turing machine (call it M_{test}) whose accept state is useful if and only if M_{in} accepts w_{in} .

M. On input $\langle M_{in}, w_{in} \rangle$:

1. Construct the following Turing machine, called M_{test} .

M_{test} . On input x :

1. Ignore the input x .
2. Simulate M_{in} on w_{in} . If M_{in} accepts, then accept. Otherwise, reject.
2. Set q to be equal to the accept state of M_{test} .
3. Simulate M_C on $\langle M_{test}, q \rangle$. If M_C rejects, then accept. Otherwise accept.

Let’s observe a few key points. At a high level, what M does is it says, “I am given the description of a Turing machine and a string as input. Let me use this information to construct a brand new Turing machine (called M_{test}), and let me use M_{test} as something I feed to my black box that answers questions about useless states.” At no time does M ever try to run or simulate the machine M_{test} — and that’s a good thing too, because in step (2) of M_{test} ’s code, the simulation may never terminate and so M_{test} may loop forever! Just because M makes a machine does not mean it has to run it.

Note that if M_{in} accepts w_{in} , then M_{test} accepts all strings. Therefore M_{test} ’s accept state is not useless, so M_C rejects in step (2), and M accepts. Conversely, if M_{in} does not accept w_{in} , then M_{test} does not accept any strings. It follows that M_{test} ’s accept

²Recall that $A_{TM} = \{ \langle M, w \rangle \mid M \text{ is a TM and } M \text{ accepts } w. \}$

state is useless, so M_C accepts in step (2), and M rejects. Because M decides A_{TM} and A_{TM} is undecidable, we have a contradiction. The contradiction implies that C is undecidable. ■

Hint: For part (c), show that the halting problem reduces to the given language. Show that, given $\langle M, x \rangle$, one can construct a TM M' such that M accepts x if and only if M' has no useless state.

3. [10 points] (From Sipser.) Let

$$f(x) = \begin{cases} 3x + 1 & x \text{ is odd} \\ x/2 & x \text{ is even} \end{cases}$$

for any natural number x . If you start with an integer x and iterate f , you obtain a sequence $x, f(x), f(f(x)), \dots$. Stop if you ever hit 1. For example, if $x = 17$, you get the sequence 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1. Extensive computer tests have shown that every starting point between 1 and a large integer gives a sequence that ends in 1. But, the question of whether all positive starting points end at 1 is unresolved; it is called the $3x + 1$ problem.

Suppose that $A_{TM} = \{\langle M, w \rangle \mid M \text{ is a TM that accepts } w\}$ were decidable by a TM H . Use H to describe a TM with an output tape that prints the answer to the $3x + 1$ problem.

Solution: We build our solution in stages. Later stages use Turing machines that we construct in earlier stages. The first step is to construct a Turing machine M_{query} that takes an input x and accepts if iterating f starting from x eventually yields 1, and loops forever otherwise.

M_{query} . On input $\langle x \rangle$:

1. If $x = 1$, then accept.
2. If x is odd, update $x \leftarrow 3x + 1$. If x is even, update $x \leftarrow x/2$.
3. Go to step 1.

Our next step is to construct a Turing machine M_{loop} which iterates over all positive integers, looking for a counter-example to the $3x + 1$ conjecture. That is, M_{loop} searches for some x such that iterating f starting from x never reaches 1 and accepts if it finds such an x . To do this, it seems tempting to have M_{loop} simulate M_{query} first on $x = 1$, next on $x = 2$, and so forth. Whenever iterating f on x yields 1, the simulation of M_{query} will eventually end and M_{loop} would then proceed to the next number $x + 1$. But what if M_{loop} actually finds a counter-example x ? In this case, the simulation of M_{query} on x will never terminate, and M_{loop} will be in the unfortunate situation that it has found what it is looking for, but it doesn't know it has found it!

To get around this, we use H . Instead of simulating M_{query} on x , we have M_{loop} check whether or not M_{query} would accept x by passing $\langle M_{\text{query}}, x \rangle$ to H .

M_{loop} . On input $\langle w \rangle$:

1. Ignore the input w .
2. For each natural number $y = 1, 2, \dots$:
 3. Run H on $\langle M_{\text{query}}, y \rangle$.
 4. If H rejects, then accept. Otherwise, continue the loop.

Finally, in order to solve the $3x + 1$ problem, we need to know whether or not M_{loop} finds a counter-example. Again, we might be tempted to simulate M_{loop} and see if it ever finds a counter-example and accepts. The problem is that there may not be any counter-example, in which case M_{loop} will loop forever and our simulation will not terminate. The trick is to use H again to see if M_{loop} finds a counter example.

M_{3x+1} . On input $\langle w \rangle$:

1. Ignore the input w .
2. Run H on $\langle M_{\text{loop}}, \varepsilon \rangle$.
3. If H accepts, then print "There is a counter-example to the $3x + 1$ conjecture." Otherwise, print "The $3x + 1$ conjecture holds."

■

4. [10 points] Give an example of an undecidable subset of $L(1^*)$. Prove that your answer is correct.

Solution: There are two reasonable ways to solve this problem. The first is to use diagonalization. Let M_1, M_2, \dots be a list of all Turing Machines. Since there are only countably many TMs, we know there is such a list (we don't even need to be able to obtain $\langle M_i \rangle$ given i , since we aren't going to show an algorithm for anything).

We define our diagonalizing language as $D = \{1^k \mid 1^k \notin L(M_k)\}$. This is clearly a subset of $L(1^*)$, since it only contains strings of the form 1^k . We also know that D does not equal $L(M_n)$ for any n , because those two sets disagree about whether to include 1^n . Since the list M_1, M_2, \dots was exhaustive, D is not the language of any TM. So not only is D undecidable, it is also unrecognizable. Bonus!

Another solution is to directly construct a language that we can reduce a known undecidable language to. We'll try to design a language that A_{TM} can be reduced to.

One nice idea is to simply make a unary encoding of strings in A_{TM} . How do we do this? Let s_1, s_2, \dots be a lexicographic list of strings in Σ^* . We know that given a string s_n , it is possible to calculate its index n in the lexicographic ordering (if only by running a lexicographic enumerator for Σ^* and counting outputs until it outputs s_n). So we design our undecidable language as $B = \{1^k \mid s_k \in A_{TM}\}$. This is surely a subset of $L(1^*)$. We just need to show a reduction from A_{TM} to B to establish B 's undecidability.

M. On input x :

1. Find the index n such that $x = s_n$. Since $x \in \Sigma^*$, it has such an index.
2. Run the decider for B on input 1^n and agree with its output.

This is a correct decider for A_{TM} , since on input s_n , it checks whether $1^n \in B$, and we defined B in such a way that $1^n \in B$ if and only if $s_n \in A_{TM}$. ■

5. [15 points] Let A be a language of Turing Machine descriptions, say, $A = \{\langle M_1 \rangle, \langle M_2 \rangle, \dots\}$. Suppose that A is Turing-recognizable, and every machine whose description appears in A is a decider. Show that there is a decidable language B that is not decided by any machine whose description appears in A .

In other words, you cannot make an exhaustive list of all the decidable languages in this way.

Hint: Consider an enumerator for A , and let $\langle M_i \rangle$ be the i th machine description output by the enumerator. Use diagonalization to construct B in such a way that B and $L(M_i)$ disagree on input s_i (the i th string in lexicographic order), for every i . Be sure to argue that B is actually decidable!

Solution: As in the hint, let E be an enumerator for A and let M_i be the i th machine whose description is output by E . We set

$$B = \{s_i \mid s_i \notin L(M_i)\}.$$

Observe that B is not the same language as, say, M_{53} because s_{53} is in exactly one of the two languages $B, L(M_{53})$. Because this is true for each i , we know that B is not decided by any machine whose description appears in A .

We still must argue that B is decidable. Given a string s_i , how do we determine if $s_i \in L(M_i)$, or equivalently, if M_i accepts s_i ? Well, first we have to get our hands on the description of M_i ; to do so, we simulate the enumerator E until it outputs $\langle M_i \rangle$. Because M_i is a machine whose description is in A , we know that M_i is a decider, so we can just simulate M_i on s_i to see if $s_i \in L(M_i)$. We construct a decider M_B for B as follows.

M_B . On input w :

1. By enumerating the strings of Σ^* in lexicographical order, find the string s_i such that $w = s_i$.
2. Simulate E until it outputs its i th description $\langle M_i \rangle$.
3. Simulate M_i on s_i .
4. If M_i rejects, then accept. Otherwise reject.

Note: there is one slight complication with this solution. In particular, if A is finite, then we need to make one slight modification to M_B to prevent it from looping forever in step 2. If $|A| = n$, then we want to include a check so that when M_B is run on a string s_i with $i > n$, we have M_B immediately accept without simulating E . (We could have also made M_B reject; the choice is arbitrary.) ■

6. [Honors; 20 points; 10 per part]

- (a) Prove that every infinite Turing-recognizable language has an infinite Turing-decidable subset.

Solution: Enumerators come to our rescue on this problem. Let A be an infinite recognizable language. It must have some enumerator E . We define the following enumerator E' :

E' .

1. Set $n = -1$.
2. Start simulating E . Each time it prints a string y :
 - If $|y| > n$, then print y and set $n = |y|$.
 - Otherwise, do nothing about y .

The variable n stores the length of the most recently printed string by E' . In this way, E' will only print a new string if it is longer than everything it has previously printed, so its output is in lexicographic order. E' will clearly only print strings that are in A , so $L(E') \subseteq A$. Finally, no matter what value n currently has, E will eventually output a string longer than n characters (since A is infinite). So E' will print an infinite number of strings.

We have shown an infinite language $L(E') \subseteq A$. Since this language has a lexicographic enumerator, it is decidable. ■

- (b) Prove or disprove: every infinite language has an infinite Turing-decidable subset.

Solution: This claim is false. To prove it, we want to construct a language B such that for every infinite decidable language A , $A \not\subseteq B$.

From part (a), we know that the language B we construct must necessarily be unrecognizable. So we can define B without any computation in mind (i.e., it can be defined from an “all-knowing” point of view, where we know whether strings are accepted by Turing Machines, we have enumerations of any countable set, etc).

There are only countably many infinite decidable languages, so let A_1, A_2, \dots be a list of them all. We define a sequence of integers $n_0 < n_1 < \dots$ inductively as follows. n_0 is zero, and n_k is defined as follows. Let x_k be the shortest string in A_k that is longer than n_{k-1} characters. Since A_k is infinite, such a string x_k exists.³ We define $n_k = |x_k| + 1$, so that $n_{k-1} < |x_k| < n_k$.

Now I claim that $B = \{1^{n_1}, 1^{n_2}, \dots\}$ is the desired language. Suppose for contradiction that there is an infinite decidable subset of B . This subset must occur somewhere on our list A_1, A_2, \dots , say, as A_i . So $A_i \subseteq B$. Let's look at the string x_i we defined above. It was chosen so that $x_i \in A_i$ and also $n_{i-1} < |x_i| < n_i$. But B does not contain any string whose length is between n_{i-1} and n_i . So $x_i \in A_i - B$, which contradicts $A_i \subseteq B$. ■

³This is where we crucially use the fact that A_k is infinite.