

CS 273: Intro to Theory of Computation, Fall 2007

Problem Set 10 (due Thursday, November 29th)

This homework contains 5 problems, one of which is bonus. It is due 12:30 in class or by noon at Elaine Wilson's office (3229 Siebel).

1. Decidability vs. Turing Recognizable

For each of the following languages, determine if it is decidable, Turing recognizable, or neither, and briefly explain why. Since all decidable languages are recognizable, you need not mention that decidable languages are recognizable.

- (a) The set of encodings of Turing machines that halt after at most 20 steps from a blank tape.
- (b) The set of encodings of Turing machines which halt, leaving at most 20 tape squares full at the end of their computation from a blank tape.
- (c) The set of encodings of Turing machines which halt after at most 20 steps on *all* inputs.
- (d) The set of encodings of Turing machines which halt, and can never move their heads left.

Solution:

Explanations here are slightly longer than what we expect, since we hope you will get a clearer idea of the argument. For example, for (b), saying that “The set is Turing recognizable but not decidable. We can build a TM that recognizes it; this machine will simulate the input TM on a blank tape till it halts and accept iff it halts leaving at most 20 symbols on its tape. Since this machine may not halt, and since halting with some number of characters is close to deciding halting, it is probably not Turing decidable.” is a satisfactory answer. The question is meant to test you on the intuitive understanding and prediction of whether languages are decidable/recognizable, and not to test formal reasoning.

- (a) The set is Turing decidable (and therefore also Turing recognizable). We can build a TM that decides it; this machine will take in an input TM, and simulate it on a blank tape for 20 transitions, and check if it halts. If it does, it accepts, else it rejects.
- (b) The set is Turing recognizable but not Turing decidable. We can build a TM that recognizes it; this machine will simulate the input TM on a blank tape till it halts and accept iff it halts leaving at most 20 symbols on its tape. It is not Turing decidable since we can always modify a TM so that it erases its tape just before accepting; we can hence reduce the halting problem to the current problem by first taking a TM as input, modifying it, and feeding it into a decider for this language.
- (c) This language is Turing decidable. Deciding whether a TM halts in 20 steps on all inputs first seems hard because one has to go through all inputs, which is an infinite set. However, note that a TM in 20 steps can read only the first 20 symbols of the input; hence testing whether the TM accepts all strings of length at most 20 in 20 steps is equivalent. We can build a TM that simulates an input TM on all these inputs for 20 steps.

(d) This set is Turing decidable (assuming input TM starts on blank tape). We can build a Turing machine that decides it: this will take an input TM and simulate it on the blank tape till it moves left or halts. If the TM doesn't halt but keeps moving right, then its behavior is entirely determined by its control state and its current tape symbol. Since there are only a finite number of each, it must either halt or come back to the same control state and tape symbol. In the latter case, we can stop the simulation because it's in an infinite loop.

2. Closure Property

Prove that the class of *Turing-recognizable* languages is closed under the union operation. In other words, if A_1 and A_2 are Turing-recognizable languages, so is $A_1 \cup A_2$.

For the definition for Turing-recognizable languages, refer to Definition 3.5 on page 142 of your textbook. Also see Theorems 1.25 and 4.22 in your textbook to get an idea of how to do the proof. For this proof, suppose you have two Turing machines, M_1 and M_2 that recognize A_1 and A_2 , respectively. Design a Turing machine M that recognizes $A_1 \cup A_2$.

Solution:

Given Turing machines M_1 recognizing A_1 and M_2 recognizing A_2 , let's build a TM M recognizing $L_1 \cup L_2$.

M works as follows:

1. Input w ;
2. $n:=1$;
3. Simulate M_1 on w for n steps; if M_1 accepts, accept and halt.
4. Simulate M_2 on w for n steps; if M_2 accepts, accept and halt.
5. Increment n ; goto 3.

The above code is easily executable by a Turing machine. Note that if neither M_1 nor M_2 accept w , then the TM will not halt.

We claim $L(M) = A_1 \cup A_2$. Take any word $w \in L(M)$. Since w gets accepted, it must be the case that either M_1 or M_2 accepted it in n steps for some n . Hence $w \in A_1 \cup A_2$. Now let $w \in A_i$, where $i = 1$ or $i = 2$. Then A_i accepts w in k steps, for some k . The TM M always completes steps 3 and 4 (for any n) and therefore will reach a stage where $n = k$. Then it will simulate A_i for n steps, which will lead to acceptance of w . Hence $w \in L(M)$. So the claim holds.

We have thus shown that Turing recognizable languages are closed under union.

3. Reduction (easy)

Suppose we know that the language

$$ALL_{CFG} = \{\langle G \rangle \mid G \text{ is a CFG such that } L(M) = \Sigma^*\}$$

is undecidable (you saw a proof of this in Tuesday's lecture and this is proved in Sipser p.197).

Use this fact to show that the language

$$EQREG_{CFG} = \{\langle G, A \rangle \mid G \text{ is a CFG and } A \text{ is a DFA and } L(G) = L(A)\}$$

is undecidable.

Prove this using reductions (by reducing ALL_{CFG} to $EQREG_{CFG}$).

Solution:

Let us show a reduction from ALL_{CFG} to $EQREG_{CFG}$. To show this, we must show that given a decider R for $EQREG_{CFG}$, we can build a decider D for ALL_{CFG} .

The decider D for ALL_{CFG} does the following:

1. Input is $\langle G \rangle$.
2. Construct a DFA A that accepts all strings over the terminal alphabet of G (take A to be the single-state DFA where the single state is both initial and final and loops on any letter to itself, for example).
3. Call the decider R for $EQREG_{CFG}$ on $\langle G, A \rangle$.
 - . If R accepts, accept; if R rejects, reject.

We claim that if R decides $EQREG_{CFG}$, then D decides ALL_{CFG} . If $L(G) = \Sigma^*$, then $L(G) = L(A)$, and hence the decider R should accept $\langle G, A \rangle$. Hence D accepts $\langle G \rangle$. If $L(G) \neq \Sigma^*$, then $L(G) \neq L(A)$, and hence the decider R will reject $\langle G, A \rangle$. Hence D rejects $\langle G \rangle$. Note that D halts on all inputs since R halts on all its inputs.

This proves that ALL_{CFG} reduces to $EQREG_{CFG}$. Since ALL_{CFG} is undecidable, it follows that $EQREG_{CFG}$ is undecidable.

It's good to *verify* whether the direction we have proved is correct, and we have indeed shown that $EQREG_{CFG}$ is undecidable. For this, note that what we have proved is that, given a decider for $EQREG_{CFG}$, we can build a decider for ALL_{CFG} . That is, we have proved that if $EQREG_{CFG}$ is decidable, then ALL_{CFG} is decidable. But we know that ALL_{CFG} is undecidable. Therefore $EQREG_{CFG}$ must be undecidable.

4. Reduction

Show that the following language is undecidable.

$$SUBSET_{TM} = \{\langle M, N \rangle \mid M \text{ and } N \text{ are TMs and } L(M) \subseteq L(N)\}.$$

Hint: Prove that we can reduce A_{TM} to $SUBSET_{TM}$. That is, show that if there exists a decider for $SUBSET_{TM}$, then there must also exist a decider for A_{TM} .

Solution:

We will show that A_{TM} reduces to $SUBSET_{TM}$. Since A_{TM} is undecidable, it follows that $SUBSET_{TM}$ is undecidable. We hence have to show that given a decider for $SUBSET_{TM}$, say R , we can build a decider D for A_{TM} . Recall that $A_{TM} = \{\langle M, w \rangle \mid M \text{ is a TM that accepts } w\}$.

The decider D for A_{TM} works as follows:

1. Input $\langle M, w \rangle$.
2. Construct a TM N_w that accepts precisely the word w .
 - . The TM N_w takes a word x as input, and if $x = w$, it accepts, and otherwise rejects.
3. Feed $\langle N_w, M \rangle$ to R , the decider for $SUBSET_{TM}$.
 - . If R accepts, then accept; if R rejects then reject.

For an input $\langle M, w \rangle$ to D , if $w \in L(M)$, then clearly $L(N_w) \subseteq L(M)$, since N_w accepts $\{w\}$. If $w \notin L(M)$, then $L(M) \not\subseteq L(N_w)$. Hence $w \in L(M)$ iff $L(N_w) \subseteq L(M)$.

Hence D accepts $\langle M, w \rangle$ iff $L(N_w) \subseteq L(M)$ iff $w \in L(M)$. Also, D clearly halts on all its inputs since R is a decider. So D decides A_{TM} , and our claim is proved.

An alternative solution is to reduce the language $E_{TM} = \{\langle M \rangle \mid L(M) = \emptyset\}$, which we know is undecidable, to $SUBSET_{TM}$. Given a decider R for $SUBSET_{TM}$, we can build a decider D for E_{TM} which takes an input TM M , constructs a TM N that accepts no input, calls R on the input $\langle M, N \rangle$, and accepts iff R accepts. D hence accepts $\langle M \rangle$ iff $L(M) \subseteq L(N)$ iff $L(M) \subseteq \emptyset$ iff $L(M) = \emptyset$. Hence D decides E_{TM} . Since we know that E_{TM} is undecidable, $SUBSET_{TM}$ must be undecidable.

5. Decidability (bonus)

Fix an alphabet Σ . For any $n \in \mathbb{N}$

$$L_n = \{\langle M \rangle \mid M \text{ is a Turing machine over } \Sigma \text{ and} \\ M \text{ writes to at most } n \text{ squares on the tape} \\ \text{during the course of the computation from a blank tape.}\}$$

Show that L_n is decidable, for any $n \in \mathbb{N}$.

Note that M need not halt in the computation and yet be in L , as long as it uses fewer than n squares on the tape during the course of its computation. For example, a code for a Turing machine which repeatedly writes 1 to the first cell of the tape is in L_2 , even though it does not halt. A code for a Turing machine which places $n + 1$ characters on the tape and then halts is not in L_n , even though it halts.

Solution:

Any machine that writes at most to the first n squares of the tape, starting from the blank tape, can be in at most a finite number of configurations. If the TM has r states, and the tape alphabet has k symbols, then the number of configurations where the tape is bounded by n is at most $m = k^n \cdot n \cdot r$ (there are k^n possible tape-configurations, there are n places where the head can be, and there are r control-states the TM can be in).

The decider for L_n is simple: It will simulate the input TM M on the blank tape for $m + 1$ steps. If during this simulation, the TM ever goes to the $(n + 1)$ 'th cell, then it rejects $\langle M \rangle$. Else, it accepts $\langle M \rangle$.

Correctness: First, it is clear that if TM M goes to the $(n + 1)$ 'th square in the simulation, then $\langle M \rangle$ does not belong to the language and hence we are correct in rejecting it. Now consider a TM M that writes to the $(n + 1)$ 'th square. Let t be the smallest number such that M writes onto the $(n + 1)$ 'th square on the t^{th} step. We claim that $t \leq m + 1$. Assume the contrary. Let $t > m + 1$. Then for the first $m + 1$ steps, the TM stays within the first n cells. Since there are only m distinct configurations where all but the first n -cells are blank, it follows that in $m + 1$ steps, the TM must repeat a configuration c . But if a TM repeats a configuration, then since it is deterministic, it will loop forever in the future, repeating that configuration again and again. Hence it will never move to the $(n + 1)$ 'th cell, which

contradicts the assumption. Hence our claim $t \leq m + 1$ is correct. Therefore, if M writes onto the $(n + 1)$ 'th cell, it must do so within $m + 1$ steps, and hence our decider will catch M doing this and reject it.

We have thus shown that our decider rejects an input TM iff the input TM writes onto the $(n + 1)$ 'th cell. It is also clear that our algorithm always halts. So, for any n , we have an algorithm that decides L_n .