NAME \_\_\_\_\_\_\_\_Model solutions\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

CS 420/520 Theory of Computation, Spring 2019 at Indiana State University, taught by Jeff Kinne

HW 17 – undecidable problems / reductions

Points - each part is graded as 1, 2/3, 1/3, or 0.

1) If the following is decidable, describe an algorithm to solve it. If it is not decidable, give a proof that it is not decidable (by reducing to an already known undecidable problem). L = {(M, x) | M is a TM that accepts x in at most n2 steps}

Claim: L is decidable.

Proof: We need to give an algorithm that gives the correct answer and always halts. It is enough to give a detailed enough description to convince the person grading the HW ;).

The basic idea is to simulate M running on input x until it has used n2 steps. We do this by adding a counter to keep track of how many steps of computation M has taken; set the counter to 0 before the simulation; increase the counter by 1 each time a step of M(x) is completed.

The question is – can this be done by a TM, and is there a problem if M(x) does not halt?

Suppose M is a 1-tape TM, and we can do our simulation with 2 tapes. One of the tapes is used as normal – it has the input, and is modified according to the simulation of M. The second tape keeps the counter of the # of steps. I claim without proof that a TM can keep a counter and increase it by 1 for each step of the simulation of M. We would need to have a separate set of states in our simulating machine to deal with increasing the counter.

When we have increased the counter we would need to check if it is still <= n2, where n = |x|. This requires computing n and also n2. I claim without proof that this can be done.

Now, what if M(x) does not halt? In this case, our simulation runs for n2 steps and then outputs “reject” because M(x) did not accept within n2 steps.

2) If the following is decidable, describe an algorithm to solve it. If it is not decidable, give a proof that it is not decidable (by reducing to an already known undecidable problem). L = { (M) | M is a TM that accepts at least one string at each input length }

Claim: L is undecidable.

Proof: We give a reduction from the language A\_TM to L. Because A\_TM is undecidable this will show that L is undecidable.

Let (M, w) be an input to A\_TM, and our goal is to determine if M(w) accepts or not. We construct the following TM M’

M’: on input x, simulate M(w) and accept iff M(w) accepts.

Notice that M’ ignores its own input and simply does the same as M(w). The question is, is M’ in L or not? If M(w) accepts, then M’ will accept on every input x; in this case M’ does indeed accept at least one string at each input length, and M’ is in L. If M(w) does not accept (either by rejecting or by never halting), then M’ will not accept any input x; in this case M’ does NOT accept at least one string at each input length, and M’ is not in L.

In all cases, M’ is in L iff M(w) accepts. This completes the reduction – if we had the answer to whether M’ is in L, we would also know whether M(w) halts.

3) If the following is decidable, describe an algorithm to solve it. If it is not decidable, give a proof that it is not decidable (by reducing to an already known undecidable problem). L = { (G, x) | G is a CFG and x is NOT a string generated by G }

Claim: L is decidable.

Proof: Theorem 4.7 of the text gives an algorithm for the following: given a CFG G and a string w, determine if G generates w. This is the opposite of what we want, so we can simply run the algorithm from Theorem 4.7 and output the opposite. Because the algorithm in the text is a deterministic algorithm, we will have the correct answer just by flipping the result from the algorithm in the text.

FWIW, the algorithm in the text has the following basic steps.   
i. Convert CFG into Chomsky Normal form.

ii. List all derivations with at most 2n-1 steps (with n the length of x)

iii. If any of those derivations is x, accept; else reject

The proof of correctness of this algorithm relies on the fact (proved in problem 2.26) that any string generated by a grammar in CNF can be generated in at most 2n-1 steps.

4) Languages that people chose –

5) Sipser (3rd edition) problem 5.30 - using Rice's Theorem to prove languages are undecidable.

Rice’s theorem applies to any property P of TM’s such that the following both hold: (i) infinitely many TM’s satisfy property P and infinitely many TM’s also do not satisfy property P, (ii) if M1 and M2 are TM’s that compute the same language then either both have property P or both do not. The theorem states that any such property is undecidable.

Our task in this problem is to take some given properties (languages) and prove that they have conditions (i) and (ii).

a) INFINITE\_TM = { M | M is a TM and L(M) is an infinite language}

Let us verify both conditions.

(i) If a TM accepts all inputs then its language is infinite (it accepts everything, so definitely infinitely many strings). It is not difficult to see that there are infinitely many TMs that accept all strings. It is also not difficult to see that there are infinitely many TMs that do not accept any strings (and therefore do not accept infinitely many strings).  
 (ii) If two TMs accept the same language, then they either both are in INFINITE\_TM or both are not.

b) { M | M is a TM and 1011 ∈ L(M)}

This is very similar to part a). It is not difficult to see that there are infinitely TMs that both contain 1011 as an accepted string, and infinitely many that don’t. It is also clear that if two TMs accept the same set of strings, either both have this property or both do not.

c) ALLTM ={ M | M is a TM and L(M)=Σ∗}

Very similar to part a).