COMP2270/6270 – Theory of Computation
Eleventh Week

School of Electrical Engineering & Computing
The University of Newcastle

Exercise 1) Consider the language $L = \{<M> : M \text{ accepts at least two strings}\}$.

a) Describe in clear English a Turing machine M that semidecides L.

M generates the strings in $\sum_M{}^*$ in lexicographic order and uses dovetailing to interleave the computation of M on those strings. As soon as two computations accept, M halts and accepts.

b) Suppose we changed the definition of L just a bit. We now consider:

$L' = \{<M> : M \text{ accepts } \textit{exactly } 2 \text{ strings}>$.

Can you tweak the Turing machine you described in part a to semidecide L'?

No. M could discover that two strings are accepted. But it will never know that there aren't any more.

Exercise 2) Consider the language $L = \{<M> : M \text{ accepts the binary encodings of the first three prime numbers}\}$.

a) Describe in clear English a Turing machine M that semidecides L.

On input <M> do:

1. Run M on `10`. If it rejects, loop.
2. If it accepts, run M on `11`. If it rejects, loop.
3. If it accepts, run M on `101`. If it accepts, accept. Else loop.

This procedure will halt and accept iff M accepts the binary encodings of the first three prime numbers. If, on any of those inputs, M either fails to halt or halts and rejects, this procedure will fail to halt.

b) Suppose (contrary to fact, as established by Theorem 19.2) that there were a Turing machine *Oracle* that decided H. Using it, describe in clear English a Turing machine M that decides L.

On input <M> do:

1. Invoke *Oracle*(<M, `10`>).
2. If M would not accept, reject.
3. Invoke *Oracle*(<M, `11`>).
4. If M would not accept, reject.
5. Invoke *Oracle*(<M, `101`>).
6. If M would accept, accept. Else reject.

Exercise 3) Show that the set D (the decidable languages) is closed under:

a) Union
b) Concatenation
c) Kleene star
d) Reverse
e) Intersection

All of these can be done by construction using deciding TMs. (Note that there's no way to do it with grammars, since the existence of an unrestricted grammar that generates some language L does not tell us anything about whether L is in D or not.)

a) Union is straightforward. Given a TM M_1 that decides L_1 and a TM M_2 that decides L_2, we build a TM M_3 to decide $L_1 \cup L_2$ as follows: Initially, let M_3 contain all the states and transitions of both M_1 and M_2. Create a new start state S and add transitions from it to the start states of M_1 and M_2 so that each of them begins in its start state with its read/write head positioned just to the left of the input. The accepting states of M_3 are all the accepting states of M_1 plus all the accepting states of M_2.

b) is a bit tricky. Here it is: If L_1 and L_2 are both in D, then there exist TMs M_1 and M_2 that decide them. From M_1 and M_2, we construct M_3 that decides L_1L_2. Since there is a TM that decides L_3, it is in D.

The tricky part is doing the construction. When we did this for FSMs, we could simply glue the accepting states of M_1 to the start state of M_2 with ε transitions. But that doesn't work now. Consider a machine that enters the state y when it scans off the edge of the input and finds a blank. If we're trying to build M_3 to accept L_1L_2, then there won't be a blank at the end of the first part. But we can't simply assume that that's how M_1 decides it's done. It could finish some other way.

So we need to build M_3 so that it works as follows: M_3 will use three tapes. Given some string w on tape 1, M_3 first nondeterministically guesses the location of the boundary between the first segment (a string from L_1) and the second segment (a string from L_2). It copies the first segment onto tape 2 and the second segment onto tape 3. It then simulates M_1 on tape 2. If M_1 accepts, it simulates M_2 on tape 3. If M_2 accepts, it accepts. If either M_1 or M_2 rejects, that path rejects.

There is a finite number of ways to carve the input string w into two segments. So there is a finite number of branches. Each branch must halt since M_1 and M_2 are deciding machines. So eventually all branches will halt. If at least one accepts, M_3 will accept. Otherwise it will reject.

c) Uses the same idea as b - nondeterminstically guess the points where it 'repeats' and run M on each section. Do this for all possible combinations, and if it is true for one then accept.

d) The new Machine reverses the input that is on the tape, then run M on it

e) Very similar idea to a. Run M1 and M2 on the input, accept only if both m1 and m2 accept.

Exercise 4) If L_1 and L_3 are in D and $L_1 \subseteq L_2 \subseteq L_3$, what can we say about whether L_2 is in D?

L_2 may or may not be in D. Let L_1 be $\varnothing$ and let L_3 be Σ. Both of them are in D. Suppose L_2 is H. Then it is not in D. But now suppose that L_2 is {a}. Then it is in D.

Exercise 5) Let L_1 and L_2 be any two decidable languages. State and prove your answer to each of the following questions:

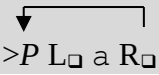
a) Is it necessarily true that L_1 - L_2 is decidable?

Yes. The decidable languages are closed under complement and intersection, so they are closed under difference.

b) Is it possible that $L_1 \cup L_2$ is regular?

Yes. Every regular language is decidable. So Let L_1 and L_2 be {a}. $L_1 \cup L_2$ = {a}, and so is regular.

Exercise 6) Construct a standard one-tape Turing machine *M* to enumerate the language A^nB^n. Assume that *M* starts with its tape equal to ❑. Also assume the existence of the printing subroutine *P*, defined in Section 20.5.1.



Exercise 7) If *w* is an element of {0, 1}*, let ¬*w* be the string that is derived from *w* by replacing every 0 by 1 and every 1 by 0. So, for example, ¬011 = 100. Consider an infinite sequence *S* defined as follows:

S_0 = 0.
$S_{n+1} = S_n \neg S_n$.

The first several elements of *S* are 0, 01, 0110, 01101001, 0110100110010110. Describe a Turing machine *M* to output *S*. Assume that *M* starts with its tape equal to ❑. Also assume the existence of the printing subroutine

P, defined in Section 20.5.1, but now with one small change: if *M* is a multitape machine, *P* will output the value of tape 1. (Hint: use two tapes.)

1. Write 0 on tape 1.
2. Do forever:
   2.1. *P*.
   2.2. Moving from left to right along tape 1, copy ¬tape 1 to tape 2.
   2.3. Moving from left to right, append tape 2 to tape 1.
   2.4. Erase tape 2.

REFERENCES

[1] Elaine Rich, Automata Computatibility and Complexity: Theory and Applications, Pearson, Prentice Hall, 2008.