CS 4510 Automata and Complexity	Section A, Lecture #21
Reductions	
Instructor: Richard Peng	Nov 12, 2018

DISCLAIMER: These notes are not an accurate representation of what I said during the class. They are mostly what I intend to say, and have not been carefully edited.

- Turing-recognizable and Turing-decidable.
- An unrecognizable problem.
- More undecidable problems via reductions.
- Quiz 10.
- Reducing post correspondence to halting.

Last week we showed that the language

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

is undecidable. That is, there does not exist a Turing machine that:

1. Take as input (some encoding of) M and w,
2. Decide whether M accepts w.

Recall that a Turing machine decides a language L if it always terminates, and accepts exactly the strings in L.

The plan today is to build upon this fact to give even more unrecognizable / undecidable languages.

1 A Turing-Unrecognizable Language

An immediate question now we've shown that there are languages that are not Turing-decidable is whether there are languages that are not Turing-recognizable. Recall that a language L is Turing recognizable if there is a Turing machine that accepts exactly the words in L, but can either reject or loop indefinitely on an input that's not in L.

We will show that $\overline{A_{TM}}$, the complement of A_{TM}, is not Turing-recognizable. The proof is by contradiction. Suppose $\overline{A_{TM}}$ is Turing-recognizable by some Turing machine M_2, Recall that A_{TM} is Turing recognizable by some Turing machine M_1. we show that we can decide on A_{TM} by running M_1 and M_2 in parallel.

Given some input w, consider alternating running M_1 and M_2 on some input w, one step each, and return the outcome when either one of them accepts. We claim in either case of $w \in A_{TM}$ or $w \notin A_{TM}$, this process will halt:

1. If w is in A_{TM}, then M_1 accepts w after k steps for some value k, and the overall process halts after $2k$ steps.

2. If w is in $\overline{A_{TM}}$, then M_2 accepts w after k steps for some value k, and the overall process halts after $2k$ steps as well.

Section 4.2 of the textbook extends this to a theorem giving that a language A is decidable if and only if it's both Turing-recognizable and co-Turing-recognizable.

2 Reductions

Note that the key to this proof is to use the fact that A_{TM} is not Turing-decidable. We did not need to examine a supposed Turing machine that recognizes $\overline{A_{TM}}$ and explicitly obtain a contradiction as with showing A_{TM} being undecidable. This is the power of mathematical proofs: we can use the difficulty of one problem to justify the difficulty of another problem.

Formally, we say that a problem A reduces to another problem B if we can solve A by solving another instance of B. Note that this means that if there is an algorithm for solving B, there would be an algorithm for solving problem A.

In other words, if we want to show that a problem B is hard, the reduction approach is to take some other problem A that we known to be hard, and reduce it to B.

One problem that we mentioned, but did not treat formally is the halting problem, which asks whether a Turing machine halts on a given input.

$$A_{Halt} = \{\langle M, w\rangle \mid M \text{ is a TM and } M \text{ either accepts or rejects } w\}.$$

Lemma 2.1. *The halting problem is undecidable.*

We can reduce A_{TM} to A_{HALT} because if M does not halt on w, M, w is not in A_{TM}. Otherwise, we can just simulate M on w until it returns either accept or reject.

Lemma 2.2. *Checking whether the language accepted by a Turing machine is empty/regular/context free is undecidable.*

For this version, note that the empty language is regular (and thus also context free). So to reduce A_{TM} to this problem, consider creating a TM $\widehat{M}$ from M and w such that:

1. $\widehat{M}$ first runs/simulates M on input w.

2. M accepts w, then $\widehat{M}$reads and accepts a non-context-free language, say ww.

3. Otherwise $\widehat{M}$ accepts nothing.

3 Post Correspondence Problem

The plan for this class is to give another undecidable problem that looks very different from the halting problem. This problem underlies the proof of the Cook-Levin theorem which we will discuss after the Thanksgiving break. However, due to the inherent complications of proofs based on Turing machine tape states, these materials are for enrichment purposes only.

Definition 3.1. The Post Correspondence Problem is givens n pairs of strings $a_1 \dots a_n$, $b_1 \dots b_n$, decide whether there is a sequence of (possibly repeating) ids $i_1 \dots i_m$ such that

$$a_{i_1} a_{i_2} \dots a_{i_m} = b_{i_1} b_{i_2} \dots b_{i_m}.$$

The main idea of this proof is to encode the tape content of a Turing machine as the difference, between the two strings and use the agreement of two tapes as reaching an accepting state in the Turing machine.

Once again we will use the special symbol # to denote separation between the tapes in the strings. We force the starting string to be

$$\frac{\#}{\#q_0 w_1 w_2 \dots w_L \#}$$

where the top term is the first string (initialized to be empty), and the bottom is the second string (initialized to the input of the Turing machine).

For this, we need three operations:

1. 'skip' over the part of the tape that does not contain the symbol corresponding to a state. This can be done with $\frac{a}{a}$ for all symbols a.

2. Transit when the state is encountere. Suppose the transition says

   $$\delta(q_1, x) = q_2, y, R$$

   then we should replace $q_1 x$ on the tape with $y q_2$. This can be done with the pairing

   $$\frac{q_1 x}{y q_2}.$$

   The mirror case of moving to the right requires one step look ahead (due to moving the q symbol one to the left), so is slightly more complicated.

3. Terminate once the state reaches q_{accept}. We can simplify this case by requiring the Turing machine to empty its tape before accepting: this is easy because it can just rewind the tape and remove all blanks. Then we simply add the symbol

   $$\frac{q_{accept}\#\#}{q_{accept}\#},$$

   where # is the special symbol denoting the separation between tapes in the string. This is the only way by which we can terminate because the bottom string would always have 1 more # otherwise.

There is one more trick in ensuring that the starting pair with empty above and input below is the only symbol that we can start with. The main idea is to put the two strings slightly 'out of synch' except at the first character via an additional special symbol.