18.404/6.5400 Recitation 6

This recitation covers configurations and the computation history method, with examples.

1 TM Configurations

A configuration of a Turing machine (textbook p.168) completely represents the status of the TM. It consists of the current:
- state,
- location of the head, and
- tape contents.

To be able to write a configuration to a tape, we usually encode it as a string by inserting the state into the tape contents at the head location. In other words, we use $uvq$, where $uv$ is the tape contents, $q$ is the state, and the head is currently at the first symbol of $v$.

1.1 Each configuration can only yield one next configuration

Given a configuration of a deterministic TM, we can always figure out what the next configuration will be. This is because a configuration contains all the inputs needed to calculate the transition function of the TM.

1.2 $A_{\text{LBA}}$ is Decidable

(textbook Theorem 5.9) Let a linear bounded automaton (LBA) be a Turing machine which is not allowed to move its head off its input. Then show that the language

$$A_{\text{LBA}} = \{ \langle M, w \rangle \mid M \text{ is an LBA that accepts string } w \}$$

is decidable.

Proof. The key is to realize that an LBA has a finite number of possible configurations. One way of seeing this is to actually calculate how many there are:

- The number of states is $|Q|$.
- The number of possible head locations is the length of the input; call it $n$.
- The number of possible tape contents is $|\Gamma|^n$, where $\Gamma$ is the tape alphabet.

So the number of possible configurations is exactly $|Q| \cdot n \cdot |\Gamma|^n$. Call this number $K$.

Now what happens if we simulate the LBA $M$ for $K$ steps? If after $K$ steps it has halted, we will know whether it has accepted or not. If it has still not halted, then it has gone through $K + 1$ configurations. But since there are only $K$ unique configurations, $M$ must have repeated a configuration somewhere, which means it must be looping.

Finally, we construct a TM which decides $A_{\text{LBA}}$:

$L = \{ \langle M, w \rangle \mid M \text{ is an LBA and } w \text{ is a string} \}$

1. Simulate $M$ on $w$ for $K$ steps.

2. If it has halted, copy the output of $M$. Otherwise, reject.
2 Computation History Method

Often, our reduction proofs in this class have a step that looks like “Simulate $M$ on $w$.” But what if we are working with a model of computation which can’t simulate Turing machines?

2.1 Computation Histories

Let the computation history of $M$ on $w$ be the sequence of configurations $C_1, C_2, \ldots, C_k$ that $M$ goes through when given input $w$. Note that for a given deterministic $M$ and input string $w$, there exists only one computation history of $M$ on $w$.

2.2 The Idea

If $M$ accepts $w$, then the computation history of $M$ on $w$ will satisfy the following three properties:

1. $C_1$ is of the form $q_0w$, where $q_0$ is the start state.
2. $C_k$ includes an accepting state.
3. For each $i \in [1, k-1]$, $C_i$ yields $C_{i+1}$ according to the transition function of $M$.

The big ideas at play are that:

• An accepting computation history of $M$ on $w$ exists if and only if $M$ accepts $w$, and
• Checking whether a given computation history is a accepting/rejecting computation history of $M$ on $w$ is easier than simulating $M$ on $w$.

2.3 When to use this method

The computation history method is often suitable for problems which:

• Deal with existence. Philosophically, this is because $M$ accepts $w$ if and only if there exists some accepting computation history of $M$ on $w$.
• Involve models of computation which are not as powerful as TMs (and thus can’t simulate them). When simulating $M$ on $w$ is not possible, it’s natural to ask whether one can instead check computation histories of $M$ on $w$. I like to describe this using the pset grader mentality:

  *I might not remember how to solve this problem by myself, but I am 100% absolutely certain that this student’s solution is wrong.*

	Indeed, the examples we’ve seen using the computation history method fall roughly under these criteria: $E_{\text{LBA}}, \text{ALL}_{\text{CFG}}$, etc.

Note: $EQ_{\text{2DIM-DFA}}$ from pset3 problem 5b doesn’t explicitly deal with existence, but we modify it to do so by making one of the 2DIM-DFA’s into one which recognizes the empty language.
2.4  $E_{2WAY-PDA}$ is Undecidable

For a better grasp of how this method is used, we covered the following example:

Let a 2WAY-PDA be a PDA which is allowed to move its head left or right on its input (but still can’t write to the tape, only to its stack). Prove that the language

$$E_{2WAY-PDA} = \{ \langle M \rangle | M \text{ is a 2WAY-PDA and } L(M) = \emptyset \}$$

is undecidable.

**Proof.** Reduce from $A_{TM}$.

Assume for sake of contradiction that we have a decider for $E_{2WAY-PDA}$, call it $D$. Then construct a decider for $A_{TM}$ as follows:

$L = \{ \langle M, w \rangle :$ On input $\langle M, w \rangle$: \}

1. Construct a 2WAY-PDA, $P_{M,w}$, which checks whether its input is an accepting computation history of $M$ on $w$.\footnote{ $P_{M,w}$ does NOT tell us whether $M$ accepts $w$. That is the job of $L$ (via $D$). The task of $P_{M,w}$ is much more limited in scope. Importantly, an accepting computation history of $M$ on $w$ might not exist; $P_{M,w}$ does not “know” this, nor does it care. In this case it will simply reject all of its inputs, oblivious to the bigger picture. It is the result of feeding $\langle P_{M,w} \rangle$ into $D$ which will tell us whether $M$ accepts $w$.} How this is done will be described below.

2. Feed $\langle P_{M,w} \rangle$ into $D$.

3. If $D$ accepts, that means there is no accepting computation history of $M$ on $w$, so $M$ does not accept $w$. In this case reject.

4. Similarly, if $D$ rejects, accept.$^\dagger$

Hopefully the overall logic in this reduction is familiar. The only new component is the structure of $P_{M,w}$, which takes as input a computation history and checks whether it is an accepting computation histories of $M$ on $w$.

The final step is to show that given $M$ and $w$, one can construct a 2WAY-PDA capable of checking computation histories of $M$ on $w$. Recall that to accomplish this, it needs to check the start configuration, the accepting configuration, and the validity of each pair of adjacent configurations. The first two of these are easy: we can hard-code the start configuration, and scan the final configuration until we find an accepting state.

To check that $C_i$ yields $C_{i+1}$ for each $i$, $P_{M,w}$ can push $C_i$ in reverse order onto its stack, then pop the stack one symbol at a time while comparing with each symbol in $C_{i+1}$.

Reversing is necessary because in a stack, symbols are popped in reverse order to how they were pushed. We can’t simply keep track of the entire configuration in finite control, because the tape contents of $M$ could get arbitrarily long. So we must instead compare one symbol at a time. Every symbol in $C_i$ should be the same as $C_{i+1}$ except for the few symbols around the head location: since in one step a TM can only write one symbol and move its head one space, we can fit the transition function check in finite control.

For more detail on how machines check computation histories, see Theorem 5.10 ($E_{LBA}$) and Theorem 5.13 ($ALL_{CFG}$) in the textbook.