# CS 121 Section 7

Harvard University

October 31 - November 2nd, 2013

## 1 Overview

This week we are covering universal TMs, TM encodings, nondeterministic TMs, dovetailing simulation, enumerators, TM algorithms, and high-level descriptions of TMs.

### 1.1 High-level descriptions

Given the ChurchTuring Thesis and representation independence, we no longer need to refer to a specific computing model or or data representation when describing an algorithm. Instead:

- Describe it as a sequence of steps operating on higher-level data types (e.g. numbers, graphs, automata, grammars).
- Each step: simple enough that it is clear it can be implemented on a reasonable model (such as a TM) using a reasonable data representation.
- Freely make use of algorithms we have seen (or are well-known, such as elementary arithmetic) as subroutines.
- Freely make use of control-flow primitives, such as loops, if-then-else, gotos, etc.

### 1.2 Encodings and Universal TMs

We can encode complex data into strings over a small alphabet. e.g. we denote the encoding of a TM $M$ as $\langle M \rangle$.

On input $\langle M, x \rangle$ a universal TM $U$ simulates $M$ when run on input $x$. i.e. it accepts/rejects/'loops' if and only if $M(x)$ accepts/rejects/loops.

### 1.3 Enumerators

A language is enumerable if its elements can be listed by a TM (with the ability to output a list of strings, not just accept/reject).

### 1.4 NTMs and Dovetailing

A nondeterministic TM (like a PDA or NFA) can follow multiple computation paths. It accepts if any of the computation paths accepts.

NTMs are equivalent to TMs. (Every language recognized by a NTM is recognized by a TM and vice versa. What about decidable languages?) We proved this equivalence by dovetailing: we simulate each possible computation path, but we have to be careful not to follow infinitely long paths.

## 2 Excercises

**Exercise 2.1.** *Show that the class of decidable languages is closed under intersection.*

**Solution:** Run $M_1$ then $M_2$ and accept iff both accept. Before you start, copy the input onto a second tape and, before running $M_2$ empty the tape and copy the input back. □

**Exercise 2.2.** *Let $L = \{\langle M \rangle |\ M$ is a DFA and for every string $w$, if $M$ accepts $w$, then $M$ also accepts $w^R\}$. Show that $L$ is decidable.*

**Solution:** Construct a decider $D$ for $L$ as follows. $D(\langle M \rangle)$:

1. Confirms that its input, $\langle M \rangle$, is a valid encoding for a DFA
2. Using the method from PS1, constructs NFA $N^R$ such that $L(M)^R = L(N^R)$
3. Using the subset construction, constructs DFA $M^R$ such that $L(M^R) = L(N^R)$
4. Constructs DFA $\overline{M^R}$ such that $L(\overline{M^R}) = \overline{L(M^R)}$ by switching the accept states and non-accepts states from $M^R$
5. Using the cross-product construction from lecture 4 it constructs DFA $M_\cap$ such that $L(M_\cap) = L(M) \cap L(\overline{M^R})$
6. Checks whether $L(M_\cap) = \emptyset$ by seeing if there is some path from the start state to any accept state. If the language is empty, then $D$ accepts. Otherwise, it rejects.

$D$ decides $L$: First, note that all steps in the construction of $M_\cap$ are guaranteed to take a finite amount of time, and so $M_\cap$ will halt. If $D$ accepts, that means that $L(M_\cap) = \emptyset$. So, there is no $w \in L(M_\cap)$, and so no $w$ in both $L(M)$ and $L(\overline{M^R})$. So, if $M$ accepts $w$, $w \in L(M)$, then $w \notin L(\overline{M^R}) \to w \in L(M^R) \to w \in L(M)^R \to w^R \in L(M)$ and so $M$ accepts $w^R$ as desired.

If $D$ rejects, then there exists some $w \in L(M_\cap) = L(M) \cap L(\overline{M^R})$. So, $M$ accepts $w$, and $w \in L(\overline{M^R}) \to w \notin L(M^R) \to w \notin L(M)^R \to w^R \notin L(M)$, so $M$ doesn't accept $w^R$ as desired. So, $M'$ decides $L$, and $L$ is decidable. □

**Exercise 2.3.** *Show that a language $L$ is decidable if and only if there is an enumerator that outputs the elements of $L$ in lexicographic order.*

**Solution:** First suppose that $L$ is decidable, and let $M$ be a decider for $L$. We construct an enumerator $E$ using the following algorithm. Let the strings in $\Sigma^*$ in lexicographic order be $w_1, w_2, \ldots$.

1. For each $w = w_1, w_2, \ldots$:
   (a) Run $M$ on input $w$.
   (b) If it accepts, emit $w_1$; otherwise, don't emit anything.

We now show that $E$ is an enumerator for $L$. For all $w \in \Sigma^*$, $E$ will eventually run $M$ on $w$ (since $M$ halts on all previous inputs), and $M$ will accept if and only if $w \in L$, so $E$ will emit $w$ if and only if $w \in L$. Also, $E$ outputs strings in lexicographic order since it tries them in lexicographic order.

Second suppose there is an enumerator $E$ that outputs the elements of $L$ in lexicographic order. If $L$ is finite, then we know $L$ is decidable (indeed, it is regular). We now show that if $L$ is infinite, $L$ is decidable. Our decider $M$ uses the following algorithm on input $w$:

1. Run $E$. For each string $w'$ emitted by $E$:
   (a) Check if it's equal to $w$; if so, halt and accept.
   (b) Check it it's lexicographically greater than $w$; if so, halt and reject.
   (c) If neither of those is true, continue.

We now argue that $M$ is a decider for $L$: if $w \in L$, then eventually $E$ will emit $w$, and at that point $M$ will halt and accept; furthermore, since $E$ emits lexicographically, $M$ will not have rejected yet because all strings so far will be lexicographically smaller than $w$. If $w \notin L$, then $E$ will never emit $w$, so $M$ will never accept; furthermore, since there are a finite number of strings lexicographically smaller than $w$, and $L$ is infinite, $E$ will eventually emit some string lexicographically greater than $w$; at this point, $M$ will halt and reject. $\square$

**Exercise 2.4.** *Let $L = \{\langle M \rangle \$ x : M$ only uses the first $|x|$ cells of the tape when run on $x\}$. Show that $L$ is decidable.*

**Solution:** There are only $l = |\Gamma|^{|x|} \cdot |x| \cdot |Q|$ (number of tapes times number of head positions times number of states) different configurations $M$ can be in. So it can only run for $l+1$ steps before either halting or repeating a configuration (pigeonhole principle). So run $M$ for $l+1$ steps and, if it doesn't use more than the first $|x|$ cells in that time, it never will. $\square$