Lecture 12

Ben Rosenberg

June 24, 2021

From last time: Algorithms for converting Turing machines between one that accepts by halting and one that accepts by final state.

Algorithm 1:

Input: Turing machine $M = (Q, \Sigma, \Gamma, \delta, q_0)$ that accepts by halting some language L;

Output: Turing machine $M_1 = (Q_1, \Sigma, \Gamma_1, \delta_1, q_1, F)$ that accepts L by final state

Construction: M_1 is M, but with every state final.

So, $Q_1 = Q, \Gamma_1 = \Gamma, \delta_1 = \delta, q_1 = q$, and $F = Q$.

Algorithm 2:

Input: Turing machine $M_1 = (Q_1, \Sigma, \Gamma_1, \delta_1, q_1, F)$ that accepts L by final state;

Output: Turing machine $M = (Q, \Sigma, \Gamma, \delta, q_0)$ that accepts by halting some language L

Construction idea: Whenever M_1 rejects, let M escape (diverge).

Problem: M_1 may reject (halt in a non-final state, on a bad string).

Whenever M_1 would reject, we will tell M to escape. For every state-symbol pair (p, q), with $p \in Q_1$ and $a \in \Gamma_1$ such that:

1. $[p, a, \_, \_, \_] \notin \delta_1$ (M_1 halts in p looking at a), and
2. $p \notin F$ (M_1 rejects there),

we introduce into δ the tuple $[, a, e, a, R]$ where $e \notin Q_1$ is a new state, and also $[e, \alpha, e, \alpha, R]$ for every $\alpha \in \Gamma$. So, we escape using e.

Then, we have:

- $Q = Q_1 \cup \{e\}$
- $\Gamma_1 = \Gamma$
- $\delta = \delta_1 \cup \{\text{new tuples}\}$

Example: Problem 9: accepts by final state exactly those strings that begin and end with 0

$M = (Q, \Sigma, \Gamma, \delta, q, F)$

$Q = \{q, \}$

$\Sigma = \{0, 1\}$

$\Gamma = \{0, 1, B, \}$

$F = \{\}$

$\delta : [q, 0, p, 0, R]; [p, 0, p, 0, R]; [p, 1, p, 1, R]; [p, B, s, B, L]; [s, 0, f, 0, L]; [s, 1, h, 1, R]; [h, B, h, B, R]$

Now, we want to construct a machine of Problem 9 that accepts by halting.

So, we need to add: $[e, 0, e, 0, R]; [e, 1, e, 1, R]; [e, B, e, B, R]; [q, 1, e, 1, R]; [q, B, e, B, R]; [s, B, e, B, R]; [h, 0, e, 0,$

Example: Problem 8: Turing machine over alphabet $\{0, 1\}$ that accepts by halting those strings that contain 11 as a substring.

$M = (Q, \Sigma, \Gamma, \delta, q)$

$Q = \{q, n, f\}$

$\Sigma = \{0, 1\}$

$\Gamma = \{0, 1, B\}$

$\delta : [q, 0, q, 0, R]; [q, 1, n, 1, R]; [q, B, q, B, R]; [n, 1, f, 1, R]; [n, 0, q, 0, R]; [n, B, q, B, R]$

This accepts by halting $(0 \cup 1)^*11(0 \cup 1)^*$.

We can make this accept by final state by simply putting Q into the position of the set of final states, F.

Definition 1: Language L is **recursively enumerable** if there exists a Turing machine that accepts L.

Here, L can accept by halting or by final state.

Definition 2: Language L is **decidable** (called "recursive" in older books) if there exists a Turing machine that accepts L and halts on every input.

So, decidable $\implies$ recursively enumerable.

Note that in definition 2, L can only accept by final state (not halting since it always halts).

Example: Problem 8: Turing machine over alphabet $\{0, 1\}$ that accepts by halting those strings that contain 11 as a substring.

$M = (Q, \Sigma, \Gamma, \delta, q, Q)$

$Q = \{q, n, f\}$

$\Sigma = \{0, 1\}$

$\Gamma = \{0, 1, B\}$

$\delta : [q, 0, q, 0, R]; [q, 1, n, 1, R]; [q, B, q, B, R]; [n, 1, f, 1, R]; [n, 0, q, 0, R]; [n, B, q, B, R]$

This accepts by halting $L = (0 \cup 1)^*11(0 \cup 1)^*$, and diverges on $\overline{L}$.

Example: Problem 9: accepts by final state exactly those strings that begin and end with 0

$M = (Q, \Sigma, \Gamma, \delta, q, F)$

$Q = \{q, \}$

$\Sigma = \{0, 1\}$

$\Gamma = \{0, 1, B, \}$

$F = \{f\}$

$\delta : [q, 0, p, 0, R]; [p, 0, p, 0, R]; [p, 1, p, 1, R]; [p, B, s, B, L]; [s, 0, f, 0, L]; [s, 1, h, 1, R]; [h, B, h, B, R]$
$[e, 0, e, 0, R]; [e, 1, e, 1, R]; [e, B, e, B, R]; [q, 1, e, 1, R]; [q, B, e, B, R]; [s, B, e, B, R]; [h, 0, e, 0, R]; [h, 1, e, 0, R].$

Accept: $0(0 \cup 1)^* \cup 0$; reject: $1(0 \cup 1)^*$; diverge: $0(0 \cup 1)^*1$.

Both (8) and (9) might be decidable, but we don't know, because by definition there might exist machines that do satisfy the "good behavior" of halting on every input.

Algorithm 3:

Input: DFA $M = (Q, \Sigma, \delta, q_0, F)$ that accepts a language L

Output: Turing machine $M_1 = (Q_1, \Sigma, \Gamma_1, \delta_1, q_1, F)$ that decides L

Construction:

- $Q_1 = Q$
- $\Sigma = \Sigma$
- $\Gamma_1 = \Sigma \cup \{B\}$
- $q_1 = q_0$
- $F = F$
- δ_1 is logically δ. Whenever we have $[p, a, t] \in \delta$, then $[p, a, t, a, R]$ is in δ_1.

In effect, the Turing machine is simulating the DFA.

Why will M_1 halt? Because M, being deterministic, will get to the end of the input, M_1 will hit the blank B, and therefore M_1 must halt since it has no transitions on B in M. (Because $B \notin \Sigma$.)

"How to fail": Let M (the finite automaton) be nondeterministic and try to use the same algorithm. We can't do this because NFAs can get stuck on an state (they can "die") if there isn't an input in δ that supports the transition. A simulator Turing machine would halt and accept even if there were not a transition for it, as long as it was in an accept state.

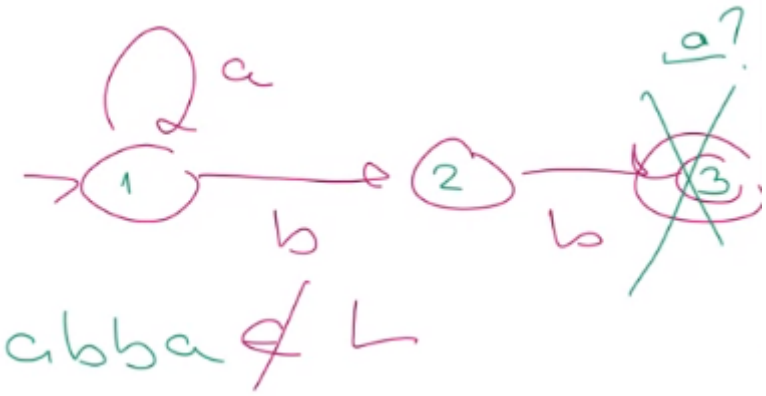

Figure 1: This Turing machine would accept abba when simulating this NFA, even though it shouldn't

In all simulations, we need a pair of simulation algorithms that witnesses equivalence between the standard Turing machine and te "extended" one.

Input: Standard Turing machine M; Output: Equivalent extended Turing machine M

Input: Extended Turing machine M; Output: Equivalent standard Turing machine M

We will omit these algorithms in this class for brevity.

Extended Turing machines

Multiple-track Turing machine For a multiple-track Turing machine, we have $\delta : (Q \times \Gamma_1) \to (Q \times \Gamma_1 \times \{L, R\})$. Here, Γ_1 is composed essentially of 5-letter words, or a 5-tuple of letters.

Turing machine with 2-way infinite tape Idea: turn it into a 2-track machine with a sentinel that says when to switch the track being processed

Multiple-tape Turing machine (different from multi-track because there are multiple heads) Can one Turing machine simulate three Turing machines?

Yes: convert to Multi-track machine; k tapes become $2k$ tracks, as each of the k gets a work track and a helper track.

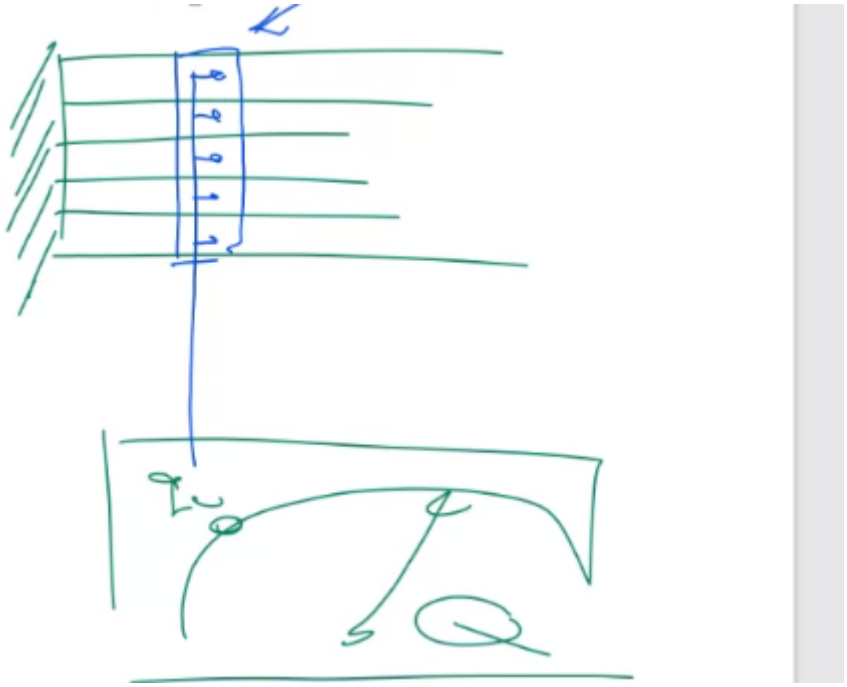

Figure 2: Multi-track Turing machine

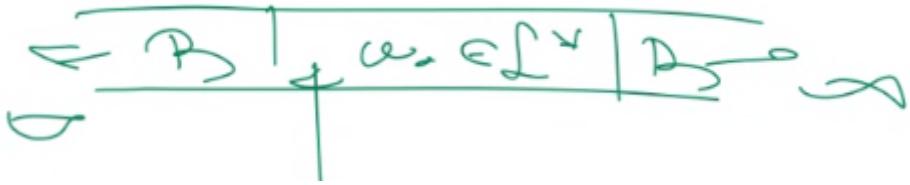

Figure 3: 2-way infinite tape

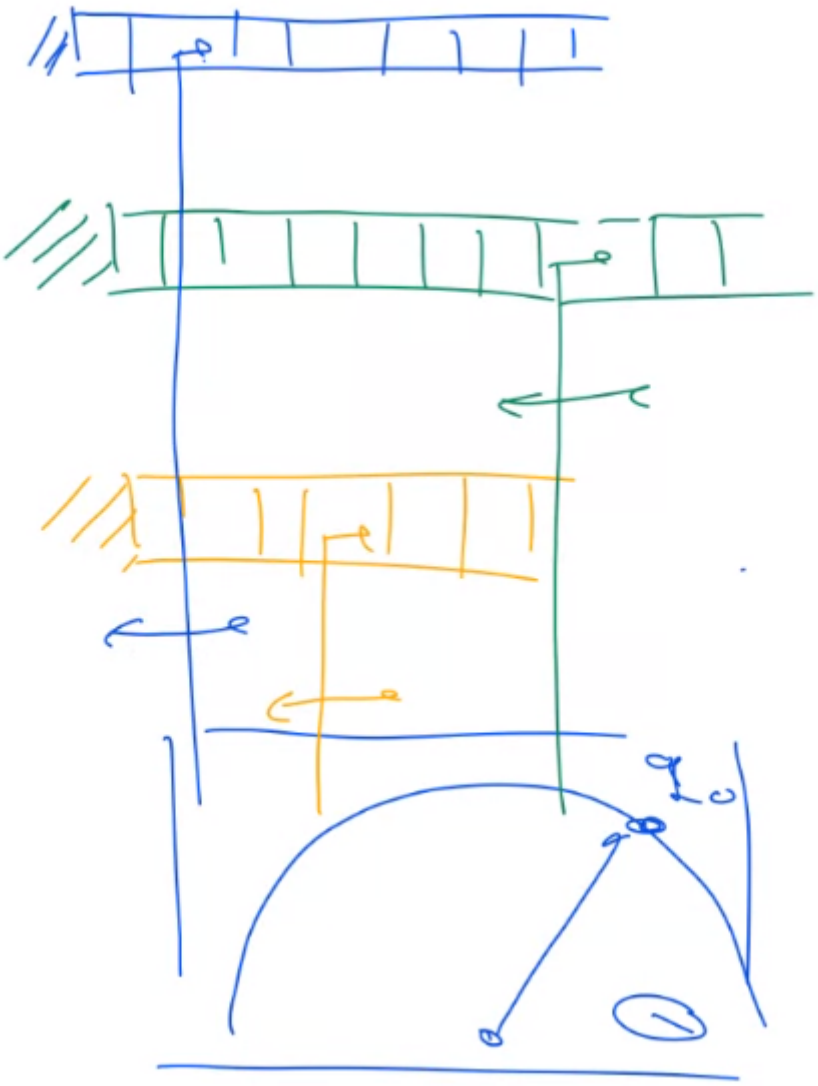

Figure 4: Multi-tape Turing machine

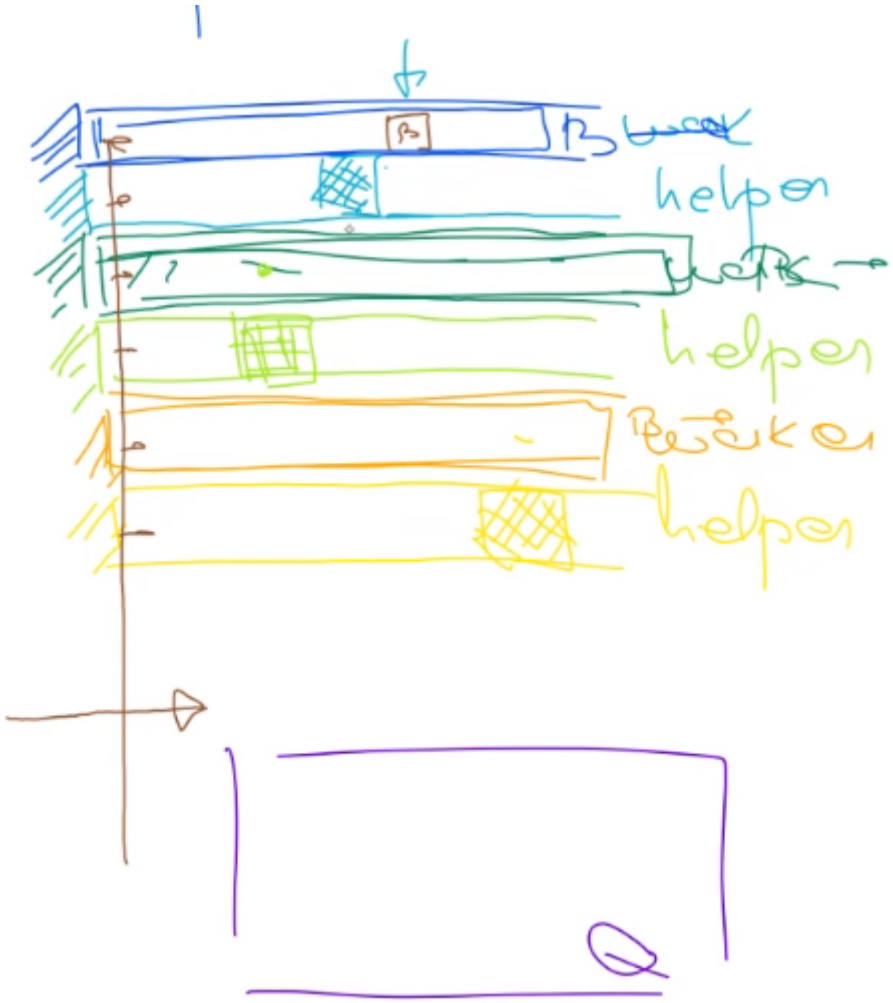

Figure 5: Multi-tape Turing machine

The helper tracks have a sentinel that indicates the current location of the above worker track's head. We erase that sentinel and rewrite it as necessary to simulate the movement of the worker tracks.

Universal Turing machine

Definition 3: $L_H = \{(M, w) | M \text{ is a Turing machine that halts on input } w\}$

Theorem 4: L_H is recursively enumerable.

There exists a **Universal Turing Machine** M_U that operates as follows:

$$M_U(M, w) \begin{cases} \text{halt if} & M(w) \searrow \\ \text{diverge if} & M(w) \nearrow \end{cases}$$

M_U is the prototype of a general *operating system*. M_U is really the only Turing machine that we build; all others are simulated by it.

Construction:

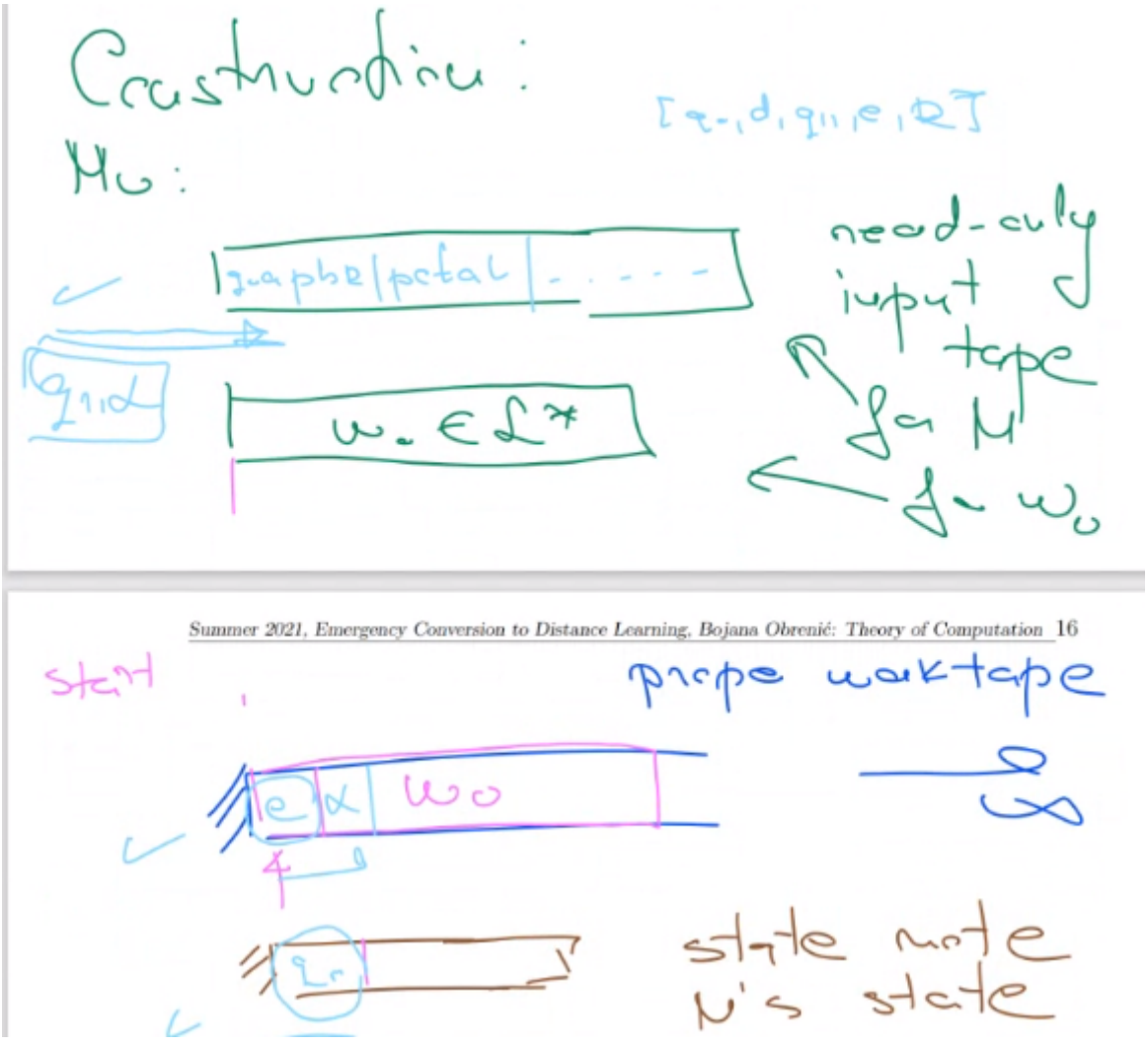

Figure 6: Universal Turing Machine illustration

This goes on forever unless M_U hits a pair (state, symbol) of M for which M has no transition in δ. This means that M would have halted there $\implies$ M_U halts.

Nondeterministic Turing machine

In a nondeterministic Turing machine $M = (Q, \Sigma, \Gamma, \delta, q_0)$, the transition function δ is a *partial* function $\delta : (Q \times \Gamma) \to \mathcal{P}((Q \times \Gamma \times \{L, R\}))$.

Nondeterministic M halts if there exists a computation sequence that leads to halting. Acceptance is similar: if there is a way for it to halt on a final state, we say it will.

Theorem 5: M_U, which is deterministic, simulates non-deterministic Turing machines.

We will upgrade M_U so that it will "discover" halting if it exists if its argument M is nondeterministic.

Upgrade to M_U: What if M_U finds several applicable tuples of M? It should find a way to do them all.

Short idea for M_U: For every M, there exists $k \in \mathbb{N}$ such that M can be written to have either 0 moves or k moves applicable to every (state, symbol) pair. Select k big enough (repeat a move where needed).

Observe that every possible computation of $\mathbb{N}$ is defined by a sequence $n_1 n_2 n_3 \ldots$ where every $1 \leq n_1 \leq k$. Every possible computation corresponds to a sequence whose elements are numbers from $[1, \ldots, k]$.

So, M_U will generate every sequence, in increasing order of length, and run it.

Recall:

$$L_H = \{(M, w) \mid M \text{ is a Turing machine that halts on input } w\}$$

L_H is recursively enumerable, since M_U accepts it:

$$M_U(M, w) \begin{cases} \text{halt if} & M(w) \searrow \\ \text{diverge if} & M(w) \nearrow \end{cases}$$

Let M_H be the Turing Machine that decides L_H:

$$M_H(M, w) \begin{cases} \text{halt and accept if} & M(w) \searrow \\ \text{halt and reject if} & M(w) \nearrow \end{cases}$$

Figure 7: To be covered next time