Lecture #7: Simulations and Turing Machine Variants
Nondeterministic Turing Machines

Lecture #7 introduced a significant variant of Turing machine — a ***multi-tape Turing machine*** — that considerably simplifies the job of showing that languages are Turing-recognizable or Turing-decidable, or that functions are Turing-computable. ***Simulations*** were described that could be used to show that the sets of Turing-recognizable and Turing-decidable languages, and Turing-computable functions, would not be changed if these were defined using multi-tape Turing machines instead of using standard (one-tape) Turing machines.

This document — which is *not* required reading — introduces yet another variant of a Turing machine that accepts, rejects or loops on strings (so that it recognizes a language, and might also decide one), namely, a ***nondeterministic Turing machine***. Simulations are sketched that can be used to prove that the sets of Turing-recognizable and Turing-decidable languages are (once again) unchanged if nondeterministic Turing machines are used instead of standard "deterministic" Turing machines when these are defined.

Definitions

A ***nondeterministic*** Turing machine is the same as a regular Turing machine except that there can be **zero, one, or *many*** moves that might be possible at any time — so that the transition function is now a total function

$$\delta : Q \times \Gamma \to \mathcal{P}(Q \times \Gamma \times \{\mathrm{L}, \mathrm{R}\}).$$

such that $\delta(q_{\text{accept}}, \sigma) = \delta(q_{\text{reject}}, \sigma) = \emptyset$ for every string $\sigma \in \Gamma$ (to ensure that q_{accept} and q_{halt} are both "halting" states).

This is (essentially) the same as the change made to the definition of a DFA in order to define an NFA.

The ***initial configuration*** of a Turing machine, for an input string $\omega \in \Sigma^\star$, is the same as for a standard (deterministic) Turing machine.

A computation of a nondeterministic Turing machine M on an input string $\omega \in \Sigma^\star$ can be represented as a ***tree***:

- The start configuration for ω is at the root.
- Each path down the tree shows one of the possible sequence of configurations that the machine can have when processing ω.

A nondeterministic Turing machine M ...

- ***accepts*** ω if ***at least one accepting configuration*** appears on the computation tree for ω.
- ***rejects*** ω if the computation tree for ω is ***finite*** and there is ***no accepting configuration*** on it.
- ***loops on*** ω otherwise — that is, when the computation tree is infinite and does not include any accepting configuration.

The definition of a Turing machine's ***recognizing*** a language (which depends on the definitions of "accepting", "rejecting" and "looping on" a string) is (otherwise) unchanged. However, the definition of a nondeterministic Turing machine's "deciding" a language is somewhat different: Let us say that a nondeterministic Turing machine M is a ***decider*** if ***all*** branches of the computation tree for M and an input $\omega \in \Sigma^\star$ are finite — so that the entire *computation tree* for ω is finite — for ***every*** input string $\omega \in \Sigma^\star$. We will say that a nondeterministic Turing machine M ***decides***[1] a language $L \subseteq \Sigma^\star$ if and only if

- M is a decider, and
- L is the language of M — that is, the set of strings in $\Sigma^\star$ that M accepts.

An Example

Suppose $\Sigma = \{\mathtt{a}, \mathtt{b}\}$ and let

$$L = \{\omega\omega \mid \omega \in \Sigma^\star\}.$$

Then a nondeterministic Turing machine that recognizes this language, using a tape alphabet

$$\Gamma = \Sigma \cup \{\sqcup\} \cup \{\mathtt{\#}, \mathtt{X}\} = \{\mathtt{a}, \mathtt{b}, \mathtt{\#}, \mathtt{X}, \sqcup\}$$

is as follows.

[1] One could also define "deciding" quite differently: One could simply say that a nondeterministic Turing machine "decides" a language $L \subseteq \Sigma^\star$ if it accepts every string in L and rejects every string in $\Sigma^\star \setminus L$. *This definition would be different* — some Turing machines would "decide" languages according to this definition, when they would not "decide" languages under the definition given first. However, the set of "Turing-decidable" languages" would still not be changed, if this other definition of "decision" was used instead.

On input $\mu \in \Sigma^\star$:

1. If the symbol that is visible on the tape is a then the machine should replace this with #, moving right, and changing to state q_a.

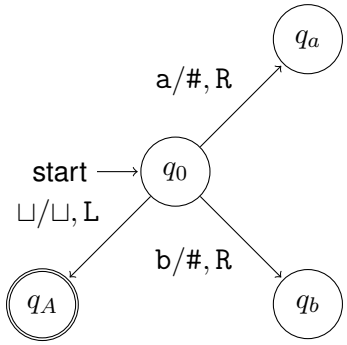
   If the symbol that is visible on the tape is b then the machine should replace this with #, moving right, and then changing to state q_b.

   If the symbol that is visible is $\sqcup$ then the machine should move left without replacing this, and ***accept***.

   Thus

   $$\delta(q_0, \mathtt{a}) = \{(q_a, \#, \mathtt{R})\}, \;\; \delta(q_0, \mathtt{b}) = \{(q_b, \#, \mathtt{R})\}, \;\; \text{and} \;\; \delta(q_0, \sqcup) = \{(q_{\text{accept}}, \sqcup, \mathtt{L})\}.$$

   A state diagram for this part of the machine is as follows.

2. If the machine is in state q_a (so that the input string began with an a) and a b is visible then the machine should sweep past it to the right without changing it and remain in state q_a (so that this step will be repeated).

   If the machine is in state q_a and $\sqcup$ is visible then the machine should sweep past it to the right, without changing it, and ***reject***.

   On the other hand, if the machine is in state q_a and an a is visible then the machine can

   - ***either guess*** that this is the first symbol in the second half of the string — replacing this a with an X, moving left, and changing to state q_L,
   - ***or guess*** that this is not the first symbol in the second half — leaving this a unchanged, moving right past it, and remaining in state q_a.

3. Similarly, if the machine is in state q_b (so that the input string began with a b) and an a is visible then the machine should sweep past it to the right without changing it and remain in state q_b (so that this step will be repeated).

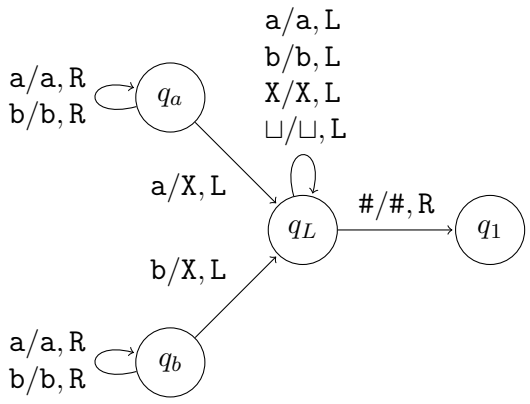
   If the machine is in state q_b and $\sqcup$ is visible then the machine should sweep past it to the right, without changing it, and ***reject***.

On the other hand, if the machine is in state q_b and a b is visible then the machine can

- ***either guess*** that this is the first symbol in the second half of the string — replacing this b with an X, moving left, and changing to state q_L,
- ***or guess*** that this is not the first symbol in the second half — leaving this b unchanged, moving right past it, and remaining in state q_b.

4. When the machine is in state q_L it should sweep left over all symbols except #, without replacing them, moving left. When # is seen then the machine should move right past it, without replacing it, and change to state q_1.

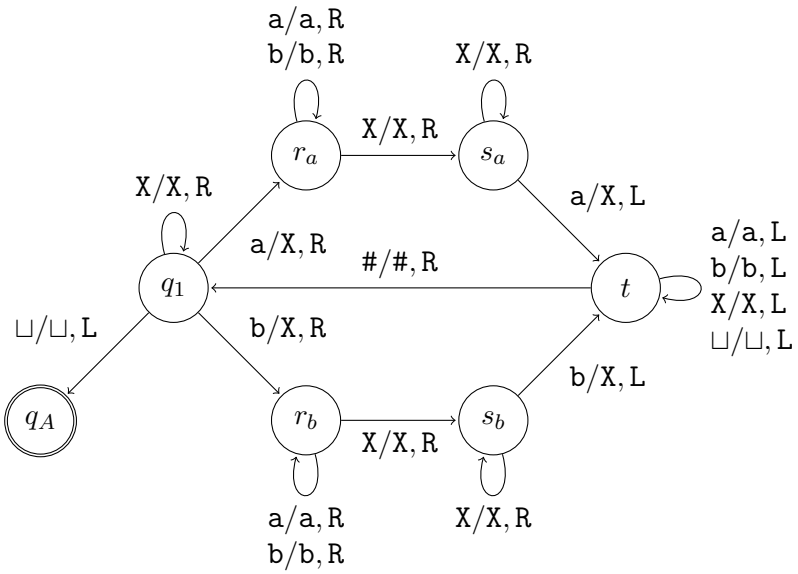
   The state diagram for the part of the machine that has now been described is as follows.



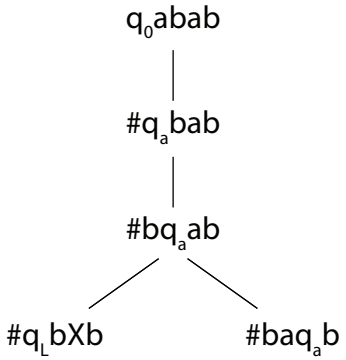
   Note: $\delta(q_a, \mathtt{a}) = \{(q_a, \mathtt{a}, \mathtt{R}), (q_L, \mathtt{X}, \mathtt{L})\}$.

5. Finally, the algorithm should repeatedly do the following.
   (a) Sweep right over any X's without replacing them. If a ␣ is seen before an a or b then ***accept***. Otherwise replace the leftmost remaining a or b with X, using the finite control to remember which of these was replaced;
   (b) Sweep right over remaining a's and b's, without replacing them, until an X is seen.
   (c) Sweep right over remaining X's without replacing them until either an a or b is seen — ***reject*** if ␣ is seen instead. If an a or b is seen and this is different from the symbol that got replaced then ***reject***. Otherwise, replace it with X and move left.
   (d) Continue moving left past symbols without replacing them, until a # is seen. Move back to the right and return to step 5(a).

The state diagram for this part of the machine is as follows.



The computation tree for the string abab begins as follows:



The ***right*** branch of this computation tree continues as follows:

$$\begin{aligned} \#\mathtt{ba}q_a\mathtt{b} &\vdash \#\mathtt{bab}q_a \\ &\vdash \#\mathtt{bab} \sqcup q_{\text{reject}}. \end{aligned}$$

On the other hand, the ***left*** branch of this computation tree continues as follows.

$$\begin{aligned}\#q_L\texttt{bXb} &\vdash q_L\#\texttt{bXb} \vdash \#q_1\texttt{bXb} \vdash \#\texttt{X}r_b\texttt{Xb}\\ &\vdash \#\texttt{XX}s_b\texttt{b} \vdash \#\texttt{X}t\texttt{XX} \vdash \#t\texttt{XXX}\\ &\vdash t\#\texttt{XXX} \vdash \#q_1\texttt{XXX} \vdash \#\texttt{X}q_1\texttt{XX}\\ &\vdash \#\texttt{XX}q_1\texttt{X} \vdash \#\texttt{XXX}q_1 \vdash \#\texttt{XX}q_{\text{accept}}\texttt{X}.\end{aligned}$$

This *this* branch of the computation tree *does* include an accepting configuration, so M ***accepts*** the input string `abab`.

It can be shown — with a bit of work — that this nondeterministic Turing machine is a decider, and that it ***decides*** the above language L.

One Simulation

Claim 1. *Let $L \subseteq \Sigma^\star$ (for an alphabet Σ).*

(a) If L is Turing-recognizable then there exists a nondeterministic Turing machine whose language is L.

(b) If L is Turing-decidable then there exists a nondeterministic Turing machine that decides L.

Proof. Let

$$M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$$

be a deterministic Turing machine M that recognizes a language $L \subseteq \Sigma^\star$. It suffices to note that if we set

$$\widehat{M} = (Q, \Sigma, \Gamma, \widehat{\delta}, q_0, q_{\text{accept}}, q_{\text{reject}})$$

to be a *nondeterministic* Turing machine with the same set Q of states, tape alphabet Γ, and with the same start state, accept state and reject state, then it suffices to define $\widehat{\delta}$ by setting

$$\widehat{\delta}(q, \sigma) = \{\delta(q, \sigma)\}$$

for each state $q \in Q \setminus \{q_{\text{accept}}, q_{\text{reject}}\}$ and every symbol $\sigma \in \Gamma$, and to set

$$\widehat{\delta}(q_{\text{accept}}, \sigma) = \widehat{\delta}(q_{\text{reject}}, \sigma) = \emptyset$$

for every symbol $\sigma \in \Gamma$.. The state diagrams for M and $\widehat{M}$ will look exactly the same. It also easy proved, then (by induction on $|\omega|$) that

$$\widehat{\delta}^\star(q_0, \omega) = \{\delta^\star(q_0, \omega)\}$$

for every string $\omega \in \Sigma^\star$. Since M and $\widehat{M}$ have the same start state, accept state, and reject state, it follows from this that $L(\widehat{M}) = L(M) = L$. Furthermore, $\widehat{M}$ ***decides*** L if M does — as needed to establish the claim. □

Another Simulation

Claim 2. *Let $L \subseteq \Sigma^\star$ (for an alphabet Σ).*

(a) If there exists a nondeterministic Turing machine whose language is L then L is Turing-recognizable.

(b) If there exists a nondeterministic Turing machine that decides L then L is Turing-decidable.

Sketch of Proof. Let $L \subseteq \Sigma^\star$, for an alphabet $\Sigma^\star$. Suppose that there exists a nondeterministic Turing machine

$$M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$$

whose language is L. In order to prove part (a) of the claim it is necessary, and sufficient, to describe a 2-tape (deterministic) Turing machine

$$\widehat{M} = (\widehat{Q}, \Sigma, \widehat{\Gamma}, \widehat{\delta}, \widehat{q}_0, \widehat{q}_{\text{accept}}, \widehat{q}_{\text{reject}}),$$

whose language is L, as well.

A ***simulation*** will be given to show such a Turing machine, $\widehat{M}$, exists: $\widehat{M}$ will simulate *all* possible executions of M on an input string $\omega \in \Sigma^\star$ by performing a ***search*** over M's computation tree on this input string.

Representation of Information: In order to do this, $\widehat{M}$ will need to represent ***configurations*** of M.

$\widehat{M}$'s tape alphabet, $\widehat{\Gamma}$, will therefore include

- All the symbols in M's tape alphabet, Γ;
- all the symbols in M's set Q of states — and it will be assumed that $Q \cap \Gamma = \emptyset$;
- one more symbol, ♠, that is not in either Γ or Q.

$\widehat{M}$ will keep the following information on its tapes.

- *Tape #1:* This initially stores the input string $\omega \in \Sigma^\star$. During this simulation it will store a sequence of ***configurations*** of M, separated by (and ending with) ♠ — and with this tape starting with two ♠'s, so that the left end of the tape can be detected.
- *Tape #2:* This is a "work tape" that will be used to update tape #1.

How Not To Do This: One way to try to simulate M's behaviour is to perform a ***depth-first search*** — simulating computations down branches of the computation tree until each ends.

Unfortunately this will not always work if M is not a "decider" — because it is possible that an infinite branch might get searched — and $\widehat{M}$ will end up "looping on ω" — even though there

is another *finite* branch in M's computation tree that ends with an accepting configuration — so that M *accepts* ω, instead.

How To Do This: Use ***breadth-first search*** instead: After t stages of this simulation all the configurations reachable from the start configuration by making t moves of M will be shown on Tape #1. Since there are only finitely many configurations that are reachable after t moves of M, for any non-negative integer t, this guarantees that an accepting configuration will be discovered, so that the input string will be accepted, whenever it belongs to the language of M.

So — while the simulation process — Tape #1 will store the sequence of all configurations that are at level t in the computation tree for the input string, for some non-negative integer t.

Initialization: Since $\widehat{M}$'s Tape #1 should store the initial configuration when the simulation of the first step is to begin, and since it initially stores the input string, it (only) needs to add three symbols — "♠♠q_0" to the beginning of the non-blank string stored on the tape, and to add one symbol "♠" to the end of it.

The initialization phase should, therefore, write the above three symbols on the leftmost cells of Tape #1 and continue to sweep right, as long as non-blank symbols are visible, shifting the non-blank (initial) contents of the tape three symbols to the right — using the finite control to remember the last three symbols seen (so that they can be written back onto the tape when needed) — adding one more non-blank symbol, "♠", at the end. The tape head should then sweep back to the leftmost cell of the tape without changing the contents of the tape. Since the only place on the tape where two copies of "♠" are next to each other is at the first two cells of the tape, the leftmost cell of the tape is easily identified.

The initialization phase should end with a change of state, so that $\widehat{M}$'s finite control also stores the information that M is in its start state, q_0, when its first step is about to begin.

Simulation of a Step of M: The simulation of a step should begin with the contents of $\widehat{M}$'s Tape #1 being copied onto Tape #2, erasing all but the first two symbols, "♠♠", that are on the first tape — with both tape heads moved back to the third cell on each tape.

Each of the configurations on Tape #2 should now be used to produce zero or more configurations that will be added onto the end of Tape #1. This process will (almost certainly) begin with a sweep to right over the representation of the configuration (which is now on Tape #2) to determine the state, $q \in Q$ that it includes, and the symbol, $\sigma \in \Gamma$, that would be visible on the tape at this point — which can be stored using $\widehat{M}$'s finite control. $\widehat{M}$'s tape head, on Tape #2, could then be moved back to the beginning of the representation of this configuration.

Recall that we are simulating a single ***fixed*** nondeterministic Turing machine M — so that M's possible moves are fixed, and can be used to define $\widehat{M}$'s "finite control" (set of states and transition function). Thus $\widehat{M}$ can include a fixed "submachine" for each state $q \in Q \setminus \{q_{\text{accept}}, q_{\text{reject}}\}$ and for each symbol $\sigma \in \Gamma$ whose transition $\delta(q, \sigma)$ should be applied. Furthermore, the submachine for q and σ will include a fixed (and finite) number of simpler ***submachines*** — one for each 3-tuple (r, τ, D) (for $r \in Q$, $\tau \in \Gamma$ and $D \in \{\mathtt{L}, \mathtt{R}\}$) that is in the *set* $\delta(q, \sigma)$.

Exercise: Figure out what these submachines will do! Consider each of the following cases.

(a) $\delta(q, \sigma)$ includes a 3-tuple $(r, \tau, \mathtt{L})$ for $r \in Q$ and $\tau \in \Gamma$.

(b) $\delta(q, \sigma)$ includes a 3-tuple $(r, \tau, \mathtt{R})$ for $r \in Q$ and $\tau \in \Gamma$.

Note that if the transition $\delta(q, \sigma)$ is now being used then $|\delta(q, \sigma)|$ configurations will be added to the end of Tape #1 to replace the configuration on Tape #2 that is now being processed.

During this process, ***rejecting*** configurations should simply be erased from Tape #2's tape as they are discovered.

If an ***accepting configuration*** is ever discovered then $\widehat{M}$ should ***accept***.

On the other hand, if this has not happened and — at the end of a stage — Tape #1's non-blank portion only stores

♠♠

then $\widehat{M}$ should ***reject*** — because the computation tree was finite, has now been completely searched, and it did not include an accepting configuration.

Otherwise, after all the transitions on Tape #2 have been processed, this tape should be erased (and the tape head should be moved back to the left). $\widehat{M}$' should change state to store M's current state, so that the simulation of the next step can begin.

Now, since M is a fixed nondeterministic Turing machine there exists a positive constant c (depending only on M) such that

$$|\delta(q, \sigma)| \leq c$$

for every state $q \in Q$ and symbol $\sigma \in \Gamma$. This can be used to prove the following (Parts (a) and (b) can be established by induction on t; part (c) is a consequence of parts (a) and (b).)

> ***Claim:*** Let $t \geq 0$. Then, after t stages of the simulation of M, the following properties are satisfied.
>
> (a) The length of each substring representing a configuration of M, on Tape #1, has length at most linear in $\max(n, t)$ if the input string $\omega \in \Sigma^\star$ has length n.
>
> (b) The number of configurations represented on Tape #1 is at most c^t.
>
> (c) The length of the non-blank portion of Tape #1 is at most linear in $\max(n, t) \times c^t$.

This claim can then be used to establish the following one.

> ***Claim:*** The number of steps used by $\widehat{M}$ to carry out the t^{th} stage of the simulation is at most linear in $\max(n, t) \times c^t$.

How To Prove This: After considering the simulation in a bit more detail you should be able to confirm that the number of steps used by $\widehat{M}$ to generate the non-blank portion of Tape #1 is only linear in the length of the non-blank portion of the tape that has been generated. The claim is a reasonably straightforward consequence of this.

Now let $\omega \in \Sigma^\star$ such that M accepts ω. Then the computation tree for ω includes an accepting configuration at some level t. It follows by the above that this configuration will be discovered (if M has not already been accepted before this) during the t^{th} simulation of a "step" of M, and that $\widehat{M}$ will also accept ω.

Suppose that M rejects ω instead: Then the computation tree for ω is finite (so that it as some finite depth, t) and does not include any accepting configuration. After the simulation of $t+1$ "steps" of M — which will require only a finite number of moves of $\widehat{M}$, by the above claims — it will be discovered that $\widehat{M}$'s Tape #1 does not store any configurations at all — and $\widehat{M}$ will also reject ω at this point.

Finally, suppose that M loops on ω — so that the computation tree for ω is infinite and does not include any accepting configurations. Then $\widehat{M}$ will never accept ω (since it must discover an accepting configuration in order to do so) and its execution on ω will never halt, since M's configuration tree has infinitely many levels (and $\widehat{M}$ can only reject ω after discovering that its computation tree has finite depth). Thus $\widehat{M}$ loops on ω too.

It follows by the above that $L(\widehat{M}) = L(M) = L$, as needed to establish part (a) of the claim. Furthermore, if M decides L then $\widehat{M}$ decides L too, as needed to establish part (b). □

Nondeterministic Multi-Tape Turing Machines

It is possible to define nondeterministic k-tape Turing machines too. It should (probably) not be hard for you to imagine how.

The simulation of a k-tape Turing machine by a regular one can be modified — in a reasonably minor way — to produce a simulation of a k-tape nondeterministic Turing machine by a one-tape nondeterministic Turing machine.

Claim 2 can then be applied, to establish that the set of Turing-recognizable languages, and Turing-decidable languages, are not changed if we use nondeterministic multi-tape Turing machines to define these sets.