Computer Science 513

Modelling Computation by Humans: Turing Machines

Instructor: Wayne Eberly

Department of Computer Science
University of Calgary

Lecture #7

Goals for Today

Goals for Today:

- Review of ***Turing machines*** which many (or all) of you will seen before in CPSC 351 or 313 and, possibly, CPSC 413
- Proof that a (partial or total) function $f : (\Sigma^\star)^n \to \Sigma^\star$ is computable, using a Turing machine, if and only if the corresponding function $\widehat{f} : \mathbb{N}^n \to \mathbb{N}$ is computable using an $\mathcal{L}$-program.

Note: While "Turing machines" are introduced in recommended references, I am *not* following these references closely, or even using the version of a "Turing machine" introduced there. I am using a version that is somewhat closer to the version of a Turing machine that you probably saw in CPSC 351 or 313 instead.

Turing Machines: What Turing Did Differently

- The model of computation that we call a “Turing machine” was described (in a somewhat different form) in a paper published in 1937.[1]
- Section 9 of this paper describes how this abstract model was developed — making it clear that Turing proceeded by “modelling computations by humans”.
- In particular, Turing developed this model by considering what people do, to perform computations, using a pencil and paper — simplifying the model by replacing a two-dimensional piece of paper (divided into cells) with a one-dimensional storage tape.

[1] A. M. Turing, “On Computable Numbers, with an Application to the Entscheidungsproblem”, *Proceedings of the London Mathematical Society*, Series 2, Volume 43 (1937), pp. 230–265.

Turing Machines: What Turing Did Differently

- At least some of the models that had already been proposed this had been criticized as being *too* abstract — it was not clear how they were related to “computation” at all.
- While we might criticize this model today as being too “low level” it was easier for others to see, when it was proposed, how it *could* be related to computation — so this model was accepted when others were not (at least, not to the same extent).

Informal Definition of a Turing Machine

A ***Turing machine***

- processes sequences of one or more strings over some ***input alphabet*** Σ like a deterministic finite automaton (DFA) does;
- has a ***finite control*** consisting of a finite set Q of ***states***, like a deterministic finite automaton does; *but*
- also has access to a (two-way) infinite ***tape*** whose *cells* store elements of a larger ***tape alphabet*** Γ:
  - $\Sigma \subseteq \Gamma$;
  - Γ also includes a special symbol, "blank" — which is drawn as $\sqcup$ in these notes; $\sqcup \notin \Sigma$.
  - Γ can include a finite number of other symbols that are not in $\Sigma \cup \{\sqcup\}$ as well.

Informal Definition of a Turing Machine

- A Turing machine also has a "tape head" which points to exactly one cell on the tape at any time.
- At the beginning of an execution of this machine on a strings $\omega_1, \omega_2, \ldots, \omega_n \in \Sigma^\star$, the initial contents of the tape and the location of the tape head are the same as they are for a Post-Turing machine, with input alphabet Σ, when it is executed on the same sequence of inputs.[2]
- The finite control is in a special state $q_0 \in Q$ that is called the ***start state***.
- This choice of initial state and organization of the tape is called the ***start configuration*** (or the ***initial configuration*** of the machine for the inputs $\omega_1, \omega_2, \ldots, \omega_n$.

[2]Most references restrict attention to the case that $n = 1$ — and this the case you probably saw if Turing machines were seen in earlier course.

Informal Definition of a Turing Machine

- The next "move" — or ***transition*** — that the machine can make will generally depend on *both*
  - its current *state* (an element of Q), and
  - the symbol (in Γ) stored at the current location of its tape head.

  When it makes its next move, the machine will
  - write a (possibly different) symbol in Γ onto the cell of the tape that is currently visible;
  - change the finite control to a (possibly different) state in Q;
  - move the location of the tape head one position to the ***left*** or one position to the ***right*** or have it ***stay*** where it is.

Informal Definition of a Turing Machine

- The machine has one more special state — which might be the start state (for a Turing machine that is computing the identity function $f : \Sigma^\star \to \Sigma^\star$) but is generally different from q_0:
  - q_{halt} — the ***halt state***.
- "Transitions" should be defined whenever the machine is a state $q \in Q$ ***except for*** q_{halt}, and for every symbol $\sigma \in \Gamma$ that might be visible on the tape when the move begins.
- Transitions should ***not*** be defined for configurations that include q_{halt}.

Formal Definition of a Turing Machine

A ***Turing machine*** is a 6-tuple,

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

where

1. Q is the (finite and nonempty) set of ***states***;
2. Σ is the (finite and nonempty) ***input alphabet***. Σ *does not* include the ***blank symbol*** $\sqcup$.
3. Γ is the (finite and nonempty) ***tape alphabet***; $\Sigma \subseteq \Gamma$, $\sqcup \in \Gamma$ — and Γ may also include a finite number of additional symbols that are not in $\Sigma \cup \{\sqcup\}$. However, $Q \cap \Gamma = \emptyset$.

Formal Definition of a Turing Machine

4. The ***transition function*** is a *partial* function

$$\delta : Q \times \Gamma \to Q \times \Gamma \times \{\mathrm{L}, \mathrm{R}, \mathrm{S}\}.$$

   - $\delta(q, \sigma)$ should be defined for every state

$$q \in Q \setminus \{q_{\text{halt}}\}$$

     and for every symbol $\sigma \in \Gamma$.
   - $\delta(q_{\text{halt}}, \sigma)$ should not be defined for any symbol $\sigma \in \Gamma$.

5. $q_0 \in Q$ is the ***start state***.
6. $q_{\text{halt}} \in Q$ is the ***halt state***.

Representing Configurations

A ***configuration*** of M provides information about

- the current state of M,
- the contents of the non-blank portion of the tape of M, and
- the location of M's tape head.

Representing Configurations

A configuration is represented by a string

$$\omega_1 \, q \, \omega_2$$

where

- $\omega_1 \in \Gamma^\star$; this string — which does not begin with $\sqcup$ — includes all the symbols on *M*'s tape, starting with the leftmost cell that is to the left of the current location of the tape head and stores a non-blank symbol, and ending with the cell immediately to the left of the location of the tape head — so that $\omega_1 = \lambda$ if there are no non-blank symbols on the tape that are to the left of the tape head;

Representing Configurations

- $q \in Q$ is the current state of M;
- $\omega_2 \in \Gamma^\star$ is a string — not ending with $\sqcup$ — that shows the contents of the remaining part of the tape that is not filled with blanks. This begins with the contents of the cell that the tape head points to and ends with the rightmost non-blank symbol on the tape, if this symbol is not part of the substring ω_1 instead.

  Thus $\omega_2 = \lambda$ if and only if the rightmost non-blank symbol on the tape is currently to the *left* of the current location of the tape head, so that $\sqcup$ is currently visible.

Halting and Non-Halting Configurations

- A configuration $\omega_1\, q\, \omega_2$ is a ***halting configuration*** if $q = q_{\text{halt}}$.
- Otherwise, it is a ***non-halting configuration*** — and the transition function can be used to move to another configuration.

  The details of this are described next.

Applying the Transition Function

Consider a non-halting configuration

$$\omega_1 \, q \, \omega_2$$

Let $\alpha \in \Gamma$ be

- the first symbol in ω_2, if $\omega \neq \lambda$, or
- $\sqcup$, otherwise.

Since $q \neq q_{\text{halt}}$, the transition $\delta(q, \alpha)$ is defined.

Now, either

- $\delta(q, \alpha) = (r, \beta, \mathtt{L})$ for $r \in Q$ and $\beta \in \Gamma$,
- $\delta(q, \alpha) = (r, \beta, \mathtt{R})$ for $r \in Q$ and $\beta \in \Gamma$, or
- $\delta(q, \alpha) = (r, \beta, \mathtt{S})$ for $r \in Q$ and $\beta \in \Gamma$.

These cases are considered separately next.

Applying the Transition Function

Case: $\delta(q, \alpha) = (r, \beta, \mathrm{L})$

- Let $\mu_2 \in \Sigma^\star$ such that $\omega_2 = \alpha\mu_2$ if $\omega_2 \neq \lambda$, and $\mu_2 = \lambda$ if $\omega_2 = \lambda$.
- Suppose that $\omega_1 \neq \lambda$, and suppose that

$$\omega_1 = \mu_1\gamma$$

where $\mu_1 \in \Gamma^\star$ and $\gamma \in \Gamma$.

Then, after the above transition is applied, *M* is in configuration

$$\mu_1\, r\, \gamma\beta\mu_2 \tag{1}$$

— because the machine has replaced a copy of α on the tape with a copy of β and then moved the tape head one position to the left.

Applying the Transition Function

Case: $\delta(q, \alpha) = (r, \beta, \mathtt{L})$

- Once again, let $\mu_2 \in \Sigma^\star$ such that $\omega_2 = \alpha\mu_2$ if $\omega_2 \neq \lambda$, and $\mu_2 = \lambda$ if $\omega_2 = \lambda$.
- If $\omega_1 = \lambda$ — so that the tape head *is* currently pointing to the left of the leftmost non-blank cell of the tape — then, after the above transition is applied, *M* is in configuration

$$r \sqcup \beta\mu_2 \tag{2}$$

— because the machine has replaced a copy of α on the tape with a copy of β, and a $\sqcup$ was stored on the cell immediately to the left of the location of the tape head before this move was made.

Applying the Transition Function

Case: $\delta(q, \alpha) = (r, \beta, \mathtt{R})$

- Once again, let $\mu_2 \in \Sigma^\star$ such that $\omega_2 = \alpha\mu_2$ if $\omega_2 \neq \lambda$, and $\mu_2 = \lambda$ if $\omega_2 = \lambda$.
- Then, after the above transition is applied, *M* is in configuration

$$\omega_1 \beta\, r\, \mu_2 \tag{3}$$

— because the machine has replaced a copy of α on the tape with a copy of β, and then moved its tape head past it to the right.

Applying the Transition Function

Case: $\delta(q, \alpha) = (r, \beta, \mathrm{S})$

- Once again, let $\mu_2 \in \Sigma^\star$ such that $\omega_2 = \alpha\mu_2$ if $\omega_2 \neq \lambda$, and $\mu_2 = \lambda$ if $\omega_2 = \lambda$.
- Then, after the above transition is applied, *M* is in configuration

$$\omega_1\, r\, \beta\, \mu_2 \tag{4}$$

— because the machine has replaced a copy of α on the tape with a copy of β, without changing the location of the tape head.

Applying the Transition Function: One Last Thing To Do

Whoops!

- It is possible that the strings shown at lines (1), (2) (3) or (4), above, start or end with one or more ␣'s.

  If that is the case then all of the initial and trailing ␣'s should be removed, to get a string describing the new configuration of the machine.

Turing Machines: Computation of a Function

- A Turing machine ***computes*** a (partial) function

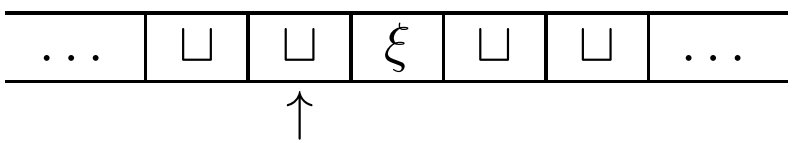
$$f : (\Sigma^\star)^k \to \Sigma^\star$$

if the following conditions are satisfied.

(a) If $\omega_1, \omega_2, \ldots, \omega_k \in \Sigma^\star$ and

$$f(\omega_1, \omega_2, \ldots, \omega_k) = \xi \in \Sigma^\star$$

then execution of the Turing machine (with these inputs) eventually ends with the machine in a halting configuration. When it does so, the tape stores ξ, surrounded with infinitely many blanks on either side, and with the tape head pointing to the blank immediately to the left of ξ:



Turing Machines: Computation of a Function

(b) On the other hand, if $\omega_1, \omega_2, \ldots, \omega_k \in \Sigma^\star$ and $f(\omega_1, \omega_2, \ldots, \omega_k)$ is undefined, then the execution of the Turing machine with these inputs never halts, at all — that is, a halting configuration is never reached.

Turing Machines: Computation of a Function

Definition: A partial or total function $f : (\Sigma^\star)^k \to \Sigma$ is ***Turing computable*** if there exists a Turing machine that computes f (as defined above).

A partial or total function $f : \mathbb{N}^k \to \mathbb{N}$ is ***Turing computable*** if there exists a positive integer n such that if

$$\Sigma = \{\sigma_1, \sigma_2, \ldots, \sigma_n\}$$

is an alphabet with size n then the function $\widehat{f} : (\Sigma^\star)^k \to \Sigma^\star$ is Turing-computable — where $\widehat{f}$ is as follows. For all $x_1, x_2, \ldots, x_k \in \mathbb{N}$, if $\omega_1, \omega_2, \ldots, \omega_k \in \Sigma^\star$ such that $\#(\omega_i) = x_i$ for $1 \leq i \leq k$, then

- if $f(x_1, x_2, \ldots, x_k)$ is defined then $\widehat{f}(\omega_1, \omega_2, \ldots, \omega_k)$ is the string $\mu \in \Sigma^\star$ such that $\#(\mu) = f(x_1, x_2, \ldots, x_k)$, and
- if $f(x_1, x_2, \ldots, x_k)$ is undefined then $\widehat{f}(\omega_1, \omega_2, \ldots, \omega_k)$ is also undefined.

Simulating a Post-Turing Program

Claim #1: Let $f : (\Sigma^\star)^k \to \Sigma$ be a (partial or total) function that is strongly computed by a Post-Turing program $\mathcal{P}$. Then f is Turing-computable.

Method of Proof: Another ***simulation***.

Details of Proof: Let $\mathcal{P} = (\Sigma, \Gamma, \mathcal{C})$ be a Post-Turing function that strongly computes the function f. Since this program "strongly computes" a function, $\Gamma = \Sigma \cup \{\sqcup\}$.

- The claim will be proved by describing a Turing machine

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

such that $\mathcal{M}$ also computes f.

Simulating a Post-Turing Program

- Suppose that $\mathcal{C}$ consists of the sequence of instructions

$$\mathcal{S}_0, \mathcal{S}_1, \ldots, \mathcal{S}_{\ell-1}$$

for some integer $k \geq 1$.

- If $\ell = 0$ — so that $\mathcal{C}$ is the "empty program" — then $k = 1$ and $f : \Sigma^\star \to \Sigma^\star$ is the identity function. It suffices to set

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

where $Q = \{q_0\}$, Σ and Γ are the same as for $\mathcal{P}$, no transitions are defined (that is, $\delta(q_0, \sigma)$ is undefined for all $\sigma \in \Gamma$) and $q_{\text{halt}} = q_0$ in order to obtain a Turing machine that also computes this function.

Simulating a Post-Turing Program

- Suppose, instead, that $\ell \geq 1$.
- Then $\mathcal{M} = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{halt}})$ will be defined to be a Turing machine such that Σ and Γ are the same as for $\mathcal{P}$ (so that, once again, $\Gamma = \Sigma \cup \{\sqcup\}$) and such that

$$Q = \{q_0, q_1, \ldots, q_{\ell-1}, q_{\text{halt}}\}$$

where, for $0 \leq i \leq \ell - 1$, the transitions for moves out of state q_i are designed to simulate an execution of the instruction $\mathcal{S}_i$ of $\mathcal{C}$.

Case: $\mathcal{S}_i$ is "PRINT σ"

- If $0 \leq i \leq \ell - 2$ then

$$\delta(q_i, \tau) = (q_{i+1}, \sigma, \mathrm{S})$$

for all $\tau \in \Gamma$. On the other hand, if $i = \ell - 1$ then

$$\delta(q_i, \tau) = (q_{\mathsf{halt}}, \sigma, \mathrm{S})$$

for all $\tau \in \Gamma$.

Case: $\mathcal{S}_i$ is " IF σ GOTO L"

- Suppose that L is used as the label for at least one statement in $\mathcal{C}$. Let j be the smallest number such that $0 \leq j \leq \ell - 1$ and L labels statement $\mathcal{S}_j$.
- Then
$$\delta(q_i, \sigma) = (q_j, \sigma, \mathrm{S}).$$
- If $0 \leq i \leq \ell - 2$ then
$$\delta(q_i, \tau) = (q_{i+1}, \tau, \mathrm{S})$$
for every symbol $\tau \in \Sigma \setminus \{\sigma\}$. On the other hand, if $i = \ell - 1$ then
$$\delta(q_i, \tau) = (q_{\text{halt}}, \tau, \mathrm{S})$$
for every symbol $\tau \in \Sigma \setminus \{\sigma\}$ instead.

Case: $\mathcal{S}_i$ is "IF σ GOTO L"

- On the other hand, suppose that L is not used as a label for *any* statement in $\mathcal{C}$.
- Then
$$\delta(q_i, \sigma) = (q_{\text{halt}}, \sigma, \mathrm{S}).$$
- If $0 \leq i \leq \ell - 2$ then
$$\delta(q_i, \tau) = (q_{i+1}, \tau, \mathrm{S})$$
for every symbol $\tau \in \Sigma \setminus \{\sigma\}$. On the other hand, if $i = \ell - 1$ then
$$\delta(q_i, \tau) = (q_{\text{halt}}, \tau, \mathrm{S})$$
for every symbol $\tau \in \Sigma \setminus \{\sigma\}$ instead.

Case: $\mathcal{S}_i$ is "RIGHT"

- If $0 \leq i \leq \ell - 2$ then

$$\delta(q_i, \tau) = (q_{i+1}, \tau, \mathrm{R})$$

for all all $\tau \in \Gamma$. On the other hand, if $i = \ell - 1$ then

$$\delta(q_i, \tau) = (q_{\text{halt}}, \tau, \mathrm{R})$$

for all $\tau \in \Gamma$.

Case: $\mathcal{S}_i$ is "LEFT"

- If $0 \leq i \leq \ell - 2$ then

$$\delta(q_i, \tau) = (q_{i+1}, \tau, \mathtt{L})$$

for all all $\tau \in \Gamma$. On the other hand, if $i = \ell - 1$ then

$$\delta(q_i, \tau) = (q_{\text{halt}}, \tau, \mathtt{L})$$

for all $\tau \in \Gamma$.

Simulating a Post-Turing Program

- ***Easy Exercise:*** Confirm that the transition function δ has now been completely defined: $\delta(q, \sigma)$ has been defined (exactly once) for all $q \in Q \setminus \{q_{\text{halt}}\}$ and $\sigma \in \Gamma$, and $\delta(q_{\text{halt}}, \sigma)$ is undefined for all $\sigma \in \Gamma$.

Simulating a Post-Turing Program

- ***Another Reasonably Easy Exercise:*** Let $k \geq 1$, $\omega_1, \omega_2, \ldots, \omega_k \in \Sigma^\star$, and suppose that $\mathcal{P}$ and $\mathcal{M}$ are both executed on a sequence of strings $\omega_1, \omega_2, \ldots, \omega_k \in \Sigma^\star$. Let $i \in \mathbb{N}$ and prove the following (by induction on i):
  - $\mathcal{M}$ has halted, when executed on these inputs, after at most i steps, if and only if $\mathcal{P}$ has.
  - If $\mathcal{M}$ has *not* halted after taking at most $i - 1$ steps, then, after the i^{th} step, $\mathcal{M}$ is in a configuration $\omega_1\, q\, \omega_2$, for $q \in Q$ and $\omega_1, \omega_2 \in \Sigma^\star$ such that
    - After the i^{th} step, the non-blank portion of $\mathcal{P}$'s tape also stores the string $\omega_1\omega_2$ — with $\mathcal{P}$'s tape head pointing to the leftmost symbol in ω_2 (or to the blank immediately to the right of ω_1 if $\omega_2 = \lambda$).
    - If $q = q_j$ for $0 \leq j \leq \ell - 1$ then $\mathcal{S}_j$ is the next instruction in $\mathcal{C}$ to be executed. Otherwise $q = q_{\text{halt}}$ and $\mathcal{M}$ and $\mathcal{P}$ are both about to halt.

Simulating a Post-Turing Program

- It follows (pretty directly), from the result obtained by completing the previous exercise, that if $\mathcal{P}$ strongly computes a function $f : (\Sigma^\star)^k \to \Sigma^\star$ then $\mathcal{M}$ computes the same function — as needed to complete the proof of the claim. □

Simulating a Turing Machine

Clam #2: Let $f : (\Sigma^\star)^k \to \Sigma$ be a (partial or total) function such that f is Turing-computable. Then there exists a Post-Turing program $\mathcal{P}$ that weakly computes this function as well.

Method of Proof; Yet another ***simulation***.

Details of Proof: Let $\mathcal{M} = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{halt}})$ be a Turing machine that computes the function f. A Post-Turing program

$$\mathcal{P} = (\Sigma, \Gamma, \mathcal{C})$$

with the same input and tape alphabets, that weakly computes f, will now be described.

Simulating a Turing Machine

- As in the previous proof there is a trivial case to deal with: If $q_0 = q_{\text{halt}}$ then $k = 1$ and f is the identify function — which is also computed by the Post-Turing program $\mathcal{P}$ whose sequence $\mathcal{C}$ of instructions is empty.
- Suppose next that $q_0 \neq q_{\text{halt}}$. Let $n \geq 1$ and $m \geq 0$ such that

$$\Sigma = \{\sigma_1, \sigma_2, \ldots, \sigma_n\}$$

and

$$\Gamma = \Sigma \cup \{\sqcup\} \cup \{\sigma_{n+1}, \sigma_{n+2}, \ldots, \sigma_{n+m}\}.$$

Simulating a Turing Machine

- Since the names of states do not effect the computation of $\mathcal{M}$ we may now assume without loss of generality that

$$Q = \{q_0, q_1, \ldots, q_{\ell-1}, q_{\text{halt}}\}$$

for some integer $\ell \geq 1$. The Post-Turing program $\mathcal{P}$ that simulates $\mathcal{M}$ will include a sequence of subprograms

$$\mathcal{S}_0, \mathcal{S}_1, \ldots, \mathcal{S}_{\ell-1}$$

such that $\mathcal{S}_i$ simulates the use of transitions out of state q_i, for $0 \leq i \leq \ell - 1$ — as described next.

Simulating a Turing Machine

- ***Final Note:*** In this description of $\mathcal{P}$, doubly-indexed labels $L_{i,j}$ will be used to improve readability. It should not be hard to see that, after this version of $\mathcal{P}$ has been completed, one can go back and replace the labels with labels from the sequence

$$A_1, A_2, A_3, \ldots$$

without changing the function that the program computes.

Simulating a Turing Machine: Simulating Transitions Out of q_i

- For $0 \leq i \leq \ell - 1$, the subprogram $\mathcal{S}_i$ that simulates transitions out of q_i should begin as follows.

$$
\begin{array}{ll}
[L_i] & \text{IF } \sqcup \text{ GOTO } L_{i,0} \\
 & \text{IF } \sigma_1 \text{ GOTO } L_{i,1} \\
 & \text{IF } \sigma_2 \text{ GOTO } L_{i,2} \\
 & \text{IF } \sigma_3 \text{ GOTO } L_{i,3} \\
 & \qquad \vdots \\
 & \text{IF } \sigma_{n+m} \text{ GOTO } L_{i,n+m}
\end{array}
$$

Simulating a Turing Machine: Simulating Transitions Out of q_i

- Note that the statement with label $L_{i,0}$ is reached if and only if $\sqcup$ is visible at the tape at this point, so that the transition $\delta(q_i, \sqcup)$ should be applied.
- Similarly, for $1 \leq j \leq n + m$, the statement with label L_{ij} is reached if and only if σ_j is visible on the tape, so that the transition $\delta(q_i, \sigma_j)$ should be applied next.
- To simplify the rest of the presentation, set σ_0 to be $\sqcup$ and set q_ℓ to be q_{halt}.

Simulating a Turing Machine: Simulating Transitions Out of q_i

- For $0 \leq j \leq n + m$, the subprogram $\mathcal{S}_i$ should continue with a subprogram $S_{i,j}$ that is as follows.
- If $\delta(q_i, \sigma_j) = (q_r, \sigma_s, \mathrm{L})$ for $0 \leq r \leq \ell$ and $0 \leq s \leq n + m$, then $\mathcal{S}_{i,j}$ is the following program.

$$
\begin{array}{ll}
[L_{i,j}] & \text{PRINT } \sigma_s \\
 & \text{LEFT} \\
 & \text{GOTO } L_r
\end{array}
$$

Simulating a Turing Machine

- If $\delta(q_i, \sigma_j) = (q_r, \sigma_s, \mathtt{R})$ for $0 \leq r \leq \ell$ and $0 \leq s \leq n + m$, then $\mathcal{S}_{i,j}$ is the following program.

$$\begin{array}{ll} [L_{i,j}] & \text{PRINT } \sigma_s \\ & \text{RIGHT} \\ & \text{GOTO } L_r \end{array}$$

- Finally, if $\delta(q_i, \sigma_j) = (q_r, \sigma_s, \mathtt{S})$ for $0 \leq r \leq \ell$ and $0 \leq s \leq n + m$, then $\mathcal{S}_{i,j}$ is the following program.

$$\begin{array}{ll} [L_{i,j}] & \text{PRINT } \sigma_s \\ & \text{GOTO } L_r \end{array}$$

Simulating a Turing Machine

- ***Exercise:*** Let $k \geq 1$, $\omega_1, \omega_2, \ldots, \omega_k \in \Sigma^\star$, and suppose that $\mathcal{P}$ and $\mathcal{M}$ are both executed on a sequence of strings $\omega_1, \omega_2, \ldots, \omega_k \in \Sigma^\star$. Let $i \in \mathbb{N}$ and prove the following (by induction on i):
  - $\mathcal{M}$ has halted, when executed on these inputs, after at most i steps, if and only if $\mathcal{P}$ has halted after simulating at most i steps of M.
  - If $\mathcal{M}$ has *not* halted after taking at most $i - 1$ steps, then, after the i^{th} step, $\mathcal{M}$ is in a configuration $\omega_1\, q\, \omega_2$, for $q \in Q$ and $\omega_1, \omega_2 \in \Sigma^\star$ such that
    - After its simulation of the i^{th} step of $\mathcal{M}$, the non-blank portion of $\mathcal{P}$'s tape also stores the string $\omega_1\omega_2$ — with $\mathcal{P}$'s tape head pointing to the leftmost symbol in ω_2 (or to the blank immediately to the right of ω_1 if $\omega_2 = \lambda$).
    - If $q = q_j$ for $0 \leq i \leq \ell - 1$ then the statement with label L_j (at the beginning of subprogram $\mathcal{S}_j$) is the next instruction in $\mathcal{C}$ to be executed. Otherwise $q = q_{\text{halt}}$ and $\mathcal{M}$ and $\mathcal{P}$ are both about to halt.

Simulating a Turing Machine

- It follows (pretty directly), from the result obtained by completing the previous exercise, that if $\mathcal{M}$ computes a function $f : (\Sigma^\star)^k \to \Sigma^\star$ then $\mathcal{P}$ weakly computes the same function — as needed to complete the proof of the claim. □

Turing Machines and Languages

- Turing machines that decide membership in languages replace the halt state q_{halt} with a *pair* of halting states — an ***accept*** state q_{accept} and a ***rejecting*** state q_{reject}.
- This kind of a Turing machine is formally modelled as a 7-tuple

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

  where
  - Q, Σ and Γ are as for Turing machines that compute functions (as defined at the beginning of the lecture);
  - $q_0, q_{\text{accept}}, q_{\text{halt}} \in Q$. It is possible that $q_0 = q_{\text{accept}}$ or that $q_0 = q_{\text{reject}}$ — but $q_{\text{accept}} \neq q_{\text{reject}}$.
  - $\delta : Q \times \Gamma \to Q \times \Gamma \times \{\mathrm{L}, \mathrm{R}, \mathrm{S}\}$ is a partial function, once again. In this case, $\delta(q, \sigma)$ is defined whenever $q \in Q \setminus \{q_{\text{accept}}, q_{\text{reject}}\}$ and $\sigma \in \Gamma$ — and $\delta(q_{\text{accept}}, \sigma)$ and $\delta(q_{\text{reject}}, \sigma)$ are both undefined for all $\sigma \in \Gamma$.

Turing Machines and Languages

- Now let $L \subseteq \Sigma^\star$. A Turing machine $\mathcal{M}$, with input alphabet Σ, ***recognizes*** the language L if the following properties are satisfied for all $\omega \in \Sigma^\star$:
  - If $\omega \in L$ then an execution of $\mathcal{M}$ on the input ω eventually halts in state q_{accept}.
  - If $\omega \notin L$ then either an execution of $\mathcal{M}$ with input ω eventually halts in state q_{reject}, or this execution of $\mathcal{M}$ never halts at all.
- The above Turing machine $\mathcal{M}$ ***decides*** the language L if $\mathcal{M}$ recognizes L, as defined above and, furthermore, the execution of $\mathcal{M}$ on input ω eventually halts, for *all* $\omega \in \Sigma^\star$.

Turing Machines and Languages

- A language $L \subseteq \Sigma^\star$ is said to be ***Turing-recognizable*** (or just ***recognizable*** if there exists a Turing machine $\mathcal{M}$ that recognizes L. L is said to be ***Turing-decidable*** (or just ***decidable***) if there exists a Turing machine $\mathcal{M}$ that decides L.

Theorem: Let $L \subseteq \Sigma^\star$ for an alphabet Σ. Then L is $\mathcal{S}$-decidable if and only if L is Turing-decidable.

Proof: Exercise. □

Variants of Turing Machines

You are, ideally, already familiar with a variety of ***variants*** of the model of computation that has been introduced in this lecture:

- A Turing machine whose tape is ***one-way infinite*** (so that the tape has a "leftmost cell".
- A ***multi-tape*** Turing machine — which has k tapes, for a positive integer k, instead of a single tape.
- Turing machines — which have languages (instead of computing functions) — that are ***nondeterministic*** instead of ***deterministic***.

The supplemental document for this lecture reviews these variants and sketches proofs of the equivalence of these models.