Computer Science 351

Introduction to Turing Machines

Instructor: Wayne Eberly

Department of Computer Science
University of Calgary

Lecture #10

Goals for Today

- Introduction to a more powerful kind of automaton — a ***Turing machine***
- We will see that this kind of machine is essentially as powerful as the CPU and memory of a real computer — it can perform any computation on strings of symbols that this part of a real computer can.

Informal Definition of a Turing Machine

A ***Turing machine***

- processes strings over some ***input alphabet*** Σ like a DFA does;
- has a ***finite control*** consisting of a finite set Q of ***states***, like a DFA does; *but*
- also has access to a (one-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 *Introduction to the Theory of Computation* and 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 string

$$\omega = \sigma_1 \sigma_2 \ldots \sigma_n \in \Sigma^\star,$$

  - the finite control is in a special state $q_0 \in Q$ that is called the ***start state*** (like a DFA's control is);
  - the first $n = |\omega|$ cells of the tape store the symbols $\sigma_1, \sigma_2, \ldots, \sigma_n$ in the string ω (in order);
  - every other cell of the tape stores the "blank" symbol $\sqcup$;
  - the tape head is pointing to the leftmost cell on the tape.

  This is called the ***start configuration*** (or the ***initial configuration*** of the machine for the input ω.

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, either one position to the ***left*** or one position to the ***right***.

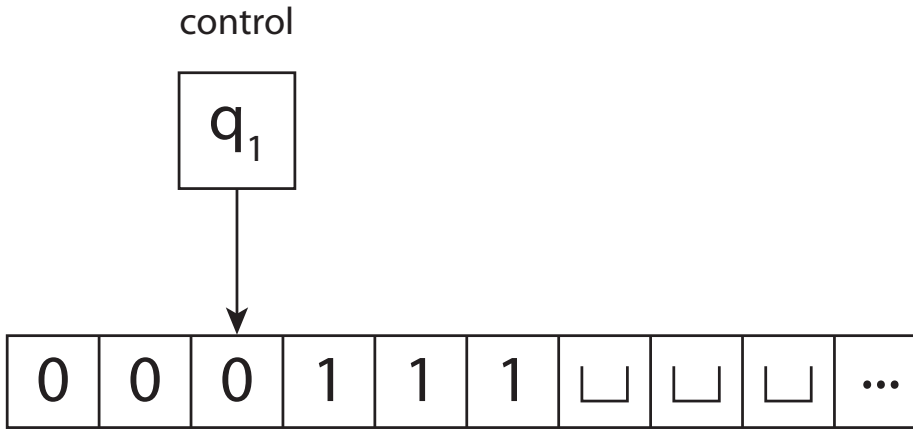
  Exception: If the tape head is already at the leftmost cell and the move would (otherwise) be to the ***left*** then the location of the tape head does not change.

Informal Definition of a Turing Machine

Suppose, for example, that a machine is executed on an input `110111`, and that after two moves, it has changed state to q_1.

Suppose that the tape head moved one position to the right during each move, and that it wrote a `0` before moving the tape head each time.

Then, after these moves, the machine's configuration looks like this:



Informal Definition of a Turing Machine

- The machine has two other special states — both of which might be q_0, but which cannot be the *same* state:
  - q_{accept} — the ***accept state***, and
  - q_{reject}— the ***reject state***.

  Note: In some pictures I will draw these as q_A and q_R, respectively, just to make the pictures simpler.
- Transitions should be defined whenever the machine is a state $q \in Q$ ***except for*** either q_{accept} or q_{reject}, 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 either q_{accept} or q_{reject} — because these are ***halting configurations***.

Formal Definition of a Turing Machine

A ***Turing machine*** is a 7-tuple,

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

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

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

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

$$q \in Q \setminus \{q_{\text{accept}}, q_{\text{reject}}\}$$

     and for every symbol $\sigma \in \Gamma$.
   - Neither $\delta(q_{\text{accept}}, \sigma)$ nor $\delta(q_{\text{reject}}, \sigma)$ should be defined for any symbol $\sigma \in \Gamma$.

5. $q_0 \in Q$ is the ***start state***.
6. $q_{\text{accept}} \in Q$ is the ***accept state***.
7. $q_{\text{reject}} \in Q \setminus \{q_{\text{accept}}\}$ is the ***reject 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.

A configuration is represented by a string

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

where

- $\omega_1 \in \Gamma^\star$; this string includes all the symbols on M's tape, starting with the leftmost cell and ending with the cell ***immediately to the left of the location of the tape head*** — so that $\omega_1 = \lambda$ if the tape head is currently pointing to the leftmost cell;

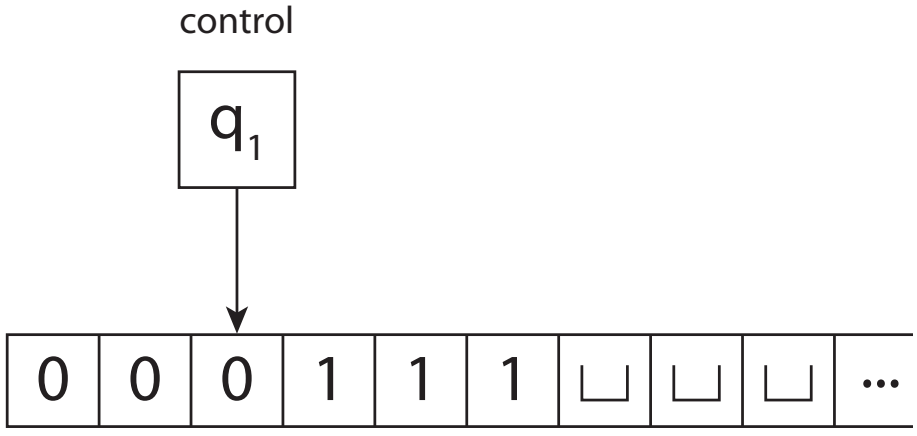
Representing Configurations

- $q \in Q$ 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, and all cells to the right of the location of the tape head store $\sqcup$ too.

Representing Configurations

For example, if M is currently as follows,



then this ***configuration*** of M is represented by the string

$$00\, q_1\, 0111$$

Halting and Non-Halting Configurations

- A configuration $\omega_1\, q\, \omega_2$ is a ***halting configuration*** if $q \in \{q_{\text{accept}}, q_{\text{reject}}\}$. In particular,
  - it is an ***accepting configuration*** if $q = q_{\text{accept}}$ and
  - it is a ***rejecting configuration*** if $q = q_{\text{reject}}$.
- 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 $q \neq \{q_{\text{accept}}, q_{\text{reject}}\}$, the transition $\delta(q, \alpha)$ is defined.

Now, either

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

These cases are considered separately next.

Applying the Transition Function

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

- 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 $\omega_1 \neq \lambda$ — so that the tape head is not currently pointing to the leftmost cell of the tape — and suppose that

  $$\omega_1 = \mu_1\gamma$$

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

  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[1] on the tape with a copy of β and then moved the tape head one position to the left.

[1] or a copy of $\sqcup$ if $\omega_2 = \lambda$

Applying the Transition Function

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

- 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 leftmost cell of the tape — then, after the above transition is applied, *M* is in configuration

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

— because the machine has replaced a copy of α[2] on the tape with a copy of β. Since the tape head could not be moved to the left the position of the tape head did not change.

[2] or a copy of $\sqcup$, if $\omega_2 = \lambda$

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[3] on the tape with a copy of β, and then moved its tape head past it to the right.

[3] or a copy of $\sqcup$, if $\omega_2 = \lambda$

Applying the Transition Function: One Last Thing To Do

Whoops!

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

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

Applying the Transition Function: Notation

Let $\mu_1, \mu_2, \omega_1, \omega_2 \in \Gamma^\star$, such that neither μ_2 nor ω_2 ends with $\sqcup$, and let $q, r \in Q$ — so that both of the strings

$$\mu_1\, q\, \mu_2 \qquad \text{and} \qquad \omega_1\, r\, \omega_2$$

represent configurations of M. In these notes,

- $\mu_1\, q\, \mu_2 \vdash \omega_1\, r\, \omega_2$ — "$\mu_1 q \mu_2$ ***yields*** $\omega_1 r \omega_2$" — means that it is possible to go from the configuration of M represented by $\mu_1\, q\, \mu_2$ to the configuration represented by $\omega_1\, r\, \omega_2$, using ***one*** move of M, as defined above, and
- $\mu_1\, q\, \mu_2 \vdash^\star \omega_1\, r\, \omega_2$ — "$\mu_1 q \mu_2$ ***derives*** $\omega_1 r \omega_2$" — means that it is possible to go from the configuration of M represented by $\mu_1\, q\, \mu_2$ to the configuration represented by $\omega_1\, r\, \omega_2$, using ***zero or more*** (but only finitely many) moves of M, as defined above.

Accepting, Rejecting, and Looping

Suppose, now, that $\omega \in \Sigma^\star$. As described above, The ***start configuration*** for ω is represented by the string $q_0\,\omega$.

- *M* ***accepts*** ω if $q_0\omega \vdash^\star \mu_1\, q_{\text{accept}}\, \mu_2$ for some strings $\mu_1, \mu_2 \in \Gamma^\star$. That is, *M* ***accepts*** ω if and only if one can go from the start configuration for ω to an accepting configuration, using a finite number of moves of *M*.
- *M* ***rejects*** ω if $q_0\omega \vdash^\star \mu_1\, q_{\text{reject}}\, \mu_2$ for some strings $\mu_1, \mu_2 \in \Gamma^\star$. That is, *M* ***rejects*** ω if and only if one can go from the start configuration for ω to a rejecting configuration, using only a finite number of moves of *M*.

Accepting, Rejecting, and Looping

These are not the only things that might happen!

- *M* ***loops on*** ω if and only if *M* neither *accepts* nor *rejects* ω, as defined above. That is, *M* ***loops on*** ω if and only if it is not possible to go from the start configuration of ω to either an accepting or rejecting configuration using a finite number of moves of *M*.

Turing Machines: A First Example

Suppose that $\Sigma = \{0, 1\}$, and that we would like to construct a Turing machine that ***accepts*** all of the strings in the language

$$L = \{0^n 1^n \mid n \geq 0\}$$

and that ***rejects*** all the other strings in $\Sigma^\star$.

Turing Machines: A First Example

This can be accomplished using a Turing machine that repeatedly sweeps over an input string, replacing the leftmost remaining `0` with an `X` and replacing the leftmost remaining `1` with a `Y`, until one of the following things happens.

1. It is discovered that the input is not a sequence of `0`'s followed by a sequence of `1`'s — at which point the input string can be ***rejected***; or
2. The input string *is* a sequence of `0`'s followed by a sequence of `1`'s, but the number of `0`'s is not the same as the number of `1`'s, so that it should be ***rejected***; or
3. The input string belongs to the above language L, so that it should be ***accepted***.

Turing Machines: A First Example

This can be carried out using a Turing machine M with tape alphabet $\Gamma = \{\mathtt{0}, \mathtt{1}, \mathtt{X}, \mathtt{Y}, \sqcup\}$ and that implements the following.

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

1. Read the symbol that is now visible, in search of a $\mathtt{0}$ to be matched.
   (a) If this symbol is $\sqcup$ then ***accept*** (because $\omega = \lambda$).
   (b) If this symbol is either $\mathtt{1}$ or $\mathtt{X}$ then ***reject***: Either the input does not consist of a sequence of $\mathtt{0}$'s followed by a sequence of $\mathtt{1}$'s, or it has too many $\mathtt{1}$'s.
   (c) If the symbol is $\mathtt{0}$ then replace this with an $\mathtt{X}$, move to the right, and go to step 2.
   (d) Finally, if the symbol is $\mathtt{Y}$ then move past it to the right, without replacing it, and go to step 4.

Turing Machines: A First Example

2. At this point, a copy of `0` has been crossed out — and a matching copy of `1` must be found and crossed out too.

   Sweep to the right past any `0`'s and `Y`'s that you see (without changing them) until you see some other symbol on the tape.

   (a) If this symbol is `1` then replace it with a `Y`, moving to the left, and go to step 3.

   (b) if this symbol is anything else then either the input did not consist of a sequence of `0`'s followed by a sequence of `1`'s at all, or there were not enough `1`'s: ***reject***.

Turing Machines: A First Example

3. A matching copy of `1` has now been found and crossed out: We need to reposition the tape head, so that we can repeat this process.

   Sweep back to the left over any `Y`'s (corresponding to `1`'s that have already been matched) or `0`'s (that have not been matched yet), without replacing them, until some other symbol is found.

   (a) If this symbol is an `X` then move the tape head back one position to the right without replacing this symbol and return to step 1.

   (b) Otherwise ***reject***[4].

[4] It is actually impossible for any other symbols to be seen, because of what has happened before. However, this rule makes it easier to complete the definition of the transition function in the next stage of this process.

Turing Machines: A First Example

4. The fact that a `Y` has now been seen, instead of a copy of `0`, means that we have run out of `0`'s at the beginning this string to be matched. With that noted, sweep to the right over all other `Y`'s, without replacing them, until a different symbol is seen.
   (a) If this symbol is a ␣ then ***accept***.
   (b) Otherwise ***reject***.

Turing Machines: A First Example

With that noted, let us continue to produce a Turing machine that implements the above algorithm.

This Turing machine will have states

$$Q = \{q_0, q_1, q_2, q_3, q_{\text{accept}}, q_{\text{reject}}\}$$

where the following is true.

- The machine is in state q_0 when step 1 of the above process is being carried out. Note that since the above process begins with this step, q_0 really is the ***start state***.
- The machine is in state q_1 when step 2 of the above process is being carried out.

Turing Machines: A First Example

- The machine is in state q_2 when step 3 of the above process is being carried out.
- The machine is in state q_3 when step 4 of the above process is being carried out.
- q_{accept} is the ***accept state***.
- q_{reject} is the ***reject state***.

Turing Machines: A First Example

This information can be used to produce the following ***transition table*** that corresponds to the Turing machine being developed. In this table q_{accept} has been rewritten as q_A and q_{reject} has been rewritten as q_R in order to make everything fit in.

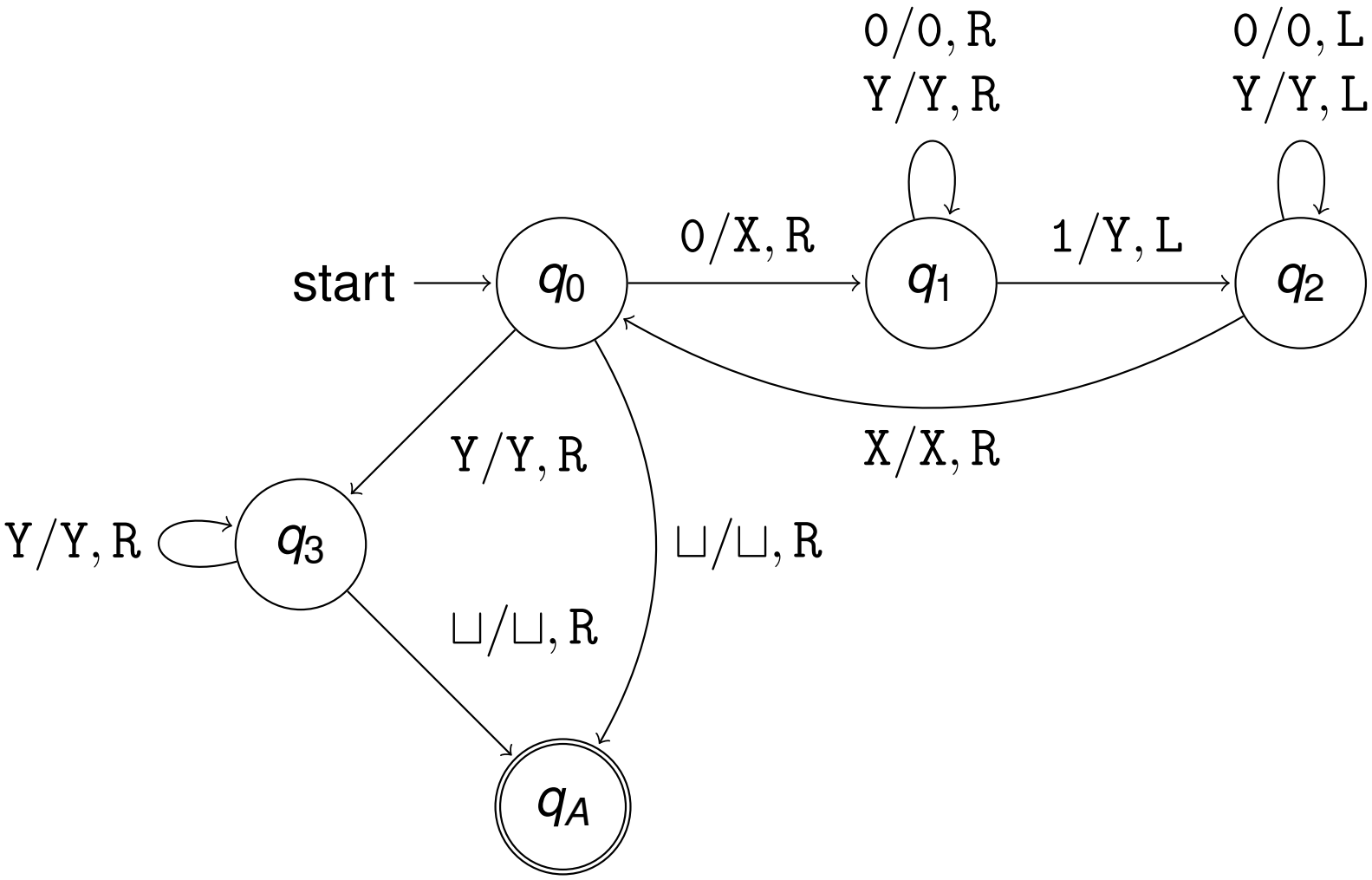
	0	1	X	Y	⊔
q_0	$(q_1, \mathtt{X}, \mathtt{R})$	$(q_R, 1, \mathtt{R})$	$(q_R, \mathtt{X}, \mathtt{R})$	$(q_3, \mathtt{Y}, \mathtt{R})$	$(q_A, ⊔, \mathtt{R})$
q_1	$(q_1, 0, \mathtt{R})$	$(q_2, \mathtt{Y}, \mathtt{L})$	$(q_R, \mathtt{X}, \mathtt{R})$	$(q_1, \mathtt{Y}, \mathtt{R})$	$(q_R, ⊔, \mathtt{R})$
q_2	$(q_2, 0, \mathtt{L})$	$(q_R, 1, \mathtt{R})$	$(q_0, \mathtt{X}, \mathtt{R})$	$(q_2, \mathtt{Y}, \mathtt{L})$	$(q_R, ⊔, \mathtt{R})$
q_3	$(q_R, 0, \mathtt{R})$	$(q_R, 1, \mathtt{R})$	$(q_R, \mathtt{X}, \mathtt{R})$	$(q_3, \mathtt{Y}, \mathtt{R})$	$(q_A, ⊔, \mathtt{R})$

Turing Machines: A First Example

This information corresponds to the following *incomplete* state diagram that is shown on the following slide.

It is incomplete because the reject state q_{reject} and transitions to it are not shown. The accept state has been renamed q_A to simplify the diagram as well.

Turing Machines: A First Example



Turing Machines: A First Example

Consider the execution of M on an input string $\omega = \texttt{0011}$. This proceeds as follows:

$$\begin{aligned} q_0\,\texttt{0011} &\vdash \texttt{X}\,q_1\,\texttt{011} \vdash \texttt{X0}\,q_1\,\texttt{11} \\ &\vdash \texttt{X}\,q_2\,\texttt{0Y1} \vdash q_2\,\texttt{X0Y1} \\ &\vdash \texttt{X}\,q_0\,\texttt{0Y1} \vdash \texttt{XX}\,q_1\,\texttt{Y1} \\ &\vdash \texttt{XXY}\,q_1\,\texttt{1} \vdash \texttt{XX}\,q_2\,\texttt{YY} \\ &\vdash \texttt{X}\,q_2\,\texttt{XYY} \vdash \texttt{XX}\,q_0\,\texttt{YY} \\ &\vdash \texttt{XXY}\,q_3\,\texttt{Y} \vdash \texttt{XXYY}\,q_3 \\ &\vdash \texttt{XXYY} \sqcup\, q_{\text{accept}}. \end{aligned}$$

Since $\texttt{XXYY} \sqcup\, q_{\text{accept}}$ represents an accepting configuration, M ***accepts*** the string $\texttt{0011}$.

Turing Machines: A First Example

Proving things about Turing machines can be tricky.

However, it *is* possible to prove the following: For every string $\omega \in \Sigma^\star$,

- if $\omega \in L$ then M ***accepts*** ω, and
- if $\omega \notin L$ then M ***rejects*** ω.

Turing Machines: A Second (Ridiculous) Example That Makes a Point

Consider another Turing machine

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

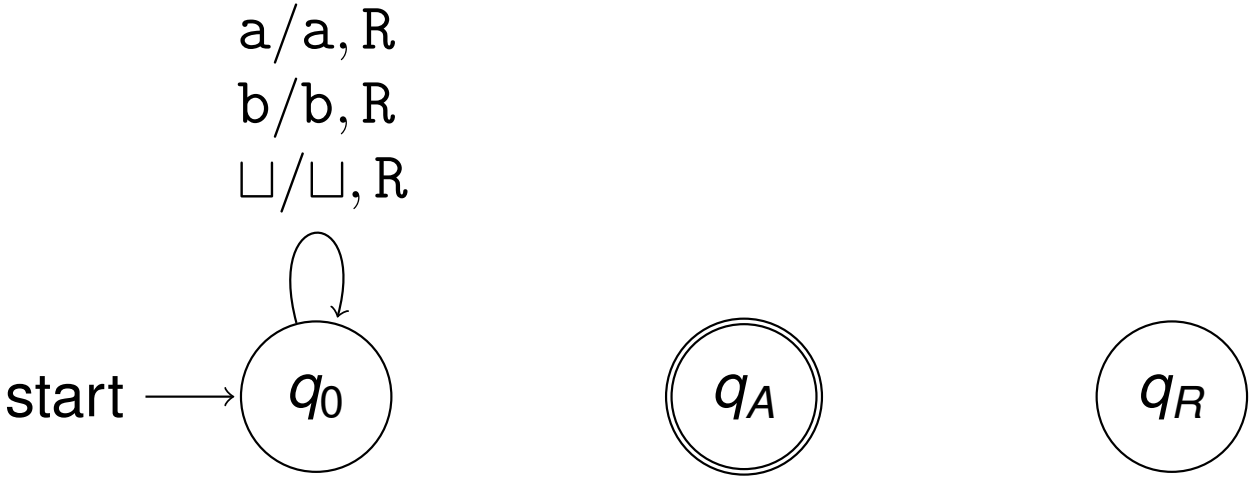
where

- $Q = \{q_0, q_{\text{accept}}, q_{\text{reject}}),$
- $\Sigma = \{\mathtt{a}, \mathtt{b}\}$
- $\Gamma = \{\mathtt{a}, \mathtt{b}, \sqcup\},$
- δ is as shown by the transition table shown on the next slide,
- q_0 is the start state, q_{accept} is the accept state, and q_{reject} is the reject state.

Turing Machines: A Second (Ridiculous) Example That Makes a Point

	a	b	⊔
q_0	$(q_0, \mathtt{a}, \mathtt{R})$	$(q_0, \mathtt{b}, \mathtt{R})$	$(q_0, \sqcup, \mathtt{R})$

A state diagram for this machine is as follows. (Once again, q_{accept} and q_{reject} are renamed q_A and q_R, respectively, to simplify the picture.)



Turing Machines: A Second (Ridiculous) Example That Makes a Point

This is a (somewhat silly) example of a Turing machine — that neither ***accepts*** nor ***rejects*** any string at all!

Instead, this machine ***loops*** on every string in $\Sigma^\star$.

Exercises:

1. Describe an even simpler Turing machine that ***accepts*** every string in $\Sigma^\star$.
2. Describe a simpler Turing machine that ***rejects*** every string in $\Sigma^\star$.

Recognition and Decision

Let $L \subseteq \Sigma^\star$

- A Turing machine M ***recognizes*** L — and L is called the ***language*** of M — if
  - M ***accepts*** every string $\omega \in L$, and
  - M either ***rejects*** or ***loops*** on every string $\omega \in \Sigma^\star$ such that $\omega \notin L$.

  A language is ***Turing-recognizable*** — or simply ***recognizable*** — if it is the language of some Turing machine.

The Turing machine in the first example ***recognizes*** the language

$$\{0^n 1^n \mid n \geq 0\}$$

and the Turing machine in the second example ***recognizes*** the language $\emptyset$.

Recognition and Decision

Let $L \subseteq \Sigma^\star$

- A Turing machine M ***decides*** L if
  - M ***accepts*** every string $\omega \in L$, and
  - M ***rejects*** every string $\omega \in \Sigma^\star$ such that $\omega \notin L$.

  A language is ***Turing-decidable*** — or simply ***decidable*** — if there is a Turing machine that decides it.

The Turing machine in the first example ***decides*** the language

$$\{0^n 1^n \mid n \geq 0\}.$$

The Turing machine in the second example does not ***decide*** any language at all, because it loops on some strings.

Turing Machines That Compute Functions

Consider the problem of computing a (partial or total) function

$$f : \Sigma_1^\star \to \Sigma_2^\star$$

using a Turing machine that is similar to a Turing machine that recognizes a language.

In particular, suppose that, for an input string $\omega \in \Sigma_1^\star$,

- if $f(\omega)$ is defined then this string should be written on the tape (with infinitely many blanks to the right) when the computation ends — with the tape head back at the leftmost cell of the tape, and
- if $f(\omega)$ is not defined then the computation should not end at all.

Turing Machines That Compute Functions

So, a deterministic Turing machine that computes a partial or total function $f : \Sigma_1^\star \to \Sigma_2^\star$ has an input alphabet, Σ_1, output alphabet Σ_2, and tape alphabet Γ.

- $\sqcup \notin \Sigma_1$, $\sqcup \notin \Sigma_2$, and $\Sigma_1 \cup \Sigma_2 \cup \{\sqcup\} \subseteq \Gamma$.

This machine also has a single ***halt state*** q_{halt} (instead of an accept and reject state).

Thus this kind of Turing machine can be modelled as a (different) 7-tuple

$$M = (Q, \Sigma_1, \Sigma_2, \Gamma, \delta, q_0, q_{\text{halt}})$$

where Q, Σ_1, Σ_2, Γ, δ, q_0 and q_{halt} are all as above.

Turing Machines That Compute Functions

The transition function, δ. must now satisfy the following requirements.

- $\delta(q, \sigma)$ is defined for every state $q \in Q \setminus \{q_{\text{halt}}\}$ and every symbol $\sigma \in \Gamma$, and
- $\delta(q_{\text{halt}}, \sigma)$ is not defined for any symbol $\sigma \in \Gamma$.

Turing Machines That Compute Functions

Such a Turing machine, M, ***computes*** a partial or total function $f : \Sigma_1^\star \to \Sigma_2^\star$ if the following properties are satisfied.

- For all $\omega \in \Sigma_1^\star$ such that $f(\omega)$ is defined, if $\mu = f(\omega) \in \Sigma_2^\star$, then $q_0\omega \vdash^\star q_{\text{halt}}\mu$ — so that, when executed on input ω, M eventually halts with $\mu = f(\omega)$ written on the leftmost cells of the tape, with infinitely many copies of $\sqcup$ to the right, and with the tape head resting at the leftmost cell of the tape, and
- for all $\omega \in \Sigma_1^\star$ such that $f(\omega)$ is not defined, M loops on ω.

A partial or total function $f : \Sigma_1^\star \to \Sigma_2^\star$ is ***Turing-computable*** (or just ***computable***) if there is a deterministic one-tape Turing machine that computes f.

Why Do We Care About Turing Machines?

- By the end of the course you will see evidence that Turing machines really *are* (in an important sense) as powerful as CPU's of modern computers, or even of computers that will exist in the future.
- These machines are simple enough that important things can be proved using them. In particular, by the end of the course you will (ideally) see how these machines can be used to identify languages that are ***undecidable*** as well as languages that are decidable.

History: Who was Turing?

Alan Mathison Turing was a British pioneering computer scientist, mathematician, logician, cryptanalyst, philosopher, mathematical biologist, and marathon and ultra distance runner.

- Turing is widely considered to be the father of theoretical computer science and artificial intelligence.
- The ***ACM A. M. Turing Award*** is named after him. Receipt of this award is generally considered to be the highest distinction in computer science — the “Nobel Prize of computing.”

History: Who was Turing?

- During the Second World War, Turing worked for the Government Code and Cypher School (GC&CS) at Bletchley Part, Britain's codebreaking center. Turing's pivotal role in cracking intercepted coded messages enabled the Allies to defeat the Nazis in many crucial engagements, including the Battle of the Atlantic. It is estimated that the work at Bletchley Park shortened the war in Europe by between two and four years.
- ***The Imitation Game*** provides a somewhat fictionalized account of this time in Turing's life.