Computer Science 351

More Turing Machine Variants and the Church-Turing Thesis

Instructor: Wayne Eberly

Department of Computer Science
University of Calgary

Lecture #14

Goal for Today

Further consideration of the relationship between Turing machine and other models of computation:

- Multi-Tape Turing machines
- Nondeterministic Turing machines
- More about this: the Church-Turing Thesis

Multi-Tape Turing Machines

- For any (fixed) positive integer k, a ***k*-tape Turing machine** is a generalization of a Turing machine that has k tapes — all of whose tape heads can move independently.
- Transitions can require that tape heads move ***left***, move ***right***, or ***stay.***
- A ***multi-tape Turing machine*** is a k-tape Turing machine, for some positive integer k.

Example

Example: Let $\Sigma = \{\mathtt{a}, \mathtt{b}\}$ and let

$$L = \{\mathtt{a}^n\mathtt{b}^n \mid n \in \mathbb{N}\}.$$

One Way To Decide Whether $\omega \in L$ — Using Two Tapes:

1. ***Accept*** if $\omega = \lambda$. Otherwise sweep right, copying all initial copies of "a" onto the second type (marking the first so it can be found later) — ***rejecting*** if ω starts with "b".
2. ***Reject*** if the next symbol seen, on the first tape, is "␣". Otherwise sweep ***right*** over copies of "b" on the first tape while matching with copies of "a" seen while sweeping back to the ***left*** on the second tape — ***rejecting*** if "a" is ever seen again on the first tape.

Example

3. ***Accept*** if the leftmost copy of "a" is detected on the second tape, and matched with the rightmost "b" on the first tape (so that "␣" is visible on the first tape after moving right) —and ***reject*** if one of these things happens before the other.

This can be implemented using a 2-tape Turing machine

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

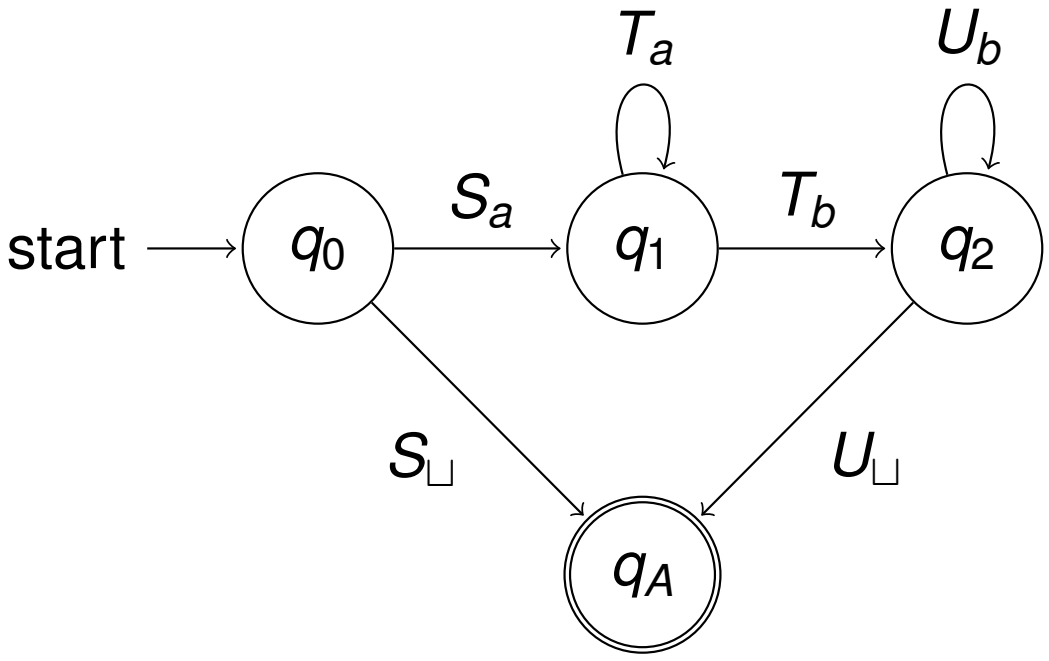
where

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

$$\Gamma = \{\mathtt{a}, \mathtt{b}, \dot{\mathtt{a}}, ␣\},$$

and transitions are as shown on the following slide.

Example



Transitions:

S_a: $\mathtt{a}, \sqcup / \dot{\mathtt{a}}, \mathtt{R}, \dot{\mathtt{a}}, \mathtt{R}$
$S_\sqcup$: $\sqcup, \sqcup / \sqcup, \mathtt{L}, \sqcup, \mathtt{L}$

T_a: $\mathtt{a}, \sqcup / \mathtt{a}, \mathtt{R}, \mathtt{a}, \mathtt{R}$
T_b: $\mathtt{b}, \sqcup / \mathtt{b}, \mathtt{R}, \sqcup, \mathtt{L}$

U_b: $\mathtt{b}, \mathtt{a} / \mathtt{b}, \mathtt{R}, \mathtt{a}, \mathtt{L}$
$U_\sqcup$: $\sqcup, \dot{\mathtt{a}} / \sqcup, \mathtt{L}, \dot{\mathtt{a}}, \mathtt{L}$

- To simplify the picture, the accepting state is shown as "q_A" instead of "q_{accept}".
- Transitions that are not shown go to the rejecting state, leaving tape contents and tape head positions unchanged.

Proof of Equivalence — Part One

Claim #1: Let $L \subseteq \Sigma^\star$.

(a) If L is Turing-recognizable then there is a k-tape Turing machine, for some integer $k \geq 1$, that recognizes L.

(b) If L is Turing-decidable then there is a k-tape Turing machine, for some integer $k \geq 1$, that decides L.

Idea of the Proof: This part is ***easy:*** All you need to do is to notice that any (regular) one-tape Turing machine is also a "k-tape Turing machine" when $k = 1$. The only other thing you need to do is to apply the definitions of "Turing-recognizable" and "Turing-decidable."

Proof of Equivalence — Part Two

Claim #2: Let $L \subseteq \Sigma^\star$. Let k be any integer such that $k \geq 1$.

(a) If there is a k-tape Turing machine M that recognizes L then there is also a one-tape Turing machine $\widehat{M}$ that recognizes L, so that L is Turing-recognizable.

(b) If there is a k-tapeTuring machine M that decides L then there is also a one-tape Turing machine $\widehat{M}$ that decides L, so that L is Turing-decidable.

Proof of Equivalence — Part Two

Sketch of Proof: Let k be a positive integer and let

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

with a language $L \subseteq \Sigma^\star$.

- The proof proceeds by describing a standard Turing machine

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

  such that $L(\widehat{M}) = L(M) = L$ — and such that $\widehat{M}$ ***decides*** L if and only if M decides L.
- This is easy to show if $q_0 = q_{\text{accept}}$ or if $q_0 = q_{\text{reject}}$ — so the case $q_0 \notin \{q_{\text{accept}}, q_{\text{reject}}\}$ will be considered.

Proof of Equivalence — Part Two

- When going from the k-tape Turing machine M to a one-tape Turing machine $\widehat{M}$ that ***simulates*** it, there will be a significant expansion of the *tape alphabet:* In particular, if M and $\widehat{M}$ each have input alphabet Σ, and M has tape alphabet Γ, then $\widehat{M}$ will have a *much larger* tape alphabet

$$\widehat{\Gamma} = \Sigma \cup \{\sqcup, \$\} \cup (\{\sqcup, \downarrow\} \times \Gamma)^k.$$

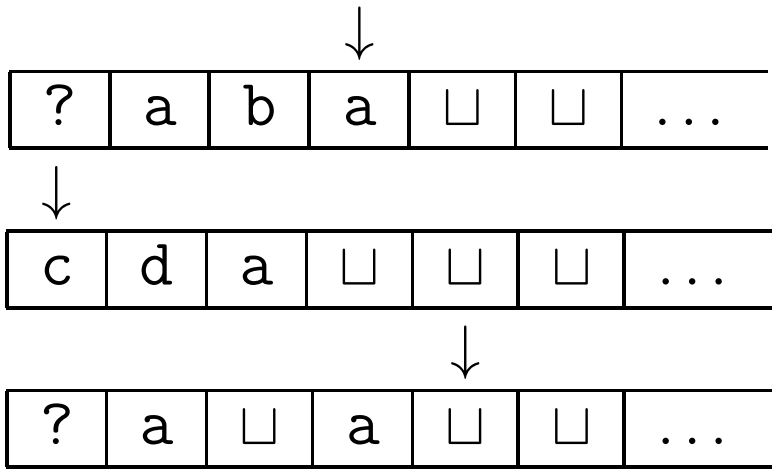
- This will allow us to view the non-blank part of $\widehat{M}$'s tape to have $2k$ ***tracks*** that can be used to store all the relevant information on M's multiple tapes at any time.
- ***Note:*** It is assumed here, that $\$ \notin \Gamma$ and that $\downarrow \notin \Gamma$.

Proof of Equivalence — Part Two

- The new symbol "$" will be used to mark the leftmost cell of $\widehat{M}$'s tape.
- Cells immediately to the right store symbols (from $\widehat{\Gamma}$) with $2k$ tracks — which are used to represent the contents of the non-blank part of M's tape.
- For $1 \leq i \leq k$,
  - Track $2i - 1$ is used to store the location of M's i^{th} tape head, by having a $\downarrow$ in the cell where the tape head is located and a $\sqcup$ in all other cells representing the non-blank part of M's i^{th} tape.
  - Track $2i$ is used to store the *contents* of the non-blank portion of M's i^{th} tape.

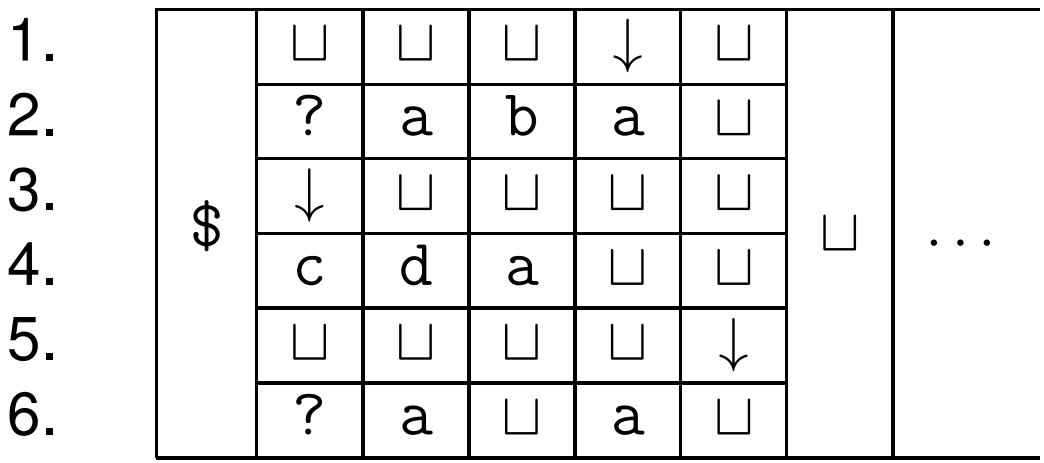
Proof of Equivalence — Part Two

Suppose, for example, that $k = 3$, $\Gamma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{d}, ?, \sqcup\}$ and the tapes of M are currently as follows.



Proof of Equivalence — Part Two

Then the contents of $\widehat{M}$'s tape might be as follows. Track numbers are shown to the left (and are not part of the tape.)



Proof of Equivalence — Part Two
Initialization Stage

- During the initialization stage the contents of $\widehat{M}$'s tape is changed, moving from
  - the contents included in a standard Turing machine's initial configuration, for an input string $\omega \in \Sigma^\star$,

  to
  - $\widehat{M}$'s *representation* of M's initial configuration — with ω stored on the first tape, all other tapes filled with copies of "$\sqcup$", and with all tape heads at the leftmost cell of the tape.
- This can be carried out using a sweep to the right, and then back to the left, using $O(|\omega|)$ steps of $\widehat{M}$.

Proof of Equivalence — Part Two
Step-By-Step Simulation

- Simulation of a step begins with a ***read phase*** in with $\widehat{M}$ sweeps from left to right, k times — using tracks $2i-1$ and $2i$ (for $i = 1, 2, \ldots, k$) to find the symbol currently visible on M's i^{th} tape. $\widehat{M}$'s finite control is to remember these symbols, along with M's current state.
- Once this information has been discovered, $\widehat{M}$ has all the information needed to identify the transition of M that should be applied.

Proof of Equivalence — Part Two
Step-By-Step Simulation

- Simulation of a step continues with an ***update phase*** in which $\widehat{M}$ sweeps from left to right, k times — updating tracks $2i-1$ and $2i$ (for $1 \leq i \leq k$) to carry out the changes needed for M's i^{th} tape.
- ***Optional:*** A "tidying" phase can be carried out to eliminate any unneeded copies of

$$(\sqcup, \sqcup, \sqcup, \sqcup, \ldots, \sqcup, \sqcup)$$

  at the right end of the non-blank part of $\widehat{M}$'s tape (to make $\widehat{M}$'s representation, of any configuration of M, unique).
- If this has not already happened, $\widehat{M}$ can move to its copy of the state that M could move to, to complete the simulation of this step.

Proof of Equivalence — Part Two
Finishing Up

- Setting $\widehat{q}_{\text{accept}}$ to be q_{accept}, and setting $\widehat{q}_{\text{reject}}$ to be q_{reject}, ensures that $\widehat{M}$ accepts ω once it simulates a step in which M accepts ω, and that $\widehat{M}$ rejects ω once it simulates a step in which M rejects ω.
- Correctness of the simulation (and completeness of the proof) can be carried out by more extensive uses of the techniques used to analyze Turing machine computations that have already been introduced.

Computing Functions

For $k \geq 1$, and alphabets Σ_1 and Σ_2, a ***k*-tape Turing machine** M **that computes a (partial or total) function** $f : \Sigma_1^\star \to \Sigma_2^\star$ satisfies the following properties:

- M has a single halting state, q_{halt}, instead of an accept state q_{accept} and reject state q_{reject}.
- The input configuration for an input string $\omega \in \Sigma_1^\star$, and the processing of moves, is the same as for a k-tape Turing machine that recognizes a language.
- If $f(\omega)$ is defined then M's execution on input ω should halt in state q_{halt}, with $f(\omega)$ written on the leftmost cells of the ***k*th tape**, with all other tapes filled with blanks, and with all tape heads at their leftmost cells.

Computing Functions

- If $f(\omega)$ is not defined then M's execution on input ω should not halt, at all — that is, M must ***loop*** on ω.

The proof, given above, can be modified to prove that a function $f : \Sigma_1^\star \to \Sigma^\star$ is ***Turing-computable*** *if and only if* there exists a k-tape Turing machine that computes f, for some positive integer k.

Nondeterministic Turing Machines

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

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

— such that $\delta(q_{\text{accept}}, \sigma) = \delta(q_{\text{reject}}, \sigma) = \emptyset$ for every symbol $\sigma \in \Gamma$.

This is (essentially) the same as the change made to the definition of a DFA in order to define an NFA (except that nothing corresponding to "λ-transitions" has been introduced).

Nondeterministic Turing Machines

Let $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$ be a nondeterministic Turing machine and let $\omega \in \Sigma^\star$.

- M ***accepts*** ω if *there exists* at least one (finite) sequence of moves, beginning in M's initial configuration for ω, that ends with M in state q_{accept}.
- M ***rejects*** ω if *every* sequence of moves of M, that begins with the initial configuration for ω, is finite — and there are no sequences of moves that end with M in state q_{accept}.
- M ***loops on*** ω otherwise.

Nondeterministic Turing Machines

Let $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$ be a nondeterministic Turing machine and let $L \subseteq \Sigma^\star$.

- M ***recognizes*** L if M accepts every string $\omega \in L$ and M either rejects or loops on every string $\omega \in \Sigma^\star$ such that $\omega \notin L$.
- M ***decides*** L if M accepts every string $\omega \in L$ and M rejects every string $\omega \in \Sigma^\star$ such that $\omega \notin L$ — so that M does not loop on any string in $\Sigma^\star$.[1]

[1] An even *stronger* condition is sometimes required, in order to say that a nondeterministic Turing machine "decides" a language — but adding this does not change the set of languages that are "decidable" by nondeterministic Turing machines.

Nondeterministic Turing Machines

Claim #3: Let $L \subseteq \Sigma^\star$ (for an alphabet L).

(a) L is Turing-recognizable if and only if there exists a nondeterministic Turing machine M such that M recognizes L.

(b) L is Turing-decidable if and only if there exists a nondeterministic Turing machine M such that M decides L.

Once again, ***simulations*** can be used to prove the above result. A supplemental document, for this lecture, includes additional information about this.

Church-Turing Thesis

- Beginning in the 1930's, a wide variety of abstract models of computation were proposed.
- While some were too limited to be useful to define "computability",[2] ***simulations*** were eventually obtained to show that all of the rest of them were "equivalent" — they all defined effectively the same sets of "recognizable" languages, "decidable" languages and "computable" functions as the ones we are now studying.
- The ***Church-Turing Thesis*** is a widely held belief that Turing machines (and the sets of languages and functions defined using them) really *do* model computability. Additional details about — quite important — thesis are given in a supplemental document about this.

[2] Many of these were still useful, for other reasons!