Computer Science 351

Multi-Tape Turing Machines, Nondeterministic Turing Machines, and the Church-Turing Thesis

Instructor: Wayne Eberly

Department of Computer Science
University of Calgary

Lecture #11

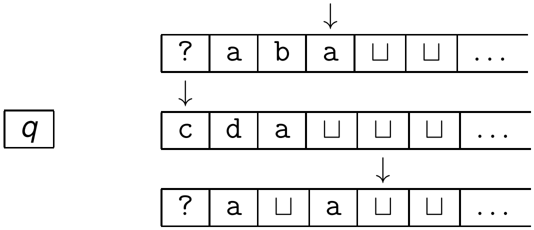
Goal for Today

- A useful variant of a Turing machine — a ***multi-tape Turing machine*** — will be introduced. A proof will be sketched that the sets of “Turing-recognizable” and “Turing-decidable” languages are not changed if these are defined using multi-tape Turing machines instead of (regular) one-tape Turing machines.
- Another variant — ***nondeterministic Turing machines*** — will also be described. The sets of “Turing-recognizable” and “Turing-decidable” languages are not changed if these are defined using nondeterministic Turing machines instead of (regular) one-tape Turing machines, either.
- the ***Church-Turing Thesis*** — a widely held belief that Turing machines (and the definitions of their languages and functions) really *do* model “computability” — will also be introduced.

Multi-Tape Turing Machines

For any (fixed) integer *k* such that $k \geq 1$, a ***k-tape Turing machine*** is a generalization of a Turing machine that has *k* tapes — all of whose tape heads can move independently.

For example, if $q \in Q$ and $\Gamma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{d}, ?, \sqcup\}$ then a configuration of a 3-tape Turing machine might look like this:



Multi-Tape Turing Machines: Representing Configurations

A ***configuration*** of a k-tape Turing machine M includes information about

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

Multi-Tape Turing Machines: Representing Configurations

Suppose that $\sharp$ is a symbol that is not in $Q \cup \Gamma$. Then there is a straightforward way to generalize the representation of configurations by strings (from the previous lecture) to obtain a representation of a configuration of a k-tape Turing machine as a string in $(Q \cup \Gamma \cup \{\sharp\})^\star$:

- For $1 \leq i \leq k$, let $\mu_i \in (Q \cup \Gamma)^\star$ be the string representing the current state, position of the i^{th} tape head and contents of the non-blank part of the i^{th} tape.
- The configuration of M can then be represented by the string
$$\mu_1 \sharp \mu_2 \sharp \ldots \sharp \mu_k \in (Q \cup \Gamma \cup \{\sharp\})^\star.$$
- This is the same as the string used to representation of M as a standard deterministic Turing machine if $k = 1$.

Multi-Tape Turing Machines

If $\omega = \sigma_1\sigma_2\ldots\sigma_n \in \Sigma^\star$ then the **start configuration** for ω is one such that

- the machine is in its ***start state*** q_0;
- the first tape is just like the tape of a one-tape Turing machine for this input: The leftmost n cells store the symbols $\sigma_1, \sigma_2, \ldots, \sigma_n$ (in order), with all other cells storing $\sqcup$. The tape head points to the leftmost cell on the tape.
- All other tapes are ***completely*** filled with $\sqcup$'s, and their tape heads point to the leftmost cell on the tape as well.

Multi-Tape Turing Machines: Initial Configuration

Then the string in $(Q \cup \Gamma \cup \{\sharp\})^\star$ representing this initial configuration is

$$q_0\omega \sharp q_0 \sharp q_0 \sharp \ldots \sharp q_0$$

that includes k copies of the symbol q_0 and $k-1$ copies of $\sharp$.

Multi-Tape Turing Machines

The ***transition function*** is now a (partial) function

$$\delta : Q \times \Gamma^k \to Q \times (\Gamma \times \{\mathtt{L}, \mathtt{R}, \mathtt{S}\})^k.$$

- If $\delta(q, \sigma_1, \sigma_2, \ldots, \sigma_k) = (r, ((\tau_1, m_1), (\tau_2, m_2), \ldots, (\tau_k, m_k)))$, where $q, r \in Q$, $\sigma_1, \sigma_2, \ldots, \sigma_k, \tau_1, \tau_2, \ldots, \tau_k \in \Gamma$, and where $m_1, m_2, \ldots, m_k \in \{\mathtt{L}, \mathtt{R}, \mathtt{S}\}$, then, if the machine is in state q and the tape head for the i^{th} tape points to a copy of σ_i, for $1 \leq i \leq k$, then — after its next move —
  - the state should change to state r,
  - the copy of σ_i visible on the i^{th} tape should be overwritten with a copy of τ_i, for $1 \leq i \leq k$, and
  - for $1 \leq i \leq k$,
    - if $m_i = \mathtt{L}$ then the i^{th} head should go left one position — unless it is already at the leftmost cell of the tape;
    - if $m_i = \mathtt{R}$ then the i^{th} tape head should go right one position;
    - if $m_i = \mathtt{S}$ then the i^{th} tape head should not move.

Multi-Tape Turing Machines

- $\delta(q, \sigma_1, \sigma_2, \ldots, \sigma_k)$ should be defined[1] for all states $q \in Q \setminus \{q_{\text{accept}}, q_{\text{reject}}\}$ and for all symbols $\sigma_1, \sigma_2, \ldots, \sigma_k \in \Gamma$.
- If $q \in \{q_{\text{accept}}, q_{\text{reject}}\}$ then $\delta(q, \sigma_1, \sigma_2, \ldots, \sigma_k)$ should *not* be defined for ***any*** symbols $\sigma_1, \sigma_2, \ldots, \sigma_k$.

Just like for regular Turing machines...

- *M **accepts*** ω if is possible to go from the start configuration for ω to an accepting configuration (which includes q_{accept}) using a finite number of moves;
- *M **rejects*** ω if it is possible to go from the start configuration for ω to a rejecting configuration (which includes q_{reject}) using a finite number of moves; and
- *M **loops*** on ω otherwise.

[1] although, sometimes, the transitions will never be used

Example: A Two-Tape Turing Machine for Palindromes

Let $\Sigma = \{\mathtt{a}, \mathtt{b}\}$ and consider the language

$$L_{Pal} = \{\omega \in \Sigma^\star \mid \omega = \omega^R\}.$$

A 2-tape Turing machine M_{Pal} — with tape alphabet

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

— will now be described.

Example: A Two-Tape Turing Machine for Palindromes

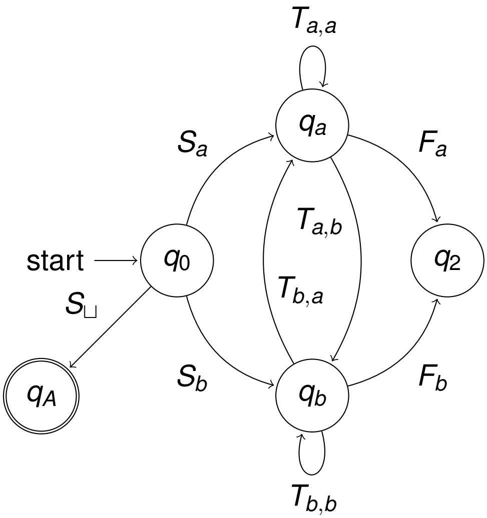
On input $\omega \in \Sigma^\star$:

1. If the first symbol visible is $\sqcup$ then ***accept***.

   Otherwise insert a leftmost # to the left of the input — shifting each symbol one position to the right in the process — while making a second copy of this on the second tape.

   A state diagram for this part of the process (with the accept state written as q_A and with the rejecting state and transitions to it left out) is as shown on the following slide.

Example: A Two-Tape Turing Machine for Palindromes



Transitions:

S_a:	$\mathtt{a}, \sqcup / \#, \mathtt{R}, \#, \mathtt{R}$
S_b:	$\mathtt{b}, \sqcup / \#, \mathtt{R}, \#, \mathtt{R}$
$S_\sqcup$:	$\sqcup, \sqcup / \sqcup, \mathtt{R}, \sqcup, \mathtt{R}$
$T_{a,a}$:	$\mathtt{a}, \sqcup / \mathtt{a}, \mathtt{R}, \mathtt{a}, \mathtt{R}$
$T_{a,b}$:	$\mathtt{b}, \sqcup / \mathtt{a}, \mathtt{R}, \mathtt{a}, \mathtt{R}$
$T_{b,a}$:	$\mathtt{a}, \sqcup / \mathtt{b}, \mathtt{R}, \mathtt{b}, \mathtt{R}$
$T_{b,b}$:	$\mathtt{b}, \sqcup / \mathtt{b}, \mathtt{R}, \mathtt{b}, \mathtt{R}$
F_a:	$\sqcup, \sqcup / \mathtt{a}, \mathtt{R}, \mathtt{a}, \mathtt{R}$
F_b:	$\sqcup, \sqcup / \mathtt{b}, \mathtt{R}, \mathtt{b}, \mathtt{R}$

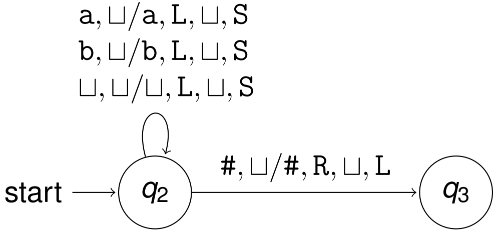
Example: A Two-Tape Turing Machine for Palindromes

2. The tape head for the first tape should be moved back to the left until # is visible, once again, while the second tape head should not move.

   When # is visible on the first tape the first tape head should move right while the second should move left — so that the first tape head is pointing to the leftmost symbol in the input while the second tape head is pointing to the rightmost symbol in the input — and the machine should go on to the third stage of this process.

   A — very simple — state diagram for this stage is as shown on the next slide.

Example: A Two-Tape Turing Machine for Palindromes


$$\begin{tikzcd}
\text{start} \arrow[r] & q_2 \arrow[loop above, "{\begin{array}{l}\mathtt{a},\sqcup/\mathtt{a},\mathtt{L},\sqcup,\mathtt{S}\\ \mathtt{b},\sqcup/\mathtt{b},\mathtt{L},\sqcup,\mathtt{S}\\ \sqcup,\sqcup/\sqcup,\mathtt{L},\sqcup,\mathtt{S}\end{array}}"] \arrow[rr, "{\#,\sqcup/\#,\mathtt{R},\sqcup,\mathtt{L}}"] & & q_3
\end{tikzcd}$$

Example: A Two-Tape Turing Machine for Palindromes

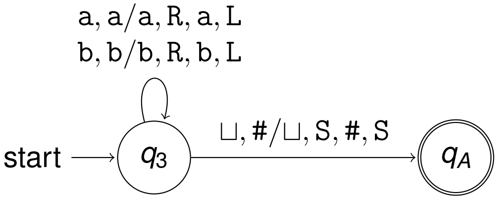
3. The first tape can now be used to read the input from left to right while the second tape can be used to read the input from right to left.

   If the input was ***not*** a palindrome then an `a` will be visible on one of these tapes at the same time as a `b` is visible on the other, and the input can then be ***rejected***.

   Otherwise, a $\sqcup$ will be seen on the first tape at the same time as # is seen on the second, and the input can be ***accepted*** instead.

   A (very simple) state diagram for this last part of the computation is as shown on the following slide.

Example: A Two-Tape Turing Machine for Palindromes


$$\begin{tikzcd}
\text{start} \arrow[r] & q_3 \arrow[loop above, "{\mathtt{a},\mathtt{a}/\mathtt{a},\mathtt{R},\mathtt{a},\mathtt{L}} \\ {\mathtt{b},\mathtt{b}/\mathtt{b},\mathtt{R},\mathtt{b},\mathtt{L}}"] \arrow[rr, "{\sqcup,\#/\sqcup,\mathtt{S},\#,\mathtt{S}}"] & & q_A
\end{tikzcd}$$

Example: A Two-Tape Turing Machine for Palindromes

Exercise: Trace the execution of this machine on the inputs

- λ
- `a`
- `abbabba`
- `abaa`

in order to better understand what is doing, and to see that it really *does* decide the language of palindromes.

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_1 that recognizes L then there is also a one-tape Turing machine M_2 that recognizes L, so that L is Turing-recognizable.

(b) If there is a k-tapeTuring machine M_1 that decides L then there is also a one-tape Turing machine M_2 that decides L, so that L is Turing-decidable.

How To Prove This: A Simulation

A ***simulation*** is something that can be presented to relate the power of two models of computation.

- These were used in Lecture #6 to prove that a language is a regular language (that is, the language of a deterministic finite automaton) if and only if it is the language of a *nondeterministic* finite automaton.

How To Prove This: A Simulation

As previously noted, in order to show that the machines described by a ***second*** model of computation are (in some sense) at least as powerful or efficient as the machines described by a ***first*** model of computation, we generally do the following:

(a) Consider an arbitrary machine M_1, of the type described by the ***first*** model of computation.

(b) Use M_1 to define another machine M_2, of the type described by the ***second*** model of computation.

(c) Prove that M_2 solves the same problem as M_1.

How To Prove This: A Simulation

When Turing machines (or other general machines) are involved, the computation of the second machine M_2, on a given instance ω often includes the following.

1. ***Initialization*** (or "*Setup*"): Use ω to produce a representation of M_1, as it would be configured at the beginning of its own execution on the same input ω.

   When describing M_1 it is generally necessary to show how M_2's storage (for Turing machines, its finite control and tape(s)) can be used to represent M_1's storage.

   A process that can be used to produce the representation of M_1's storage, for the beginning of an execution on input ω, *from the input ω itself*, should generally be presented and analyzed in order to carry out this step.

How To Prove This: A Simulation

(b) ***Step-by-Step Simulation:*** Consider each of the possible "moves" that M_1 could make and each of the possible configurations $\mathcal{C}_1$ that M_1 might be in, before this move is made.

Describe a process that can be used by M_2 to begin with its representation of $\mathcal{C}_1$ and "implement the move"' by producing a representation of $\mathcal{C}_2$, where $\mathcal{C}_2$ is the configuration of M_1 that would be reached (from $\mathcal{C}_1$) when this move is executed.

Prove the correctness of this process.

How To Prove This: A Simulation

(c) ***Completion*** (or "*Cleanup*"): If M_1's execution on the input ω ends then, when this is detected, M_2 uses its representation of M_1's final configuration, $\mathcal{C}_f$, on this input string, to move to the configuration $\widehat{\mathcal{C}_f}$ that *it* should have when its execution on the same input should end.

Once again, a process that can be used by M_2 to do this should be described and proved to be correct.

Simulating Multi-Tape Turing Machines: Eliminating a Trivial Case

Let k be a positive integer, let Σ be an alphabet, and let

$$M_1 = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$$

be a k-tape deterministic Turing machine as described above, with language $L \subseteq \Sigma^\star$.

- Note that if $q_0 = q_{\text{accept}}$ then $L = \Sigma^\star$ and it suffices to set M_2 to be a one-tape Turing machine such that $q_0 = q_{\text{accept}}$ as well, in order to establish the desired result for Claim #2: Both M_1 and M_2 accept every string $\omega \in \Sigma^\star$ (so that $L = \Sigma^\star$) without taking any steps at all.

Simulating Multi-Tape Turing Machines: Eliminating a Trivial Case

- The desired result for claim #2 is also easily established (in much the same way) if $q_0 = q_{\text{reject}}$, so that $L = \emptyset$.
- We will therefore assume, for the rest of the proof of Claim #2, that $q_0 \notin \{q_{\text{accept}}, q_{\text{reject}}\}$. In this case $T \geq 1$.

Simulating Multi-Tape Turing Machines: Organization of Data

- When going from the k-tape Turing machine M_1 to a one-tape Turing machine M_2 that ***simulates*** it, there will be a significant expansion of the *tape alphabet:* In particular, if M_1 and M_2 each have input alphabet Σ, and M_1 has tape alphabet Γ, then M_2 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 M_2's tape to have $2k$ ***tracks*** that can be used to store all the relevant information on M_1's multiple tapes at any time.
- ***Note:*** It is assumed here, that $\$ \notin \Gamma$ and that $\downarrow \notin \Gamma$. The new special symbols \$ and $\downarrow$ should be renamed, otherwise.

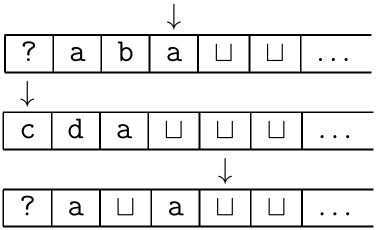
Simulating Multi-Tape Turing Machines: Organization of Data

For $1 \leq i \leq k$,

- Track $2i - 1$ is used to store the location of M_1'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 nonblank part of M_1's i^{th} tape.
- Track $2i$ is used to store the *contents* of the non-blank portion of M_1's i^{th} tape.

Simulating Multi-Tape Turing Machines: Organization of Data

Suppose, for example, that $k = 3$, $\Gamma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{d}, ?, \sqcup\}$ and the tapes of M_1 are currently as follows.



Simulating Multi-Tape Turing Machines: Organization of Data

Then the contents of M_2's tape might be as follows. Track numbers are shown to the left (and are not part of the tape.)

1.	$	⊔	⊔	⊔	↓	⊔	⊔	. . .
2.		?	a	b	a	⊔		
3.		↓	⊔	⊔	⊔	⊔		
4.		c	d	a	⊔	⊔		
5.		⊔	⊔	⊔	⊔	↓		
6.		?	a	⊔	a	⊔		

Simulating Multi-Tape Turing Machines: Organization of Data

M_2'*s* ***finite control*** must also be significantly expanded: Let $\widehat{Q}$ be the states in M_2.

- To begin, let us include $Q \cup \{\widehat{q}_0\}$ where Q is the set of states of M_1, $\widehat{q}_0 \notin Q$, and $\widehat{q}_0$ is the (new) start state of M_2.
- So far, $|Q| + 1$ states have been included in $\widehat{Q}$. Additional states will be described and included, as the details of the simulation are presented.

Simulating Multi-Tape Turing Machines: Initialization

Let us require that the simulation of every move of M_1 by M_2 should begin with M_2's tape head resting on the leftmost cell to the ***right*** of \$ on its tape — that is, the ***second*** cell.

In order to move from M_2's initial state on input ω to the representation of M_1's initial configuration on the same input string, M_2's moves should include

- a single sweep to the right over the non-blank portion of the tape in order to ***update the tape*** — starting with the symbol after \$ and continuing with the rest of the representation of the tapes, as given above, along with
- a single sweep back to the left[2] in order to ***reposition the tape head***.

[2]with one last step, back to the right

Simulating Multi-Tape Turing Machines: Initialization

- During the sweep right, it is necessary to use the finite control to remember a single symbol in Σ — because it is originally found on one cell of the tape but will be represented on the cell to the right of that.
- It is also necessary to remember whether the leftmost cell of M_1's tape has been represented yet — in order to know whether $\downarrow$ or $\sqcup$ should used to represent locations of tape heads when the next symbol is written onto M_2's tape.
- This "initialization" phase can be carried out by introducing (at most) $2|\Sigma| + 2$ additional states to $\widehat{Q}$.
- This part of the simulation always ends. Indeed, the number of steps of M_2 needed is at most 3 if the input string is empty, or at most $2n + 1$ if the input string has positive length n.

Simulating Multi-Tape Turing Machines: Simulation of a Move

Let τ_1, τ_2 be configurations of M_1 such that $\tau_1 \vdash \tau_2$. It is necessary to describe and analyze a process used by M_2 to move from a representation of τ_1 to a representation of τ_2.

- The state non-halting state q included in τ_1 (or, later, the state included in configuration τ_2) will be remembered using the finite control.
- During a ***read*** phase the symbols visible on each tape of M_1 (as given by τ_1) will be discovered using M_2's tape and remembered using the finite control.
- During a ***write*** phase M_2's tape updated to include the symbols given by τ_2 and update tape head location.
- A final ***tidying*** phase will be needed to complete the move to the representation of τ_2.

Simulating Multi-Tape Turing Machines: Simulation of a Move

During the ***read*** phase the symbol visible on M_1's i^{th} tape is discovered, for each integer i from 1 to k (in increasing order). For each integer i such that $1 \leq i \leq k$:

- During a sweep to the ***right*** a symbol storing $2k$ tracks such that track $2i - 1$ stores $\downarrow$ (instead of $\sqcup$) is searched for. When this is found M_2 changes state, in order to remember the state q of M_1, the symbols on the first $i - 1$ tapes that were discovered before this, and the symbol in Γ that has been found in track $2i$ — the symbol visible on M_1's i^{th} tape.
- A sweep back to the ***left*** (with one last step right) repositions the tape head.

Simulating Multi-Tape Turing Machines: Simulation of a Move

- The number of states that must be added to $\widehat{Q}$ to implement this phase is (at most)

$$(|Q| - 2)\left(|\Gamma|^k - 2\frac{|\Gamma|}{|\Gamma| - 1}\right).$$

- Suppose that, for $1 \leq i \leq k$, the i^{th} tape head of M_1 has distance ℓ_i from the leftmost cell of the tape. Then this iteration of the "read" phase can be carried out using at most $2\sum_{i=1}^{k} \max(\ell_i, 1)$ steps of M_2.

Simulating Multi-Tape Turing Machines: Simulation of a Move

Suppose that M_1 is in non-halting state $q \in Q$ when in configuration τ_1 and, for $1 \leq i \leq k$, the symbol $\sigma_i \in \Gamma$ is visible on M_1's i^{th} tape. Then some transition

$$\delta(q, (\sigma_1, \sigma_2, \dots, \sigma_k)) = (r, ((\widehat{\sigma}_1, d_1), (\widehat{\sigma}_2, d_2), \dots, (\widehat{\sigma}_k, d_k)))$$

(where $r \in Q$, $\widehat{\sigma}_1, \widehat{\sigma}_2, \dots, \widehat{\sigma}_k \in \Gamma$, and $d_1, d_2, \dots, d_k \in \{\mathtt{L}, \mathtt{R}, \mathtt{S}\}$) — that should be applied to move from configuration τ_1 to configuration τ_2 — can now be determined.

Since M_1 is a ***fixed*** Turing machine, we can think of M_1's transitions as being included in the definition of M_2's finite control: We can move to state of M_2 for the beginning of the write phase, that remembers the transition of M_1 that was used, as part of the final step of M_2's read phase.

Simulating Multi-Tape Turing Machines: Simulation of a Move

During the ***write*** phase — when the above transition is to be applied — the contents of tracks $2i-1$ and $2i$ in cells are changed on M_2's tape, in order to reflect a change to the contents of M_1's i^{th} tape and a movement of its tape head. This will be carried out for $i = k, k-1, \ldots, 1$, that is, by decreasing order of i.[3] For each i...

- An initial sweep ***right*** is needed to locate the cells of M_2's tape that must be updated. A constant number of steps are needed to update the tape.
- A sweep back to the ***left*** (with one last step right) is needed to reposition the tape head.

[3]This order is not necessary but it simplifies the description of M_2.

Simulating Multi-Tape Turing Machines: Simulation of a Move

- The number of states that must be added to $\widehat{Q}$ to implement this phase is (at most)

$$4k \cdot (|Q| - 2) \cdot |\Gamma|^k.$$

- Suppose that, for $1 \leq i \leq k$, the i^{th} tape head of M_1 has distance ℓ_i from the leftmost cell of the tape when this "write" phase begins — so that it has distance at most $\ell_i + 1$ from the leftmost cell of the tape when this "write" phase ends. Then the number of steps used by M_2 during this execution of the "write" phase is at most

$$2\left(\sum_{i=1}^{k} \ell_i\right) + 4k.$$

Simulating Multi-Tape Turing Machines: Simulation of a Move

The ***tidying*** phase removes unnecessary cells (showing each of M_1's tapes showing $\sqcup$, with none of the tape heads resting on the cell) from the right end of the non-blank part of M_2's tape.[4] This can be carried out using a single sweep to the right, followed by a sweep back to the left.

- At most $3 \cdot |Q|$ states must be added to $\widehat{Q}$ to implement this phase.
- Let K be the number of cells in the non-blank part of M_2's tape when the simulation of this step of M_1 begins. Then the number of steps of M_2 needed to implement this "tidying" phase is at most $2K + 2$.

[4]This is not really necessary, but it makes the simulation a bit easier to describe in more detail.

Simulating Multi-Tape Turing Machines: Simulation of a Move

It follows that the total number of states added to $\widehat{Q}$ to implement the simulation of a move, is at most

$$\begin{aligned}
&(|Q|-2)\cdot\left(|\Gamma|^k - 2\frac{|\Gamma|}{|\Gamma|-1}\right) && \text{(for the “read” phase)}\\
&\quad + 4k\cdot(|Q|-2)\cdot|\Gamma|^k && \text{(for the “write” phase)}\\
&\quad + 3\cdot|Q| && \text{(for the “cleanup” phase)}\\
&= 4k\cdot(|Q|-2)|\Gamma|^k + (|Q|-2)|\Gamma|^k\\
&\quad + \left(3 - 2\frac{|\Gamma|}{|\Gamma|-1}\right)\cdot(|Q|-2) + 6.
\end{aligned}$$

Simulating Multi-Tape Turing Machines: Simulation of a Move

If ℓ_i is the distance of M_1's i^{th} tape head from the leftmost cell, when this machine is in configuration τ_1, for $1 \leq i \leq k$, and K is the number of cells in the non-blank part of M_2's tape when the simulation of the move from τ_1 to τ_2 begins, then it follows that the total number of steps of M_2 to simulate this move is at most

$$4\left(\sum_{i=1}^{k} \max(\ell_i, 1)\right) + 2K + 4k + 2.$$

Simulating Multi-Tape Turing Machines: Simulation of a Move

The next two claims are easily proved by induction on the number of moves of M_1 that have been simulated so far.

Subclaim #3: Suppose that $t \geq 0$ and that the first t moves of the execution of M_1 on input $\omega \in \Sigma^\star$ have been performed. Then each tape head of M_1 is at most t positions away from the leftmost cell of its tape. That is, $\ell_i \leq t$ for $1 \leq i \leq k$.

Proof: Easy induction on t. □

Simulating Multi-Tape Turing Machines: Simulation of a Move

Subclaim #4: Suppose that $t \geq 0$. Then the number K of non-blank cells on M_2's tape, after the first t moves of M_1 have been simulated, is at most

$$\max(2, |\omega| + 1, t + 1).$$

Proof: Easy induction on t. □

Simulating Multi-Tape Turing Machines: Simulation of a Move

Subclaim #5: If $t \geq 0$ and M_1 makes at least t moves when it is executed on an input string $\omega \in \Sigma^\star$ with length n, then the number of moves used by M_2 to simulate M's t^{th} move is at most linear in $k \times t$.

How To Prove This: Apply the results from the previous two subclaims to continue the analysis.

Simulating Multi-Tape Turing Machines: Continuation of Analysis

Subclaim #6: Suppose that M_1 makes at least t moves when executed on an input string $\omega \in \Sigma^\star$ with length n.

Then the total number of moves used by M_2 to initialize its tape and then simulate the first t moves of M_1 is at most linear in $n + k \times t^2$.

How To Prove This: Use the analysis of the Initialization phase above, along with Subclaim #5, to write down a ***summation*** for the total number of steps used by $\widehat{M}$ — and then notice that this sum really is at most linear in $n + k \times t^2$.

Simulating Multi-Tape Turing Machines: Cleanup Phase

There is very little "cleanup" needed for this simulation: M_2 should simply move to its accept state (respectively, its reject state) after simulating a move in which M_1 moves to its accept state (respectively, its reject state).

- No additional states of M_2, are needed to carry this out, and at most one more step is used.

Simulating Multi-Tape Turing Machines

Subclaim #7: Let $\omega \in \Sigma^\star$.

(a) If M_1 accepts ω then M_2 also accepts ω.
(b) If M_1 rejects ω then M_2 also rejects ω.
(c) If M_1 loops on ω then M_2 also loops on ω.

Proof: Suppose that M_1 accepts or rejects ω. Then it does so after making t moves for some integer $t \geq 0$. Parts (a) and (b) now follow by Subclaim #6 (and the “Cleanup Phase” described above).

Part (c) follows by an inspection of these details too: M_2 only accepts (or rejects) after confirming that M_1 would too.

Simulating Multi-Tape Turing Machines

Subclaim #8: Let $L \subseteq \Sigma^\star$.

(a) If M_1 recognizes L Then M_2 recognizes L as well.

(b) If M_1 decides L then M_2 decides L too.

How To Prove This: Use the definition of what it means for a Turing machine to *recognize* or *decide* a language, along with Subclaim #7.

Note: This completes a proof of Claim #2.

Simulating Multi-Tape Turing Machines

Confession: This lecture has included a proof of Theorem 3.13 in *Introduction to the Theory of Computation* — but the proof in that book is a different one!

Instead of using $2k$ tracks, the simulation in the textbook uses a much smaller tape alphabet: $\widehat{\Gamma}$ includes

- Γ,
- a symbol $\dot{\sigma}$ for every symbol $\sigma \in \Gamma$, and
- a symbol # that is assumed not to belong to Γ.

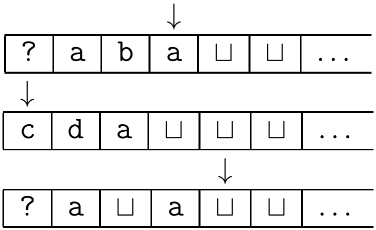
The non-blank part of M_2's tape now looks like

$$\# T_1 \# T_2 \# \ldots \# T_k \#$$

where T_i is the non-blank part of M_1's i^{th} tape — with $\dot{\sigma}$ appearing instead of σ at the location of the tape head.

Simulating Multi-Tape Turing Machines

For example, if M_1's tapes look like this:



then the non-blank portion of M_2's tape would look like this:

#?ab$\dot{\mathtt{a}}$#$\dot{\mathtt{c}}$da#?a ⊔ a$\dot{\sqcup}$#

Some details of the simulation in the proof in the above book are a bit more complicated (and, potentially confusing) than the details of the simulation in these notes. This is why the *other* standard proof of Claim #2 is given in these notes.

Simulating Multi-Tape Turing Machines

- It is also possible to define ***multi-tape Turing machines that compute functions*** and prove similar results concerning these.
- These are described in a supplemental document for this lecture.

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 (partial) function

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

- If $q \in Q \setminus \{q_{\text{accept}}, q_{\text{reject}}\}$ then $\delta(q, \sigma)$ should be defined for every symbol $\sigma \in \Gamma$.

  However, it is possible that $\delta(q, \sigma) = \emptyset$, so that there is no “next move” that can be used if M is in state q, and σ is visible on the tape.
- If $q \in \{q_{\text{accept}}, q_{\text{reject}}\}$ then $\delta(q, \sigma)$ should *not* be defined for *any* 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.

Nondeterministic Turing Machines

Let $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, qR)$ 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*, beginning with the initial configuration for ω is finite — either because the rejecting state has been reached or a “non-halting” state has been reached, but there is move that can be made (because *M* in a state *q*, and σ is now visible, where $\delta(q, \sigma) = \emptyset$).
- *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 $\Sigma^\star$.[5]

[5] There is an even *stronger* condition that 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. See the supplemental document for details, if you are interested in this.

Nondeterministic Turing Machines

Claim #9: 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",[6] ***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.

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

Expectations

Expectations:

- Students will not be asked about the details of the proofs of the claims, given in this lecture, on a quiz, exam, or assignment in this course!
- However, it is important that you understand (and remember) the ***results*** that has been given: If one-tape Turing machines are replaced, in definitions, by multi-tape Turing machines — or by nondeterministic Turing machines — then the set of languages that are *recognizable*, or *decidable*, is not changed.

Expectations

- The idea that one kind of machine can ***simulate*** the computation of a *different kind of machine* is ***extremely important***.
- It will be proved that Turing machines can simulate other models of computation too. It is now sufficient to prove that a ***multi-tape*** Turing machine can simulate these other models to prove this.
- On the other hand, if I am trying to show you that a problem ***cannot*** be solved then I will probably discuss (hypothetical) one-tape Turing machines, instead — because that will make this easier to prove.