COMP3630 / COMP6363

week 5: **Introduction to Turing Machines**

This Lecture Covers Chapter 8 of HMU: Introduction to Turing Machines

slides created by: Dirk Pattinson, based on material by
Peter Hoefner and Rob van Glabbeck; with improvements by Pascal Bercher

convenor & lecturer: Pascal Bercher

The Australian National University

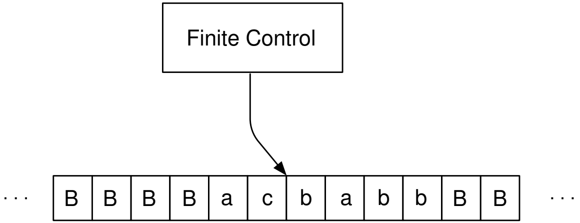
Semester 1, 2023

Content of this Chapter

- Turing Machine
- Extensions of Turing Machines (and PDAs)
- Restrictions of Turing Machines

Additional Reading: Chapter 8 of HMU.

Turing Machine: Informal Definition



- An tape extending infinitely in both sides
- A reading head that can edit tape, move right or left.
- A finite control.
- A string is accepted if finite control reaches a final/accepting state

Turing Machine: Formal Definition

A Turing machine $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$ is comprised of:

Turing Machine: Formal Definition

A Turing machine $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$ is comprised of:

- Q: finite set of states

Turing Machine: Formal Definition

A Turing machine $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$ is comprised of:

- Q: finite set of states
- Σ: finite set of input symbols

Turing Machine: Formal Definition

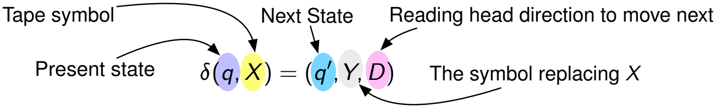
A Turing machine $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$ is comprised of:

- Q: finite set of states
- Σ: finite set of input symbols
- Γ: finite set of tape symbols such that $\Sigma \subseteq \Gamma$

Turing Machine: Formal Definition

A Turing machine $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$ is comprised of:

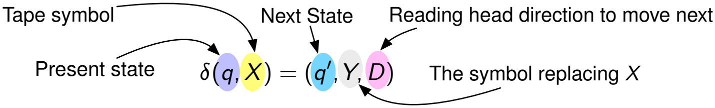
- Q: finite set of states
- Σ: finite set of input symbols
- Γ: finite set of tape symbols such that $\Sigma \subseteq \Gamma$
- δ: (deterministic) transition function. δ is a **partial function** over $Q \times \Gamma$, where the first component is viewed as the present state, and the second is viewed as the tape symbol read. If $\delta(q, X)$ is defined, then



Turing Machine: Formal Definition

A Turing machine $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$ is comprised of:

- Q: finite set of states
- Σ: finite set of input symbols
- Γ: finite set of tape symbols such that $\Sigma \subseteq \Gamma$
- δ: (deterministic) transition function. δ is a **partial function** over $Q \times \Gamma$, where the first component is viewed as the present state, and the second is viewed as the tape symbol read. If $\delta(q, X)$ is defined, then

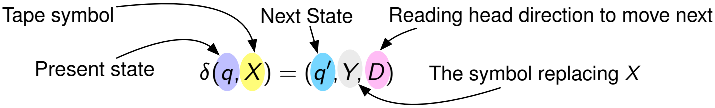


- $B \in \Gamma \setminus \Sigma$ is the blank symbol. All but a finite number of tape symbols are Bs.

Turing Machine: Formal Definition

A Turing machine $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$ is comprised of:

- Q: finite set of states
- Σ: finite set of input symbols
- Γ: finite set of tape symbols such that $\Sigma \subseteq \Gamma$
- δ: (deterministic) transition function. δ is a **partial function** over $Q \times \Gamma$, where the first component is viewed as the present state, and the second is viewed as the tape symbol read. If $\delta(q, X)$ is defined, then

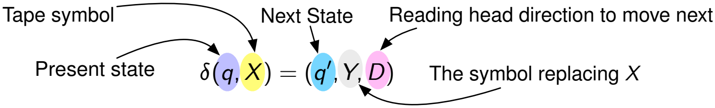


- $B \in \Gamma \setminus \Sigma$ is the blank symbol. All but a finite number of tape symbols are Bs.
- q_0: the initial state of the TM.

Turing Machine: Formal Definition

A Turing machine $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$ is comprised of:

- Q: finite set of states
- Σ: finite set of input symbols
- Γ: finite set of tape symbols such that $\Sigma \subseteq \Gamma$
- δ: (deterministic) transition function. δ is a **partial function** over $Q \times \Gamma$, where the first component is viewed as the present state, and the second is viewed as the tape symbol read. If $\delta(q, X)$ is defined, then

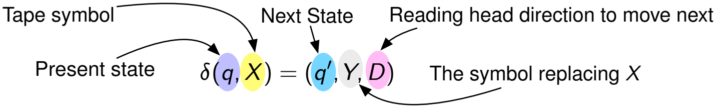


- $B \in \Gamma \setminus \Sigma$ is the blank symbol. All but a finite number of tape symbols are Bs.
- q_0: the initial state of the TM.
- F: the set of final/accepting states of the TM.

Turing Machine: Formal Definition

A Turing machine $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$ is comprised of:

- Q: finite set of states
- Σ: finite set of input symbols
- Γ: finite set of tape symbols such that $\Sigma \subseteq \Gamma$
- δ: (deterministic) transition function. δ is a **partial function** over $Q \times \Gamma$, where the first component is viewed as the present state, and the second is viewed as the tape symbol read. If $\delta(q, X)$ is defined, then



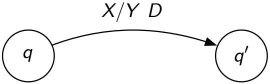
- $B \in \Gamma \setminus \Sigma$ is the blank symbol. All but a finite number of tape symbols are Bs.
- q_0: the initial state of the TM.
- F: the set of final/accepting states of the TM.

- Head **always** moves to the left or right. Being stationary is not an option. It can also be defined with such an option, see tutorial.

Describing TMs

- Turing machines can be defined by describing δ using a transition table.
- They can also be defined using transition diagrams (with labels appropriately altered)

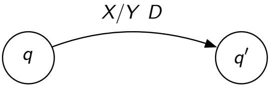
If $\delta(q, X) = (q', Y, D)$



Describing TMs

- Turing machines can be defined by describing δ using a transition table.
- They can also be defined using transition diagrams (with labels appropriately altered)

If $\delta(q, X) = (q', Y, D)$

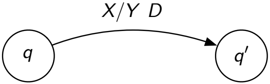


A TM that accepts any binary string that does not contain 111

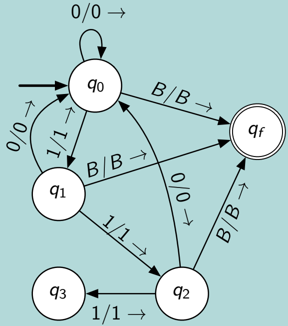
Describing TMs

- Turing machines can be defined by describing δ using a transition table.
- They can also be defined using transition diagrams (with labels appropriately altered)

If $\delta(q, X) = (q', Y, D)$



A TM that accepts any binary string that does not contain 111

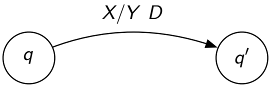


This encodes a DFA (almost).
Can you see why?

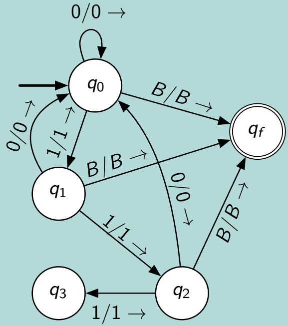
Describing TMs

- Turing machines can be defined by describing δ using a transition table.
- They can also be defined using transition diagrams (with labels appropriately altered)

If $\delta(q, X) = (q', Y, D)$



A TM that accepts any binary string that does not contain 111



This encodes a DFA (almost). Can you see why?

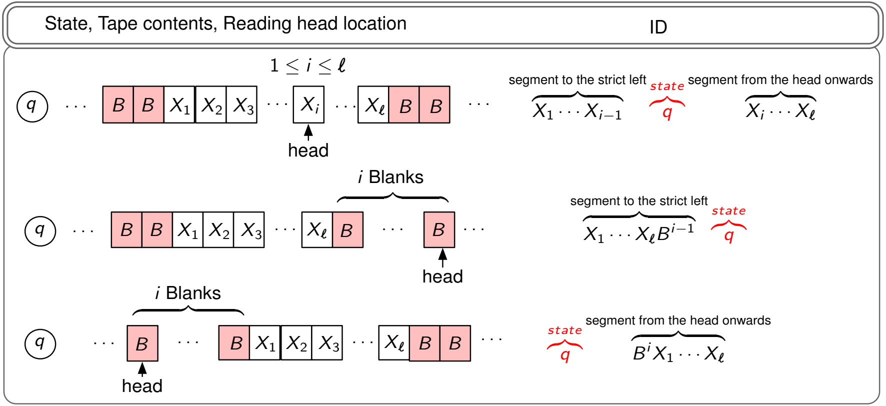
Because we never manipulate the tape and terminate once the String is read. The only difference is that not all edges are defined, but this can be fixed with a trap state.

Instantaneous Descriptions of TMs

- An instantaneous description (or configuration) of a TM is a complete description of the system that enables one to determine the trajectory of the TM as it operates.

Instantaneous Descriptions of TMs

- An instantaneous description (or configuration) of a TM is a complete description of the system that enables one to determine the trajectory of the TM as it operates.
- The instantaneous description or configuration or ID of a TM contains 3 parts:
  (a) The (finite, non-trivial) portion of tape to the left of the reading head;
  (b) the state that the TM is presently in; and
  (c) the (finite, non-trivial) portion of the tape to the right of the reading head.



'Moves' of a TM

- Just as in the case of a PDA, we use $\underset{M}{\vdash}$ to indicate a single move of a TM M, and $\overset{*}{\underset{M}{\vdash}}$ to indicate zero or a finite number of moves of a TM.

Present ID	Transition	**Next** ID
$X_1 \cdots X_{i-1} q X_i \cdots X_\ell$	$\delta(q, X_i) = (q', Y, R)$	$X_1 \cdots X_{i-1} Y q' X_{i+1} \cdots X_\ell$
$(1 < i < \ell)$	$\delta(q, X_i) = (q', Y, L)$	$X_1 \cdots X_{i-2} q' X_{i-1} Y X_{i+1} \cdots X_\ell$

'Moves' of a TM

- Just as in the case of a PDA, we use $\underset{M}{\vdash}$ to indicate a single move of a TM M, and $\overset{*}{\underset{M}{\vdash}}$ to indicate zero or a finite number of moves of a TM.

Present ID	Transition	**Next** ID
$X_1 \cdots X_{i-1} q X_i \cdots X_\ell$ $(1 < i < \ell)$	$\delta(q, X_i) = (q', Y, R)$	$X_1 \cdots X_{i-1} Y q' X_{i+1} \cdots X_\ell$
	$\delta(q, X_i) = (q', Y, L)$	$X_1 \cdots X_{i-2} q' X_{i-1} Y X_{i+1} \cdots X_\ell$
$X_1 \cdots X_\ell B^{i-1} q$	$\delta(q, B) = (q', Y, R)$	$X_1 \cdots X_\ell B^{i-1} Y q'$
	$\delta(q, B) = (q', Y, L)$	$\begin{cases} X_1 \cdots X_{\ell-1} q' X_\ell Y & i = 1 \\ X_1 \cdots X_\ell B^{i-2} q' B Y & i > 1 \end{cases}$

'Moves' of a TM

- Just as in the case of a PDA, we use $\underset{M}{\vdash}$ to indicate a single move of a TM M, and $\overset{*}{\underset{M}{\vdash}}$ to indicate zero or a finite number of moves of a TM.

Present ID	Transition	**Next** ID
$X_1 \cdots X_{i-1} q X_i \cdots X_\ell$ $(1 < i < \ell)$	$\delta(q, X_i) = (q', Y, R)$	$X_1 \cdots X_{i-1} Y q' X_{i+1} \cdots X_\ell$
	$\delta(q, X_i) = (q', Y, L)$	$X_1 \cdots X_{i-2} q' X_{i-1} Y X_{i+1} \cdots X_\ell$
$X_1 \cdots X_\ell B^{i-1} q$	$\delta(q, B) = (q', Y, R)$	$X_1 \cdots X_\ell B^{i-1} Y q'$
	$\delta(q, B) = (q', Y, L)$	$\begin{cases} X_1 \cdots X_{\ell-1} q' X_\ell Y & i = 1 \\ X_1 \cdots X_\ell B^{i-2} q' B Y & i > 1 \end{cases}$
$q B^i X_1 \ldots X_\ell$	$\delta(q, B) = (q', Y, R)$	$\begin{cases} Y q' X_2 \cdots X_\ell & i = 0 \\ Y q' B^{i-1} X_1 \cdots X_\ell & i > 0 \end{cases}$
	$\delta(q, B) = (q', Y, L)$	$\begin{cases} q' B Y X_2 \cdots X_\ell & i = 0 \\ q' B Y B^{i-1} X_1 \cdots X_\ell & i > 0 \end{cases}$

Language accepted by a TM

- A string w is in the language accepted by a TM M iff $q_0 w \overset{*}{\underset{M}{\vdash}} \alpha p \beta$ for some $p \in F$.

Language accepted by a TM

- A string w is in the language accepted by a TM M iff $q_0 w \overset{*}{\underset{M}{\vdash}} \alpha p \beta$ for some $p \in F$.
- Another possible notion of acceptance is to require a TM to halt (i.e., no further transitions are possible).

Language accepted by a TM

- A string w is in the language accepted by a TM M iff $q_0 w \overset{*}{\underset{M}{\vdash}} \alpha p \beta$ for some $p \in F$.
- Another possible notion of acceptance is to require a TM to halt (i.e., no further transitions are possible).
- It is always possible to design a TM such that the TM halts when it reaches a final state without changing the language the TM accepts.

Language accepted by a TM

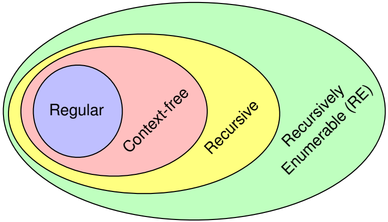
- A string w is in the language accepted by a TM M iff $q_0 w \overset{*}{\underset{M}{\vdash}} \alpha p \beta$ for some $p \in F$.
- Another possible notion of acceptance is to require a TM to halt (i.e., no further transitions are possible).
- It is always possible to design a TM such that the TM halts when it reaches a final state without changing the language the TM accepts.
- However, we cannot require (all) TMs to halt for all inputs.

Language accepted by a TM

- A string w is in the language accepted by a TM M iff $q_0 w \overset{*}{\underset{M}{\vdash}} \alpha p \beta$ for some $p \in F$.
- Another possible notion of acceptance is to require a TM to halt (i.e., no further transitions are possible).
- It is always possible to design a TM such that the TM halts when it reaches a final state without changing the language the TM accepts.
- However, we cannot require (all) TMs to halt for all inputs.
- A language L is **recursively enumerable** if it is accepted by some TM.

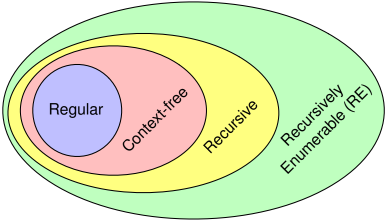
Language accepted by a TM

- A string w is in the language accepted by a TM M iff $q_0 w \overset{*}{\underset{M}{\vdash}} \alpha p \beta$ for some $p \in F$.
- Another possible notion of acceptance is to require a TM to halt (i.e., no further transitions are possible).
- It is always possible to design a TM such that the TM halts when it reaches a final state without changing the language the TM accepts.
- However, we cannot require (all) TMs to halt for all inputs.
- A language L is **recursively enumerable** if it is accepted by some TM.
- A language L is **recursive** if it is accepted by a TM that **always** halts on its input.



Language accepted by a TM

- A string w is in the language accepted by a TM M iff $q_0 w \overset{*}{\underset{M}{\vdash}} \alpha p \beta$ for some $p \in F$.
- Another possible notion of acceptance is to require a TM to halt (i.e., no further transitions are possible).
- It is always possible to design a TM such that the TM halts when it reaches a final state without changing the language the TM accepts.
- However, we cannot require (all) TMs to halt for all inputs.
- A language L is **recursively enumerable** if it is accepted by some TM.
- A language L is **recursive** if it is accepted by a TM that **always** halts on its input.



- (Another important class is the context-sensitive languages. They sit between the context-free and recursive languages.)

On Acceptance, Rejection, Termination, and Deciding I/III

- The primary purpose of a TM is to recognize/accept a language.
- Since TMs might not always terminate, we differentiate between deciding and accepting. The fundamental difference here is halting (on the non-accepted words).

On Acceptance, Rejection, Termination, and Deciding I/III

- The primary purpose of a TM is to recognize/accept a language.
- Since TMs might not always terminate, we differentiate between deciding and accepting. The fundamental difference here is halting (on the non-accepted words).
- Recall the following definitions:
  - A TM halts if in the current state there's no transition for the current symbol.
  - A TM accepts a word if there exists a transition to an accepting state when that word is read.
  - a TM rejects a word if it's not accepted and the TM halts.

On Acceptance, Rejection, Termination, and Deciding I/III

- The primary purpose of a TM is to recognize/accept a language.
- Since TMs might not always terminate, we differentiate between deciding and accepting. The fundamental difference here is halting (on the non-accepted words).
- Recall the following definitions:
  - A TM halts if in the current state there's no transition for the current symbol.
  - A TM accepts a word if there exists a transition to an accepting state when that word is read.
  - a TM rejects a word if it's not accepted and the TM halts.
- Now, what does that mean/imply for a TM M?
- We clarify this on the next slide in detail, but first a few general notes!

On Acceptance, Rejection, Termination, and Deciding I/III

- The primary purpose of a TM is to recognize/accept a language.
- Since TMs might not always terminate, we differentiate between deciding and accepting. The fundamental difference here is halting (on the non-accepted words).
- Recall the following definitions:
  - A TM halts if in the current state there's no transition for the current symbol.
  - A TM accepts a word if there exists a transition to an accepting state when that word is read.
  - a TM rejects a word if it's not accepted and the TM halts.
- Now, what does that mean/imply for a TM M?
- We clarify this on the next slide in detail, but first a few general notes!
- Usually, there is no need to define outgoing edges for accepting states. Because as soon as we enter such a state, the input word is accepted! (So why proceed?)

On Acceptance, Rejection, Termination, and Deciding I/III

- The primary purpose of a TM is to recognize/accept a language.
- Since TMs might not always terminate, we differentiate between deciding and accepting. The fundamental difference here is halting (on the non-accepted words).
- Recall the following definitions:
  - A TM halts if in the current state there's no transition for the current symbol.
  - A TM accepts a word if there exists a transition to an accepting state when that word is read.
  - a TM rejects a word if it's not accepted and the TM halts.
- Now, what does that mean/imply for a TM M?
- We clarify this on the next slide in detail, but first a few general notes!
- Usually, there is no need to define outgoing edges for accepting states. Because as soon as we enter such a state, the input word is accepted! (So why proceed?)
  - Thus, when you pick/design a TM for accepting a given language L (which you know must exist by assumption if L is in a certain class), you are allowed to do so using a "reasonable" TM that always halts on accepted words.

On Acceptance, Rejection, Termination, and Deciding I/III

- The primary purpose of a TM is to recognize/accept a language.
- Since TMs might not always terminate, we differentiate between deciding and accepting. The fundamental difference here is halting (on the non-accepted words).
- Recall the following definitions:
  - A TM halts if in the current state there's no transition for the current symbol.
  - A TM accepts a word if there exists a transition to an accepting state when that word is read.
  - a TM rejects a word if it's not accepted and the TM halts.
- Now, what does that mean/imply for a TM M?
- We clarify this on the next slide in detail, but first a few general notes!
- Usually, there is no need to define outgoing edges for accepting states. Because as soon as we enter such a state, the input word is accepted! (So why proceed?)
  - Thus, when you pick/design a TM for accepting a given language L (which you know must exist by assumption if L is in a certain class), you are allowed to do so using a "reasonable" TM that always halts on accepted words.
  - However, if you have to judge properties of a given TM (e.g., whether some TM M always halts, or accepts a particular word etc.), then you have to deal with *any* TM – reasonable or not... (*Motivation:* You might want to judge properties of somebody's "program". And we'd like to know whether we actually can!)

On Acceptance, Rejection, Termination, and Deciding II/III

- “Accepting w”

On Acceptance, Rejection, Termination, and Deciding II/III

- "Accepting w" implies $w \in L(M)$. It does not imply halting since we can define outgoing edges in accepting states. It is however noteworthy that usually we write "accept and halt" since we usually pick the TM ourselves and thus use a reasonable one that doesn't have outgoing edges from accepting states.

On Acceptance, Rejection, Termination, and Deciding II/III

- "Accepting w" implies $w \in L(M)$. It does not imply halting since we can define outgoing edges in accepting states. It is however noteworthy that usually we write "accept and halt" since we usually pick the TM ourselves and thus use a reasonable one that doesn't have outgoing edges from accepting states.
- "Not accepting w"

On Acceptance, Rejection, Termination, and Deciding II/III

- "Accepting w" implies $w \in L(M)$. It does not imply halting since we can define outgoing edges in accepting states. It is however noteworthy that usually we write "accept and halt" since we usually pick the TM ourselves and thus use a reasonable one that doesn't have outgoing edges from accepting states.
- "Not accepting w" implies $w \notin L(M)$. This is however not the same as rejecting. We can "not accept w" either by rejecting w (and thus terminating) or by looping forever. In particular this means that "Not accepting" does not imply halting.

On Acceptance, Rejection, Termination, and Deciding II/III

- "Accepting w" implies $w \in L(M)$. It does not imply halting since we can define outgoing edges in accepting states. It is however noteworthy that usually we write "accept and halt" since we usually pick the TM ourselves and thus use a reasonable one that doesn't have outgoing edges from accepting states.
- "Not accepting w" implies $w \notin L(M)$. This is however not the same as rejecting. We can "not accept w" either by rejecting w (and thus terminating) or by looping forever. In particular this means that "Not accepting" does not imply halting.
- "Rejecting w"

On Acceptance, Rejection, Termination, and Deciding II/III

- "Accepting w" implies $w \in L(M)$. It does not imply halting since we can define outgoing edges in accepting states. It is however noteworthy that usually we write "accept and halt" since we usually pick the TM ourselves and thus use a reasonable one that doesn't have outgoing edges from accepting states.
- "Not accepting w" implies $w \notin L(M)$. This is however not the same as rejecting. We can "not accept w" either by rejecting w (and thus terminating) or by looping forever. In particular this means that "Not accepting" does not imply halting.
- "Rejecting w" implies $w \notin L(M)$. It also implies halting since this is the only way we can reject a word.

On Acceptance, Rejection, Termination, and Deciding II/III

- "Accepting w" implies $w \in L(M)$. It does not imply halting since we can define outgoing edges in accepting states. It is however noteworthy that usually we write "accept and halt" since we usually pick the TM ourselves and thus use a reasonable one that doesn't have outgoing edges from accepting states.
- "Not accepting w" implies $w \notin L(M)$. This is however not the same as rejecting. We can "not accept w" either by rejecting w (and thus terminating) or by looping forever. In particular this means that "Not accepting" does not imply halting.
- "Rejecting w" implies $w \notin L(M)$. It also implies halting since this is the only way we can reject a word.
- "Not rejecting w"

On Acceptance, Rejection, Termination, and Deciding II/III

- "Accepting w" implies $w \in L(M)$. It does not imply halting since we can define outgoing edges in accepting states. It is however noteworthy that usually we write "accept and halt" since we usually pick the TM ourselves and thus use a reasonable one that doesn't have outgoing edges from accepting states.
- "Not accepting w" implies $w \notin L(M)$. This is however not the same as rejecting. We can "not accept w" either by rejecting w (and thus terminating) or by looping forever. In particular this means that "Not accepting" does not imply halting.
- "Rejecting w" implies $w \notin L(M)$. It also implies halting since this is the only way we can reject a word.
- "Not rejecting w" neither implies $w \in L(M)$ nor $w \notin L(M)$. This is because if we do not reject w we could either accept it (then, $w \in L(M)$) or we could loop forever without traversing an accepting state (then, $w \notin L(M)$). It thus also neither implies halting (since we could loop forever as just mentioned) nor does it imply not halting (since we could accept and halt).

On Acceptance, Rejection, Termination, and Deciding II/III

- "Accepting w" implies $w \in L(M)$. It does not imply halting since we can define outgoing edges in accepting states. It is however noteworthy that usually we write "accept and halt" since we usually pick the TM ourselves and thus use a reasonable one that doesn't have outgoing edges from accepting states.
- "Not accepting w" implies $w \notin L(M)$. This is however not the same as rejecting. We can "not accept w" either by rejecting w (and thus terminating) or by looping forever. In particular this means that "Not accepting" does not imply halting.
- "Rejecting w" implies $w \notin L(M)$. It also implies halting since this is the only way we can reject a word.
- "Not rejecting w" neither implies $w \in L(M)$ nor $w \notin L(M)$. This is because if we do not reject w we could either accept it (then, $w \in L(M)$) or we could loop forever without traversing an accepting state (then, $w \notin L(M)$). It thus also neither implies halting (since we could loop forever as just mentioned) nor does it imply not halting (since we could accept and halt).
- "Halting on w"

On Acceptance, Rejection, Termination, and Deciding II/III

- "Accepting w" implies $w \in L(M)$. It does not imply halting since we can define outgoing edges in accepting states. It is however noteworthy that usually we write "accept and halt" since we usually pick the TM ourselves and thus use a reasonable one that doesn't have outgoing edges from accepting states.
- "Not accepting w" implies $w \notin L(M)$. This is however not the same as rejecting. We can "not accept w" either by rejecting w (and thus terminating) or by looping forever. In particular this means that "Not accepting" does not imply halting.
- "Rejecting w" implies $w \notin L(M)$. It also implies halting since this is the only way we can reject a word.
- "Not rejecting w" neither implies $w \in L(M)$ nor $w \notin L(M)$. This is because if we do not reject w we could either accept it (then, $w \in L(M)$) or we could loop forever without traversing an accepting state (then, $w \notin L(M)$). It thus also neither implies halting (since we could loop forever as just mentioned) nor does it imply not halting (since we could accept and halt).
- "Halting on w" neither implies $w \in L(M)$ nor $w \notin L(M)$. It also does not imply rejection. This is all the case because we can halt both in an accepting and in a rejecting state (without previously having traversed an accepting state).

On Acceptance, Rejection, Termination, and Deciding II/III

- "Accepting w" implies $w \in L(M)$. It does not imply halting since we can define outgoing edges in accepting states. It is however noteworthy that usually we write "accept and halt" since we usually pick the TM ourselves and thus use a reasonable one that doesn't have outgoing edges from accepting states.
- "Not accepting w" implies $w \notin L(M)$. This is however not the same as rejecting. We can "not accept w" either by rejecting w (and thus terminating) or by looping forever. In particular this means that "Not accepting" does not imply halting.
- "Rejecting w" implies $w \notin L(M)$. It also implies halting since this is the only way we can reject a word.
- "Not rejecting w" neither implies $w \in L(M)$ nor $w \notin L(M)$. This is because if we do not reject w we could either accept it (then, $w \in L(M)$) or we could loop forever without traversing an accepting state (then, $w \notin L(M)$). It thus also neither implies halting (since we could loop forever as just mentioned) nor does it imply not halting (since we could accept and halt).
- "Halting on w" neither implies $w \in L(M)$ nor $w \notin L(M)$. It also does not imply rejection. This is all the case because we can halt both in an accepting and in a rejecting state (without previously having traversed an accepting state).
- "Not halting on w"

On Acceptance, Rejection, Termination, and Deciding II/III

- "Accepting w" implies $w \in L(M)$. It does not imply halting since we can define outgoing edges in accepting states. It is however noteworthy that usually we write "accept and halt" since we usually pick the TM ourselves and thus use a reasonable one that doesn't have outgoing edges from accepting states.
- "Not accepting w" implies $w \notin L(M)$. This is however not the same as rejecting. We can "not accept w" either by rejecting w (and thus terminating) or by looping forever. In particular this means that "Not accepting" does not imply halting.
- "Rejecting w" implies $w \notin L(M)$. It also implies halting since this is the only way we can reject a word.
- "Not rejecting w" neither implies $w \in L(M)$ nor $w \notin L(M)$. This is because if we do not reject w we could either accept it (then, $w \in L(M)$) or we could loop forever without traversing an accepting state (then, $w \notin L(M)$). It thus also neither implies halting (since we could loop forever as just mentioned) nor does it imply not halting (since we could accept and halt).
- "Halting on w" neither implies $w \in L(M)$ nor $w \notin L(M)$. It also does not imply rejection. This is all the case because we can halt both in an accepting and in a rejecting state (without previously having traversed an accepting state).
- "Not halting on w" implies almost nothing. It only implies that we are not rejecting, because rejection implies halting. That's because we only know that we loop forever, which can even happen after a word was accepted.

On Acceptance, Rejection, Termination, and Deciding III/III

Now, what does it mean that a language L is decidable (= recursive, in R)?

On Acceptance, Rejection, Termination, and Deciding III/III

Now, what does it mean that a language L is decidable (= recursive, in R)?

- If L is decidable, it means that there is a total (i.e., always halting) TM that judges for all $w \in \Sigma^*$ whether $w \in L$ (or not).
- Thus, there is a TM M with $L = L(M)$, and TM halts on all $w \in \Sigma^*$, whether $w \in L(M)$ or whether $w \notin L(M)$.

On Acceptance, Rejection, Termination, and Deciding III/III

Now, what does it mean that a language L is decidable ($=$ recursive, in R)?

- If L is decidable, it means that there is a total (i.e., always halting) TM that judges for all $w \in \Sigma^*$ whether $w \in L$ (or not).
- Thus, there is a TM M with $L = L(M)$, and TM halts on all $w \in \Sigma^*$, whether $w \in L(M)$ or whether $w \notin L(M)$.
- Note that while a TM does not *have* to halt after accepting a word, we know that there is no point in continuing after a word was accepted. Therefore, we can just pick a TM without such pointless transitions for accepting states. Thus, since we choose that TM, we can assume that it always halts for all words that are accepted.
  (But recall that this is not the case if you need to decide whether "a given TM" has certain properties – then everything goes!)

On Acceptance, Rejection, Termination, and Deciding III/III

Now, what does it mean that a language L is decidable (= recursive, in R)?

- If L is decidable, it means that there is a total (i.e., always halting) TM that judges for all $w \in \Sigma^*$ whether $w \in L$ (or not).
- Thus, there is a TM M with $L = L(M)$, and TM halts on all $w \in \Sigma^*$, whether $w \in L(M)$ or whether $w \notin L(M)$.
- Note that while a TM does not *have* to halt after accepting a word, we know that there is no point in continuing after a word was accepted. Therefore, we can just pick a TM without such pointless transitions for accepting states. Thus, since we choose that TM, we can assume that it always halts for all words that are accepted.
(But recall that this is not the case if you need to decide whether "a given TM" has certain properties – then everything goes!)

What does it mean that a language L is semi-decidable (= recursively enumerable, in RE)?

On Acceptance, Rejection, Termination, and Deciding III/III

Now, what does it mean that a language L is decidable (= recursive, in R)?

- If L is decidable, it means that there is a total (i.e., always halting) TM that judges for all $w \in \Sigma^*$ whether $w \in L$ (or not).
- Thus, there is a TM M with $L = L(M)$, and TM halts on all $w \in \Sigma^*$, whether $w \in L(M)$ or whether $w \notin L(M)$.
- Note that while a TM does not *have* to halt after accepting a word, we know that there is no point in continuing after a word was accepted. Therefore, we can just pick a TM without such pointless transitions for accepting states. Thus, since we choose that TM, we can assume that it always halts for all words that are accepted.
  (But recall that this is not the case if you need to decide whether "a given TM" has certain properties – then everything goes!)

What does it mean that a language L is semi-decidable (= recursively enumerable, in RE)?

- There is a TM M with $L = L(M)$ that halts on all words in L. (As above f.a. $w \in L$.)
- But if $w \notin L$, our TM could either reject or not halt (without accepting). Thus, it might loop forever and therefore, potentially (if L is not in R), not halt.

On Acceptance, Rejection, Termination, and Deciding III/III

Now, what does it mean that a language L is decidable (= recursive, in R)?

- If L is decidable, it means that there is a total (i.e., always halting) TM that judges for all $w \in \Sigma^*$ whether $w \in L$ (or not).
- Thus, there is a TM M with $L = L(M)$, and TM halts on all $w \in \Sigma^*$, whether $w \in L(M)$ or whether $w \notin L(M)$.
- Note that while a TM does not *have* to halt after accepting a word, we know that there is no point in continuing after a word was accepted. Therefore, we can just pick a TM without such pointless transitions for accepting states. Thus, since we choose that TM, we can assume that it always halts for all words that are accepted. *(But recall that this is not the case if you need to decide whether "a given TM" has certain properties – then everything goes!)*

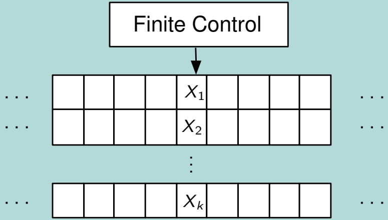
What does it mean that a language L is semi-decidable (= recursively enumerable, in RE)?

- There is a TM M with $L = L(M)$ that halts on all words in L. (As above f.a. $w \in L$.)
- But if $w \notin L$, our TM could either reject or not halt (without accepting). Thus, it might loop forever and therefore, potentially (if L is not in R), not halt.
- Note that $R \subsetneq RE$, and therefore being in RE, does not mean that we are not in R! We might be since for any $L \in R$ holds $L \in RE$.

Multiple-Track TMs

Multiple-track TM

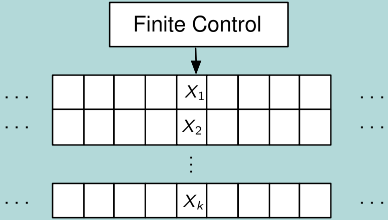
- We do not provide a formal definition (but assume you could provide one).
- There are k tracks, each having symbols written on them. They are essentially tapes, but we call them that way since they are not independent.



Multiple-Track TMs

Multiple-track TM

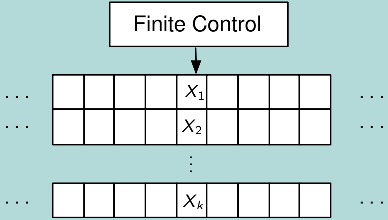
- We do not provide a formal definition (but assume you could provide one).
- There are k tracks, each having symbols written on them. They are essentially tapes, but we call them that way since they are not independent.
- The machine can only read symbols from each tape corresponding to **one** location, i.e., all symbols in a column at any one time.
- Likewise, all tapes move simultaneously in the same direction.



Multiple-Track TMs

Multiple-track TM

- We do not provide a formal definition (but assume you could provide one).
- There are k tracks, each having symbols written on them. They are essentially tapes, but we call them that way since they are not independent.
- The machine can only read symbols from each tape corresponding to **one** location, i.e., all symbols in a column at any one time.
- Likewise, all tapes move simultaneously in the same direction.

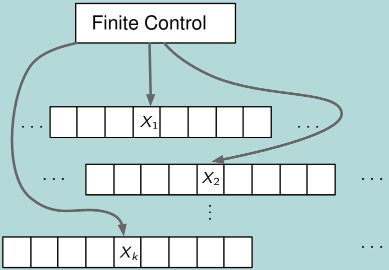


- A k-track TM with tape alphabet Γ has the same langauge-acceptance power as a TM with tape alphabet Γ^k. (E.g., each cell contains the “symbol” $(X_1, \ldots, X_k)$)

Multi-tape TMs

Multiple-tape TM

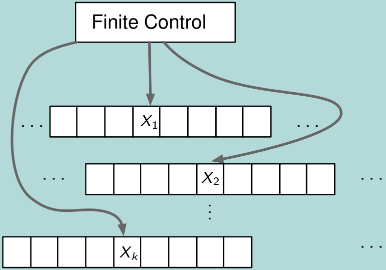
- We (again) don't provide a formal definition (but assume you could provide one).
- There are k (independent) tapes, each having symbols written on them.



Multi-tape TMs

Multiple-tape TM

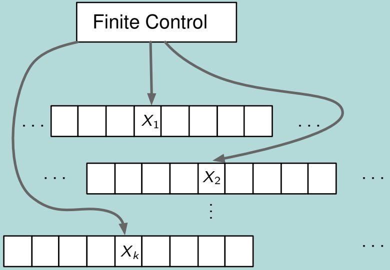
- We (again) don't provide a formal definition (but assume you could provide one).
- There are k (independent) tapes, each having symbols written on them.
- The machine can read each tape independently, i.e., the symbols read from each tape need not correspond to the same location.



Multi-tape TMs

Multiple-tape TM

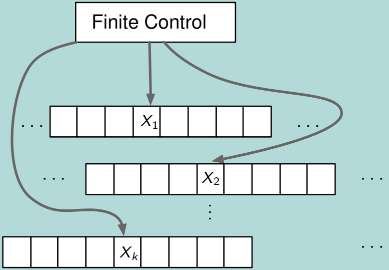
- We (again) don't provide a formal definition (but assume you could provide one).
- There are k (independent) tapes, each having symbols written on them.
- The machine can read each tape independently, i.e., the symbols read from each tape need not correspond to the same location.
- After all tapes are read, all tape transitions must happen (now we can also stay with a head – the entire TM is for convenience anyway!).



Multi-tape TMs

Multiple-tape TM

- We (again) don't provide a formal definition (but assume you could provide one).
- There are k (independent) tapes, each having symbols written on them.
- The machine can read each tape independently, i.e., the symbols read from each tape need not correspond to the same location.
- After all tapes are read, all tape transitions must happen (now we can also stay with a head – the entire TM is for convenience anyway!).



- The rest stays the same (e.g., one set of states, acceptance, etc.).

Multi-tape TMs

Theorem 8.2.1

Every language that is accepted by a multi-tape TM is also recursively enumerable (i.e., accepted by some 'standard' TM).

Proof of Theorem 8.2.1

Multi-tape TMs

Theorem 8.2.1

Every language that is accepted by a multi-tape TM is also recursively enumerable (i.e., accepted by some 'standard' TM).

Proof of Theorem 8.2.1

- Let L be accepted by a k-tape TM M. We'll devise a $2k$-track TM M' that accepts L.

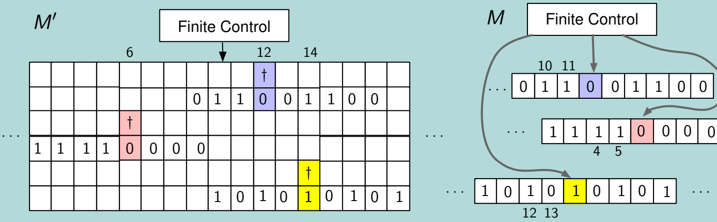
Multi-tape TMs

Theorem 8.2.1

Every language that is accepted by a multi-tape TM is also recursively enumerable (i.e., accepted by some 'standard' TM).

Proof of Theorem 8.2.1

- Let L be accepted by a k-tape TM M. We'll devise a $2k$-track TM M' that accepts L.
- Every even tape of M' has the same alphabet as that of the k-tape TM. The $2i^{\text{th}}$ track of M' contains exactly the same contents as the i^{th} tape of M.



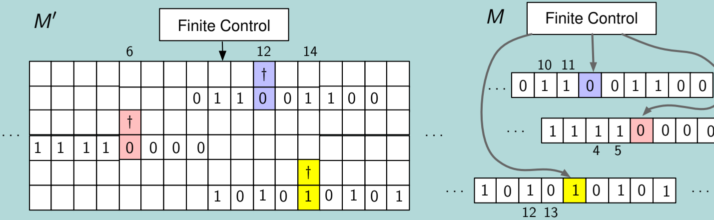
Multi-tape TMs

Theorem 8.2.1

Every language that is accepted by a multi-tape TM is also recursively enumerable (i.e., accepted by some 'standard' TM).

Proof of Theorem 8.2.1

- Let L be accepted by a k-tape TM M. We'll devise a $2k$-track TM M' that accepts L.
- Every even tape of M' has the same alphabet as that of the k-tape TM.
  The $2i^{\text{th}}$ track of M' contains exactly the same contents as the i^{th} tape of M.
- Every odd track has an alphabet $\{B, \dagger\}$, and contains a single $\dagger$.
  The $2i-1^{\text{th}}$ track of M' contains $\dagger$ at the location where the i^{th} head of M is located.

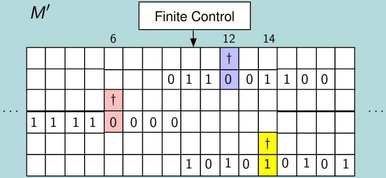


Multi-tape TMs

Proof of Theorem 8.2.1 (1 of 3)

What is the main problem we need to solve?

- In the Multi-tape TM M, heads move independently, whereas in the Multi-track TM M' they do not. So the heads can diverge:



(But M' has just a single head position!)

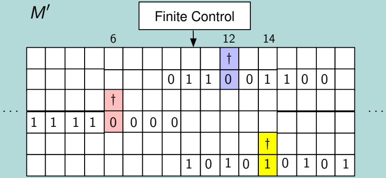
So, how to solve it?

Multi-tape TMs

Proof of Theorem 8.2.1 (1 of 3)

What is the main problem we need to solve?

- In the Multi-tape TM M, heads move independently, whereas in the Multi-track TM M' they do not. So the heads can diverge:



(But M' has just a single head position!)

So, how to solve it?

- Make sure that in each transition of M, we visit all heads of M'.
- "Store" all head positions in a state with k (number of tapes) entries.

Multi-tape TMs

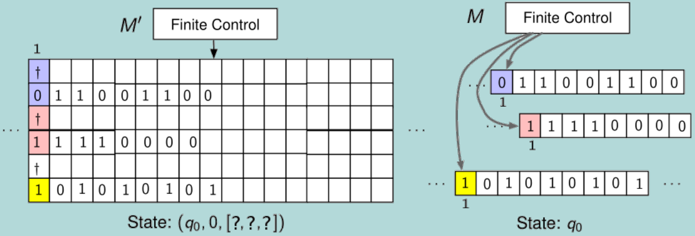
Proof of Theorem 8.2.1 (2 of 3)

- The state of M' has 3 components: (a) the state of M; (b) the number of †s to its head's strict left; and (c) a k-length tuple from $(\Gamma \cup \{?\})^k$.

Multi-tape TMs

Proof of Theorem 8.2.1 (2 of 3)

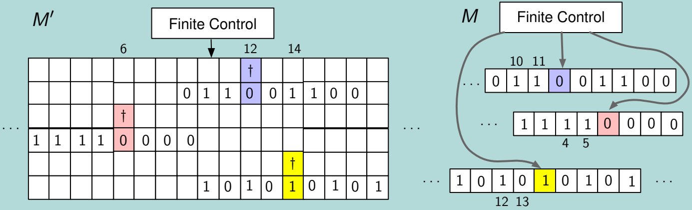
- The state of M' has 3 components: (a) the state of M; (b) the number of †s to its head's strict left; and (c) a k-length tuple from $(\Gamma \cup \{?\})^k$.
- At the beginning of the sweep, the head of M' is at the location of the leftmost † and the state of M' is $(q, 0, [?, \cdots, ?])$. The head moves to the right uncovering †s and the corresponding track symbols (are stored in the third component of the state).



Multi-tape TMs

Proof of Theorem 8.2.1 (2 of 3)

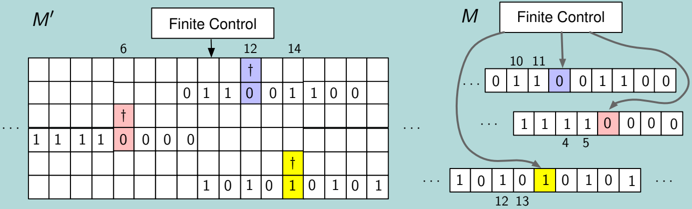
- The state of M' has 3 components: (a) the state of M; (b) the number of †s to its head's strict left; and (c) a k-length tuple from $(\Gamma \cup \{?\})^k$.
- At the beginning of the sweep, the head of M' is at the location of the leftmost † and the state of M' is $(q, 0, [?, \cdots, ?])$. The head moves to the right uncovering †s and the corresponding track symbols (are stored in the third component of the state).
- Each move of M takes multiple moves of M', and is a sweep of the tape from the location of the leftmost † to that of the rightmost † and back performing the changes to tracks that M would do to its corresponding tapes.
- The right sweep ends when the second component is k.



Multi-tape TMs

Proof of Theorem 8.2.1 (3 of 3)

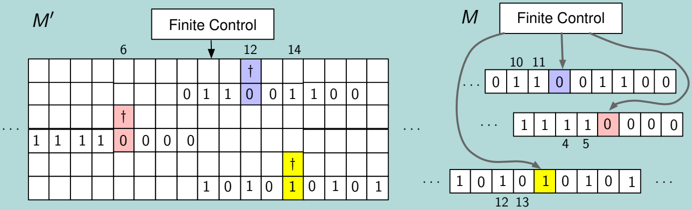
- At this stage (once the i in $(q, i, [\gamma_1, \cdots, \gamma_k])$ is k and all γ_j are set), M' knows the head symbols M will have read, and knows what actions to take.



Multi-tape TMs

Proof of Theorem 8.2.1 (3 of 3)

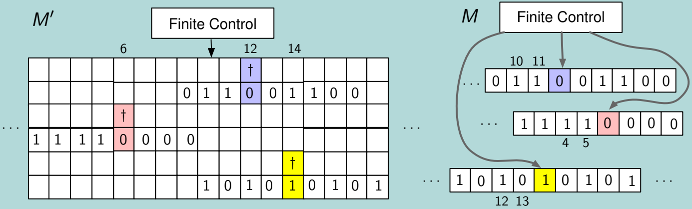
- At this stage (once the i in $(q, i, [\gamma_1, \cdots, \gamma_k])$ is k and all γ_j are set), M' knows the head symbols M will have read, and knows what actions to take.
- It then sweeps left making appropriate changes to the tracks (just like M does to its tape) each time a † is encountered. M' also moves the †s accordingly.



Multi-tape TMs

Proof of Theorem 8.2.1 (3 of 3)

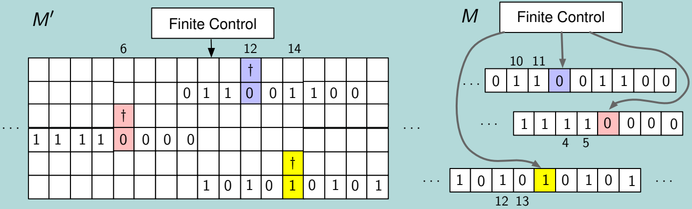
- At this stage (once the i in $(q, i, [\gamma_1, \cdots, \gamma_k])$ is k and all γ_j are set), M' knows the head symbols M will have read, and knows what actions to take.
- It then sweeps left making appropriate changes to the tracks (just like M does to its tape) each time a † is encountered. M' also moves the †s accordingly.
- The left sweep ends when the second component is zero. At this time, M' would have completed moving the †s and the track contents; they'll now match those of M.



Multi-tape TMs

Proof of Theorem 8.2.1 (3 of 3)

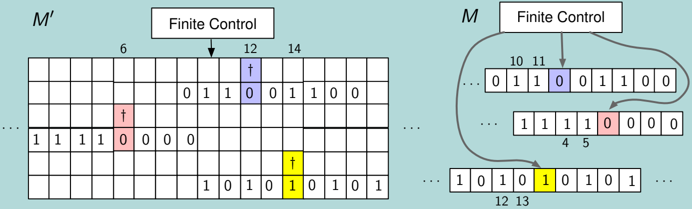
- At this stage (once the i in $(q, i, [\gamma_1, \cdots, \gamma_k])$ is k and all γ_j are set), M' knows the head symbols M will have read, and knows what actions to take.
- It then sweeps left making appropriate changes to the tracks (just like M does to its tape) each time a † is encountered. M' also moves the †s accordingly.
- The left sweep ends when the second component is zero. At this time, M' would have completed moving the †s and the track contents; they'll now match those of M.
- M' then moves the state to $(q', 0, [?, \cdots, ?])$ and starts the next sweep if q' is not a final state.



Multi-tape TMs

Proof of Theorem 8.2.1 (3 of 3)

- At this stage (once the i in $(q, i, [\gamma_1, \cdots, \gamma_k])$ is k and all γ_j are set), M' knows the head symbols M will have read, and knows what actions to take.
- It then sweeps left making appropriate changes to the tracks (just like M does to its tape) each time a † is encountered. M' also moves the †s accordingly.
- The left sweep ends when the second component is zero. At this time, M' would have completed moving the †s and the track contents; they'll now match those of M.
- M' then moves the state to $(q', 0, [?, \cdots, ?])$ and starts the next sweep if q' is not a final state.
- Note that M' mimics M and hence the languages accepted are identical.



Multi-tape TMs

- The running time of a TM M with input w is the number of moves M makes before it halts. (If it does not, the running time is ∞).

Multi-tape TMs

- The running time of a TM M with input w is the number of moves M makes before it halts. (If it does not, the running time is ∞).
- The time complexity $T_M : \{0, 1, \ldots\} \to \{0, 1, \ldots\} \cup \{\infty\}$ of a TM M is defined as follows:
  - $T_M(n) :=$ maximum running time of M for an input w of length n symbols.

Multi-tape TMs

- The running time of a TM M with input w is the number of moves M makes before it halts. (If it does not, the running time is ∞).
- The time complexity $T_M : \{0, 1, \ldots\} \to \{0, 1, \ldots\} \cup \{\infty\}$ of a TM M is defined as follows:
  - $T_M(n) :=$ maximum running time of M for an input w of length n symbols.

Theorem 8.2.2

The time taken for M' in Theorem 8.2.1 to process n moves of M is $O(n^2)$.

Multi-tape TMs

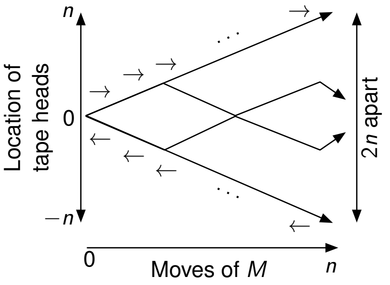
- The running time of a TM M with input w is the number of moves M makes before it halts. (If it does not, the running time is ∞).
- The time complexity $T_M : \{0, 1, \ldots\} \to \{0, 1, \ldots\} \cup \{\infty\}$ of a TM M is defined as follows:
  - $T_M(n) :=$ maximum running time of M for an input w of length n symbols.

Theorem 8.2.2

The time taken for M' in Theorem 8.2.1 to process n moves of M is $O(n^2)$.

Outline of Proof of Theorem 8.2.2

- In the ith move of M, any two heads of M can be at most $2i$ locations apart.



Multi-tape TMs

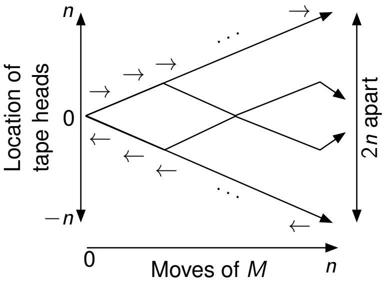
- The running time of a TM M with input w is the number of moves M makes before it halts. (If it does not, the running time is ∞).
- The time complexity $T_M : \{0, 1, \ldots\} \to \{0, 1, \ldots\} \cup \{\infty\}$ of a TM M is defined as follows:
  - $T_M(n) :=$ maximum running time of M for an input w of length n symbols.

Theorem 8.2.2

The time taken for M' in Theorem 8.2.1 to process n moves of M is $O(n^2)$.

Outline of Proof of Theorem 8.2.2

- In the ith move of M, any two heads of M can be at most $2i$ locations apart.
- Each sweep then requires $4i$ moves of M'.



Multi-tape TMs

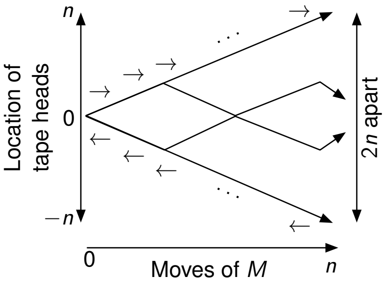
- The running time of a TM M with input w is the number of moves M makes before it halts. (If it does not, the running time is ∞).
- The time complexity $T_M : \{0, 1, \ldots\} \to \{0, 1, \ldots\} \cup \{\infty\}$ of a TM M is defined as follows:
  - $T_M(n) :=$ maximum running time of M for an input w of length n symbols.

Theorem 8.2.2

The time taken for M' in Theorem 8.2.1 to process n moves of M is $O(n^2)$.

Outline of Proof of Theorem 8.2.2

- In the ith move of M, any two heads of M can be at most $2i$ locations apart.
- Each sweep then requires $4i$ moves of M'.
- Each track update requires $\Theta(k)$ time.



Multi-tape TMs

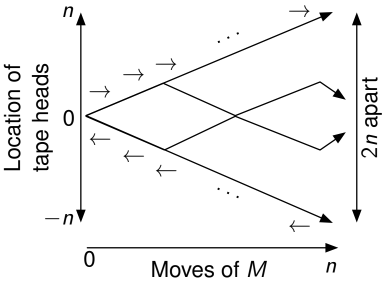
- The running time of a TM M with input w is the number of moves M makes before it halts. (If it does not, the running time is ∞).
- The time complexity $T_M : \{0, 1, \ldots\} \to \{0, 1, \ldots\} \cup \{\infty\}$ of a TM M is defined as follows:
  - $T_M(n) :=$ maximum running time of M for an input w of length n symbols.

Theorem 8.2.2

The time taken for M' in Theorem 8.2.1 to process n moves of M is $O(n^2)$.

Outline of Proof of Theorem 8.2.2

- In the ith move of M, any two heads of M can be at most $2i$ locations apart.
- Each sweep then requires $4i$ moves of M'.
- Each track update requires $\Theta(k)$ time.
- So n moves in M need $O(n^2)$ moves in M'.



Non-deterministic TMs

Non-deterministic TM: $\delta(q, X)$ is a set of triples representing possible moves.

Non-deterministic TMs

Non-deterministic TM: $\delta(q, X)$ is a set of triples representing possible moves.

Theorem 8.2.3

For every non-deterministic TM M, there is a TM N such that $L(M) = L(N)$.

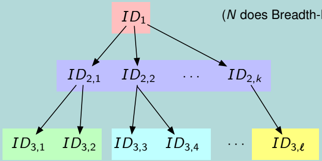
Non-deterministic TMs

Non-deterministic TM: $\delta(q, X)$ is a set of triples representing possible moves.

Theorem 8.2.3

For every non-deterministic TM M, there is a TM N such that $L(M) = L(N)$.

Outline of Proof of Theorem 8.2.3



(N does Breadth-First exploration of IDs of M)

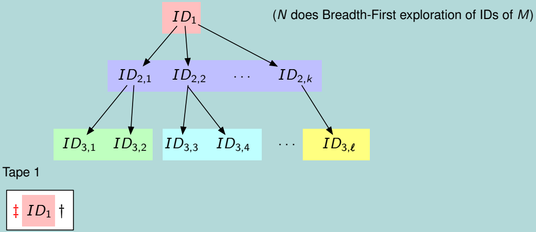
Non-deterministic TMs

Non-deterministic TM: $\delta(q, X)$ is a set of triples representing possible moves.

Theorem 8.2.3

For every non-deterministic TM M, there is a TM N such that $L(M) = L(N)$.

Outline of Proof of Theorem 8.2.3



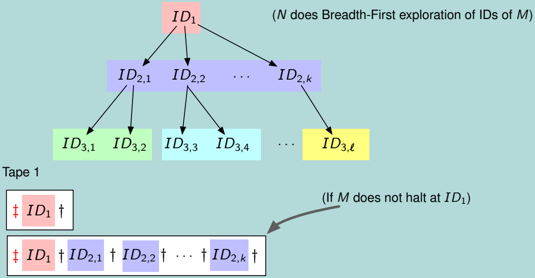
Non-deterministic TMs

Non-deterministic TM: $\delta(q, X)$ is a set of triples representing possible moves.

Theorem 8.2.3

For every non-deterministic TM M, there is a TM N such that $L(M) = L(N)$.

Outline of Proof of Theorem 8.2.3



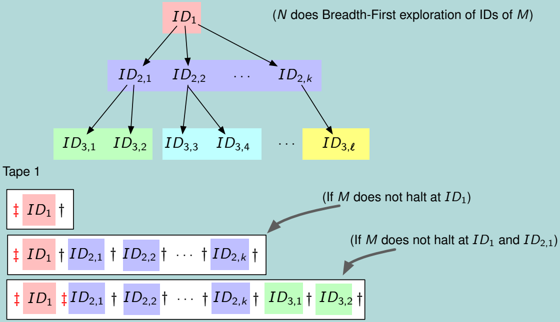
Non-deterministic TMs

Non-deterministic TM: $\delta(q, X)$ is a set of triples representing possible moves.

Theorem 8.2.3

For every non-deterministic TM M, there is a TM N such that $L(M) = L(N)$.

Outline of Proof of Theorem 8.2.3



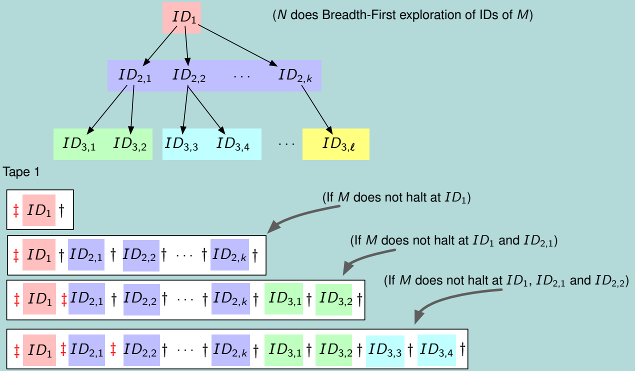
Non-deterministic TMs

Non-deterministic TM: $\delta(q, X)$ is a set of triples representing possible moves.

Theorem 8.2.3

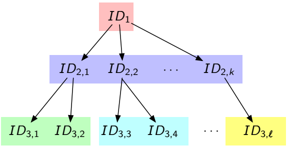
For every non-deterministic TM M, there is a TM N such that $L(M) = L(N)$.

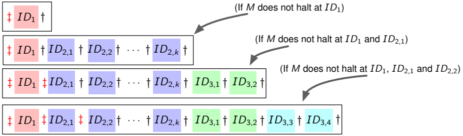
Outline of Proof of Theorem 8.2.3



Outline of Proof of Theorem 8.2.3

- We can devise a 2-tape TM N that simulates M.
- N first replaces the content of the first tape by ‡ followed by the ID that M is initially in, which is then followed by a special symbol †, which serves as ID separator. (N uses the second tape as scratch tape to enable this operation).
- If the ID corresponds to a final state, N accepts (as would M).
- If not, N then identifies all possible choices for the next IDs for M and enters each one of them followed by † at the right end of its first tape. (Again, N uses the second tape as scratch tape to enable this operation.)
- N then searches for † to the right of ‡, changes the † to a ‡ (to signify that it is processing the succeeding ID), and processes that ID in the similar way.
- N halts at an ID iff M would at that ID.





TM Semi-infinite Tape

A TM with a semi-infinite tape is a TM that only has blanks on one of its sides, but not on the other.

Phrased (slightly) more formally:
A TM with a semi-infinite tape is a TM that can never move to left of the left-most input symbol.

We don't provide a formal definition, but a way of simulating this is by providing a special symbol, placed on the left of the input, and defining the transitions to always go to the right when this is read.

TM Semi-infinite Tape

Theorem 8.3.1

Every recursively enumerable language is also accepted by a TM with semi-infinite tape.

TM Semi-infinite Tape

Theorem 8.3.1

Every recursively enumerable language is also accepted by a TM with semi-infinite tape.

Outline of Proof of Theorem 8.3.1

TM Semi-infinite Tape

Theorem 8.3.1

Every recursively enumerable language is also accepted by a TM with semi-infinite tape.

Outline of Proof of Theorem 8.3.1

- Given a TM M that accepts a language L, construct a two-track TM M' as follows.

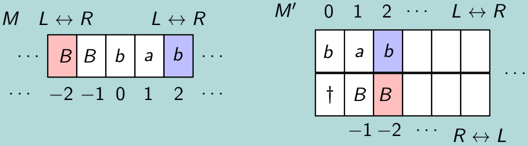
TM Semi-infinite Tape

Theorem 8.3.1

Every recursively enumerable language is also accepted by a TM with semi-infinite tape.

Outline of Proof of Theorem 8.3.1

- Given a TM M that accepts a language L, construct a two-track TM M' as follows.
- The first/second tracks of M' are the right/left parts of the tape of M.



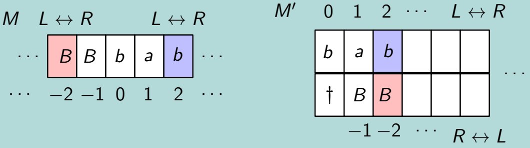
TM Semi-infinite Tape

Theorem 8.3.1

Every recursively enumerable language is also accepted by a TM with semi-infinite tape.

Outline of Proof of Theorem 8.3.1

- Given a TM M that accepts a language L, construct a two-track TM M' as follows.
- The first/second tracks of M' are the right/left parts of the tape of M.
- First, write a special symbol, say † at the leftmost part of the second track; this indicates to M' that a left move is not to be attempted at this location.



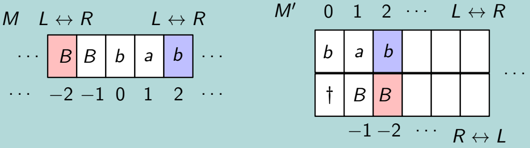
TM Semi-infinite Tape

Theorem 8.3.1

Every recursively enumerable language is also accepted by a TM with semi-infinite tape.

Outline of Proof of Theorem 8.3.1

- Given a TM M that accepts a language L, construct a two-track TM M' as follows.
- The first/second tracks of M' are the right/left parts of the tape of M.
- First, write a special symbol, say † at the leftmost part of the second track; this indicates to M' that a left move is not to be attempted at this location.
- At any time, M' keeps track of whether M is to the right or left of its start location.



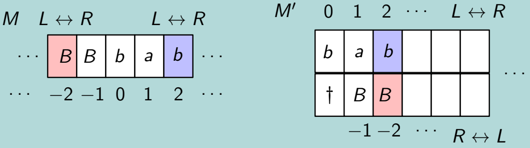
TM Semi-infinite Tape

Theorem 8.3.1

Every recursively enumerable language is also accepted by a TM with semi-infinite tape.

Outline of Proof of Theorem 8.3.1

- Given a TM M that accepts a language L, construct a two-track TM M' as follows.
- The first/second tracks of M' are the right/left parts of the tape of M.
- First, write a special symbol, say † at the leftmost part of the second track; this indicates to M' that a left move is not to be attempted at this location.
- At any time, M' keeps track of whether M is to the right or left of its start location.
- If M is to the strict right of its start location, M' mimics M on the first track. If M is to the strict left of its start location, M' mimics M on second track, but with the head directions reversed. M' detects the start by the † symbol.



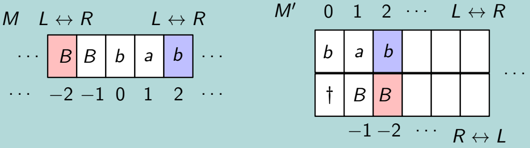
TM Semi-infinite Tape

Theorem 8.3.1

Every recursively enumerable language is also accepted by a TM with semi-infinite tape.

Outline of Proof of Theorem 8.3.1

- Given a TM M that accepts a language L, construct a two-track TM M' as follows.
- The first/second tracks of M' are the right/left parts of the tape of M.
- First, write a special symbol, say † at the leftmost part of the second track; this indicates to M' that a left move is not to be attempted at this location.
- At any time, M' keeps track of whether M is to the right or left of its start location.
- If M is to the strict right of its start location, M' mimics M on the first track. If M is to the strict left of its start location, M' mimics M on second track, but with the head directions reversed. M' detects the start by the † symbol.
- It can be formally shown that M' accepts a string iff M accepts it.



Multi-stack Machines

A multistack machne is a PDA with several independent stacks (i.e., one can be popping a symbol, while another is pushing a symbol).

Multi-stack Machines

A multistack machne is a PDA with several independent stacks (i.e., one can be popping a symbol, while another is pushing a symbol).

Theorem 8.4.1

Every recursively enumerable language is accepted by a two-stack PDA

Multi-stack Machines

A multistack machne is a PDA with several independent stacks (i.e., one can be popping a symbol, while another is pushing a symbol).

Theorem 8.4.1

Every recursively enumerable language is accepted by a two-stack PDA

Outline of Proof of Theorem 8.4.1

- Let each stack again contain a bottom-most start symbol.
- Let ID $= x_{-3}x_{-2}x_{-1}qx_0x_1x_2$, i.e., $w = x_{-3}x_{-2}x_{-1}x_0x_1x_2$, and head read reads x_0
- Let stack-1 be $x_0x_1x_2$ (top to bottom) the head and its right and stack-2 be $x_{-1}x_{-2}x_{-3}$ the head's left part in reversed order.
- What if we move the head to the right? Then, ID' $= x_{-3}x_{-2}x_{-1}x_0q'x_1x_2$. We can easily do this with our stacks:
  - How should the stack now look like?
  - stack-1: x_1x_2 and stack-2: $x_0x_{-1}x_{-2}x_{-3}$.
  - But that's just a simple pop and push!
- Moving to the left, and changing the symbol that's written can be simulated as well.

Multi-stack Machines

Outline of Proof of Theorem 8.4.1, cont'd

- Remaining problem: How to fill the stacks initially?
- Recall: stack-1 contains the head and its right and stack-2 the head's left part in reversed order.

Multi-stack Machines

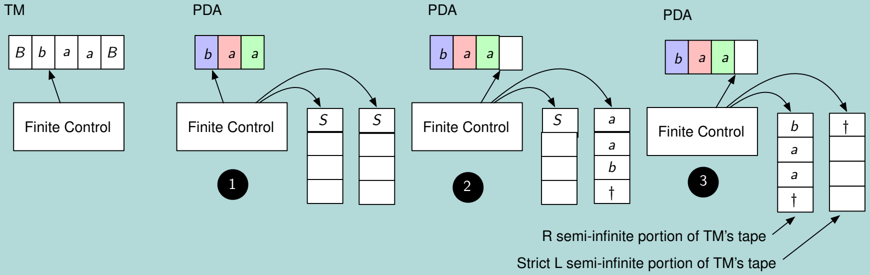
Outline of Proof of Theorem 8.4.1, cont'd

- Remaining problem: How to fill the stacks initially?
- Recall: stack-1 contains the head and its right and stack-2 the head's left part in reversed order.
- Initial configuration is $q_0 w$, so stack-1 should be w and stack-2 "empty".

Multi-stack Machines

Outline of Proof of Theorem 8.4.1, cont'd

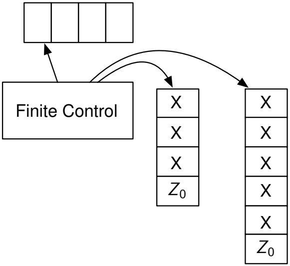
- Remaining problem: How to fill the stacks initially?
- Recall: stack-1 contains the head and its right and stack-2 the head's left part in reversed order.
- Initial configuration is $q_0 w$, so stack-1 should be w and stack-2 "empty".
- We achieve this by the following procedure:



- I.e., run to the right filling stack-2, then run back putting it on stack-1.

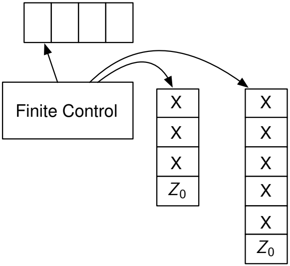
Counter Machines

- A counter machine is a multi-stack machine whose stack alphabet contains two symbols: Z_0 (stack end marker) and X
- Z_0 is initially in the stack.
- Z_0 may be replaced by $X^i Z_0$ for some $i \geq 0$
- X may be replaced by X^i for some $i \geq 0$.



Counter Machines

- A counter machine is a multi-stack machine whose stack alphabet contains two symbols: Z_0 (stack end marker) and X
- Z_0 is initially in the stack.
- Z_0 may be replaced by $X^i Z_0$ for some $i \geq 0$
- X may be replaced by X^i for some $i \geq 0$.
- A counter machine effectively stores a non-negative number.



Counter Machines

Theorem 8.4.2

Every recursively enumerable language is accepted by a three-counter machine

Outline of Proof of Theorem 8.4.2

Counter Machines

Theorem 8.4.2

Every recursively enumerable language is accepted by a three-counter machine

Outline of Proof of Theorem 8.4.2

- We know a two-stack PDA can simulate any TM.
- We'll show that a 3-counter machine can simulate any (two stack) PDA.

Counter Machines

Theorem 8.4.2

Every recursively enumerable language is accepted by a three-counter machine

Outline of Proof of Theorem 8.4.2

- We know a two-stack PDA can simulate any TM.
- We'll show that a 3-counter machine can simulate any (two stack) PDA.
- WLOG, let the stack alphabet of $\Gamma = \{0, 1, \ldots, r-1\}$.
- Suppose stack 1/2 contains Y_1(top)$, \ldots, Y_k$. Then counter stores $Y_1 + rY_2 + \cdots + r^{k-1}Y_k$. E.g., if stack is $1, 5, 7$, interpret it as 157.

Counter Machines

Theorem 8.4.2

Every recursively enumerable language is accepted by a three-counter machine

Outline of Proof of Theorem 8.4.2

- We know a two-stack PDA can simulate any TM.
- We'll show that a 3-counter machine can simulate any (two stack) PDA.
- WLOG, let the stack alphabet of $\Gamma = \{0, 1, \ldots, r-1\}$.
- Suppose stack 1/2 contains Y_1(top)$, \ldots, Y_k$. Then counter stores $Y_1 + rY_2 + \cdots + r^{k-1}Y_k$. E.g., if stack is $1, 5, 7$, interpret it as 157.
- The third counter is used to change the two stack contents.
- Popping the top symbol from a stack (say A) $=$ finding quotient when $Y_1 + rY_2 + \cdots + r^{k-1}Y_k$ is divided by r.
  - pop r X's from stack A, and push a single X on the third stack. Repeat until all Xs are exhausted on the stack where popping is performed.
  - Now empty stack A and copy the third stack contents onto stack A.
- Change Y_1 to some Y_1' requires adding or subtracting, which is done by popping or pushing the corresponding number of Xs.

Counter Machines

Outline of Proof of Theorem 8.4.2

- pushing a symbol Z onto a stack (say A) $=$ compute $rC + Z$ where C is the number presently stored in the stack A.
  - pop one X from stack A, and push r Xs on the third stack.
  - Finally push Z Xs onto the third stack. Now empty stack A and copy the third stack contents onto stack A.
- Since the above three are the only operations needed to simulate a TM on a two-stack PDA, we can stimulate a 2-stack PDA and hence a TM using a 3-counter machine.

Counter Machines

Outline of Proof of Theorem 8.4.2

- pushing a symbol Z onto a stack (say A) = compute $rC + Z$ where C is the number presently stored in the stack A.
  - pop one X from stack A, and push r Xs on the third stack.
  - Finally push Z Xs onto the third stack. Now empty stack A and copy the third stack contents onto stack A.
- Since the above three are the only operations needed to simulate a TM on a two-stack PDA, we can stimulate a 2-stack PDA and hence a TM using a 3-counter machine.

Theorem 8.4.3

Every recursively enumerable language is accepted by a two-counter machine

Outline of Proof of Theorem 8.4.3

- The key idea: simulate three counters using one, and use the other for manipulations.
- The first counter stores $2^i 3^j 5^k$ where i, j, k are the contents of the 3-counter machine.
- Updates to the stack involve: (a) divide by 2, 3, or 5; (b) multiply by 2, 3, or 5; or (c) identify if i or j or k is zero (check divisibility).
- Each operation can be easily seen to be done with a spare counter.