Foundations of Computing

Lecture 12

Arkady Yerukhimovich

February 27, 2024

Outline

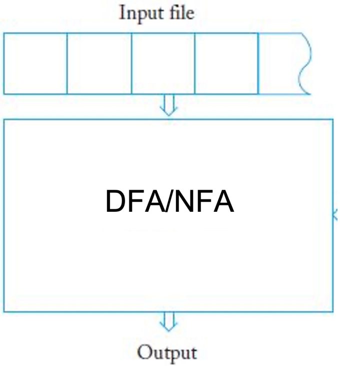
1. Lecture 10+11 Review
2. Models of Computation
3. The Turing Machine
4. Formalizing Turing Machines

Lecture 10+11 Review

- Equivalence of CFGs and PDAs
- CFL Pumping Lemma
- Using the CFL Pumping Lemma

Outline

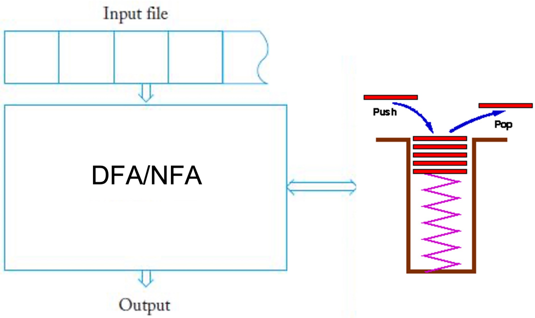
Finite Automata



Recall:

- An NFA/DFA has no external storage
- Only memory must be encoded in the finite number of states
- Can only recognize regular languages

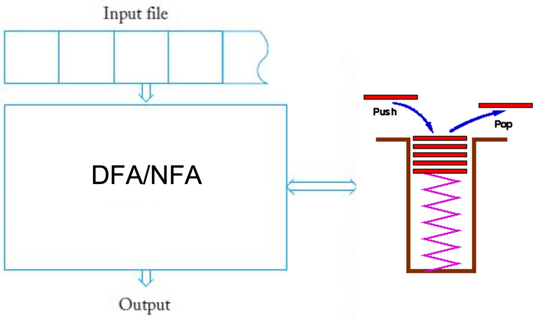
Pushdown Automata (PDA)



A PDA consists of:

- An NFA for a control unit
- A Stack for storage

Pushdown Automata (PDA)



A PDA consists of:

- An NFA for a control unit
- A Stack for storage

Recall:

- Can only access memory in LIFO fashion
- Can only recognize context-free languages

A Model for General Computation

Question

All the prior models of computation couldn't recognize some simple languages. Can we develop a computation model that captures all languages that can be computed on any computer?

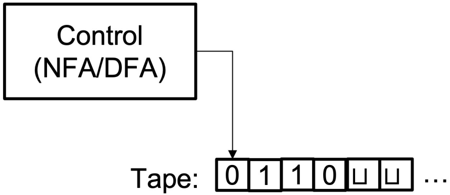
Our Goal

One model to rule them all!

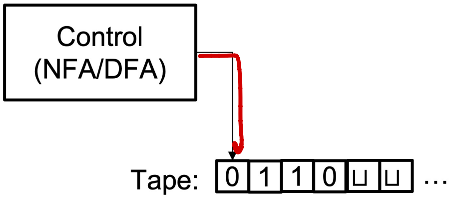
Outline

1. Lecture 10+11 Review
2. Models of Computation
3. The Turing Machine
4. Formalizing Turing Machines

The Turing Machine



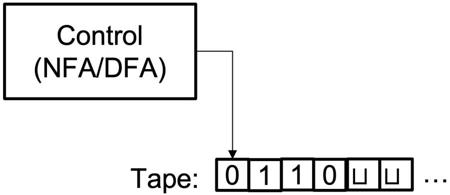
The Turing Machine



Key Differences:

- A TM can read and write to its tape

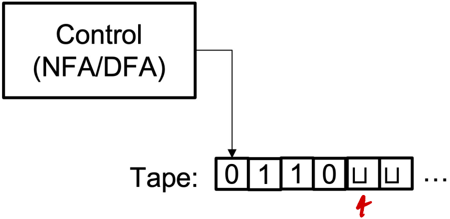
The Turing Machine



Key Differences:

- A TM can read and write to its tape
- The read/write head can move to the right and to the left

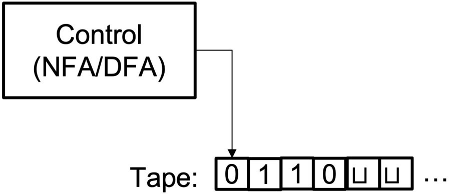
The Turing Machine



Key Differences:

- A TM can read and write to its tape
- The read/write head can move to the right and to the left
- No separate input tape, input written onto memory tape at start

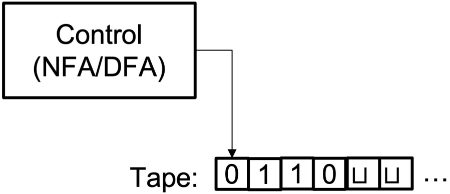
The Turing Machine



Key Differences:

- A TM can read and write to its tape
- The read/write head can move to the right and to the left
- No separate input tape, input written onto memory tape at start
- The memory tape is infinite

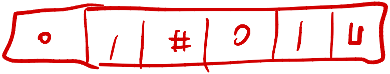
The Turing Machine



Key Differences:

- A TM can read and write to its tape
- The read/write head can move to the right and to the left
- No separate input tape, input written onto memory tape at start
- The memory tape is infinite
- Control FA has accept and reject states that are immediately output if entered

An Algorithm for M:
On input string s (written on the tape):



An Example: TM To Recognize $L = \{w\#w \mid w \in \{0,1\}^*\}$

An Algorithm for M:

On input string s (written on the tape):

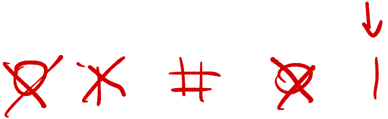
1. Scan the input to check that it contains exactly one # symbol, if not reject.

An Example: TM To Recognize $L = \{w\#w \mid w \in \{0,1\}^*\}$

An Algorithm for M:

On input string s (written on the tape):

1. Scan the input to check that it contains exactly one # symbol, if not reject.
2. Zigzag to corresponding positions on each side of the # and see if they contain same symbol. If not, reject. Cross off symbols as they are checked



An Example: TM To Recognize $L = \{w\#w \mid w \in \{0,1\}^*\}$

An Algorithm for M:

On input string s (written on the tape):

1. Scan the input to check that it contains exactly one # symbol, if not reject.
2. Zigzag to corresponding positions on each side of the # and see if they contain same symbol. If not, reject. Cross off symbols as they are checked
3. When all symbols to the left of # have been crossed off, check that no uncrossed-off symbols remain to the right of #. If any symbols remain, reject, otherwise accept.

An Example: TM To Recognize $L = \{w\#w \mid w \in \{0,1\}^*\}$

Recognizing $s = 011000\#011000$:

An Example: TM To Recognize $L = \{w\#w \mid w \in \{0,1\}^*\}$

Recognizing $s = 011000\#011000$:

$$\textcolor{red}{0}11000\#011000 \sqcup \cdots$$

An Example: TM To Recognize $L = \{w\#w \mid w \in \{0,1\}^*\}$

Recognizing $s = 011000\#011000$:

$$\textcolor{red}{0}11000\#011000 \sqcup \cdots$$

$$x11000\#\textcolor{red}{0}11000 \sqcup \cdots$$

An Example: TM To Recognize $L = \{w\#w \mid w \in \{0,1\}^*\}$

Recognizing $s = 011000\#011000$:

$$\textcolor{red}{0}11000\#011000 \sqcup \cdots$$

$$x11000\#\textcolor{red}{0}11000 \sqcup \cdots$$

$$x\textcolor{red}{1}1000\#x11000 \sqcup \cdots$$

An Example: TM To Recognize $L = \{w\#w \mid w \in \{0,1\}^*\}$

Recognizing $s = 011000\#011000$:

$$\textcolor{red}{0}11000\#011000\sqcup\cdots$$
$$x11000\#\textcolor{red}{0}11000\sqcup\cdots$$
$$x\textcolor{red}{1}1000\#x11000\sqcup\cdots$$
$$xx1000\#x\textcolor{red}{1}1000\sqcup\cdots$$

An Example: TM To Recognize $L = \{w\#w \mid w \in \{0,1\}^*\}$

Recognizing $s = 011000\#011000$:

$$\textcolor{red}{0}11000\#011000\sqcup\cdots$$
$$x11000\#\textcolor{red}{0}11000\sqcup\cdots$$
$$x\textcolor{red}{1}1000\#x11000\sqcup\cdots$$
$$xx1000\#x\textcolor{red}{1}1000\sqcup\cdots$$
$$\cdots$$
$$xxxxxx\#xxxxxx\textcolor{red}{\sqcup}\cdots$$

An Example: TM To Recognize $L = \{w\#w \mid w \in \{0,1\}^*\}$

Recognizing $s = 011000\#011000$:

$$\textcolor{red}{0}11000\#011000 \sqcup \cdots$$
$$x11000\#\textcolor{red}{0}11000 \sqcup \cdots$$
$$x\textcolor{red}{1}1000\#x11000 \sqcup \cdots$$
$$xx1000\#x\textcolor{red}{1}1000 \sqcup \cdots$$
$$\cdots$$
$$xxxxxx\#xxxxxx\textcolor{red}{\sqcup} \cdots$$
$$accept$$

What is an algorithm?

Algorithms

What is an algorithm?

- A collection of simple instructions for carrying out some task

Algorithms

What is an algorithm?

- A collection of simple instructions for carrying out some task
- A process according to which it can be determined by a finite number of operations – Hilbert 1900

Algorithms

What is an algorithm?

- A collection of simple instructions for carrying out some task
- A process according to which it can be determined by a finite number of operations – Hilbert 1900

Algorithms are critical to understand solutions / complexity of a problem

Algorithms

What is an algorithm?

- A collection of simple instructions for carrying out some task
- A process according to which it can be determined by a finite number of operations – Hilbert 1900

Algorithms are critical to understand solutions / complexity of a problem

- To show how to solve a problem, we design an algorithm

Algorithms

What is an algorithm?

- A collection of simple instructions for carrying out some task
- A process according to which it can be determined by a finite number of operations – Hilbert 1900

Algorithms are critical to understand solutions / complexity of a problem

- To show how to solve a problem, we design an algorithm
- To reason about languages accepted by NFA/PDA, we designed algorithms

Algorithms

What is an algorithm?

- A collection of simple instructions for carrying out some task
- A process according to which it can be determined by a finite number of operations – Hilbert 1900

Algorithms are critical to understand solutions / complexity of a problem

- To show how to solve a problem, we design an algorithm
- To reason about languages accepted by NFA/PDA, we designed algorithms
- How can we reason about the limits of what an algorithm can compute?

Turing Machines and Algorithms

Church-Turing Thesis (1936)

Anything that can be computed by an algorithm can be computed by a Turing Machine

Turing Machines and Algorithms

Church-Turing Thesis (1936)

Anything that can be computed by an algorithm can be computed by a Turing Machine

Observations:

Turing Machines and Algorithms

Church-Turing Thesis (1936)

Anything that can be computed by an algorithm can be computed by a Turing Machine

Observations:

- While unproven, all modern computers satisfy Church-Turing thesis

Turing Machines and Algorithms

Church-Turing Thesis (1936)

Anything that can be computed by an algorithm can be computed by a Turing Machine

Observations:

- While unproven, all modern computers satisfy Church-Turing thesis
- To prove that some problem cannot be solved by an algorithm, enough to reason about Turing Machines

Turing Machines and Algorithms

Church-Turing Thesis (1936)

Anything that can be computed by an algorithm can be computed by a Turing Machine

Observations:

- While unproven, all modern computers satisfy Church-Turing thesis
- To prove that some problem cannot be solved by an algorithm, enough to reason about Turing Machines
- This means that Turing Machines give an abstraction to capture “feasible computation”

Turing Machines and Algorithms

Church-Turing Thesis (1936)

Anything that can be computed by an algorithm can be computed by a Turing Machine

Observations:

- While unproven, all modern computers satisfy Church-Turing thesis
- To prove that some problem cannot be solved by an algorithm, enough to reason about Turing Machines
- This means that Turing Machines give an abstraction to capture "feasible computation"

Outline

Turing Machine – Formal Definition

A Turing machine is a 7-tuple:

Turing Machine – Formal Definition

A Turing machine is a 7-tuple:

1. Q – set of states

Turing Machine – Formal Definition

A Turing machine is a 7-tuple:

1. Q – set of states
2. Σ – input alphabet (not including blank symbol $\sqcup$)

Turing Machine – Formal Definition

A Turing machine is a 7-tuple:

1. Q – set of states
2. Σ – input alphabet (not including blank symbol $\sqcup$)
3. Γ – tape alphabet, where $\sqcup \in \Gamma$ and $\Sigma \subseteq \Gamma$

Turing Machine – Formal Definition

A Turing machine is a 7-tuple:

1. Q – set of states
2. Σ – input alphabet (not including blank symbol $\sqcup$)
3. Γ – tape alphabet, where $\sqcup \in \Gamma$ and $\Sigma \subseteq \Gamma$
4. $\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$ – transition function

Turing Machine – Formal Definition

A Turing machine is a 7-tuple:

1. Q – set of states
2. Σ – input alphabet (not including blank symbol $\sqcup$)
3. Γ – tape alphabet, where $\sqcup \in \Gamma$ and $\Sigma \subseteq \Gamma$
4. $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$ – transition function
5. $q_0 \in Q$ – start state
6. $q_{accept} \in Q$ – accept state
7. $q_{reject} \in Q$ – reject state

Transition function: $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$

Turing Machine – Formal Definition

A Turing machine is a 7-tuple:

1. Q – set of states
2. Σ – input alphabet (not including blank symbol $\sqcup$)
3. Γ – tape alphabet, where $\sqcup \in \Gamma$ and $\Sigma \subseteq \Gamma$
4. $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$ – transition function
5. $q_0 \in Q$ – start state
6. $q_{accept} \in Q$ – accept state
7. $q_{reject} \in Q$ – reject state

Transition function: $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$
On state q and tape input γ:

Turing Machine – Formal Definition

A Turing machine is a 7-tuple:

1. Q – set of states
2. Σ – input alphabet (not including blank symbol $\sqcup$)
3. Γ – tape alphabet, where $\sqcup \in \Gamma$ and $\Sigma \subseteq \Gamma$
4. $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$ – transition function
5. $q_0 \in Q$ – start state
6. $q_{accept} \in Q$ – accept state
7. $q_{reject} \in Q$ – reject state

Transition function: $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$
On state q and tape input γ:

- move control to state q',

Turing Machine – Formal Definition

A Turing machine is a 7-tuple:

1. Q – set of states
2. Σ – input alphabet (not including blank symbol $\sqcup$)
3. Γ – tape alphabet, where $\sqcup \in \Gamma$ and $\Sigma \subseteq \Gamma$
4. $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$ – transition function
5. $q_0 \in Q$ – start state
6. $q_{accept} \in Q$ – accept state
7. $q_{reject} \in Q$ – reject state

Transition function: $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$
On state q and tape input γ:

- move control to state q',
- write γ' to the tape,

Turing Machine – Formal Definition

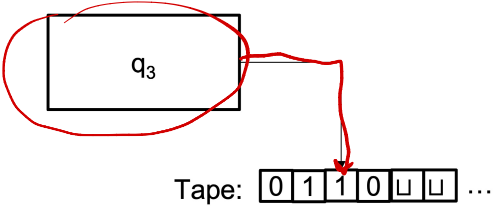
A Turing machine is a 7-tuple:

1. Q – set of states
2. Σ – input alphabet (not including blank symbol $\sqcup$)
3. Γ – tape alphabet, where $\sqcup \in \Gamma$ and $\Sigma \subseteq \Gamma$
4. $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$ – transition function
5. $q_0 \in Q$ – start state
6. $q_{accept} \in Q$ – accept state
7. $q_{reject} \in Q$ – reject state

Transition function: $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$
On state q and tape input γ:

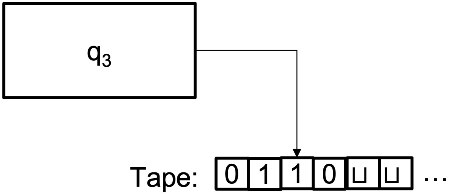
- move control to state q',
- write γ' to the tape,
- and move the tape head one spot to either Left or Right

Computing on a Turing Machine



Configuration of a TM

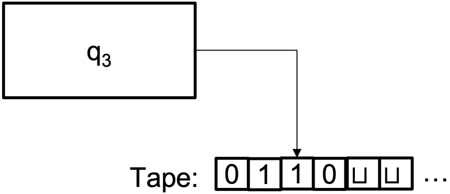
Computing on a Turing Machine



Configuration of a TM

- Describes the state of a TM computation

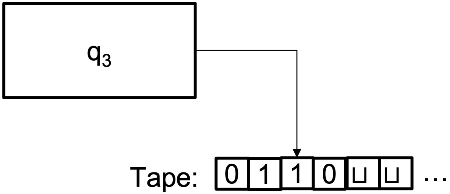
Computing on a Turing Machine



Configuration of a TM

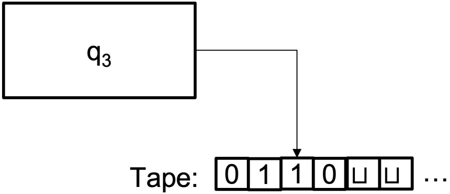
- Describes the state of a TM computation
- Current state of control, state of tape, location of tape head

Computing on a Turing Machine



Configuration of a TM

- Describes the state of a TM computation
- Current state of control, state of tape, location of tape head
- Example: $01q_310$



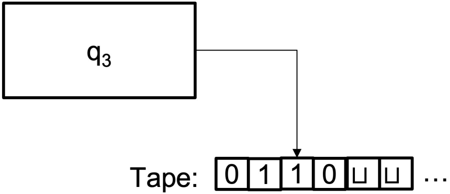
Configuration of a TM

- Describes the state of a TM computation
- Current state of control, state of tape, location of tape head
- Example: $01q_310$

Definitions:

- Configuration C_1 **yields** C_2, if M can go from C_1 to C_2 in a single step

Computing on a Turing Machine

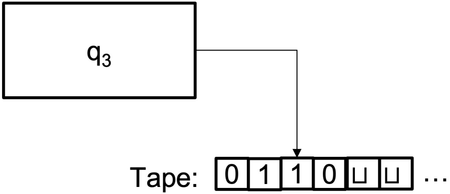


Configuration of a TM

- Describes the state of a TM computation
- Current state of control, state of tape, location of tape head
- Example: $01q_310$

Definitions:

- Configuration C_1 **yields** C_2, if M can go from C_1 to C_2 in a single step
- start configuration of M on input s – configuration q_0s



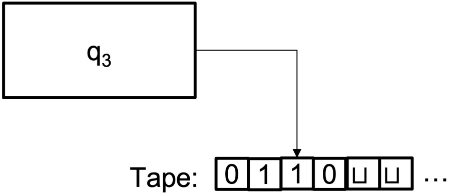
Configuration of a TM

- Describes the state of a TM computation
- Current state of control, state of tape, location of tape head
- Example: $01q_310$

Definitions:

- Configuration C_1 **yields** C_2, if M can go from C_1 to C_2 in a single step
- start configuration of M on input s – configuration q_0s
- accepting configuration – any config with state q_{accept}

Computing on a Turing Machine



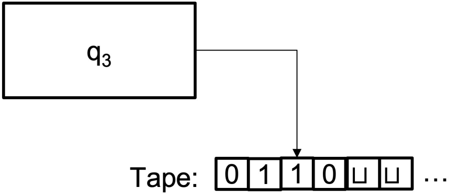
Configuration of a TM

- Describes the state of a TM computation
- Current state of control, state of tape, location of tape head
- Example: $01q_310$

Definitions:

- Configuration C_1 **yields** C_2, if M can go from C_1 to C_2 in a single step
- start configuration of M on input s – configuration q_0s
- accepting configuration – any config with state q_{accept}
- rejecting configuration – any config with state q_{reject}

Computing on a Turing Machine



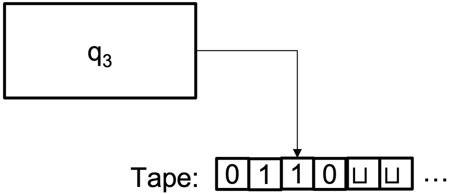
Configuration of a TM

- Describes the state of a TM computation
- Current state of control, state of tape, location of tape head
- Example: $01q_310$

Definitions:

- Configuration C_1 **yields** C_2, if M can go from C_1 to C_2 in a single step
- start configuration of M on input s – configuration q_0s
- accepting configuration – any config with state q_{accept}
- rejecting configuration – any config with state q_{reject}
- halting configuration – accepting or rejecting configs

Computing on a Turing Machine



Configuration of a TM

- Describes the state of a TM computation
- Current state of control, state of tape, location of tape head
- Example: $01q_310$

Definitions:

- Configuration C_1 **yields** C_2, if M can go from C_1 to C_2 in a single step
- start configuration of M on input s – configuration q_0s
- accepting configuration – any config with state q_{accept}
- rejecting configuration – any config with state q_{reject}
- halting configuration – accepting or rejecting configs

Computing on a Turing Machine

A TM accepts an input s if there exists a sequence of configs $C_1, C_2, \ldots, C_k$ where

Computing on a Turing Machine

A TM accepts an input s if there exists a sequence of configs $C_1, C_2, \ldots, C_k$ where

1. C_1 is the start configuration of M on input s

Computing on a Turing Machine

A TM accepts an input s if there exists a sequence of configs $C_1, C_2, \ldots, C_k$ where

1. C_1 is the start configuration of M on input s
2. Each C_i yields C_{i+1}

Computing on a Turing Machine

A TM accepts an input s if there exists a sequence of configs $C_1, C_2, \ldots, C_k$ where

1. C_1 is the start configuration of M on input s
2. Each C_i yields C_{i+1}
3. C_k is an accepting configuration

Computing on a Turing Machine

A TM accepts an input s if there exists a sequence of configs $C_1, C_2, \ldots, C_k$ where

1. C_1 is the start configuration of M on input s
2. Each C_i yields C_{i+1}
3. C_k is an accepting configuration

Language $L(M)$

The collection of strings that M accepts

Characterizing Computability of Languages

Characterizing Computability of Languages

Definition: Recursively enumerable languages

Characterizing Computability of Languages

Definition: Recursively enumerable languages

A language L is *Turing-recognizable* or *recursively enumerable* if some TM M recognizes it

Characterizing Computability of Languages

Definition: Recursively enumerable languages

A language L is *Turing-recognizable* or *recursively enumerable* if some TM M recognizes it

- M halts and accepts all strings in L

Characterizing Computability of Languages

Definition: Recursively enumerable languages

A language L is *Turing-recognizable* or *recursively enumerable* if some TM M recognizes it

- M halts and accepts all strings in L
- M may not halt on strings not in L – does not necessarily have to reject

Characterizing Computability of Languages

Definition: Recursively enumerable languages

A language L is *Turing-recognizable* or *recursively enumerable* if some TM M recognizes it

- M halts and accepts all strings in L
- M may not halt on strings not in L – does not necessarily have to reject

Definition: Decidable languages

A language L is *decidable* or *recursive* if some TM M decides it

Characterizing Computability of Languages

Definition: Recursively enumerable languages

A language L is *Turing-recognizable* or *recursively enumerable* if some TM M recognizes it

- M halts and accepts all strings in L
- M may not halt on strings not in L – does not necessarily have to reject

Definition: Decidable languages

A language L is *decidable* or *recursive* if some TM M decides it

- M halts on all inputs, accepting those in L and rejecting those not in L

Another Example

Consider $L = \{0^{2^n} \mid n \geq 0\}$

TM algorithm M for recognizing L:
On input s:

Another Example

Consider $L = \{0^{2^n} \mid n \geq 0\}$

TM algorithm M for recognizing L:
On input s:

1. If the tape has exactly one 0, accept

Another Example

Consider $L = \{0^{2^n} \mid n \geq 0\}$

TM algorithm M for recognizing L:
On input s:

1. If the tape has exactly one 0, accept
2. If the tape has an odd number of 0's, greater than 1, reject

Another Example

Consider $L = \{0^{2^n} \mid n \geq 0\}$

TM algorithm M for recognizing L:
On input s:

1. If the tape has exactly one 0, accept
2. If the tape has an odd number of 0's, greater than 1, reject
3. Sweep left to right across tape, crossing out every other 0

Another Example

Consider $L = \{0^{2^n} \mid n \geq 0\}$

TM algorithm M for recognizing L:
On input s:

1. If the tape has exactly one 0, accept
2. If the tape has an odd number of 0's, greater than 1, reject
3. Sweep left to right across tape, crossing out every other 0
4. Return the head to the left-hand end of the tape

Another Example

Consider $L = \{0^{2^n} \mid n \geq 0\}$

TM algorithm M for recognizing L:
On input s:

1. If the tape has exactly one 0, accept
2. If the tape has an odd number of 0's, greater than 1, reject
3. Sweep left to right across tape, crossing out every other 0
4. Return the head to the left-hand end of the tape
5. Go to step 1

Another Example

Consider $L = \{0^{2^n} \mid n \geq 0\}$

TM algorithm M for recognizing L:
On input s:

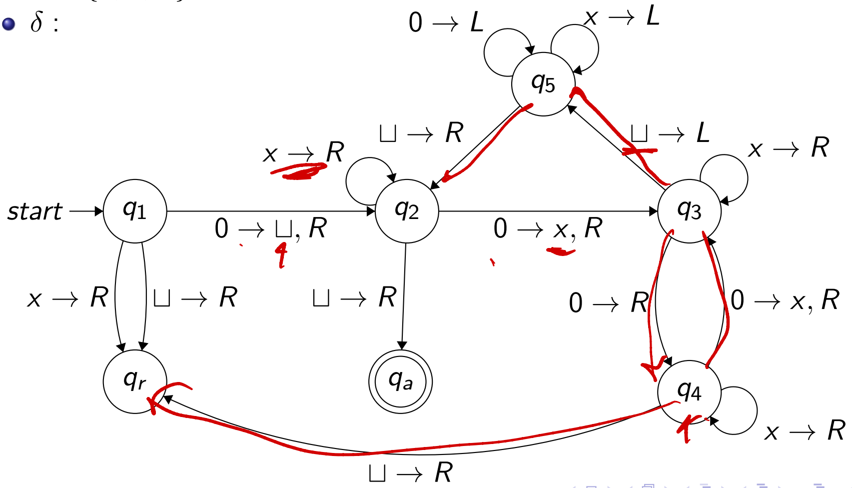
1. If the tape has exactly one 0, accept
2. If the tape has an odd number of 0's, greater than 1, reject
3. Sweep left to right across tape, crossing out every other 0
4. Return the head to the left-hand end of the tape
5. Go to step 1

Making M Formal

- $Q = \{q_1, q_2, q_3, q_4, q_5, q_a, q_r\}$
- $\Sigma = \{0\}$
- $\Gamma = \{0, x, \sqcup\}$
- δ :

Making M Formal

- $Q = \{q_1, q_2, q_3, q_4, q_5, q_a, q_r\}$
- $\Sigma = \{0\}$
- $\Gamma = \{0, x, \sqcup\}$
- δ :



Running M on $w = 0000$