Lecture 16: Decidable Languages

CSC 320: Foundations of Computer Science

Quinton Yong

quintonyong@uvic.ca

University of Victoria

Descriptions of Turing Machines

Formal Description:

- Fully describes TM's states, transition function, etc.

Implementation Description:

- Use English prose to describe how TM moves tapes head(s) and the way that it stores data on its tape(s)
- No details of states or transition function

High-level Description

- Use English prose to describe TM like an algorithm, ignoring implementation details
- No need to describe how TM manages tape or tape head

Encoding of TM Inputs

- The input to a TM is always an **input string** on the tape
- TM's can take **objects** as inputs, which need to be represented as a string

- In high-level descriptions, we use the notation $\langle \boldsymbol{O} \rangle$ to denote the **encoding of object $\boldsymbol{O}$ as a string**
- $\boldsymbol{O}$ can be any object such as a graph or even a DFA, PDA, or TM
- Encodings of several objects $\boldsymbol{O_1}, \boldsymbol{O_2}, \dots, \boldsymbol{O_k}$ is denoted $\langle \boldsymbol{O_1}, \boldsymbol{O_2}, \dots, \boldsymbol{O_k} \rangle$

Encoding of TM Inputs

Example 1: High level TM

M = "On input $\langle G \rangle$, where G is a graph,

… "

Example 2: Language description

- L: language consisting of **all strings representing undirected graphs that are connected**
- We write $L = \{ \langle G \rangle \mid G \text{ is a connected undirected graph} \}$

Encoding of G as a string

Decision Problem Languages

Recall:

- A **decision problem** is a problem with a **yes or no answer**
- Languages can be used to represent decision problems, where strings in the language are all **yes-instances** of the problem

Decision problem: Is the given undirected graph connected?

- Input: An undirected graph $\boldsymbol{G}$
- Language: $\boldsymbol{L} = \{\langle \boldsymbol{G} \rangle \mid \boldsymbol{G}$ is a connected undirected graph $\}$

Decision problem: Does the given undirected, weighted graph have a spanning tree with weight at most $\boldsymbol{k}$

- Input: An undirected, weighted graph $\boldsymbol{G}$ and positive integer $\boldsymbol{k}$
- Language: $\boldsymbol{L} = \{\langle \boldsymbol{G} = (\boldsymbol{V}, \boldsymbol{E}), \boldsymbol{k} \rangle \mid \boldsymbol{k}$ is a positive integer and $\boldsymbol{G}$ is an undirected, weighted graph which has a spanning tree of weight at most $\boldsymbol{k}$ $\}$

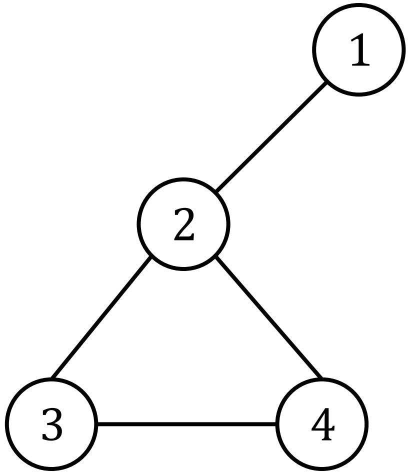
Encoding of Graph Inputs

To construct a TM which recognizes / decides languages such as

$$\boldsymbol{L} = \{\,\langle \boldsymbol{G} \rangle \mid \boldsymbol{G} \text{ is a connected undirected graph}\,\}$$

we need to **encode graphs as strings** to input into a TM.

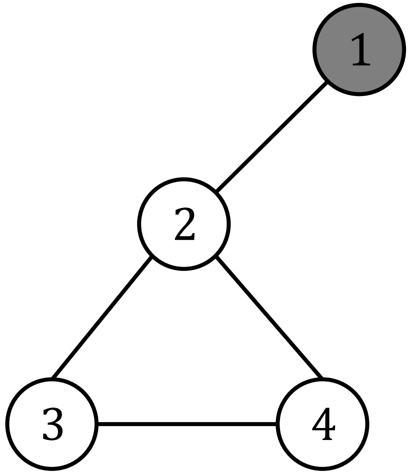
- We can encode a graph as follows:



$$\langle \boldsymbol{G} \rangle = \underbrace{(1, 2, 3, 4)}_{\text{vertices } \boldsymbol{V}}\underbrace{((1, 2), (2, 3), (2, 4), (3, 4))}_{\text{edges } \boldsymbol{E}}$$

- Tape head can move between $\boldsymbol{V}$ and $\boldsymbol{E}$ to "traverse" the graph
- TM can immediately **reject** if input is not in the correct format

High-level Description of Decider



Construct a **decider** for the following language:

$$\boldsymbol{L} = \{ \langle \boldsymbol{G} \rangle \mid \boldsymbol{G} \text{ is a connected undirected graph} \}$$

$M =$ "On input $\langle \boldsymbol{G} \rangle$:

1. Select first node of $\boldsymbol{G}$; **mark it**

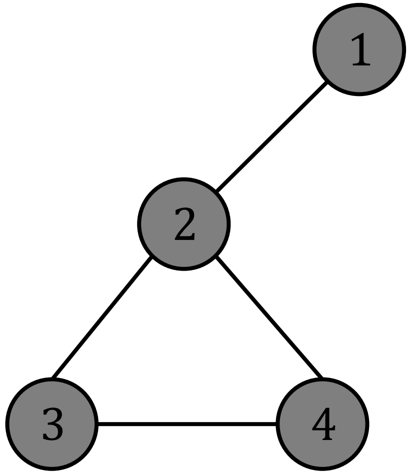
2. Repeat the following until no new nodes are marked:

   - For each node $\boldsymbol{v}$ in $\boldsymbol{G}$, **mark** $\boldsymbol{v}$ if it's incident to an edge whose other endpoint is already marked

3. Scan all nodes of $\boldsymbol{G}$

   - If all nodes are marked, **accept**
   - Otherwise, **reject**

High-level Description of Decider



Construct a **decider** for the following language:

$$\boldsymbol{L} = \{\, \langle \boldsymbol{G} \rangle \mid \boldsymbol{G} \text{ is a connected undirected graph} \,\}$$

$M =$ "On input $\langle \boldsymbol{G} \rangle$:

1. Select first node of $\boldsymbol{G}$; **mark it**

2. Repeat the following until no new nodes are marked:
   - For each node $\boldsymbol{v}$ in $\boldsymbol{G}$, **mark $\boldsymbol{v}$** if it's incident to an edge whose other endpoint is already marked

3. Scan all nodes of $\boldsymbol{G}$
   - If all nodes are marked, **accept**
   - Otherwise, **reject**

More Decidable Languages / Problems

- $A_{DFA} = \{\langle D, w\rangle \mid D \text{ is a } \mathbf{DFA} \text{ that accepts input string } \mathbf{w}\}$
  - Decider takes as input the encoding of a DFA D and a string w and **accepts** if the DFA D accepts the string w, or **rejects** otherwise

- $A_{NFA} = \{\langle N, w\rangle \mid N \text{ is an } \mathbf{NFA} \text{ that accepts input string } \mathbf{w}\}$

- $A_{REX} = \{\langle R, w\rangle \mid R \text{ is a regular expression that generates string } w\}$

- $E_{DFA} = \{\langle A\rangle \mid A \text{ is a } \mathbf{DFA} \text{ and } L(A) = \emptyset\}$

- $EQ_{DFA} = \{\langle A, B\rangle \mid A \text{ and } \mathbf{B} \text{ are } \mathbf{DFAs} \text{ and } L(A) = L(B)\}$

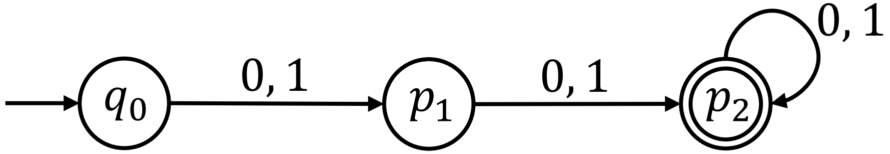
Encoding of DFAs, NFAs, PDAs, and TMs

- Certain Turing machines take DFAs, NFAs, PDAs, or TMs as input for the language that they recognize / decide

- Example: $\boldsymbol{A_{DFA}} = \{\langle \boldsymbol{D}, \boldsymbol{w} \rangle \mid \boldsymbol{D} \text{ is a } \mathbf{DFA} \text{ that accepts input string } \mathbf{w}\}$

- We will show how to encode a **DFA as a binary string**, and PDAs / TMs can be encoded in a similar way

Binary Encoding of a DFA



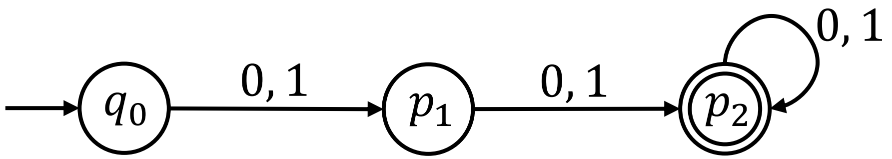
- $\boldsymbol{D} = (\boldsymbol{Q}, \boldsymbol{\Sigma}, \boldsymbol{\delta}, \boldsymbol{q_0}, \boldsymbol{F})$

  $= (\{q_0, p_1, p_2\}, \{0, 1\},$

  $\boldsymbol{\delta}(q_0, 0) = p_1, \boldsymbol{\delta}(q_0, 1) = p_1, \boldsymbol{\delta}(p_1, 0) = p_2, \boldsymbol{\delta}(p_1, 1) = p_2, \boldsymbol{\delta}(p_2, 0) = p_2, \boldsymbol{\delta}(p_2, 1) = p_2,$

  $q_0, \{p_2\})$

Binary Encoding of a DFA

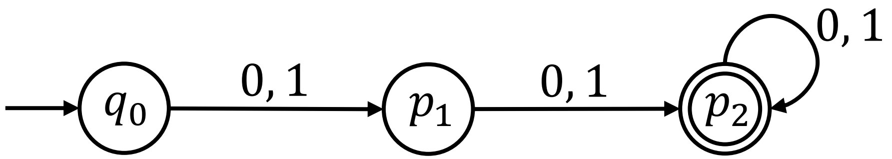


- States will be represented using unary notation: $\boldsymbol{q_0} \coloneqq \mathbf{1}$, $\boldsymbol{p_1} \coloneqq \mathbf{11}$, $\boldsymbol{p_2} \coloneqq \mathbf{111}$
- Alphabet will be represented using unary notation: $\mathbf{0} \coloneqq \mathbf{1}$, $\mathbf{1} \coloneqq \mathbf{11}$
- Items within $\boldsymbol{Q}, \boldsymbol{\Sigma}, \boldsymbol{\delta}, \boldsymbol{F}$ separated by $\mathbf{0}$
  - e.g. $\boldsymbol{Q} \coloneqq \underbrace{\mathbf{1}}_{q_0}\mathbf{0}\underbrace{\mathbf{11}}_{p_1}\mathbf{0}\underbrace{\mathbf{111}}_{p_2}$, $\boldsymbol{\Sigma} = \underbrace{\mathbf{1}}_{0}\mathbf{0}\underbrace{\mathbf{11}}_{1}$

Binary Encoding of a DFA

$\boldsymbol{Q}$: $\boldsymbol{q_0} \coloneqq \mathbf{1}$, $\boldsymbol{p_1} \coloneqq \mathbf{11}$, $\boldsymbol{p_2} \coloneqq \mathbf{111}$

$\boldsymbol{\Sigma}$: $\mathbf{0} \coloneqq \mathbf{1}$, $\mathbf{1} \coloneqq \mathbf{11}$

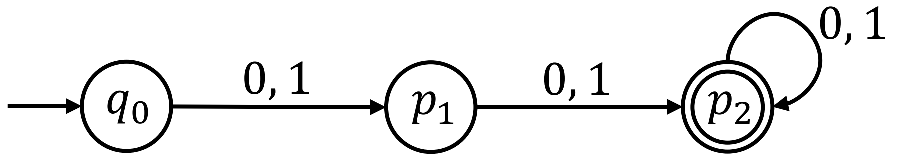


- Each transition in $\boldsymbol{\delta}$ can be encoded using 3 unary encodings (separated by $\mathbf{0}$'s)
  - $\boldsymbol{\delta}(\boldsymbol{q_0}, \mathbf{0}) = \boldsymbol{p_1}$ is encoded as $\underbrace{\mathbf{1}}_{\boldsymbol{q_0}}\mathbf{0}\underbrace{\mathbf{1}}_{\mathbf{0}}\mathbf{0}\underbrace{\mathbf{11}}_{\boldsymbol{p_1}}$
- All encoded transitions separated by $\mathbf{0}$'s (TM knows to read transition format)
  - $\boldsymbol{\delta}(q_0, 0) = p_1, \boldsymbol{\delta}(q_0, 1) = p_1, \boldsymbol{\delta}(p_1, 0) = p_2$ encoded $\underbrace{\mathbf{101011}}_{\boldsymbol{\delta}(q_0, 0) = p_1}\mathbf{0}\mathbf{1011011}\mathbf{0}\mathbf{11010111}$
- 5-tuple sections separated by $\mathbf{00}$
  - E.g. $(\boldsymbol{Q}, \boldsymbol{\Sigma}, \dots) \coloneqq \underbrace{\mathbf{10110111}}_{\boldsymbol{Q}}\mathbf{00}\underbrace{\mathbf{1011}}_{\boldsymbol{\Sigma}}\mathbf{00}\dots$

Binary Encoding of a DFA

$\boldsymbol{Q} : \boldsymbol{q_0} \coloneqq \mathbf{1},\ \boldsymbol{p_1} \coloneqq \mathbf{11},\ \boldsymbol{p_2} \coloneqq \mathbf{111}$

$\boldsymbol{\Sigma} : \mathbf{0} \coloneqq \mathbf{1},\ \mathbf{1} \coloneqq \mathbf{11}$



- $\boldsymbol{D} = (\boldsymbol{Q}, \boldsymbol{\Sigma}, \boldsymbol{\delta}, \boldsymbol{q_0}, \boldsymbol{F})$

  $= (\{q_0, p_1, p_2\}, \{0, 1\},$

  $\boldsymbol{\delta}(q_0, 0) = p_1, \boldsymbol{\delta}(q_0, 1) = p_1, \boldsymbol{\delta}(p_1, 0) = p_2, \boldsymbol{\delta}(p_1, 1) = p_2, \boldsymbol{\delta}(p_2, 0) = p_2, \boldsymbol{\delta}(p_2, 1) = p_2,$

  $q_0, \{p_2\})$

- Encoding:

$$\underbrace{10110111}_{\boldsymbol{Q}}00\underbrace{1011}_{\boldsymbol{\Sigma}}00\underbrace{101011010110110110101110110110111011101011101110110111}_{\boldsymbol{\delta}}00\underbrace{1}_{\boldsymbol{q_0}}00\underbrace{111}_{\boldsymbol{F}}$$

More Decidable Languages / Problems

- $\boxed{A_{DFA} = \{\langle D, w\rangle \mid D \text{ is a } \mathbf{DFA} \text{ that accepts input string } \mathbf{w}\}}$
  - Decider takes as input the encoding of a DFA D and a string w and **accepts** if the DFA D accepts the string w, or **rejects** otherwise

- $A_{NFA} = \{\langle N, w\rangle \mid N \text{ is an } \mathbf{NFA} \text{ that accepts input string } \mathbf{w}\}$

- $A_{REX} = \{\langle R, w\rangle \mid R \text{ is a regular expression that generates string } w\}$

- $E_{DFA} = \{\langle A\rangle \mid A \text{ is a } \mathbf{DFA} \text{ and } L(A) = \emptyset\}$

- $EQ_{DFA} = \{\langle A, B\rangle \mid A \text{ and } \mathbf{B} \text{ are } \mathbf{DFAs} \text{ and } L(A) = L(B)\}$

A_{DFA} is Decidable

Show that the language

$$A_{DFA} = \{\langle D, w\rangle \mid D \text{ is a } \mathbf{DFA} \text{ that accepts input string } \mathbf{w}\}$$

is decidable by constructing a TM M_1 that decides A_{DFA}

What the decider M_1 must do:

- M_1 takes as input the encoding of a DFA D and string w
- M_1 **accepts** if D accepts w
- M_1 **rejects** if D does not accept w
- M_1 must not infinitely loop

A_{DFA} is Decidable

Show that the language

$$A_{DFA} = \{\langle \boldsymbol{D}, \boldsymbol{w} \rangle \mid \boldsymbol{D} \text{ is a } \mathbf{DFA} \text{ that accepts input string } \mathbf{w}\}$$

is decidable by constructing a TM $\boldsymbol{M_1}$ that decides $\boldsymbol{A_{DFA}}$

High level description:

$\boldsymbol{M_1}$ = "On input $\langle \boldsymbol{D}, \boldsymbol{w} \rangle$, where $\boldsymbol{D}$ is a DFA and $\boldsymbol{w}$ is a string:

- Simulate DFA $\boldsymbol{D}$ on input $\boldsymbol{w}$
  - If simulation ends in an accept state, **accept**
  - If simulation ends in a non-accepting state, **reject**

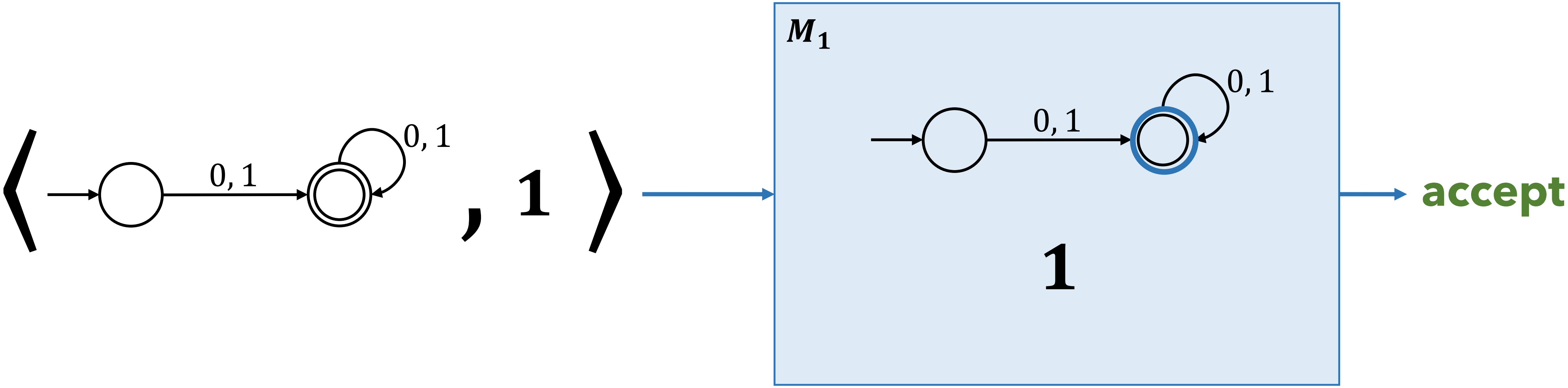
Note: Simulating a DFA on an input will always halt

A_{DFA} is Decidable

High level description:

M_1 = "On input $\langle D, w\rangle$, where D is a DFA and w is a string:

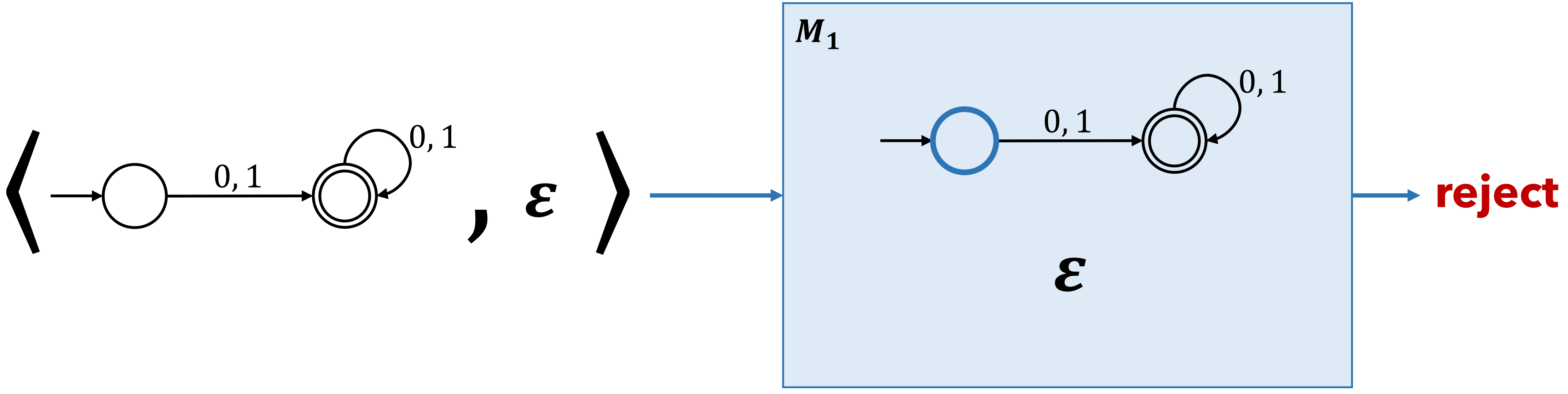
- Simulate DFA D on input w
  - If simulation ends in an accept state, **accept**
  - If simulation ends in a non-accepting state, **reject**



A_{DFA} is Decidable

High level description:

M_1 = "On input $\langle D, w \rangle$, where D is a DFA and w is a string:
- Simulate DFA D on input w
  - If simulation ends in an accept state, **accept**
  - If simulation ends in a non-accepting state, **reject**



More Decidable Languages / Problems

- $A_{DFA} = \{\langle D, w\rangle \mid D$ is a **DFA** that accepts input string $\mathbf{w}\}$
  - Decider takes as input the encoding of a DFA D and a string w and **accepts** if the DFA D accepts the string w, or **rejects** otherwise

- $\boxed{A_{NFA} = \{\langle N, w\rangle \mid N \text{ is an } \mathbf{NFA} \text{ that accepts input string } \mathbf{w}\}}$

- $A_{REX} = \{\langle R, w\rangle \mid R$ is a regular expression that generates string $w\}$

- $E_{DFA} = \{\langle A\rangle \mid A$ is a **DFA** and $L(A) = \emptyset\}$

- $EQ_{DFA} = \{\langle A, B\rangle \mid A$ and **B** are **DFAs** and $L(A) = L(B)\}$

A_{NFA} is Decidable

Show that the language

$$\boldsymbol{A_{NFA}} = \{\langle \boldsymbol{N}, \boldsymbol{w} \rangle \mid \boldsymbol{N} \text{ is an } \mathbf{NFA} \text{ that accepts input string } \mathbf{w}\}$$

is decidable by constructing a TM $\boldsymbol{M_2}$ that decides $\boldsymbol{A_{NFA}}$

High level description:

$\boldsymbol{M_2}$ = "On input $\langle \boldsymbol{N}, \boldsymbol{w} \rangle$, where $\boldsymbol{N}$ is an NFA and $\boldsymbol{w}$ is a string:

- Convert NFA $\boldsymbol{N}$ to an equivalent DFA $\boldsymbol{N'}$
- Run previous decider $\boldsymbol{M_1}$ on $\langle \boldsymbol{N'}, \boldsymbol{w} \rangle$
  - If $\boldsymbol{M_1}$ accepts, **accept**
  - If $\boldsymbol{M_1}$ rejects, **reject**

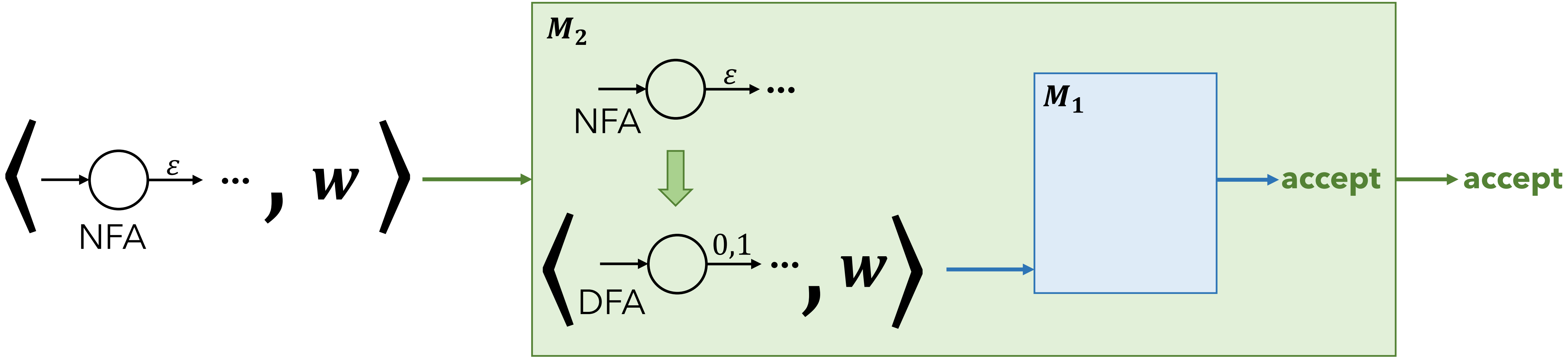
Note: NFA to DFA conversion is a known (halting) algorithm, and $\boldsymbol{M_1}$ is a decider so will always halt

A_{NFA} is Decidable

High level description:

M_2 = "On input $\langle N, w \rangle$, where N is an NFA and w is a string:

- Convert NFA N to an equivalent DFA N'
- Run previous decider M_1 on $\langle N', w \rangle$
  - If M_1 accepts, **accept**
  - If M_1 rejects, **reject**

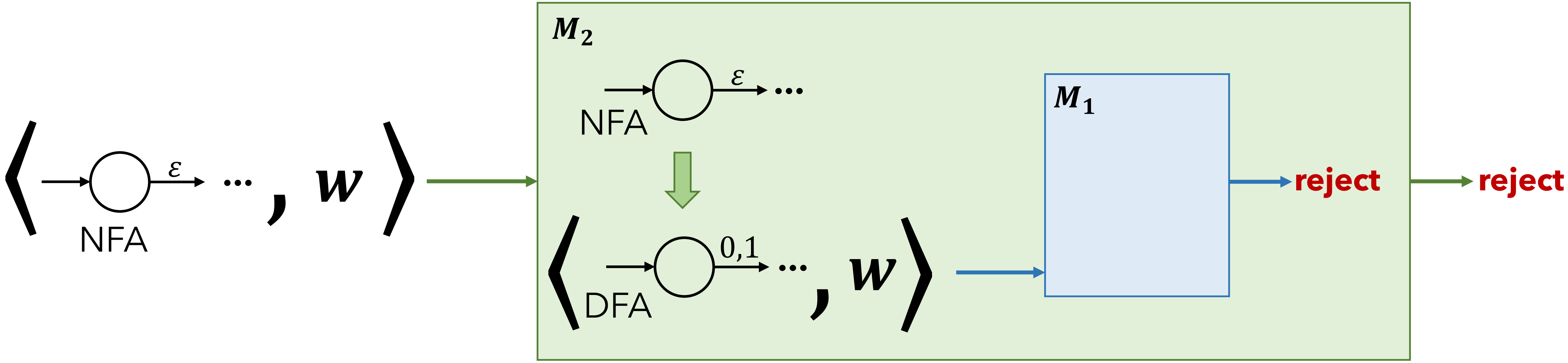


A_{NFA} is Decidable

High level description:

M_2 = "On input $\langle N, w \rangle$, where N is an NFA and w is a string:

- Convert NFA N to an equivalent DFA N'
- Run previous decider M_1 on $\langle N', w \rangle$
  - If M_1 accepts, **accept**
  - If M_1 rejects, **reject**



More Decidable Languages / Problems

- $A_{DFA} = \{\langle D, w\rangle \mid D \text{ is a } \mathbf{DFA} \text{ that accepts input string } \mathbf{w}\}$
  - Decider takes as input the encoding of a DFA D and a string w and **accepts** if the DFA D accepts the string w, or **rejects** otherwise

- $A_{NFA} = \{\langle N, w\rangle \mid N \text{ is an } \mathbf{NFA} \text{ that accepts input string } \mathbf{w}\}$
- $\boxed{A_{REX} = \{\langle R, w\rangle \mid R \text{ is a regular expression that generates string } w\}}$
- $E_{DFA} = \{\langle A\rangle \mid A \text{ is a } \mathbf{DFA} \text{ and } L(A) = \emptyset\}$
- $EQ_{DFA} = \{\langle A, B\rangle \mid A \text{ and } \mathbf{B} \text{ are } \mathbf{DFAs} \text{ and } L(A) = L(B)\}$

A_{REX} is Decidable

Show that the language

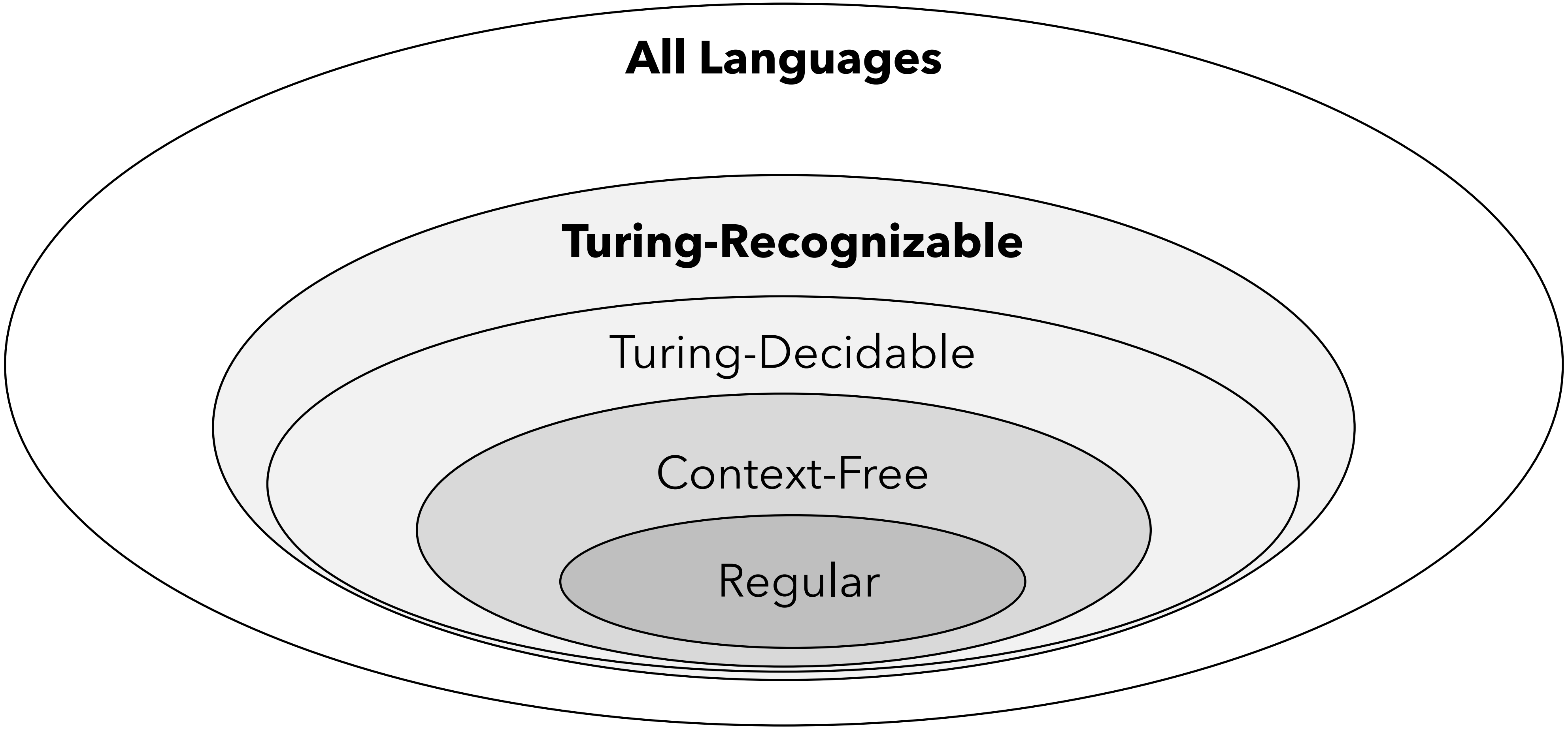
$$A_{REX} = \{\langle R, w\rangle \mid R \text{ is a regular expression that generates string } w\}$$

is decidable by constructing a TM M_3 that decides A_{NFA}

High level description:

M_3 = "On input $\langle R, w\rangle$, where R is a regular expression and w is a string:

- Convert regular expression R to an equivalent NFA R'
- Run previous decider M_2 on $\langle R', w\rangle$
  - If M_2 accepts, **accept**
  - If M_2 rejects, **reject**



Question: Is every language Turing-recognizable?

Are all languages Turing-recognizable?

Recall:

- The set of all languages over $\boldsymbol{\Sigma}$ is $\mathcal{P}(\boldsymbol{\Sigma}^*)$, which is **uncountably infinite**
- $\boldsymbol{\Sigma}^*$ is **countably infinite**

How many Turing machines are there?

- TMs can be encoded over some alphabet $\boldsymbol{\Sigma}$
- We can **enumerate TM's** by enumerating all strings in $\boldsymbol{\Sigma}^*$ and omitting strings that don't any TM
- Hence, the number of Turing-machines and Turing-recognizable languages is **countably infinite**

Therefore, **not all languages are Turing-recognizable** because there are more languages than Turing machines