Theory of Automata and Languages

Turing Machines and Turing-recognizable Languages - 2

Fall 2024

Sharif University of Technology

Mehran Moeini Jam



Overview

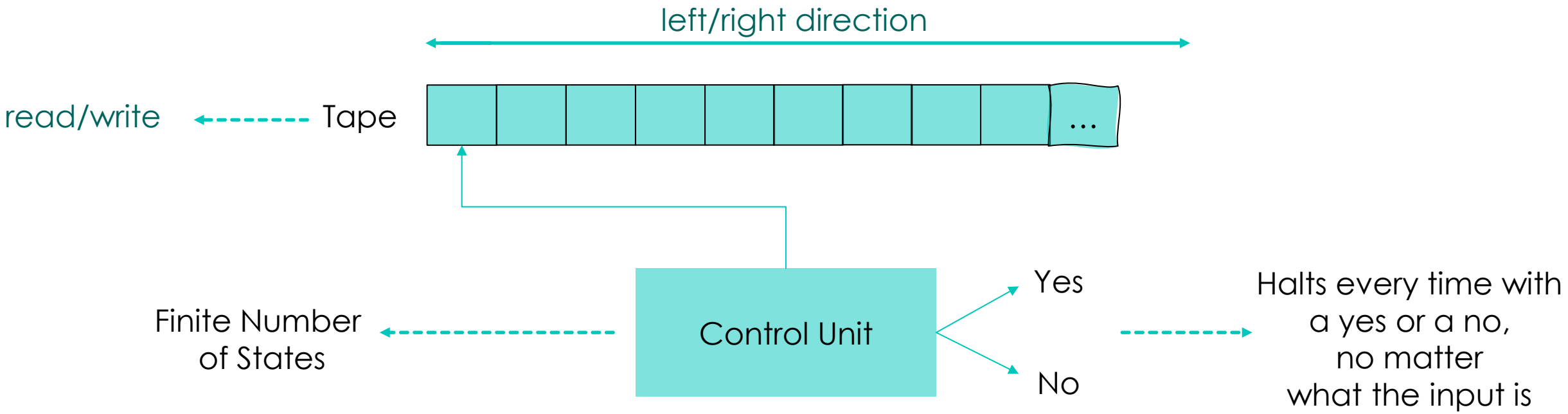
- Decidable Languages
- Decidable Problems for Regular and Context-free Languages
- Turing Machine (as a transducer)
- Computable Functions
- Undecidable Languages
- Reduction
- Turing Machine (as an enumerator)

Recall: Turing-decidable Languages

- Recall that decider Turing machines are Turing machines that halt on every input, indifferent to whether the input string belongs to their language or not.

- The set of all languages for which there exist a decider Turing machine, is called the class of Turing-decidable languages, or simply decidable (also called the class of recursive languages).

Decider Turing Machine (as an acceptor)



Decidable Problems for Regular Languages

- Consider the following language:

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

Decidable Problems for Regular Languages

- Consider the following language:

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

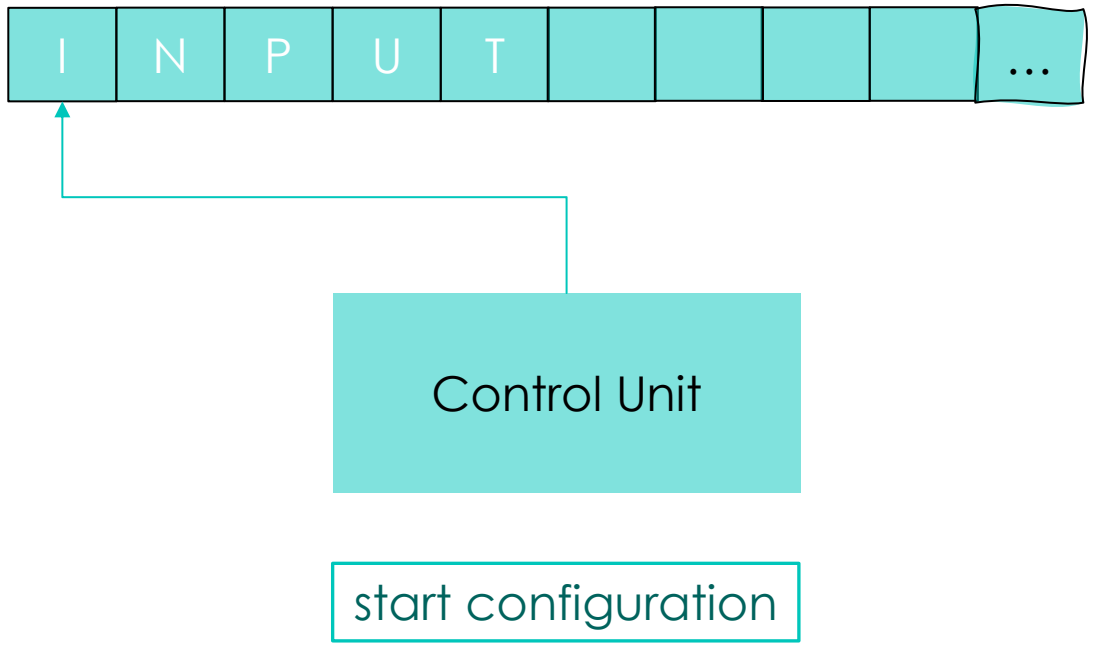
- The following Decider TM decides the above language:

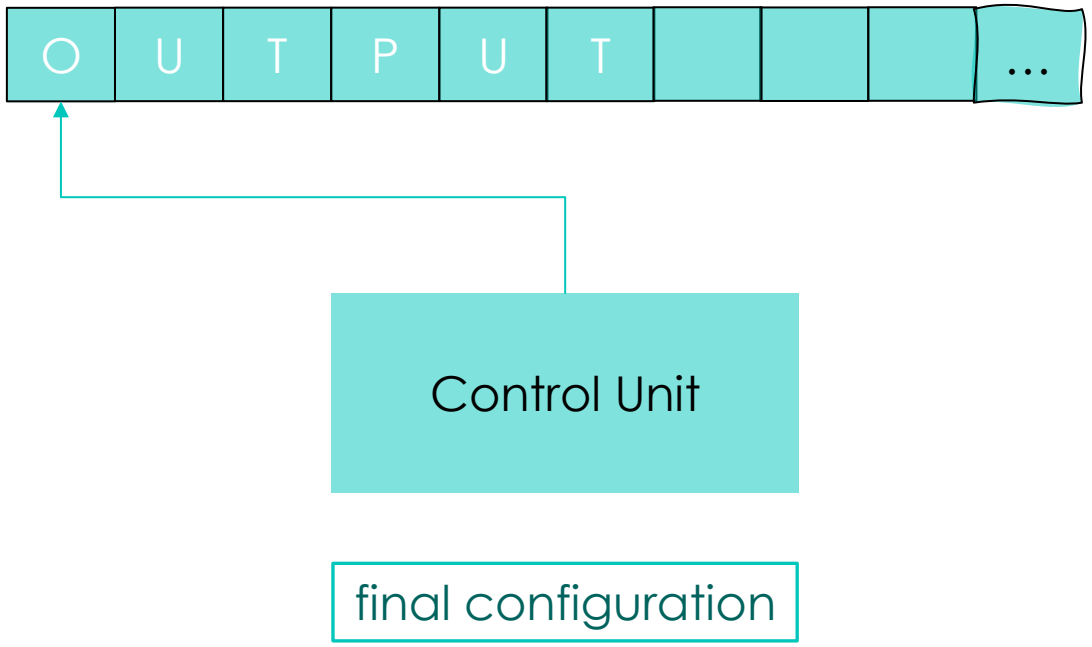
M = “On input <B, w>, where B is a DFA and w is a string:

1. Simulate B on input w.

2. If the simulation ends in an accept state, accept . If it ends in a non-accepting state, reject .”

Turing Machine (as a function/transducer)





Deterministic Turing Machine (as a transducer)

A Deterministic Turing Machine is a 7-tuple $(Q, \Sigma, \Gamma, \delta, q_0, q_h)$ where

- Q is a finite set called states,
- Σ is a finite set called the input alphabet, not containing the symbol u
- Γ is a finite set called the tape alphabet, where $u \in \Gamma$ and $\Sigma \subseteq \Gamma$
- $\delta : Q \times \Gamma \longrightarrow Q \times \Gamma \times \{L, R\}$ is the transition function,
- $q_0 \in Q$ is the start state,
- $q_h \in Q$ is the halting state.

Computable Partial Functions

- Consider T be a Turing machine as a transducer, and $f : \Sigma^* \longrightarrow \Sigma^*$ a partial function, we say that T computes f when for every w in the domain of f (a subset of Σ^*) :

$$q_0 w \quad \vdash \quad q_{acc} f(w)$$

- We call this function a computable partial function. It can be easily proved that a language $L \subseteq \Sigma^*$ is Turing-recognizable if and only if there exists such a function f whose domain (or range) is L.

Computable Partial Functions

- We first consider the if (←) part. We know that when ∫ is a computable partial function, there exist a TM T_f (as a function) that computes it. So T_f can simply distinguish between the strings in and out of the domain and accept the string after the computation is done.

- For the only if (→) part, consider the acceptor TM T_a for L and change it so that it erases the tape upon entering the accept state and before halting. So there is a function ∫ that maps each string in L to the empty string.

Computable Total Functions

- We also define $f : \Sigma^* \longrightarrow \Sigma^*$ to be a computable total function, when f is a total function and there is a Turing machine T that computes f.

- Note that this machine halts for every possible input, so T is effectively a decider Turing machine, similar to what we had seen previously. Try to find the connection between these Turing machines and the class of Turing-decidable languages.

Computable Functions (Extention)

- Note that the definition of computable functions can be extended in such a way that the function accepts more than one input.

- In this case, consider $f : (\Sigma^*)^k \longrightarrow \Sigma^*$ to be a function, we say that Turing machine T computes f when for every $(w_1, w_2, ..., w_k)$ in the domain of f,

$$q_0 w_1 \sqcup w_2 \sqcup \ldots \sqcup w_k \quad \overset{*}{\vdash} \quad q_{acc} f(w_1, w_2, ..., w_k)$$

(remember that $\sqcup$ is the blank symbol)

Reduction

- Consider languages A and B defined on the same alphabet Σ, we say that A is mapping reducible (also: many-one reducible) to B, written $A \leq_m B$ when there is a computable total function $f : \Sigma^* \longrightarrow \Sigma^*$ where for every $w \in \Sigma^*$,

$$w \in A \quad \Leftrightarrow \quad f(w) \in B$$

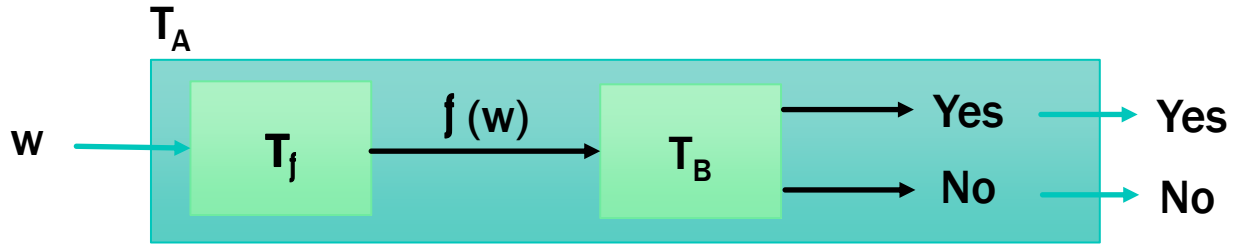
Reduction

- Consider languages A and B defined on the same alphabet Σ, we say that A is mapping reducible (also: many-one reducible) to B, written $A \leq_m B$ when there is a computable total function $f : \Sigma^* \longrightarrow \Sigma^*$ where for every $w \in \Sigma^*$,

$$w \in A \quad \Leftarrow \quad f(w) \in B$$

$$w \in A \quad \Rightarrow \quad f(w) \in B \quad \equiv \quad w \notin A \quad \Leftarrow \quad f(w) \notin B$$

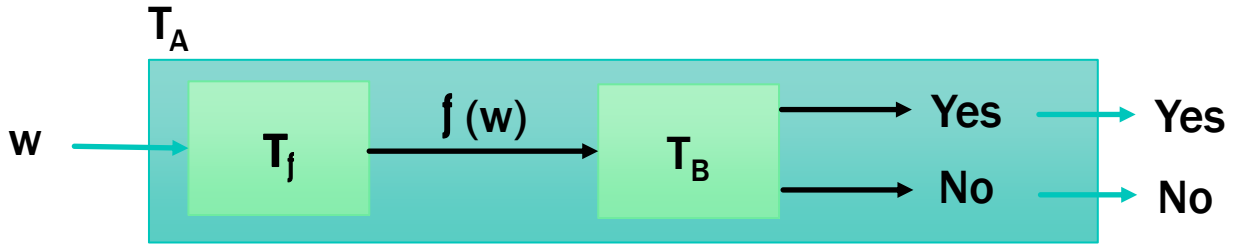
- We can build a TM T_A that recognizes A by using the TM T_B that recognizes B:



Theorems

- **Theorem 1**: Consider $A \leq_m B$ where A and B are languages over the same alphabet Σ. If B is Turing-recognizable, then A is also Turing-recognizable.

- **Theorem 2**: Consider $A \leq_m B$ where A and B are languages over the same alphabet Σ. If B is Turing-decidable, then A is also Turing-decidable.



Decidable Problems for Regular Languages

- Consider the following language:

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

- The following Decider TM decides the above language:

M = “On input <B, w>, where B is a DFA and w is a string:

1. Simulate B on input w.

2. If the simulation ends in an accept state, accept . If it ends in a non-accepting state, reject .”

Decidable Problems for Regular Languages

- Consider the following language:

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

Decidable Problems for Regular Languages

- Consider the following language:

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

- The following Decider TM decides the above language:

N = "On input <C, w>, where C is a NFA and w is a string:

1. Convert NFA C to an equivalent DFA B, using the procedure previously presented
2. Run TM M (from previous slides) on input <B, w>.
3. If M accepts, accept and if it rejects, reject."

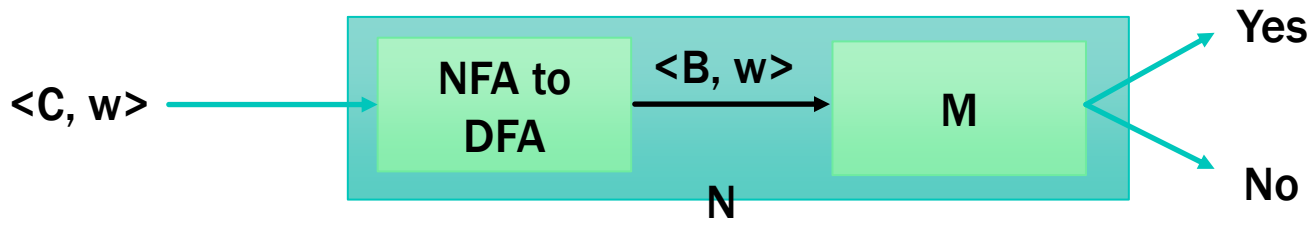
Decidable Problems for Regular Languages

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

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

N = "On input <C, w>, where C is a NFA and w is a string:

1. Convert NFA C to an equivalent DFA B, using the procedure previously presented
2. Run TM M (from previous slides) on input <B, w>.
3. If M accepts, accept and if it rejects, reject."



Decidable Problems for Regular Languages

- Consider the following language:

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

Decidable Problems for Regular Languages

- Consider the following language:

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

- The following Decider TM decides the above language:

P = “On input <R, w>, where R is a regular expression and w is a string:

1. Convert regular expression R to an equivalent NFA A by using the procedure previously presented

2. Run TM N on input <A, w>.

3. If N accepts, accept and if N rejects, reject .”

Decidable Problems for Regular Languages

- Consider the following language:

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

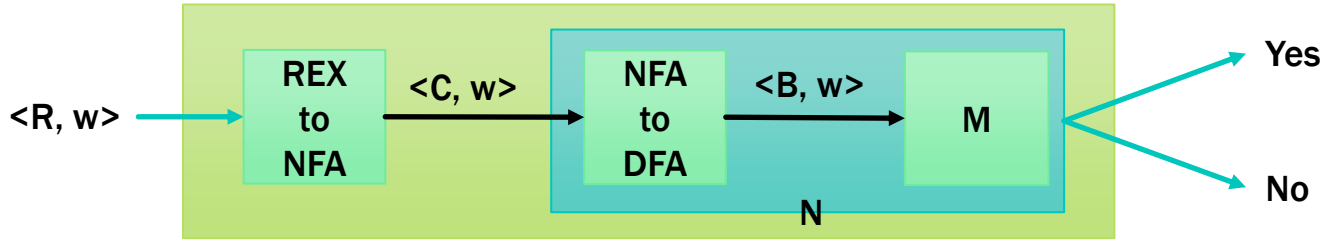
- The following Decider TM decides the above language:

P = “On input <R, w>, where R is a regular expression and w is a string:

1. Convert regular expression R to an equivalent NFA A by using the procedure previously presented

2. Run TM N on input <A, w>.

3. If N accepts, accept and if N rejects, reject .”



Decidable Problems for Regular Languages

- Consider the following language:

$$E_{DFA} = \{ < A > \mid A \text{ is a DFA and } L(A) = \emptyset \}$$

Decidable Problems for Regular Languages

- Consider the following language:

$$E_{DFA} = \{ < A > \mid A \text{ is a DFA and } L(A) = \emptyset \}$$

- The following Decider TM decides the above language:

T = “On input <A>, where A is a DFA:

1. Mark the start state of A.

2. Repeat until no new states get marked:

3. Mark any state that has a transition coming into it from any state that is already marked.

4. If no accept state is marked, accept and otherwise, reject .”

Decidable Problems for Regular Languages

- Consider the following language:

$$EQ_{DFA} = \{ < A, B > \mid A \text{ and } B \text{ are DFAs and } L(A) = L(B) \}$$

Decidable Problems for Regular Languages

- Consider the following language:

$$EQ_{DFA} = \{ < A, B > \mid A \text{ and } B \text{ are DFAs and } L(A) = L(B) \}$$

- We know that $L(C) = (L(A) \cap L(B)^c) \cup (L(A)^c \cap L(B))$

- The following Decider TM decides the above language:

On input < A, B > , where A and B are DFAs:

1. Construct DFA C as described above.

2. Run TM T on input <C>.

3. If T accepts, accept and if T rejects, reject ."

Decidable Problems for Context-free Languages

Consult the textbook.

Undecidable Languages

- Consider the following language (Self-accepting):

$$SA_{TM} = \{ \langle T \rangle \mid T \text{ is a TM and } \langle T \rangle \in L(T) \}$$

- The above language is Turing-recognizable. Consider the following non-decider TM which corresponds to a semi-algorithm:

TM_1 = "On input $\langle T \rangle$ where T is a Turing Machine,

1. Run T on input $\langle T \rangle$,

3. If T accepts, accept and if T rejects, reject"

Undecidable Languages

- Consider the following language (Non-self-accepting):

$$NSA_{TM} = \{ \langle T \rangle \mid T \text{ is a TM and } \langle T \rangle \notin L(T) \}$$

- We assume that the above language is decidable by a decider TM D :



Undecidable Languages

- Consider the following language (Non-self-accepting):

$$NSA_{TM} = \{ \langle T \rangle \mid T \text{ is a TM and } \langle T \rangle \notin L(T) \}$$

- We assume that the above language is decidable by a decider TM D.



※

Undecidable Languages

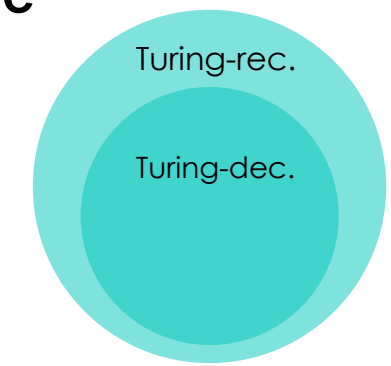
- We proved that the language NSA is undecidable (not Turing-decidable).

- One immediate conclusion is that the language SA is also undecidable, because we know that the class of Turing-decidable languages is closed under the complement operator and assuming SA to be decidable leads to a contradiction.

Undecidable Languages

- So there exist languages (like the language SA) that are Turing-recognizable but not Turing-decidable.

- We now have the tool to prove as an immediate corallory that the class of Turing-decidable languages is a proper subset of the class of Turing-recognizable languages.



Undecidable Languages

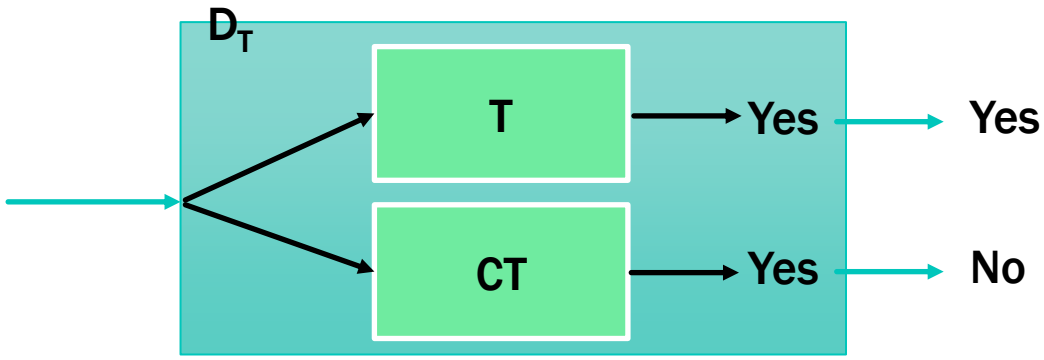
- We proved that the language NSA is undecidable (not Turing-decidable).

  Now someone might ask whether it is Turing-recognizable, like the language SA.

  The answer is again No.

Theorem

- **Theorem**: Turing-recognizble languages are **not closed** under the **complement** operator.
- **Proof**: Assume otherwise, then we can construct the following TM D_T that **decides** <u>every</u> Turing-recognizble language, which we know is a contradiction because we have Turing-recognizble languages that are not decidable.

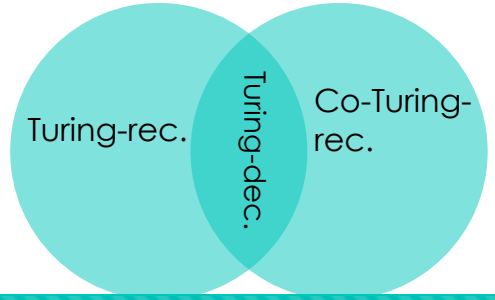


- Consider $L(CT) = L(T)^c$
- T and CT are both non-decider TMs

Undecidable Languages

- If the language NSA were Turing-recognizable, then according to the previous theorem, both languages SA and NSA must be decidable, which is a contradiction.

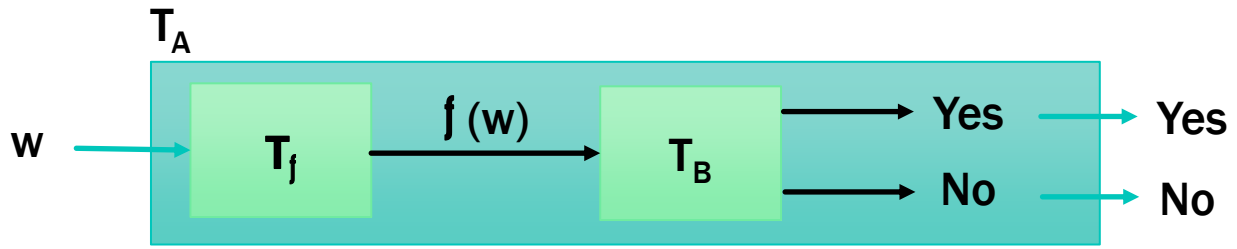
- The class of languages where their complement is Turing-recognizable is called the class of Co-Turing-recognizable languages. We conclude (by the construction in the previous slide) that if a language is both Turing-recognizable and Co-Turing-recognizable, then it is Turing-decidable.



Proof of Undecidability by using Reduction

- Recall the following theorem: Consider $A \leq_m B$ where A and B are languages over the same alphabet Σ. If B is Turing-decidable, then A is also Turing-decidable.

- Corollary: Consider $A \leq_m B$ where A and B are languages over the same alphabet Σ. If A is undecidable, then B is also undecidable.



Proof of Undecidability by using Reduction

- Consider the following language:

$$A_{TM} = \{ < T, w > \mid T \text{ is a TM that accepts the string } w \}$$

- To prove the above language undecidable, we reduce from the language SA.

  Consider T_f to be the following transducer TM :

T_f = “On input < T > where T is a Turing Machine,

1. Output < T , <T> >”

Proof of Undecidability by using Reduction

- Consider the following language:

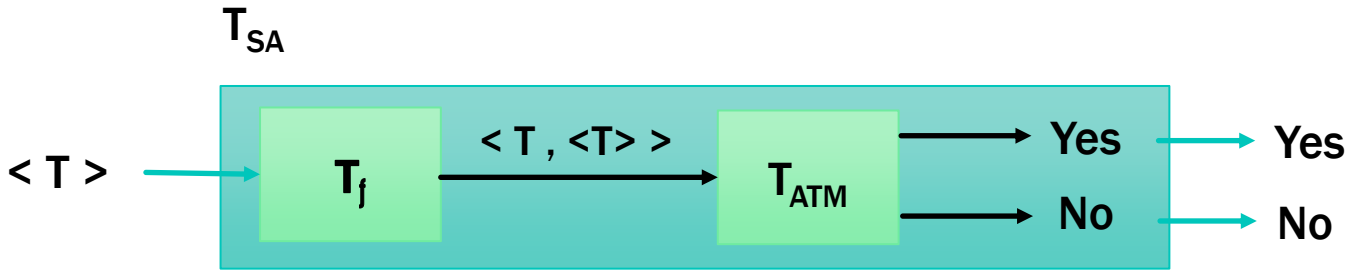
$$A_{TM} = \{ < T, w > \mid T \text{ is a TM that accepts the string } w \}$$

- To prove the above language undecidable, we reduce from the language SA.

  Consider T_f to be the following transducer TM :

T_f = "On input < T > where T is a Turing Machine,

1. Output < T , <T> >



Proof of Undecidability by using Reduction

- Consider the following language:

$$HALT_{TM} = \{ < T, w > \mid T \text{ is a TM that halts on input string } w \}$$

- To prove the above language undecidable, we reduce from the language A_{TM} .

  Consider T_f to be the transducer TM depicted in the next page.

Proof of Undecidability by using Reduction

$$HALT_{TM} = \{ < T, w > \mid T \text{ is a TM that halts on input string } w \}$$

T_f = "On input < T, w > where T is a Turing Machine and w is a string,

1. Build a TM M where

   M = "On input < w' > where w' is a string,

   1. Discard w',

   2. Run T on input w,

   3. If T accepts, accept ..."

2. Output < M, w >"

Proof of Undecidability by using Reduction

T_f = "On input < T, w > where T is a Turing Machine and w is a string,

1. Build a TM M where

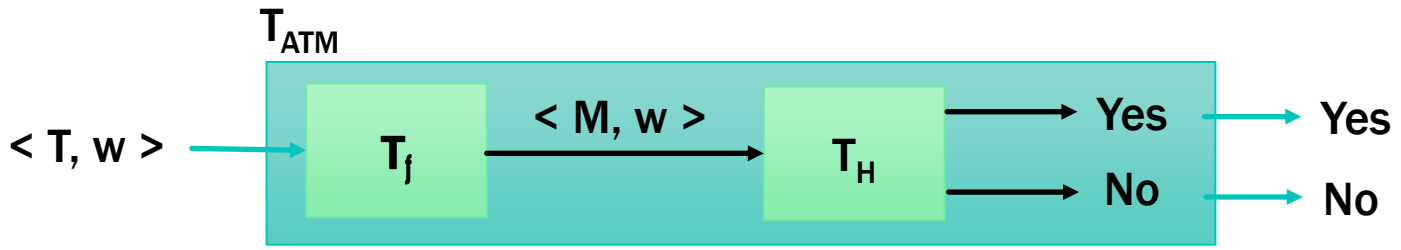
   M = "On input < w' > where w' is a string,

   1. Discard w',

   2. Run T on input w,

   3. If T accepts, accept ..."

2. Output < M, w >"



Proof of Undecidability by using Reduction

T_f = "On input < T, w > where T is a Turing Machine and w is a string,

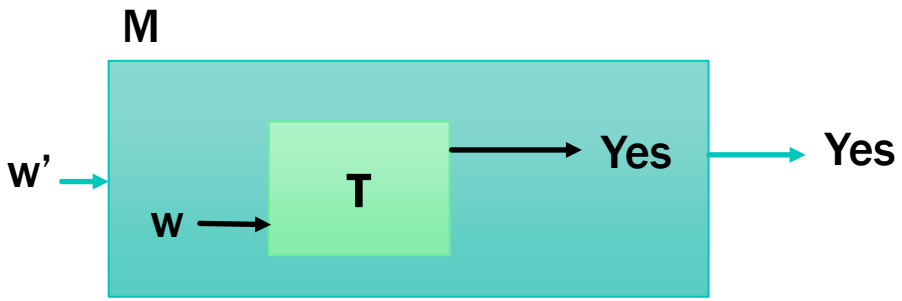
1. Build a TM M where

M = "On input < w' > where w' is a string,

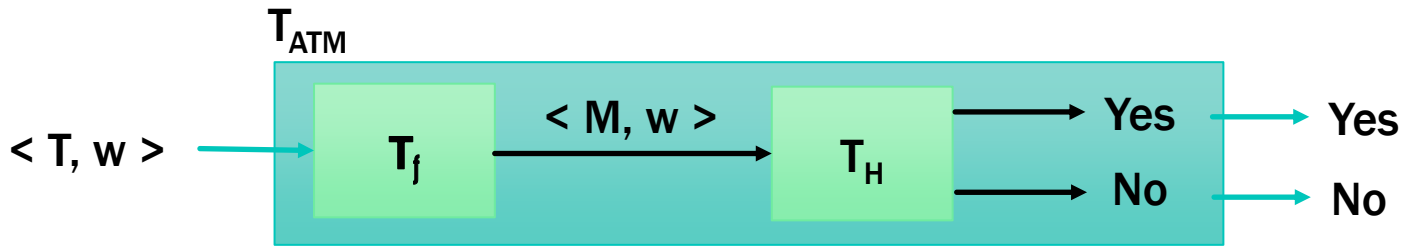
1. Discard w',

2. Run T on input w,

3. If T accepts, accept ..."



2. Output < M, w >"



Proof of Undecidability by using Reduction

T_f = "On input < T, w > where T is a Turing Machine and w is a string,

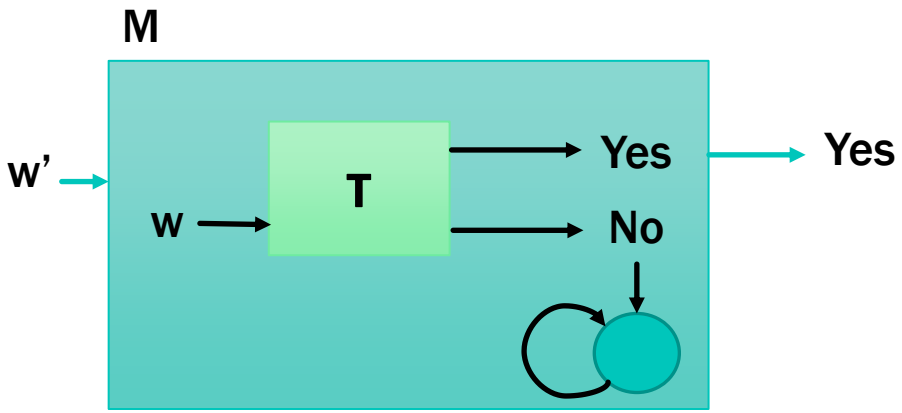
1. Build a TM M where

M = "On input < w' > where w' is a string,

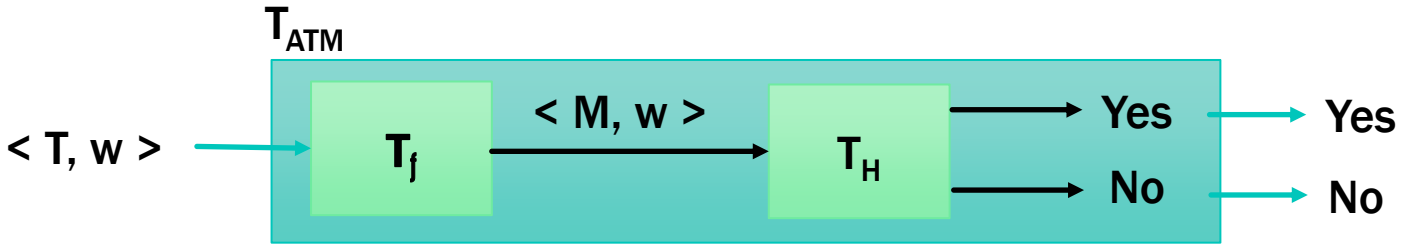
1. Discard w',

2. Run T on input w,

3. If T accepts, accept and if T rejects, loop forever (don't halt)"



2. Output < M, w >"



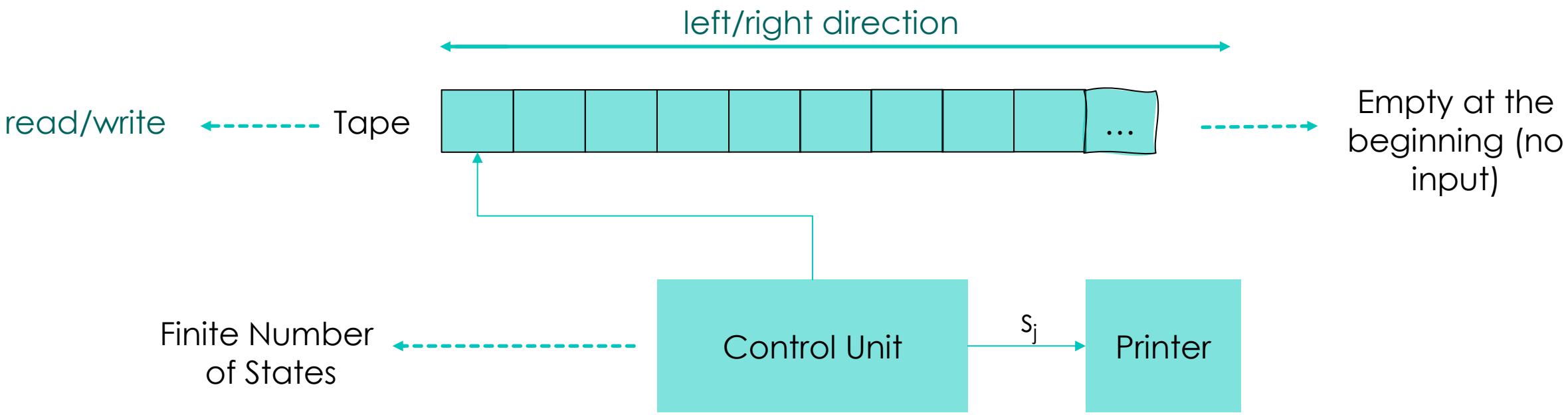
Proof of Undecidability by using Reduction

- Consider the following language:

$$EQ_{TM} = \{ < T_1, T_2 > \mid T_1 \text{ and } T_2 \text{ are TMs and } L(T_1) = L(T_2) \}$$

- The above language can also be proved to be undecidable, by using a reduction from the language A_{TM} . Try to complete the proof yourself.

Turing Machine (as an enumerator)



Theorem

- **Theorem**: A language is Turing-recognizable if and only if some enumerator Turing Machine enumerates it.

- **Proof**: First we prove the if (←) part . The acceptor TM M works as follows:
- M = “On input w:

    1. Run enumerator TM E. Every time that E outputs a string, compare it with w.

    2. If w ever appears in the output of E, accept.”

Theorem

- **Proof**: Now we do the only if (→) part. Say that $s_1, s_2, s_3, \ldots$ is a list of all possible strings in Σ^*. Enumerator E works as follows:

- E = "Ignore the input if any was provided,

    1. Repeat the following for i = 1, 2, 3, ...

        1. Run M for i steps on each input $s_1, s_2, \ldots, s_i$.

        2. If any computations accept, print out the corresponding s_j ."

