Handout 3: TMs (Decidable and Recognizable)

Gianna Baker, Alice Lin, Charlie Mei

updated 26 October 2026

1 Turing Machines

Definition: Turing Machine

a Turing Machine is the 7-tuple $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{acc}, q_{rej})$ where:

- Q = the set of states (finite)
- Σ = input alphabet (does not contain blank symbol)
- Γ = tape alphabet ($\sqcup \in \Gamma$, $\Sigma \subseteq \Gamma$)
- δ = transition function $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$
- q_0 = start state
- q_{acc} = accept state
- q_{rej} = reject state (cannot be equal to accept state)

A Turing Machine (TM) can be thought of like a very simple computer with unlimited tape memory. Any ordinary algorithm can be modeled by a Turing machine. If no Turing machine can solve a problem, no ordinary algorithm can solve it either. A TM consists of a finite set of states, an infinite tape, a read/write head, and rules describing what to do at each step.

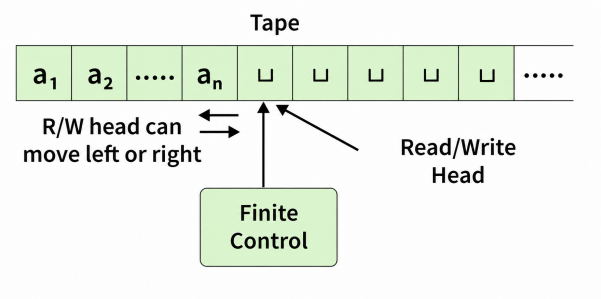


Figure 1: Tape Structure

1. **Tape**: stores input, intermediate data, and output. It extends indefinitely in the standard model.
2. **Tape head**: reads one symbol at a time, may overwrite it, and moves left or right.
3. **State**: represents what the machine is currently doing.
4. **Transition Function**: specifies the next state, the symbol to write, and the direction to move.

Transitions

A transition is a single step of instruction that dictates what the machine writes and where it moves to next based on its current state and tape symbol it read. We can get familiar with the ones we will see below:

Label	Action
$0 \to R$	Read 0, leave it unchanged, move right (shorthand for $0 \to 0, R$)
$1 \to L$	Read 1, leave it unchanged, move left
$B \to R$	Read blank, leave it blank, move right
$1 \to \#, R$	Read 1, replace 1 with #, move right
$0, 1, B \to L$	On any of these symbols, move left without changing it

Table 1: Understanding Common Transition Labels (uses binary input alphabet)

Configurations and Tracing Computation

A machine's configuration is its instantaneous situation: the tape contents, head position, and current state. The following is a good tracing procedure to use:

1. Write down the initial tape and head position.
2. Identify the current state and the symbol under the head.
3. Find the matching outgoing transition.
4. Apply the write, move, and state change.
5. Repeat until the machine accepts or rejects (or until you're convinced the machine is going to run forever, or until you give up)

Example: Listing Configurations of a TM

Consider the following Turing Machine M where:

- $Q = \{q_0, q_1, q_{\text{acc}}, q_{\text{rej}}\}$
- $\Sigma = \{0, 1\}$
- $\Gamma = \{0, 1, B, X\}$
- The start state is q_0, the accept state is q_{acc} and the reject state is q_{rej}.

$$\delta(q, a) = (q', b, D)$$

means that the machine, in state q reading symbol a, writes b, moves in direction D, and enters state q'. The transition function is δ given by:

Current State	Read Symbol	Action
q_0	0	$(q_0, 0, R)$
q_0	1	(q_1, X, R)
q_0	B	(q_{rej}, B, R)
q_1	0	$(q_1, 0, R)$
q_1	1	$(q_1, 1, R)$
q_1	B	(q_{acc}, B, L)

Question: List all configurations when M runs on input $w = 00101$.
Answer: We represent a configuration by writing the current state immediately before the symbol being read. The initial configuration is:

$$q_0\,00101$$

The machine scans right over the first two 0's:

$$q_0\,00101$$
$$0\,q_0\,0101$$
$$00\,q_0\,101$$

It reads the first 1, replaces it with X, and enters q_1:

$$00X\,q_1\,01$$

It then scans right until reaching the blank symbol:

$$00X0\,q_1\,1$$
$$00X01\,q_1\,B$$

Finally, it moves left and enters the accepting state:

$$00X0\,q_{\text{acc}}\,1$$

The machine halts and accepts the input.

Three Possible Outcomes of a Computation

There are three different possibilities/outcomes from giving a TM an input:



Accept

The machine reaches q_{acc} and halts. The input is accepted and belongs to $L(M)$.



Reject

The machine reaches q_{rej} and halts. The input is not in $L(M)$.



Loop forever

The machine never reaches a halting state and it neither accepts nor rejects that input. The input is not in $L(M)$.

It is important to note that the accept and reject states are halting states. Sometimes, we do not specify all transition, if a transition is missing it means we go to the reject state and halt (the transition would be of the form $a, q \to a, q_{\text{rej}}, R$).

Definition: $L(M)$

The language of a Turing Machine M is

$$L(M) = \{w \in \Sigma^* \mid M \text{ accepts } w\}.$$

It contains exactly the strings that cause the machine to accept. A string is **not** in the language if the machine rejects it or loops forever on it.

Example: Finding the Language of a TM

Consider the following Turing Machine M over the alphabet $\Sigma = \{0, 1\}$.
Algorithm: On input w:

1. Scan the input from left to right.
2. If the input contains two consecutive 0's, reject.
3. Otherwise, count the number of 1's in the input.
4. If the number of 1's is odd, accept.
5. If the number of 1's is even, move right forever.

Question: What is the language $L(M)$?
Answer: The machine accepts exactly those strings that contain no consecutive 0's and have an odd number of 1's. Therefore,

$$L(M) = \left\{ w \in \{0,1\}^* \;\middle|\; \begin{array}{l} w \text{ contains no consecutive 0's, and} \\ w \text{ contains an odd number of 1's} \end{array} \right\}.$$

Important: Strings containing consecutive 0's are rejected, while strings with no consecutive 0's and an even number of 1's cause the machine to loop forever. Neither type of string belongs to $L(M)$.
For example:

- $1 \in L(M)$: one 1, no consecutive 0's.
- $1011 \in L(M)$: three 1's, no consecutive 0's.
- $1001 \notin L(M)$: contains consecutive 0's, so the machine rejects.
- $1010 \notin L(M)$: has two 1's, so the machine loops forever.

In class, we saw that several variations of Turing Machines are equivalent to the standard definition. These models may differ in how their tapes or heads operate, but they can recognize and decide exactly the same languages as a standard TM.

- **Stay-put TM:** The tape head can move left, right, or stay in place. A standard TM can simulate a stay-put move by moving right and then left, without changing the symbols.
- **Doubly infinite tape:** The tape extends infinitely in both directions. A standard TM can simulate this by encoding the two halves of the tape on a single, one-sided tape.
- **Multi-tape TM:** The machine has multiple tapes, each with its own read/write head. A standard

TM can simulate this by encoding all tapes and their head positions on a single tape and scanning to simulate each transition.

These variations do not increase the computational power of Turing Machines; they may only make certain computations more convenient or efficient to describe.

2 Input-Output TMs

Unlike a recognizer or decider, (which determines whether an input belongs to a language) a function-computing TM transforms an input string into an output string. The TM doesn't have accept / reject states, instead it has an halting state q_{halt}: the TM halts as soon as it enters it. Initially the tape contains the input x and then it performs a sequence of transitions to manipulate the tape. When it halts, the tape contains the desired output $f(x)$.

Definition: TM Computable

A function $f : \Sigma^* \to \Sigma^*$ is TM computable if there exists an input-output TM M such that on all inputs x M halts with $f(x)$ written on the tape.

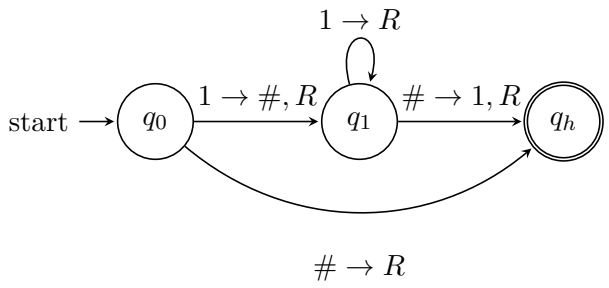
Function	Input	Output
Prepending a delimiter	w	$\#w$
Unary addition	$1^n\#1^m$	$\#1^{n+m}$
Binary addition	$x\#y$	$\#(x+y)$
Unary multiplication	$1^n\#1^m$	1^{nm}
Binary multiplication	$x\#y$	$x \cdot y$

Table 2: Some Computable Functions

Example: Unary Addition

We want to construct a TM that computes $f(1^n\#1^m) = \#1^{n+m}$. The algorithm is as follows:

1. Replace the first 1 with a #
2. Keep moving right until reaching the original #
3. Replace the # with a 1
4. Halt



Note the bottom transition is for if $n = 0$, as our input is already in the form of the needed output so we automatically accept.

3 TM-Recognizable

A language L is **Turing-recognizable** is there exists a Turing machine M such that $L(M) = L$, namely

$$w \in L \Leftrightarrow M \text{ eventually accepts } w.$$

For strings outside L, the machine may reject or run forever (does not need to halt on nonmembers).

Proof Template: TM-Recognizable

To prove L is TM-recognizable:

- construct algorithm M
- prove
  - $x \in L \Rightarrow M$ accepts x
  - $x \notin L \Rightarrow M$ rejects x **OR** M runs forever on x

Examples

Example 1: A_{TM}

Problem: Prove that

$$A_{TM} = \{\langle M, w\rangle \mid M \text{ accepts } w\}$$

is TM-recognizable.
Proof: We Construct a Turing machine R.

On input $\langle M, w\rangle$ where M is a TM and w a string:

1. Run M on input w. (If M runs forever on w, R continues simulating forever.)
2. If M accepts w, R accepts.
3. If M rejects w, R rejects.

If $\langle M, w\rangle \in A_{TM}$, then M eventually accepts w, so R eventually accepts $\langle M, w\rangle$.
If $\langle M, w\rangle \notin A_{TM}$, then either M rejects w, in which case R rejects, or M runs forever in w, in which case R runs forever on $\langle M, w\rangle$. In both cases R does not accept.
Therefore, R recognizes A_{TM}.

Note: we did not explicitly address the case that the input to R is malformed and is not a valid encoding of $\langle M, w\rangle$. In this case R rejects immediately – that's what the line "On input $\langle M, w\rangle$ where M is a TM and w a string" implies by convention.

Example 2: $RA_{TM,DFA}$

Problem: Prove that

$$RA_{TM,DFA} = \left\{ \langle M, D \rangle \;\middle|\; \begin{array}{l} M \text{ is a TM, } D \text{ is a DFA, } \exists\, w \in \Sigma^* \text{ s.t.} \\ M \text{ rejects } w,\ D \text{ accepts } w \end{array} \right\}$$

is TM-recognizable.

Proof: Here's a TM (algorithm R) recognizing $RA_{TM,DFA}$:
R: "On input x:
    if $x \neq \langle M, D \rangle$, reject. (1)
    else:
        for $i = 1, 2, 3, \ldots$ (2)
            for all $s \in \Sigma^*$ such that $|s| \leq i$: (3)
                Run M on s for i steps. (4)
                if M rejects, (5)
                    Run D on s and if D accepts, accept." (6)

Analysis:

- $x \in RA_{TM,DFA} \Rightarrow x = \langle M, D \rangle$ where M is a TM, D is a DFA, and there is a string w such that
  (a) M rejects w in k (finite) steps $\Rightarrow$ in iteration $i = \max(|w|, k)$, M rejects w in line (5), and
  (b) D accepts w (a DFA always halts after finitely many steps) $\Rightarrow$ R accepts x in line (6).
- $x \notin RA_{TM,DFA} \Rightarrow$ either
  (a) the input is not a valid encoding, i.e. $x \neq \langle M, D \rangle \Rightarrow$ R rejects x in line (1), or
  (b) $x = \langle M, D \rangle$ for TM M and DFA D but there is no such w, so for every s, M does not reject s or D does not accept s. Line (6) never accepts $\Rightarrow$ R runs forever.

So R recognizes $RA_{TM,DFA}$, and $RA_{TM,DFA}$ is TM-recognizable. ■

The above example used "dovetailing" for the loop in lines 2-6, which allowed us to guarantee that if the input is a yes-instance, we will eventually reach an iteration that accepts. It would NOT work if we just replaced lines 2-6 with:
for all $s \in \Sigma^*$, run M on s, if M rejects, run D on s and if D accepts, accept.
Do you see the problem?

4 TM-Decidable

Proof Template: TM-Decidable

To prove L is TM-decidable:

- construct algorithm M
- prove
  - $x \in L \Rightarrow M$ accepts x
  - $x \notin L \Rightarrow M$ rejects x

Unlike a general TM (which always recognizes some langauge), a TM decider **MUST** halt on all inputs (every string is either accepted or rejected). It cannot loop and thus has stricter requirements a TM must meet. The implication is that L is Decidable $\implies L$ is Recognizable.

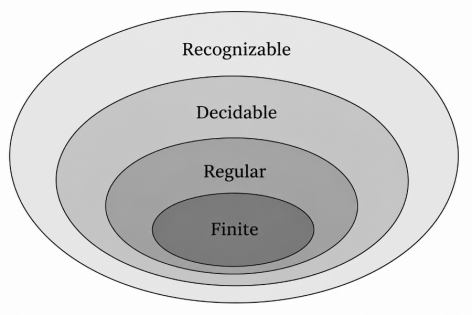


Figure 2: Relationship between Languages

Examples

Example 1: Containing a Substring

Problem: Prove that the language

$$L_1 = \{w \in \{0,1\}^* \mid w \text{ contains } 11\}$$

is TM-decidable.

Proof: Construct a Turing Machine D that on input w:

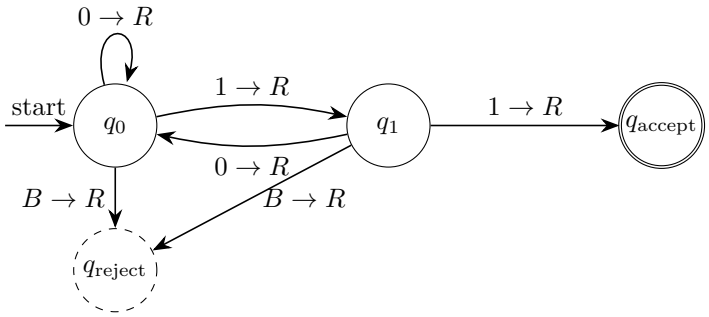
1. It scans w from left to right.
2. If it finds two consecutive 1's it accepts.
3. If it reaches the end of the input without finding two consecutive 1's it rejects.

If $w \in L_1$, then w contains two consecutive 1's and D eventually finds them and accepts w.

If $w \notin L_1$, then w does not contain two consecutive 1's and D reaches the end on the input and rejects w.

Since w is finite, D always reaches either an accepting or rejecting step after finitely many moves.

Thus, D accepts every string in L_1, rejects every string not in L_1, and halts on every input. Therefore, D decides L_1.



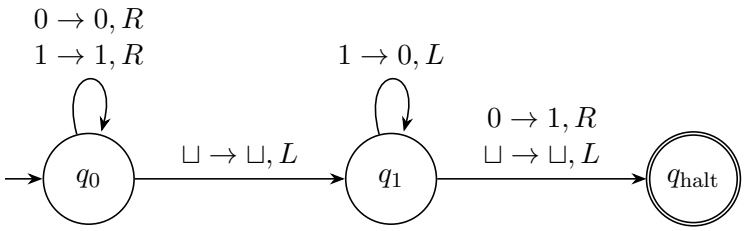
You might notice that the above TM looks a lot like the DFA for the language. Indeed, all regular

languages are decidable, and it's easy to construct a TM for them. However, TMs are much more powerful than DFAs – see some examples in the lecture notes (and textbook, and below). In fact, a TM can solve any problem that you could solve using any reasonable model of computation (e.g., your favorite programming language on your laptop, or a super computer, etc). The Church-Turing thesis says that a TM is a good notion of "algorithm" and captures any reasonable notion of computation (ignoring efficiency).

5 Practice Problems

Problem 1: Listing Configurations and TM Outputs

Consider the input-output TM $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{halt}})$ where $Q = \{q_0, q_1, q_{\text{halt}}\}$, $\Sigma = \{0, 1\}$, $\Gamma = \{0, 1, \sqcup\}$ drawn bellow.



(a) Provide the complete sequence of configurations of M when run on input 100.

(b) What is the output of M on 10011? On input 11?

(c) What function is computed by M?

Problem 2: Finding the Language of a TM

For each of the following Turing Machines, describe its language $L(M)$ using set notation. Be careful to distinguish between accepting, rejecting, and looping forever.

(a) Consider a TM M over $\Sigma = \{0, 1\}$.

Algorithm: On input w:

(a) Scan the entire input and count the number of 0's and 1's.
(b) If w contains the substring 00, reject.
(c) If the number of 0's is greater than the number of 1's, accept.
(d) If the number of 0's is equal to the number of 1's, loop forever.
(e) Otherwise, reject.

Describe $L(M)$. What happens on the inputs $\epsilon, 001, 010, 1001, 10100, 1110$?

(b) Consider a TM N over $\Sigma = \{0, 1\}$.

Algorithm: On input w:

(a) If $w = \epsilon$, accept.
(b) Scan the input from left to right, keeping track of the number of symbols read.
(c) If the end of the input is reached and the length of w is divisible by 3, accept.
(d) If the end of the input is reached and the length of w is even, loop forever.
(e) Otherwise, reject.

Describe $L(N)$. What happens on the inputs $\epsilon, 0, 01, 101, 0101, 10101, 101010$?

Problem 3: Proving Recognizability

For each language below, prove that it is TM-recognizable.

(a)
$$A3_{TM} = \left\{ \langle M, w_1, w_2, w_3 \rangle \;\middle|\; \begin{array}{l} M \text{ is a TM, } w_1, w_2, w_3 \text{ are strings, and} \\ M \text{ accepts } w_1, w_2, \text{ and } w_3 \end{array} \right\}$$

(b)
$$MOD3_{TM} = \left\{ \langle M \rangle \;\middle|\; \begin{array}{l} M \text{ is a TM, } \exists\, w \in \Sigma^* \text{ s.t.} \\ |w| \bmod 3 = 0, M \text{ accepts } w \end{array} \right\}$$

(c)
$$AA_{TM,TM} = \left\{ \langle M_1, M_2 \rangle \;\middle|\; \begin{array}{l} M_1, M_2 \text{ are TMs, } \exists\, w \in \Sigma^* \text{ s.t.} \\ M_1 \text{ accepts } w, M_2 \text{ accepts } w \end{array} \right\}$$

Problem 4: Proving Decidability

For each language below, prove that it is TM-decidable.

(a)
$$L_2 = \left\{ \langle D \rangle \;\middle|\; D \text{ is a DFA and } L(D) = \Sigma^* \right\}$$

(b)
$$L = \left\{ \langle M, w, k \rangle \;\middle|\; \begin{array}{l} M \text{ is a Turing machine, } w \in \Sigma^*, \text{ and } k \geq |w|, \\ M \text{ visits at most } k \text{ distinct tape cells} \\ \text{when run on input } w \end{array} \right\}.$$

(c)
$$CYCLE = \left\{ \langle G, k \rangle \;\middle|\; G \text{ is a graph that contains at least one cycle of size } k \right\}$$

Recall that a cycle is a sequence of vertices connected by edges that starts and ends at the same vertex.