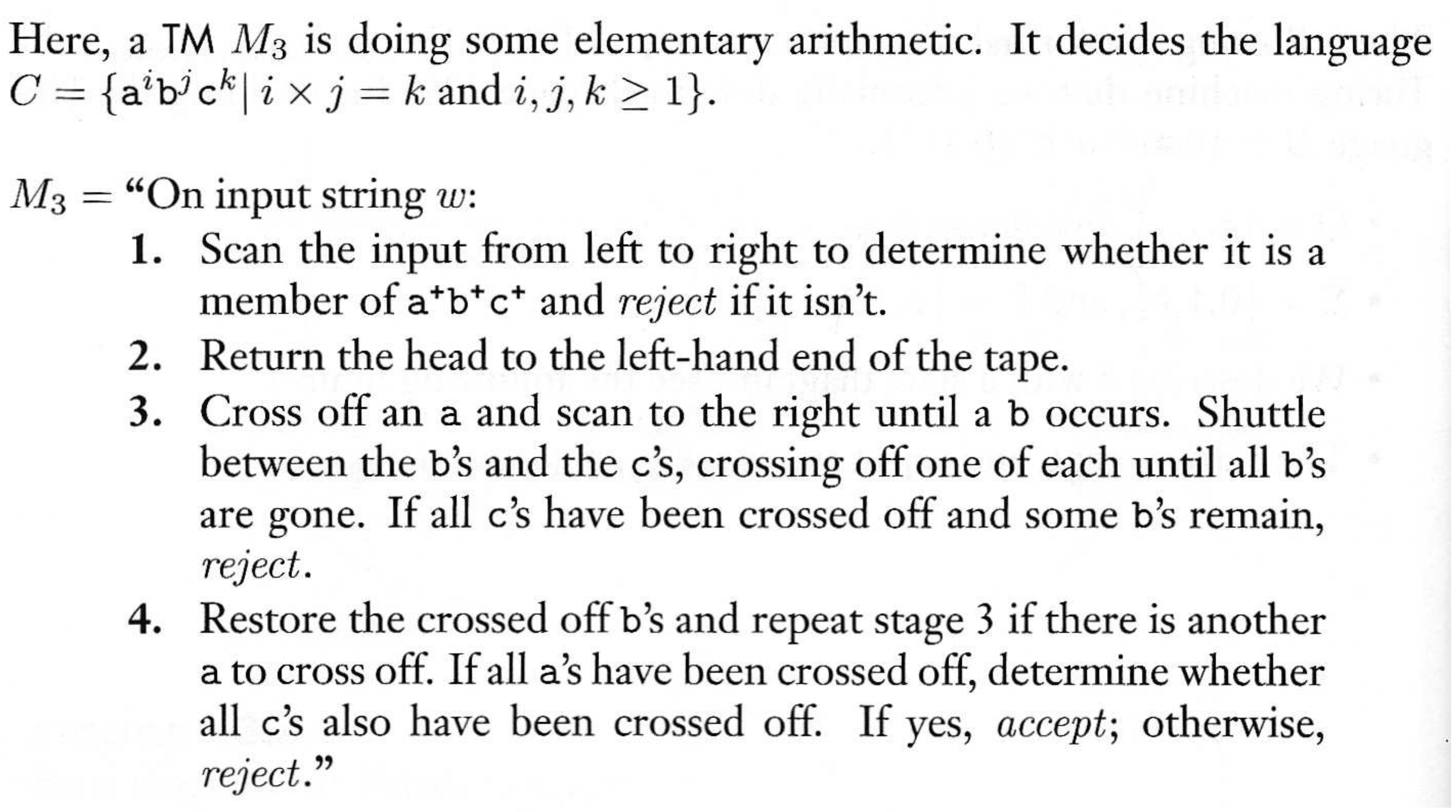
Alternate Definition

Definition: A ***Turing-recognizable language*** is a formal language for which there exists a Turing machine that will halt and accept when presented with any string in the language as input but may either halt and reject or loop forever when presented with a string not in the language. Contrast this to ***(Turing-)decidable languages***, which require that the Turing machine halts in all cases.

Source: Wikipedia

Decidable Languages

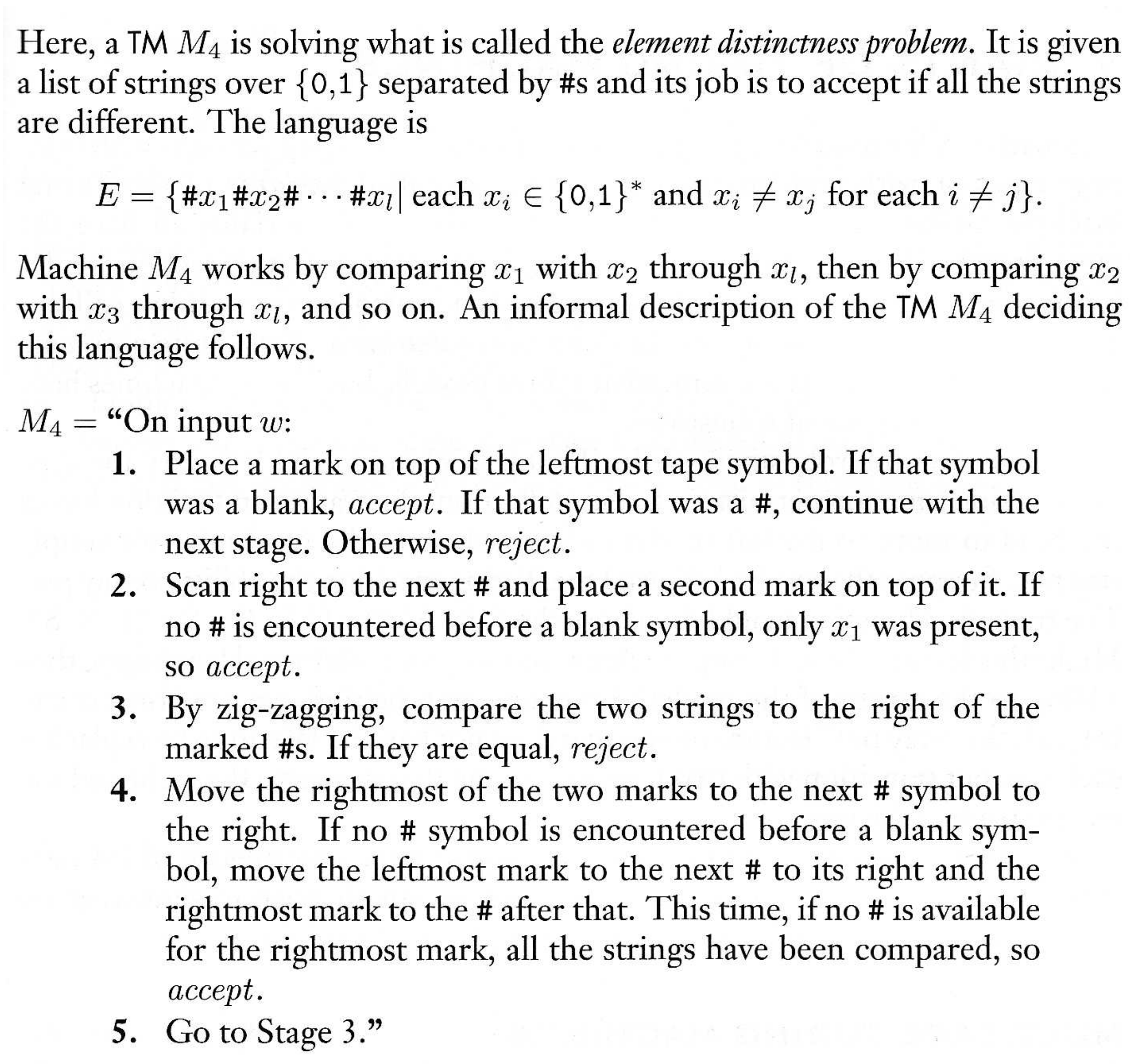


Here, a TM M_3 is doing some elementary arithmetic. It decides the language $C = \{\mathtt{a}^i\mathtt{b}^j\mathtt{c}^k \mid i \times j = k \text{ and } i, j, k \geq 1\}$.

M_3 = "On input string w:

1. Scan the input from left to right to determine whether it is a member of $\mathtt{a}^+\mathtt{b}^+\mathtt{c}^+$ and *reject* if it isn't.
2. Return the head to the left-hand end of the tape.
3. Cross off an a and scan to the right until a b occurs. Shuttle between the b's and the c's, crossing off one of each until all b's are gone. If all c's have been crossed off and some b's remain, *reject*.
4. Restore the crossed off b's and repeat stage 3 if there is another a to cross off. If all a's have been crossed off, determine whether all c's also have been crossed off. If yes, *accept*; otherwise, *reject*."

Decidable Languages



Here, a TM M_4 is solving what is called the *element distinctness problem*. It is given a list of strings over $\{0,1\}$ separated by #s and its job is to accept if all the strings are different. The language is

$$E = \{\#x_1\#x_2\#\cdots\#x_l \mid \text{each } x_i \in \{0,1\}^* \text{ and } x_i \neq x_j \text{ for each } i \neq j\}.$$

Machine M_4 works by comparing x_1 with x_2 through x_l, then by comparing x_2 with x_3 through x_l, and so on. An informal description of the TM M_4 deciding this language follows.

M_4 = "On input w:

1. Place a mark on top of the leftmost tape symbol. If that symbol was a blank, *accept*. If that symbol was a #, continue with the next stage. Otherwise, *reject*.
2. Scan right to the next # and place a second mark on top of it. If no # is encountered before a blank symbol, only x_1 was present, so *accept*.
3. By zig-zagging, compare the two strings to the right of the marked #s. If they are equal, *reject*.
4. Move the rightmost of the two marks to the next # symbol to the right. If no # symbol is encountered before a blank symbol, move the leftmost mark to the next # to its right and the rightmost mark to the # after that. This time, if no # is available for the rightmost mark, all the strings have been compared, so *accept*.
5. Go to Stage 3."

Variants of TM’s

As in the case of FA’s we can construct different variants of the Turing machine. And, as in the case of FA’s, we can show that all these variants have the same computational power: they all recognize the same languages.[a]

The most important variants:

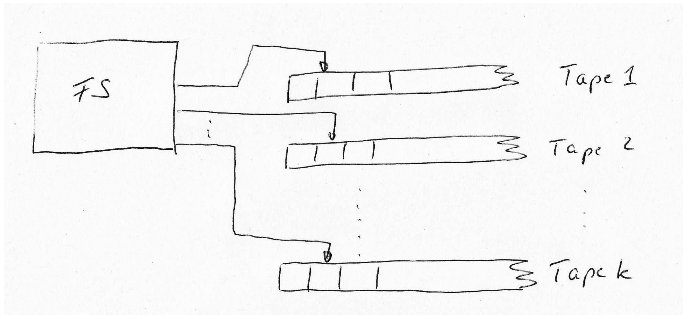
- multi-tape TM’s
- nondeterministic TM’s

[a] This is not to be confused with *complexity*, some of the variants have lower computational complexity than others for the same task.

Multi-tape TM's

A ***multi-tape Turing machine*** is a 7-tuple, $(Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$, where Q, Σ, Γ are all finite sets and

1. Q is the set of states,
2. Σ is the input alphabet not containing the ***blank symbol*** $\sqcup$,
3. Γ is the tape alphabet, where $\sqcup \in \Gamma$ and $\Sigma \subseteq \Gamma$,
4. $\delta : Q \times \Gamma^k \longrightarrow Q \times \Gamma^k \times \{L, R\}^k$ is the transition function with k the number of tapes,
5. $q_0 \in Q$ is the start state,
6. $q_{\text{accept}} \in Q$ is the accept state, and
7. $q_{\text{reject}} \in Q$ is the reject state, where $q_{\text{reject}} \neq q_{\text{accept}}$.



NOTE: For convenience sake we always load the input onto tape 1.

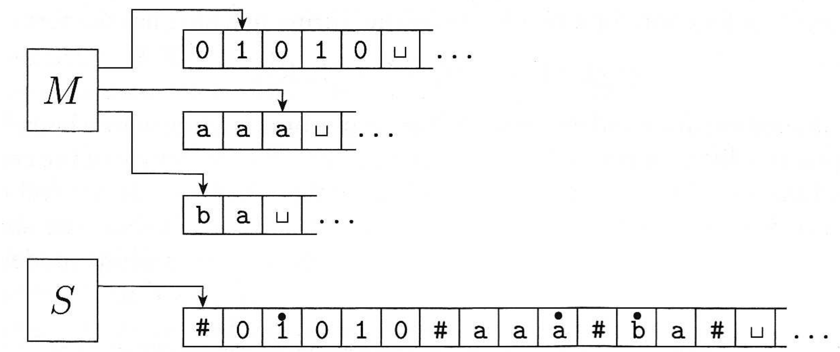
Multi-tape TM's

Theorem: Multi- and Single-tape TM's are equivalent.

Proof Sketch: We show equivalence by demonstrating that each machine can simulate the other.

(a) To show that a multi-tape TM can simulate a single-tape TM is trivial because a single-tape TM is a special case of the multi-tape TM.

(b) A single-tape TM can simulate a multi-tape TM by simulating the k tapes of the multi-tape machine on its single tape. This requires appropriate separation markers between the virtual tapes and memory for the k virtual tape heads. Graphically,



□

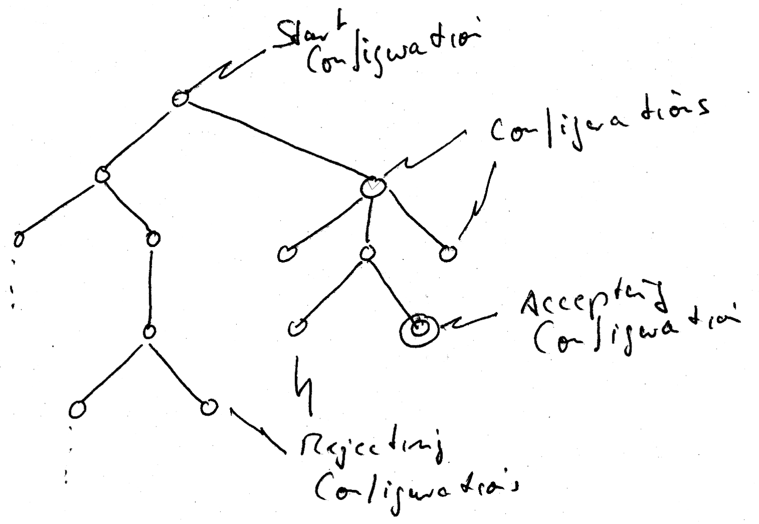
Multi-tape TM’s

Corollary: A language is Turing-recognizable iff some multi-tape TM recognizes it.

Nondeterministic TM's

A ***nondeterministic Turing machine*** is a 7-tuple, $(Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$, where Q, Σ, Γ are all finite sets and

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



NOTE: If all branches of computation in an NTM halt on all inputs then we call it a ***decider***.

Nondeterministic TM’s

Theorem: Nondeterministic TM’s are equivalent to deterministic TM’s.

Proof Sketch: We show equivalence by showing that each TM can simulate the other.

(a) To show that nondeterministic TM’s can simulate deterministic TM’s is trivial because deterministic TM’s are a special case of nondeterministic TM’s.

(b) We can simulate an NTM N with a TM D by having D search through the tree of nondeterministic computation configurations for accepting configurations in a *breadth first* manner. $\Box$

Corollary: A language is Turing-recognizable iff some nondeterministic TM recognizes it

Corollary: A language is decidable iff some nondeterministic TM decides it

Encodings

Since TM's are models of general computations we are able to express algorithms over more structured problems than just structures of strings.

We express more general encodings as $\langle$something$\rangle$. The thing to keep in mind is that this encoding should be "straight forward" in the sense that the process of encoding should not hide steps that might lead to erroneous conclusions when computing with TM's.[a]

Example: Let A be the language of all connected undirected graphs, formally

$$A = \{\langle G\rangle | G \text{ is a connected undirected graph}\}.$$

We could envision that $\langle G\rangle$ is a list of vertices and the edges between them easily obtained by scanning the graph G.

[a] This will become especially important when we talk about computational complexity.

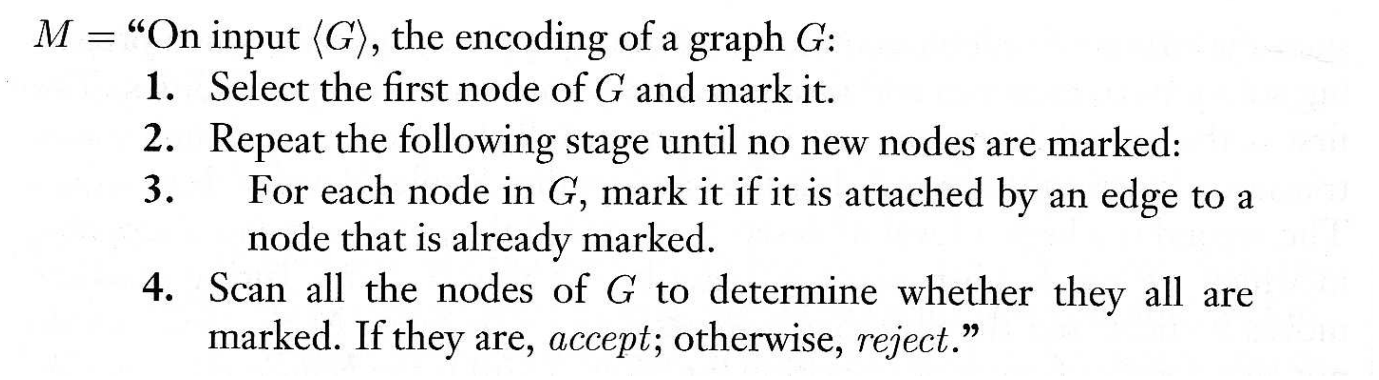
Encodings

Example: Show that the language

$$A = \{\langle G\rangle | G \text{ is a connected undirected graph}\}.$$

is decidable.

Proof: We show decidability by constructing a decider for the language.



```
M = "On input ⟨G⟩, the encoding of a graph G:
    1. Select the first node of G and mark it.
    2. Repeat the following stage until no new nodes are marked:
    3.     For each node in G, mark it if it is attached by an edge to a
           node that is already marked.
    4. Scan all the nodes of G to determine whether they all are
       marked. If they are, *accept*; otherwise, *reject*."
```