

COMPUTATION: DAY 6

BURTON ROSENBERG
UNIVERSITY OF MIAMI

1. TURING MACHINES

Wednesday, 25 March 2026

1.1 The Turing Machine was introduced by Alan Turing in his 1936 paper On Computable Numbers, with an Application to the Entscheidungsproblem. While the paper ranges more widely, our concern is with a question posed by David Hilbert and formalized as the Entscheidungsproblem:

Is there a general, mechanical method that can decide whether any given mathematical statement is true or false?

1.2 Any proof system may be regarded as such a mechanical method: a finite collection of rules by which statements may be derived from one another.

1.3 A proof system is said to be sound if every statement it proves is true, and complete if every true statement can be proven within the system. Hilbert's question may then be understood as asking whether there exists a mechanical proof system that is both sound and complete.

1.4 For sufficiently strong proof systems, the answer is no. In such systems, one may insist on soundness, and be assured that whatever is proven is true, but then completeness must be lost: there will be true statements that cannot be proven.

2. TURING MACHINE FORMAL DEFINITION

2.1 A Turing Machine is a finite automata endowed with a semi-infinite tape, consisting of a sequence of cells, each cell containing one from a finite set of tape symbols. The automata and the tape interact with a head, positioned over a cell, that can read, write and move one step left or one step right, according to the programming of the finite automata. Formally, a Turing Machine is,

$$M = \langle Q, \Sigma, \Gamma, \delta, q_s, q_a, q_r \rangle$$

where,

- Q is a finite set of states,
- Σ is a finite set of symbols comprising the alphabet of the language,
- Γ is a finite set of symbols that can appear on the tape,
 - The tape alphabet includes the language alphabet, $\Sigma \subset \Gamma$,
 - There is a special blank symbol $\sqcup \in \Gamma$, $\sqcup \notin \Sigma$, that fills the uninitialized infinite portion of the tape.
- δ is the transition function, $\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$,
- $q_s, q_a, q_r \in Q$ are respectively the start, accept and reject states, and $q_a \neq q_r$.

2.2 A *computation* by a Turing Machine M has the following steps.

Initialize: The computation is prepared,

- A string $s \in \Sigma^*$ is written on the tape, with no blanks to the left of s ;
- The Turing Machine current state is q_s ; and
- the tape head is positioned over the leftmost cell.

Compute: The Turing Machine computes in steps of application of δ . This involves.

- reading and writing a tape symbol at the tape head;
- transitioning the state; and
- moving the head left or right one cell.

Finalize: The computation either,

- halts by achieving q_r or q_a , or
- never reaches either of these two states — it runs forever never deciding for accept or reject.

2.3 The result of a computation is in general one of three situations,

$$M(s) = \begin{cases} T & \text{if } q_a \text{ is achieved} \\ F & \text{if } q_r \text{ is achieved} \\ \perp & \text{the machine does not halt} \end{cases}$$

Definition 2.1. A language $\mathcal{L} \subseteq \Sigma^*$ is *recursively enumerable* if there exists a Turing Machine M such that $\mathcal{L} = \mathcal{L}(M)$ where we define,

$$\mathcal{L}(M) = \{ s \in \Sigma^* \mid M(s) = T \}$$

Definition 2.2. A language $\mathcal{L} \subseteq \Sigma^*$ is *recursive* if there exists a Turing Machine M such that $\mathcal{L} = \mathcal{L}(M)$ and the set,

$$\{ s \in \Sigma^* \mid M(s) = \perp \}$$

is empty.

Theorem 2.1. If a language and its complement are both recursively enumerable, the language is recursive.

2.4 Recursively enumerable is also called *recognizable* and recursive is also called *decidable*. When the complement of a language is recursively enumerable, the language is called co-recursively enumerable.

3. AN EXAMPLE OF A RECURSIVE LANGUAGE

3.1 The language $\{ a^i b^j c^k \mid i \geq 0 \}$ is recursive. Here is how we will accept it. The tape alphabet is $\{ a, b, c, \$, x \}$ and we begin with the tape,

$$\vdash w \sqcup \dots$$

where $w = \{ a, b, c \}^*$. I break the algorithm down into three subtasks,

Step 1: Preparation

- If the input tape is,

$$\vdash \sqcup \dots$$

accept.

- If the input tape is,

$$\vdash a^i b^j c^k \sqcup \dots$$

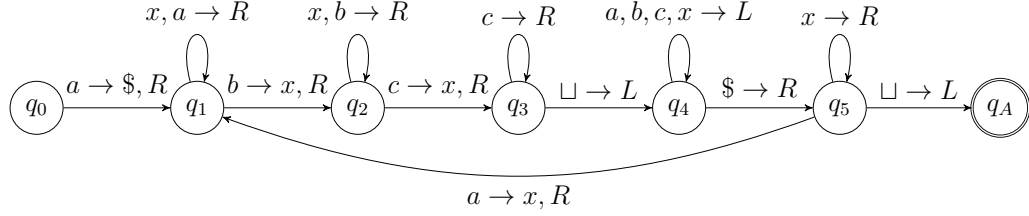
with $i, j, k > 0$; transform it to

$$\vdash \$ a^{i-1} x b^{j-1} x c^{k-1} \sqcup \dots$$

rewind to the end marker and go to step two.

- If w is not of these forms, halt with reject.

Step 2: Termination condition

FIGURE 1. Turing machine recognizing $a^i b^j c^i$, $i > 0$

- If the input tape is,

$$\neg \$x^+ \sqcup \dots$$

accept.

- Else rewind to the end marker and go to step three.

Step 3: Loop transformation

- Loop invariant: the tape contains the string,

$$\neg \$x^* a^i x^* b^j x^* c^k \sqcup \dots$$

with $i, j, k \geq 0$.

- If $i, j, k > 0$ transform the tape to

$$\neg \$x^* a^{i-1} x^* b^{j-1} x^* c^{k-1} \sqcup \dots$$

then rewind to the end marker and go to step two.

- This transformation is accomplished by replacing by x the first a , b and c encountered in a rightward sweep. Attempt this transformation and if it fails the machine rejects.

3.2 The Turing machine given in Figure 1 is an optimized version of the above algorithm. The notation on the arrow between states q_0 and q_1 ,

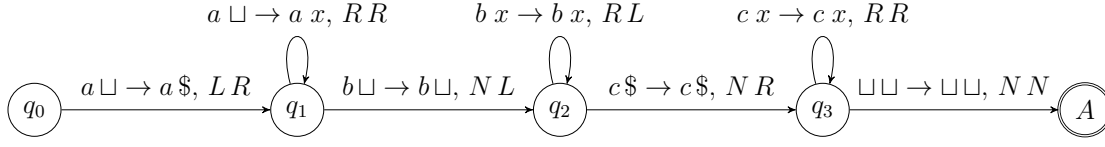
$$a \rightarrow \$, R$$

summarizes the state transition

$$\delta(q_0, a) = (q_1, \$, R).$$

To avoid any clutter, arrows to the reject state are not drawn. Also multiple arrows can be combined with a comma separated list of symbols and the write symbol can be omitted if it will be unchanged by the transition. So the loop on state q_1 with arrow

$$x, a \rightarrow R$$

FIGURE 2. 2-Tape Turing machine for $a^i b^i c^i$, $i > 0$

combines the transtions,

$$\delta(q_1, x) = (q_1, x, R), \quad \delta(q_1, a) = (q_1, a, R).$$

4. MULTITAPE TURING MACHINES

4.1 A deterministic Turing Machine can be defined for more than one tape. Having multiple tapes does give a Turing Machine increased power to decide languages, but only in a certain amount of speed up. The value of these machines is having clear explanations of the explanation of non-deterministic machines in terms of an advice oracle, and in the description of a universal Turing Machine, essentially a stored program computer. Because there exists a Turing Machine universal for all Turing Machines, we get an important result of the limits of computation by Turing Machines.

4.2 To define a k -tape turing machine, we update the transition function to

$$\delta : Q \times \Gamma^k \rightarrow Q \times \Gamma^k \times \{L, R\}^k$$

with the interpretation that the transition depends on all k tape symbols, and they call all be updated, and the heads can move independently left or right on account of the state transition.

4.3 There is increased power. The language $a^i b^i c^i$ is now recognizable in time $O(n)$. Figure 2 is a 6 state TM (counting the reject state not shown) that runs in $O(n)$ steps and recognizes this language.

4.4 However, a single tape Turing Machine can simulate a k -tape Turing machine in several obvious but tedious methods.

Theorem 4.1. A k -tape machine with a time bound of $f(n)$ for a recursive function can be simulated by a one tape turing machine in time $O(f(n)^2)$.¹

¹Hopcroft, Motwani and Ullman p. 347

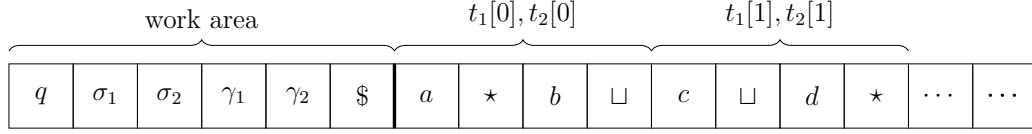


FIGURE 3. Tape layout for the 2 tape simulation

4.5 Notice that this time increase is mostly going from two to one tape, as going from k to two tapes can be simulated in time $O(f(n) \log f(n))$.²

4.6 These bounds are *upper bounds*, meaning there is a way of doing the simulation within the given time bound. One wants to know if there are other, faster ways. There can be practical consequences of *lower bounds* that say which speed-ups are impossible. The following theorem gives a connection between how complex a problem is in terms of the run time of an algorithm recognizing the language and the complexity of the problem in terms of the machine models explored in this course.

Theorem 4.2. If a one-tape turning machine recognizes a language in $o(n \log n)$ time (step count) then the language is regular.

4.7 We give a hint as to how a one-tape Turing S machine can simulate a two-tape Turing machine T .

4.8 The bulk of the tape is organized into repeating blocks of four cells: two cells encode the first tape, and two cells encode the second tape. Within each pair, the first cell stores the tape symbol, and the second indicates whether the head of the corresponding tape is positioned at that location. In Figure 3, the symbols in the first two cells of tape one are a, c , with the head over a ; the symbols in the first two cells of tape two are b, d , with the head over d .

4.9 The leftmost portion of the tape is a work area, consisting of six consecutive cells. The sixth cell contains a special symbol that marks the separation between the work area and the repeating tape-content blocks.

4.10 The very first cell contains the symbol representing the current state of the simulated two-tape machine, drawn from a tape alphabet enriched with a unique symbol for each state. The next two cells store the symbols currently scanned on the first and second tapes. The transition function is then consulted on these three cells, and the resulting five symbols — the new state, the two write-back symbols, and the two direction symbols—are written into the first five cells of the work area.

²Hennie and Stearns, 1966

4.11 The way the simulating machine S consults the two-tape machine T is by transitioning to a special query state q_q . For each transition,

$$\delta_T(q, \sigma_1, \sigma_2) = (q', \sigma_3, \sigma_4, \gamma_1, \gamma_2),$$

there is a path which leads from q_q to a state unique for the sequence q, σ_1, σ_2 . From that state a sequence of states writes $q', \sigma_3, \sigma_4, \gamma_1, \gamma_2$ into the first five cells of the work area. Then transition via state q_e back to the simulator S program.

$$\begin{aligned} \delta_S(q_q, q) &= (q_1, \sqcup, R) \\ \delta_S(q_1, \sigma_1) &= (q_2, \sqcup, R) \\ \delta_S(q_2, \sigma_2) &= (q_3, \sqcup, R) \\ \delta_S(q_3, \cdot) &= (q_4, \sqcup, R) \\ \delta_S(q_4, \cdot) &= (q_5, \gamma_2, L) \\ \delta_S(q_5, \cdot) &= (q_6, \gamma_1, L) \\ \delta_S(q_6, \cdot) &= (q_7, \sigma_4, L) \\ \delta_S(q_7, \cdot) &= (q_8, \sigma_3, L) \\ \delta_S(q_8, \cdot) &= (q_e, q', N) \end{aligned}$$

4.12 The overall scheme of the one-tape simulating machine S can be described in three main stages:

- (1) **Scan and collect symbols:** S scans the tape to locate the head markers of the simulated two-tape machine and brings the corresponding symbols into the work area. This provides the current state and the two tape symbols needed to consult the transition function.
- (2) **Query the transition function:** S enters the special query state q_q and the sequence of states rewrites the work tape and exits by q_e .
- (3) **Update the tape:** From q_e , S performs a forward scan of the tape to update the contents according to the result stored in the work area. This includes writing the new symbols, moving the head markers left or right as indicated, and updating the current state in the first cell of the work area.

5. NONDETERMINISTIC TURING MACHINES

5.1 The concept of a nondeterministic Turing Machine is that there are multiple computation paths, and the machine explores them in some fashion. Therefore the transition function describes the possibilities of transitions, but does not indicate

which one will contribute to a given computation, As a consequence of having multiple paths, we have to define carefully the notion of accepting a string. A particular computation path is of one of three kinds,

- (1) the computation ends at the accepting state;
- (2) the computation path ends by arriving at a transition to the empty set;
- (3) or the computation path does not end.

Definition 5.1. A non-deterministic Turing Machine is a 6-tuple,

$$N = \langle Q, \Sigma, \Gamma, \delta, q_s, q_a \rangle$$

where Q is a finite set of states, Σ is a finite language alphabet, $\Gamma \supset \Sigma$ is a finite tape alphabet, containing the blank symbol $\sqcup$ which is not in the input alphabet, $q_s \in Q$ is the start state and $q_a \in Q$ is the accepting state. The transition function maps state, tape alphabet pairs to a set of possible next-state, next-letter, next-action triples.

$$\delta : Q \times \Gamma \longrightarrow \mathcal{P}(Q \times \Gamma \times \{L, R\}).$$

Definition 5.2. A *computation* in a (multi-tape) non-deterministic Turing Machine is a sequence of state transition, tape writings, and tape actions allowed by the transition function that begins at with the input written in the language alphabet in the leftmost tape cells of the tape, and the result of the semi-infinite tape filled with blanks (if multi-tape: and all other tapes fill with blanks), and the head (if multitape: all heads) at that leftmost cell. A computation terminates on entering the accept state, or if the transition function yields the null-set.

Definition 5.3. A string is *accepted* if some computation accepts the string. There is no definition for rejecting a string.

5.2 [Advice] A nondeterministic Turing machine is no more powerful than a deterministic Turing machine for recursive enumerable languages. Any deterministic machine for a recursive language is easily transformed to a non-deterministic Turing machine accept the same language. A non-deterministic machine can be made

deterministic by adding a tape on which *advice* is written that resolves the non-deterministic choice of a transition at each step.³ For this, we introduce a two input deterministic machine that takes advice to treat a non-deterministic machine as a deterministic machine when provided with *advice* that rectifies the non-determinism.

Theorem 5.1. Given a non-deterministic machine $N(s)$ there is a deterministic machine $M(a, s)$, with a read-only *advice tape* which moves right on every transition such that for a recursively enumerable language $\mathcal{L}(N)$,

$$s \in \mathcal{L}(N) \iff \exists a \text{ s.t. } M(a, s) \text{ accepts.}$$

5.3 [Proof] Let

$$N = \langle Q, \Sigma, \Gamma, \delta, q_s, q_a \rangle$$

be a non-deterministic Turing machine.

Let $q_r \notin Q$ be a reject state.. Let d be the maximum set size of any value of δ . Let $\chi = [1, 2, \dots, d]$ be an indexing set to order all the choices in a value of δ ,

$$\delta(q, s) = \langle d_i \rangle_{i \in \chi}$$

with the conventions,

- (1) If the arity of some particular $\delta(q, s)$ is less than d , repeat transitions.
- (2) If the transition function yields the empty set, $d_i = q_r$ for all $i \in \chi$.

The deterministic M which takes advice is defined as,

$$M = \langle Q \sqcup \{q_r\}, \Sigma, \Gamma \sqcup \chi, \delta', q_s, q_a, q_r \rangle$$

where δ' is a promotion of δ to use the advice tape according to,

$$\delta'(q, s, a) = \begin{cases} \delta(q, s)_a & \text{if } a \neq \sqcup \\ (q_r, s, R) & \text{otherwise} \end{cases}$$

Suppose $s \in \mathcal{L}(N)$. Then there is a computation in N that leads to q_a . Note the index in M for each transition in the computation in N . Create the advice string a from the sequence of indices. Then $M(a, s)$ accepts. Suppose there is an a such that $M(a, s)$ accepts. The sequence of transitions in M as suggested by a is a sequence of transitions in N that also leads to an accepting state. So $s \in \mathcal{L}(N)$. $\square$

³There is a subtlety for recursive languages, as rejecting a string is not a single thing for non-deterministic machines as it is for deterministic machines. One way to approach this is to note that a recursive language is both recursively enumerable and co-recursively enumerable. Have the recursive language decided by two non-deterministic machines, one recognizing the language and the other recognizing the complement of the language. There might be more elegant ways of handling this, but this way has the advantage of keeping before us an essential difficulty.

Theorem 5.2. A non-deterministic machine is no more powerful than a deterministic machine.

5.4 [Proof] For the nondeterministic machine $N(s)$ let $M(a, s)$ be the deterministic as described in the previous theorem. Now enumerate all possible advice in lexicographic order. If $s \in \mathcal{L}(N)$ then there is an a such that $M(a, s)$ accepts, and the enumeration will find it. So the deterministic machine which is the combination of enumerating χ^* followed by testing each advice by M accepts exactly $s \in \mathcal{L}(N)$. So the language of N is recursively enumerable.

In the other direction, we note again that a deterministic machine can be made a non-deterministic machine without changing the set it accepts. $\square$

5.5 We return to the question of sets which are recursive. They do not naturally fit in the structure of a non-deterministic machine. As was mentioned in the footnote, it can be better approached by considering the recursive enumerability of the complement of the language in question.

Corollary 5.1. A language is recursive if and only if there is a pair of non-deterministic Turing machines, one accepting the language, and the other accepting the complement of the language.

5.6 [Proof] Given a deterministic Turing machine M which decides the language, create two non-deterministic Turing machines. Both make a formal change that replaces the transition function from a value to a singleton set containing that value.

On one, replace any transition to the singleton set of the rejecting state with a transition to the empty set. On the other, replace any transition to the singleton set of the accepting state with a transition to the empty set, then consider the rejecting state to be the accepting state.

Given a pair of non-deterministic Turing machines, N_T accepting the set and N_F accepting the complement of the set, consider the deterministic lifts M_T and M_F . Create a single Turing machine by: enumerating all advice in lexicographic order, alternately running M_T and M_F until one or the other accepts. Because they accept for complementary sets, exactly one must eventually halt with proper advice. $\square$