COMPUTATION: DAY 5

BURTON ROSENBERG
UNIVERSITY OF MIAMI

Contents

1. Turing Machines

Thursday, 27 March 2025

The Turing Machine was introduced by Alan Turing in a paper concerning the Halting Problem. In it, he proved that the problem of whether a finitely described mathematical procedure can always be made to come to a decision. For instance, given a notion of true and false statements, whether a proof system can be made to always prove the true statements and disprove the false statements.

And the answer is no. Proof is not entirely useless because the true statements can be proven. However, the false statements are more accurately described as those statements which cannot be shown to be true. Some false statements can be given finite demonstrations for being false, but in general, this is not possible.

2. Turing Machine Formal Definition

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, $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$,
- $q_s, q_a, q_r \in Q$ are respectively the start, accept and reject states, and $q_a \neq q_r$.

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.

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,

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

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,

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

is empty.

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

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

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

$$\dashv\ w \sqcup \ldots$$

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

Step 1: Preparation

- If the input tape is,
$$\dashv\ \sqcup \ldots$$
accept.
- If the input tape is,
$$\dashv\ a^i b^j c^k \sqcup \ldots$$
with $i, j, k > 0$; transform it to
$$\dashv\ \$\, a^{i-1} x\, b^{j-1} x\, c^{k-1} \sqcup \ldots$$
rewind to the end marker and go to step two.
- If w is not of these forms, halt with reject.

Step 2: Termination condition

- If the input tape is,
$$\dashv\ \$\, x^+ \sqcup \ldots$$
accept.
- Else rewind to the end marker and go to step three.

Step 3: Loop transformation

- Loop invariant: the tape contains the string,
$$\dashv \$\, x^* a^i x^* b^j x^* c^k \sqcup \ldots$$
with $i, j, k \geq 0$.
- If $i, j, k > 0$ transform the tape to
$$\dashv \$\, x^* a^{i-1} x^*\, b^{j-1} x^*\, c^{k-1} \sqcup \ldots$$
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.

4. Multitape Turing Machines

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.

To define a k-tape turing machine, we update the transition function to
$$\delta : Q \times \Gamma^k \to Q \times \Gamma^k \times \{ L, R \}^k$$
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.

There is increased power. The language $a^i b^i c^i$ is now recognizable in time $O(n)$. I leave to the reader a program that runs in $O(n)$ steps and recognizes this language.

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]

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].

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.

[1] Hopcroft, Motwani and Ullman p. 347

[2] Hennie and Stearns, 1966

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.

5. Nondeterministic Turing Machines

The concept of a nondeterministic Turing Machine is that their 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 has one accepting state, and no rejecting states and a transition function,

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

A string is accepted if some computation accepts the string. The string is rejected if all computation paths halt and no computation path accepts the string.

5.1. **Advice.** A nondeterministic Turing machine is no more powerful than a deterministic Turing machine. Any deterministic Turing machine can be turned into a nondeterministic machine without changing what languages are recursive or recursively enumerable.[3]

A nondeterministic (Turing) machine can be simulated by a deterministic machine which methodically searches through all possible computations for an accepting computation.

[3]There is a subtlety for recursive languages, as rejecting a string is not a single thing for nondeterministic 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.

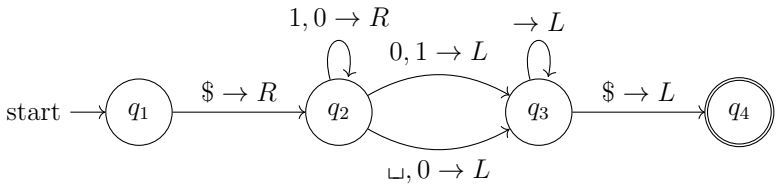


FIGURE 1. Lexicographically advancing the advice (adding one)

(1) The first step is to take the nondeterministic one tape machine $\mathcal{N}(s)$ and modify it to a deterministic two tape machine $\mathcal{A}(a, s)$ where the additional tape is the advice tape upon which an *advice string* is written. The advice tape is a read-only tape that is advanced to the right one step with every transition.
(2) Pick an indexing for the possible transitions,

$$\delta(q, s) = \{\, \chi \,\} = \langle\, \chi_1, \chi_2, \ldots, \chi_d \,\rangle$$

and promote the function δ to δ' that uses the advice tape symbol to chose the transition,

$$\delta'(q, s, a) = \delta(q, s)(a) = \chi_a$$

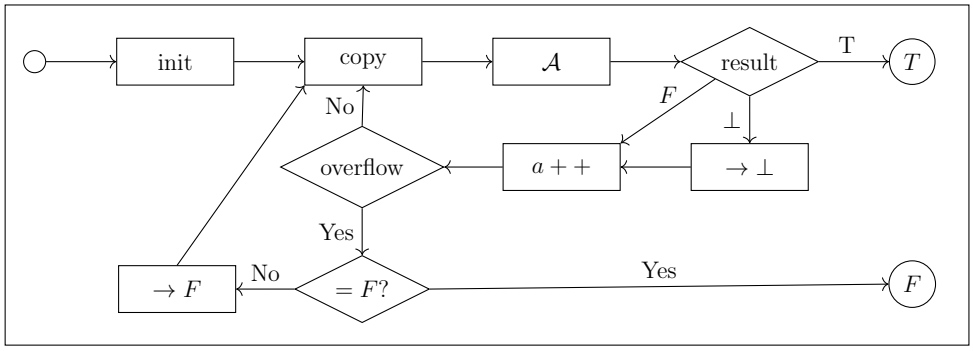
The symbol set for the advice tape needs at least d symbols. To make matters simple, assign a χ_a for all a even if that means repeating an assignment, that is, $\chi_a = \chi_{a'}$ for distinct a and a'.
(3) The machine $\mathcal{A}$ has four distinguished states. The start, accept, reject and terminated states. The start and accept states of $\mathcal{A}$ are those of $\mathcal{N}$. The reject state is entered when δ gives the empty set, as a deterministic machine must go somewhere. And the terminated state is entered when the advice runs out, that is, the advice head reads a blank on the advice tape.
(4) As a stand alone machine, $\mathcal{A}$ starts in the start state, accepts in the accept state and rejects in the reject or terminated states. As part of the search machine, these states entry and exits points as control is passed between $\mathcal{A}$ and the search machine.

Theorem 5.1. For any recursively enumerable language L recognized by the nondeterministic machine $\mathcal{N}(s)$ there is a deterministic machine $\mathcal{A}(a, s)$ such that,

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

In addition, if $\mathcal{N}(s)$ decides the language, then,

$$s \notin L \iff \exists\, n \in \mathbb{Z} \text{ s.t. } \forall\, a, |a| > n, \mathcal{A}(a, s) \text{ rejects.}$$



- `Init`: initialize the advice tape with the empty string and a mark to decide reject.
- `Copy`: erase the work tape and copy the input tape to the work tape.
- $\mathcal{A}$: Run the advice machine on the work tape and the advice tape.
- `result`: The result is sorted out.
- $\to \perp$: mark the advice tape if the result was a terminated, to signal some advice at this length did not reject.
- $++a$: advance the advice on the advice tape.
- `overflow`: if the advice advances in length, a choice is made to possible reject the input.
- $=?F$: if the tape mark was not updated, all advice rejects, so reject.
- $\to F$: mark the tape.

FIGURE 2. Searching for Advice

5.2. **Enumerating advice.** The machine $\mathcal{A}$ will be a subroutine called a large number of times inside of the run of a four tape search machine $\mathcal{S}$. Each run of $\mathcal{A}$ will start with the original input on its input tape, and the next in an sequence of advice strings written on its advice tape, exhausting all advice strings, given from shortest (the empty string)

If ever $\mathcal{A}$ exits back to $\mathcal{S}$ through its accept state, $\mathcal{S}$ accepts. Otherwise $\mathcal{A}$ exits back to $\mathcal{S}$ through the reject or terminated state. It the machine $\mathcal{S}$ is convinced that all future exits are through the reject state, it will reject. Else it can continue looking forever at longer and longer advice strings.

Advice strings must be enumerated in size order, and so that all possible advice strings are attempted. An example of how this can be done is given in Fig. 1.

The machine $\mathcal{S}$ is convinced that all future exist from $\mathcal{A}$ will be through the reject state if all exits for all advice strings of a given length were through the reject state. Longer advice won't change things. The machine $\mathcal{S}$ knows when the length of the advice string increases, so some carrying out this check is simple.

The machine $\mathcal{S}$ will have two tapes in addition to the input and advice tape. To avoid (or invite) confusion, rename the input tape of $\mathcal{A}$ the work tape of $\mathcal{S}$. The input tape of $\mathcal{S}$ will never be written. It will be copied to the work tape before each run of $\mathcal{A}$. We cannot assume anything of $\mathcal{A}$. When it exits, we can make no assumption where the work tape head is, or what cells were written. To erase completely and find the leftmost cell of the work tape, a fourth tape, called the guide tape, will be used.

The heads of both the work and guides tapes will start together over the leftmost cell, and will unfailingly move together left and right. Where the head is on the guide tape is also where the head is on the work tape, and the cells used by the work tape are the cells used by the guide tape. The leftmost cell of the guide tape will have a distinct marking. Any cell scanned by the guide tape will get an in-use marking. To erase and rewind both tapes, in tandem move right to a blank on the guide tape, move left while writing blanks on both the work and guide tapes, and stop when the guide tape is over the end of tape marker.

By the way, a small variety in this end marker can be used to store the needed information to carry out the rejection logic. A flowchart for the overall workings of $\mathcal{S}$ is given in Fig. 2.

6. Universal Turing Machines

A feature of our computers is that the hardware is fixed and the utility of the machine is determined by software. Even the software is under the control of software, as there is software to download new software onto our computers, install and run the software. This is also a fact about Turing Machines. The state, which is encoded in the transition function δ can be a university program to carry out a program description that is written onto of the tapes before the start of the computation. For any Turing Machine, the δ can be transcribed into a string, which is the program to be run, and the *Universal Turing Machine* can run it.

A UTM will need a universal set for Σ, Γ and Q. We represent each as a binary code over $\mathcal{B} = \{ 0, 1 \}$. Our universal language for all these elements of a Turing Machine is as it is on our practical computers, strings of 0's and 1's. We consider how to transcribe our generally described TM into one over the language B^*.

Given the tape alphabet Γ we have an injective map to strings $\mathcal{B}^k$ with k sufficiently large so that $| \Gamma | \leq 2^k$.

Given the state set Q we have an injective map to string $\mathcal{B}\mathcal{B}^+$ with the preassigned mapping of the start state to 00, the accept state to 01, the reject state to 10 the reject state, and 11 a reserved symbol.

In representing the map δ as a sting in $\mathcal{B}^*$, the string 0 notates the "move left" action and the string 1 notates the"move right" action.

A transition is encoded as a string as,

$$(q, \gamma, q', \gamma', a) \implies \mathcal{T} = \mathcal{B}\mathcal{B}^+ \sqcup \mathcal{B}^k \sqcup \mathcal{B}\mathcal{B}^+ \sqcup \mathcal{B}^k \sqcup \mathcal{B}$$

And the entire transition table written on the program tape as,

$$\dashv \left(\sqcup \mathcal{T} \right)^+ \sqcup \sqcup$$

A working tape encoded as,

$$\dashv \left(\sqcup \mathcal{B}^k \right)^* \sqcup \sqcup$$

The current tape head position is over the blank to the left of the k-length tape symbol, and is initialized to the leftmost cell on the tape (which is a blank).

An additional "current state" tape has the symbol of the current state written on it, and is initialized to the well-known name of the start state 00. After each step this tape checked for equality to 01 or 10, and on match the universal machine enters its own accept or reject state.

The program that is written into the states of the universal machine is a loop which marches down the program tape looking for a match between the current state and the current input symbol, and on match replaces these with the symbols in the matched transition $\mathcal{T}$. The tape head is advanced according to the last symbol in $\mathcal{T}$. Encountering a double-blank means the table is incomplete and this can be taken as an implied transition to a reject state.

6.1. **Enumeration of Turing Machines.** We have described a UTM, but I wish to go further and describe that machine as $U_i(j)$, a machine on two integer indices, i and j. We will then speak of the computation of the i-th machine on the j-th input. We will have then comprised all possible computations into a single integer function,

$$U_i(j) : \mathbb{Z} \times \mathbb{Z} \to \{ T, F, \perp \}.$$

We need to enumerate all possible TM's. Our UTM as a strict syntax for programs, a subset of strings over $\mathcal{B}^*$. This string set is Regular, so we can write a regular expression accepting strings that are TM descriptions, and create a TM that enumerates all strings in $\mathcal{B}^*$ in lexicographic order applying a TM program for the regular expression to decide which strings are programs. As a string matches, it is written on an output tape, and the the sequence of output programs is then our enumeration of programs. If the i-th program is required, this enumerator is run with a counter that stops at the i-th output and writes just that program on a program tape for subsequent use by $U_i(j)$.

Likewise to find the j-th input. In detail, this is dependent on the program, as the tape symbol set, dependent on k, will be determined by the program syntax.

However, we now have a program which given i and j, writes the i-th program on the program tape, the j-th input on the work tape and then starts the UTM.

7. The Entscheidungsproblem

Theorem 7.1. The set,

$$A_{TM} = \{\, (i,j) \in \mathbb{N} \times \mathbb{N} \,|\, U_i(j) \,\}$$

is recursively enumerable but not recursive.

Proof: The set is recursively enumerable because it is recognized by $U_i(j)$.

We prove it is not recursive, suppose A_{TM} were recursive. We will show a contradiction. If A_{TM} were recursive, there is a recursive function $H(i,j)$ which is true when $U_i(j)$ is true, and false otherwise,

$$H(i,j) = \begin{cases} T & U_i(j) = T \\ F & \text{otherwise} \end{cases}$$

The function H resolves the undecided cases of $U_i(j)$. We can then define the recursive function D,

$$D(i) := \neg H(i,i)$$

by running H until its result, then negating it as the result of D. Since D is computable, there is a TM that computes it. Let that be the j-th TM,

$$D(i) = U_j(i).$$

We then ask for the result of giving D as input this index.

$$D(j) = T \implies U_j(j) = T \implies H(j,j) = T \implies D(j) = F.$$

Therefore $D(j)$ cannot be true.

$$D(j) = F \implies U_j(j) \neq T \implies H(j,j) = F \implies D(j) = T.$$

So $D(j)$ cannot be false either. Therefore there can be no H that decides A_{TM}. $\square$.

Theorem 7.2. The complement set of A_{TM} is not recursively enumerable. That is, you cannot recognize non-halting.