COMPUTATION: DAY 5

BURTON ROSENBERG
UNIVERSITY OF MIAMI

Contents

1. Turing Machines

Monday, 5 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,

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

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,

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

is empty.

Recursively enumerable is also called *recognizable* and recursive is also called *decidable*.

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. Complexity Theory

Given a recursive language, and a Turing Machine program accepting the language, we can count the number of steps required to come a decision whether to accept the string or not. The *Computability Theory* might be seen as not caring about this step count, as it is concerned with what can be done and what cannot be done. Consideration of this step count, as well as the use of other computation resource needed to decide a recursive language, is the subject of the *Complexity Theory*.

Complexity theory wishes to determine something about the problem that is solved, not the particular program that solves it. The program itself has to elements, the algorithm used and the coding of the algorithm. All these things make the exact number of steps arbitrary. However, the order of magnitude of the step count, as a function of input size, for the best possible program to solve the problem, is not arbitrary. That is the complexity of the problem.

The above program as an order of magnitude runtime of $O(n^2)$ for input of length n. This is because we have a loop that includes sweeping back and forth over the string twice. A single sweep is no more than $k\,n$ steps for some k. The sweep in the loop replaces some letters by x's so after not more than n sweeps the algorithm runs out of letters to replace.

This does not mean the complexity of recognizing the language is exactly $O(n^2)$, as there might be a faster algorithm. However, complexity theory has some very powerful results. For instance,

Theorem 4.1. If a one-tape turning machine recognizes a language in $o(n \log n)$ time (step count) then the language is regular.

Therefore perhaps we can solve this problem in $O(n \log n)$, but not faster.

5. Nondeterministic Turing Machines

The non-deterministic counterpart of the (deterministic) Turing machine differs in that,

(1) The transition map δ maps $Q\times\Gamma$ is a set of actions, a subset of $Q\times\Gamma\times\{ L, R \}$,

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

This gives two new situations — one of under–determinacy when this subset has more than one element and one of over–determinacy when the result is the empty set.

(2) Non-determinism introduces the possibility of some computation paths terminating in an accept state and others do not. We define accepting as the existence of a terminating computation path that accepts. Other paths, if any, do not matter.

(3) We will have no rejecting states. A string is rejected if all possible computation paths terminate and the string is not accepted.

As sort of memory aid for the definition of accepting, rejecting and non-halting is the strength diagram,

$$T > \perp > F$$

where any accept path wins, and if none, any non-terminating path wins.

A deterministic TM is a non-deterministic TM for which the transition function gives a singleton set or the empty set, where where the empty set is assigned to exactly the reject and accept states, and the designation of "reject" is removed.

6. 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 6.1. 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].

7. Advice and the Power of Nondeterminism

The nondeterministic choices determine the computation path. An *oracle* is a hypothetical correspondent to the Turing Machine which chooses an advice string in some space of advice strings Ω^*, and presents that to the NTM to resolve its non-determinism. Let $N(s)$ be the computation of an NTM on string s. An oracle machine is a two tape deterministic machine M with an additional *terminated* state $\perp$. The computation on string s and advice a is denoted $M(s, a)$. The rules for such advice strings are that, for language $S \subseteq \Sigma^*$,

- For any s such that $N(s) = T$ there is an advice string $a \in \Omega^*$ such the $M(s, a) = T$;
- For all s such that $N(s) \neq T$, then no advice string $a \in \Omega^*$ does $M(s, a)$ return true. It may either reject or run out the advice string and enter the terminated state.
- Halting advice cannot be retracted. That is if $M(s, a) = T$ then $M(s, a') = T$ for all a' for which a is a prefix, and likewise for F.

In the case of $N(s) = F$, the machine M needs fairly extensive evidence to reject at string,

$$N(s) = F \implies \exists\, k \text{ s.t. } \forall a \in \Omega^k, M(s, a) = F$$

Here we use that advice cannot be retracted, so a decision to reject on shorter advice applies to all longer advice.

We note that such a machine M is easily constructed from a machine N by indexing the set result of $\delta(q, \gamma)$ with elements from Ω,

$$\delta_N(q, \gamma) = \{\, ((q', \gamma', a)_\omega \,|\, \omega \in \Omega \,\}$$

and then selecting,

$$\delta_M(q, \gamma, \omega) = \delta(q, \gamma)(\omega).$$

[1] Hopcroft, Motwani and Ullman p. 347

[2] Hennie and Stearns, 1966

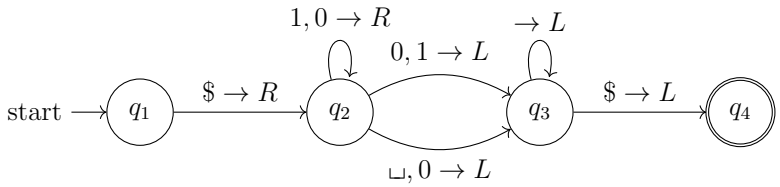


FIGURE 1. Lexicographic next of the advice

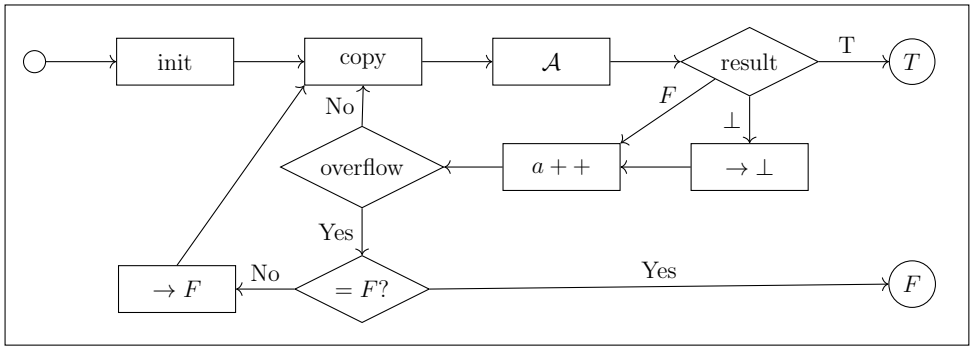
With every TM step the advice tape head advances one to the right. The blank symbol on the advice tape will terminate the computation and return the value $\perp$.

7.1. **The simulation argument.** From an NTM machine N the oracle machine M is constructed, as described above. For an s in the language, a systematic search can be made for an a such that $M(s, a)$. This is best described by a *multi-tape Turing Machine* which is set to have k tapes, and k heads that read and written and moved over with each state transition. We we later show how multiple tapes are a convenience, and do not change the class of languages recognized or decided by Turing machines.

7.2. **Enumerating advice.** The simulation will try all advice by enumerating the strings in Ω in lexicographic ordering. This means, shorter string first, having placed in some arbitrary order the elements of Ω. The increment that increases the length is a significant event, because if all advice of a given length are rejections, that the string can be rejected. An example of this for $\mathcal{B} = \{ 0, 1 \}$ is Fig. 1. Note that the state $q3$ can be split into two states to remember whether there was a change of length in the advice.

7.3. **Input and the work tape.** In addition to the input and advice tape, the third tape is a work tape. The input tape is copied to the work tape. The machine M is started on the work and advice tape. The halting states T, F or $\perp$, rather than halting, return to the simulator for a decision on further action. If the return is through the accepting state, than the simulator accepts: the current advice is an a such that $M(s, a) = T$. Otherwise the simulator will restart M with the next advice string in lexicographic order. A flowchart is on Fig. 2.

7.4. **Recognizing rejection.** An additional detail is that when the advice tape advices to the next length, it is somehow marked signaling that no $\perp$ outcomes have occurred at this length. When the simulation ends transition for a terminated state, this mark is cleared. At the next increment of the advice tape, if the mark is still present, all advice strings of this length have returned F and the simulator will halt rejecting.



- `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.
- $++a$: advice 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. Simulating a NTM with a TM

Theorem 7.1. If N is an NTM that recognizes a language, the simulator will recognize that language. If N decides the language, the simulator will decide that language.

As any MT is an NTM with $|\Omega| = 1$, the converse is also true.

Theorem 7.2. NTM's and NT's recognize and decide exactly the same languages.

8. 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.

8.1. **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.

9. The Entscheidungsproblem

Theorem 9.1. The set,

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

is RE but not recursive.

Proof: The set is recursive 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}. $\square$.