THE UNIVERSAL TURING MACHINE

Mihai-Valentin DUMITRU
mihai.dumitru2201@upb.ro

October 2023

We have seen various examples of Turing Machines designed to solve some particular problem. For each problem we encountered, we created a new machine: we defined a new 6-tuple (usually implicitly, by just writing the transition table), specifying what states the machine has, what tape symbol it uses, what transitions it can perform etc.

We will now show that it is possible to create a single machine capable of doing anything that some Turing Machine can do. In particular, this one machine could *solve* any *solvable* problem. We will call this a *"Universal Turing Machine"* (UTM).

A UTM receives as input the *encoding* of a pair (M, w), consisting of some Turing Machine M and some string w, then decodes the description of M in order to simulate how M would run when given input w.

The only thing that a UTM has to "know" how to do is read a Turing Machine description and correctly apply its effects on the string w.

In order to prove such a machine can exist, we need to first design a proper encoding for arbitrary Turing Machines, then specify the implementation of a UTM: how it can simulate another machine based on its encoding.

1 Encoding Turing Machines

1.1 Dealing with different alphabets

Remember that the definition of a Turing Machine involves *alphabets*. If each machine can have its own alphabet of arbitrary size, holding arbitrary symbols, how can we hope to design a single machine to simulate all others?

It seems like we would have to specify a single machine with a single input alphabet and a single tape alphabet that is somehow capable of "representing" every possible alphabet; an impossible task.

Instead, our Universal Turing Machine will only be "universal" in regards to all possible machines over a single specific alphabet, namely $\Sigma_b = \{0, 1\}$.

However, the loss in generality is not as great as it might first seem, for reasons we will explore below.

For any alphabet $\Sigma = \{s_1, s_2, ..., s_n\}$, we can create a *"binary encoding"* of possible strings.

Definition 1.1. A binary encoding of an alphabet $\Sigma = \{s_1, s_2, ..., s_n\}$ is an injection:

$$enc : \Sigma^* \to \{0, 1\}^*$$

with the property that:

$$\forall w \in \Sigma^* (w = \sigma_1\sigma_2...\sigma_m;\ \sigma_1, \sigma_2, ..., \sigma_m \in \Sigma), enc(w) = enc(\sigma_1)enc(\sigma_2)...enc(\sigma_m)$$

Here we abused notation a bit; in the expression $w = \sigma_1\sigma_2...\sigma_m$, each σ_i refers to some *symbol* from Σ; however, in the final expression $enc(\sigma_1)enc(\sigma_2)...enc(\sigma_m)$, each σ_i refers to *a string consisting of a single symbol*.

Put into words, the binary encoding of a word is the concatenation of the encodings of its symbols.

Theorem 1.1. There exists a binary encoding $enc : \Sigma^* \to \{0, 1\}^*$, such that for any Turing Machine $M = (Q, \Sigma, \Gamma, B, q_1, \delta)$, there exists a Turing Machine $M_b = (Q_b, \{0, 1\}, \Gamma_b, B_b, q_{1_b}, \delta_b)$ such that:

$$\forall w \in \Sigma^*, (M[w] \to v) \Rightarrow (M_b[enc(w)] \to enc(v))$$

Proof. We'll denote the symbols of Σ: $\sigma_1, \sigma_2, ..., \sigma_n$, where $n = |\Sigma|$.

Our encoding function will map each σ_i to a string of 1, followed by i 0s. E.g., $enc(\sigma_4) = 10000$

M_b will have a copy of of all states of M ($Q \subseteq Q_b$), as well as all tape symbols of M ($\Gamma \subseteq \Gamma_b$). The blank symbol of M is the blank symbol of M_b.

The initial state of M_b will be the initial state of M.

Like in the multitape-to-single-tape simulation, we will construct a machine M_b that simulates, in turn, each transition of M, by going through multiple *phases*.

1. For any transition of the form:

   $\delta(q, g) = (r, h, D)$, with $q \in Q, r \in Q', g, h \in \Gamma \setminus \Sigma, D \in \{-, \rightarrow\}$

   M_b will have a transition:

   $\delta_b(q, g) = (r, h, D)$

   I.e. if the transition **doesn't involve input symbols and the head won't move left**, there's nothing special that needs to be done.

2. For any transition of the form:

   $\delta(q, g) = (r, h, \leftarrow)$, with $q \in Q, r \in Q', g, h \in \Gamma \setminus \Sigma$

   M_b will have a transition:

   $\delta_b(q, g) = (r_{aux}, h, \leftarrow)$

   and:

   - $\delta_b(r_{aux}, 0) = (r_{aux}, 0, \leftarrow)$
   - $\delta_b(r_{aux}, 1) = (r, 1, -)$
   - $\delta_b(r_{aux}, c) = (r, c, -)$, for any $c \in \Gamma \setminus \Sigma$

   If the transition doesn't involve input symbols and the head moves left, we might encounter a binary encoding of an input symbol; if we do, we need to skip over the 0s of our encoding to place M_b's head over the initial 1.

   If we encounter a non-input symbol, we just go to the next state.

3. For any transition of the form:

   $\delta(q, g) = (r, \sigma_j, D)$, with $q \in Q, r \in Q', g \in \Gamma \setminus \Sigma, \sigma_j \in \Sigma, D \in \{\leftarrow, -, \rightarrow\}$

   M_b will have a transition: $\delta_b(q, g) = (r\_shift\_j\_D, 1, \rightarrow)$

   Where the purpose of the state $r\_shift\_j\_D$ is to shift the rest of the tape contents right by j symbols, filling the space created with j 0s, transition to state r and move the *simulated head* in direction D.

   Moving the simulated head to the right, means moving the actual head just one cell to the right of the last inserted 0.

   Keeping the simulated head still, means moving the actual head to the left, past the inserted 0s, until it reaches a 1.

   Moving the simulated head to the left, means moving the actual head left, past the inserted 0s, until it reaches a 1, then moving it once more to the left; if there is a 0 there, move it to the left until a 1 is encountered.

   We won't present explicitly the states and transition rules needed to achieve this effect, but they should seem reasonable.

4. From any state $q \in Q$, when encountering a 1, M_b will need to determine the symbol from Σ whose encoding it just came across. To do this, it will go right, counting each 0 and memorizing the number of 0s (using states). So M_b will have the following transitions:

   - $\delta_b(q, 1) = (q\_sym, 1, \rightarrow)$

- $\delta_b(q\_sym, 0) = (q\_sym_1, 0, \rightarrow)$
- $\delta_b(q\_sym_1, 0) = (q\_sym_2, 0, \rightarrow)$
- $\delta_b(q\_sym_2, 0) = (q\_sym_3, 0, \rightarrow)$
- ...
- $\delta_b(q\_sym_{n-1}, 0) = (q\_sym_n, 0, \rightarrow)$

From each q_sym_i on any non-0 symbol, M_b will transition to a state whose purpose is to shift the right part of the tape left, by i symbols, deleting the 0s that form the encoding of σ_i.

Then, looking at $\delta(q, \sigma_i) = (r, g, D)$, it will either overwrite the remaining 1 with g, if $g \in \Gamma \setminus \Sigma$, or apply the same mechanism as in item 3 to shift the right part of the tape to the right and insert a new symbol σ_j.

□

We can formulate and prove a similar theorem for accepting, rejecting or never halting on an input. Now we can focus strictly on machines whose input alphabet is $\{0, 1\}$.

2 Encoding Turing Machines as binary strings

Like in the example above, where we encoded symbols of some alphabet Σ, we shall use a unary encoding for states, symbols and directions.

Let $M = (Q, \{0, 1\}, \Gamma, B, q_0, \delta)$.

Remember that Q' is a *finite* set of states, so there must be some natural number n, such that $|Q'| = n$[†] Thus we can rename all states of Q' to $q_1, q_2, q_3, ..., q_n$, with the convention:

- q_1 is the initial state
- q_2 is the accepting state Y
- q_3 is the rejecting state N
- q_4 is the halting state H

The index of the other states in Q' can be arbitrary.

Similarly, Γ is a *finite* set of tape symbols, so there must be some natural number r, such that $|\Gamma| = r$[‡]. Thus we can rename all symbols of Γ to $g_1, g_2, g_3, ..., g_r$, with the convention:

- g_1 is 0
- g_2 is 1
- g_3 is the blank symbol

The index of the other symbols in Γ can be arbitrary.

Right now, we can binary encode the first five elements of our 6-tuple. We proceed to design an encoding for the transition function, but first, we will employ the following convention about head movement direction:

- d_1 is *left* ($\leftarrow$)
- d_2 is *right* ($\rightarrow$)
- d_3 is *hold* ($-$)

[†] n must be at least four, because each TM Q' consists at least of an initial states and the three final states Y, N, H.
[‡] r must also be at least three, because our TMs have at least three tape symbols: 0, 1 and □.

With the above conventions, each transition T has the form:

$\delta(q_i, g_j) = (q_k, g_l, d_m)$

We encode such a transition as:

$enc(T) = 0^i 10^j 10^k 10^l 10^m$

Where "0^i" means "the symbol 0 repeated i times. Note this is just a sequence of the *unary encodings* of each number, separated by a 1.

Remember the domain and range of the transition function:

$\delta : Q \times \Gamma \to Q' \times \Gamma \times \{\leftarrow, -, \rightarrow\}$

Each Turing Machine has a finite number of transitions, exactly $|Q| \cdot |\Gamma|$, or $n \cdot r$.

To encode the entire transition function, we simply take the encoding for all the transitions and concatenate them, separating them with a "11".

$enc(\delta) = enc(T_1)11enc(T_2)11enc(T_3)11...enc(T_{nr-1})enc(T_{nr})$

The order that the transitions appear in is irrelevant and can be arbitrary.

Note that the encoding of a transition cannot contain two consecutive 1s, so there is no ambiguity as to where a transition's encoding ends.

As you will soon see, it turns out that all the relevant information about our machine M is contained in the encoding of the transition function. Thus, we conclude this section with:

$$enc(M) = enc(\delta)$$

3 Example encodings

We provide an example encoding for a Turing Machine previously studied: `isEven`.

	0	1	□
q_1	$q_1, 0, \rightarrow$	$q_1, 1, \rightarrow$	$q_2, \square, \leftarrow$
q_2	$Y, 0, -$	$N, 1, -$	$N, \square, -$

First, we need to map the internal states and tape symbols to new names based on our convention.

$q_1 \Longrightarrow q_1$

$Y \Longrightarrow q_2$

$N \Longrightarrow q_3$

$H \Longrightarrow q_4$

$q_2 \Longrightarrow q_5$

$0 \Longrightarrow g_1$

$1 \Longrightarrow g_2$

$\square \Longrightarrow g_3$

Let's take the transition $\delta(q_2, 0) = (Y, 0, -)$ and rewrite it according to the renaming scheme above:

$\delta(q_5, g_1) = (q_2, g_1, d_3)$

Next, we encode it as a binary string:

$$\underbrace{00000}_{q_2} 1 \underbrace{0}_{0} 1 \underbrace{00}_{Y} 1 \underbrace{0}_{0} 1 \underbrace{000}_{-}$$

We then to do this for every transition and concatenate all of them, separated by "11"s (remember that the order is irrelevant, but we'll use the ordering in which they appear above):

$0101010100110100101001001101000100000100010110000001010010100011000001001000010010001000 \hookleftarrow$

11001000100010001000

(The line break, $\hookleftarrow$ is not part of the string, the string just wouldn't fit inside this page).

Recall that the input to the Universal Turing Machine is a tuple (M, w), where M is a Turing Machine and w is a word on which to simulate M.

To encode this tuple, we will simply concatenate the machine's encoding with that of the input, separated by three consecutive ones, "111"[§].

$enc((M, w)) = enc(M)111w$

With this in mind, let's proceed to describing the Universal Turing Machine.

4 Formal definition of the UTM

Definition 4.1. A *"Universal Turing Machine"* U is a Turing Machine such that for any Turing Machine $M = (Q, \{0, 1\}, \Gamma, B, q_1, \delta)$ and string w, $U[enc((M, w))] \equiv M[w]$.

This isn't exactly true, but we opted to write it in a concise form. If $M[w]$ computes some string v, then $U[enc((M, w))]$ will actually compute $enc(v)$.

5 UTM behavior

A UTM is not *unique*, there's an infinite number of machines which respect Definition 4; they may possess various interesting characteristics, but our goal in designing one will be to keep its description clear and concise.

In order to easily visualize our Universal Machine (and write less text) we shall design it as a three-tape Turing Machine. Combined with our single-tape simulation for multitape TMs, we could then obtain a single-tape Universal Turing Machine.

The three-tapes will have the following roles:

1. The first tape will contain the input: the encoding of a (M, w) tuple. The UTM will treat this as a "read-only" tape – it will never modify any symbol on it.

2. The second tape will contain M's current state during the simulation, encoded as a string of zeros. E.g. if M is in state $\mathbf{q_5}$, the second tape will contain five 0s: "00000"

3. The third tape will simulate the contents of M's tape; in any given configuration of U, it will contain the encoding of exactly those symbols that are part of the current configuration of M.

First, the UTM goes through an *initialization* phase, where it properly initializes the second tape with the encoding of of M's initial state, q_1 and the third tape with the encoding w.

Recall the single tape simulation of a multitape Turing Machine; similarly, U will precisely simulate the effect of each transition of M, by using multiple transitions. We will refer to the simulation of one transition of M as a *"phase"*.

At the beginning of each phase, U's first and second heads will be positioned over the first non-blank symbol of their respective tape; the third head will match where M's head should be (on the first 0 of the encoding of M's "current symbol"). Each phase consists of the following steps:

1. Move the first and second heads in tandem to the right, while they're both reading 0.

   If the first head reaches a 1 at the same time that the third head reaches a blank, it means that on the first tape we've reached a transition relevant to the current state of M: go to item 3.

[§]Our construction is carefully designed so that no string of three consecutive ones can appear inside the encoding of a Turing Machine.

2. If we're not reading the encoding of a transition concerning the current state, we need to continue searching; reset the second head to the leftmost non-blank symbol and move the first head right until it reaches two consecutive ones, then on the cell immediately to the right of them. Go to item 1.

3. On the first tape, skip the following 1, then move the first and third head in tandem, while they're both reading 0.

   If the first head reaches a 1 at the same time that the third head reaches a 1 or a blank, it means that on the first tape we've reached a transition relevant to the current state and the currently read symbol of M: go to item 5.

4. If we're not reading the encoding of a transition concerning the current symbol, we need to continue searching; move the third head left until it reaches a 1 (or blank, for the first symbol), then one cell to the right. Move the first head right until it reaches two consecutive ones, then on the cell immediately to the right of them. Go to item 3.

5. Assuming we're in state q_i reading symbol g_j, we must now perform the necessary changes for M's transition:

$$\delta(q_i, g_j) = (q_k, g_l, d_m)$$

   (a) Erase the contents of the second tape. On the first tape, skip over a 1 and start copying each following 0 onto the second tape until reaching another 1. Then reset the second head onto the leftmost non-blank symbol.

   (b) on the third tape (the head should be on the 1 immediately after the current symbol's encoding, or on a blank if the current symbol was the last symbol), go left and remove 0s until reaching a 1. By "removal" we mean shifting the entire right part of the tape, one cell to the left.

   Once the symbol is erased, on the first tape, skip the 1 and go on the first of the following l 0s. On the third tape, insert a 0 to the right. By "insertion", we mean shifting the entire right part of the tape, one cell to the right, then writing a 0 in the gap that appears.

   Repeat these steps (move the first head one cell to the right; insert one 0 on the third tape) until the first head reaches a 1. We have finished overwriting symbol g_j with symbol g_l.

   (c) Move the first head one cell to the right and count the 0s (memorizing them with the help of auxiliary states).

   We now know the direction the head of M should move:

   - If the head of M should stay where it is, move the third head left until it reaches a 1, then one cell to the right (remember that were positioned on the last 0 of the encoding of the written symbol; we do this to go to the first 0 of its encoding).

   - If the head of M should move left, move the third head left until it reaches a 1, then once more, then one cell to the right.

   - If the head of M should move right, move the third head two cells to the right.

Doing this repeatedly, U will behave the same as M: if $M[w]$ never halts, neither will $U[enc((M, w))]$'s simulation; if $M[w]$ accepts, so will $U[enc((M, w))]$ etc.

6 References and further reading

The Universal Machine plays a crucial role in Turing's 1936 article: "On computable numbers, with an application to the Entscheidungsproblem" [1]. Turing introduces a special notation that allows him to "metaprogram" machines by parametrizing some state templates; this is described in section 4, "Abbreviated tables". In section 5, he defines an encoding format for machines, as a base-10 number. Then sections 6 and 7 present the detailed construction of such a Universal Turing Machine.

"The annotated Turing" [2] covers these sections in chapters 7, 8 and 9. It also provides later corrections by Emil Post and Donald Davies that fix some bugs in Turing's article.

The construction in this lecture is adapted from the sections 9.1.2 and 9.2.3 of the textbook "Automata Theory, Languages and Computation", third edition [3].

Neither our construction, nor Turing's, actually provides an explicit transition table for the Universal Machine, on the basis that it would be too much effort to do so for no considerable benefit. But it would be interesting to "see" such a machine in a way that allows us to appreciate its complexity. Fortunately, in his 1989 book "The Emperor's New Mind" [4], Roger Penrose dedicates around three physical pages of the last part of his second chapter to printing out in full the binary encoding of a UTM.

Bibliography

[1] Alan Mathison Turing et al. "On computable numbers, with an application to the Entscheidungsproblem". In: *J. of Math* 58.345-363 (1936), p. 5.

[2] Charles Petzold. *The annotated Turing: a guided tour through Alan Turing's historic paper on computability and the Turing machine*. Wiley Publishing, 2008.

[3] John E Hopcroft, Rajeev Motwani, and Jeffrey D Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, 2006.

[4] Roger Penrose. *The emperor's new mind*. Oxford University Press, 1989.