Lecture 5

Gidon Rosalki

2025-04-27

1 Calling TMs as procedures / functions

(We have seen: Numbers, adding symbols, movement, adding letters above through ordered pairs)

1.1 Example - counting computation steps

Task: Given a TM M, we want to create the TM M', such that if M stops on input w, then M' will also stop on w, and $M'(w) = M(w)\#t$, where t is the binary representation of the number of steps that M needed to compute w. The concept is that we will preserve on the tape what M does, and then a #, followed by a counter. We are making the assumption that # is not part of the original alphabet. So, we then do a computation step of M, and then increment the counter, and then go back and compute the next step of M, increment the counter, and so on. Let there be S:

1. Runs right, until finding the #
2. If there is no number to the right of #, then S writes 0 (perhaps we have not yet started the counter)
3. Add 1 to the counter
4. We will finish in state s_F

M':

1. Add # to the right of the input, and return to its beginning
2. Transition to the start state of M
3. Simulate a step of calculation of M
4. "Call" to S to increment the counter
5. Return to the original position, and state
6. Return to 3

So, to create M', considering M, in state q, with the transition a, b, R, then in M', this will be from q, following the transition $\alpha, \hat{\alpha}, R$ to the state q, s_0. We are adding the state (q, s) for every state q of M and s of S. Additionally, considering a transition in S of s to s' following α, β, D, then in M' we will have from (q, s) to (q, s'), following α, β, D, until we eventually reach the state (q, s_F). From here we need to add the following transitions: A loop for every non signed α, the transition back to (q, s_F) α, α, L. For when we do find a letter with a symbol, then we follow the transition $\overline{a}, b, R$ to q' in the original TM M.

Let us assume that q is the final state of M. So for M', it is no longer a final state, and it still has the transition which adds a line above the letter (except we will skip the adding of the line, since it is unnecessary from the final state), and continues on to (q, s_0). We will then finish at (q, s_F).

1.2 TM as input to a TM

In order to give a TM T the input TM M, we need to convert M into a string.

Definition 1.1. $< M >$ *the string representation of the TM M*

We will explain the TM M as the string over $\{0, 1, \#, |\}$. W.l.o.g. $\Sigma \subseteq \Gamma \subseteq \mathbb{N}$. Additionally, w.l.o.g. $Q \subseteq \mathbb{N}$. Also w.l.o.g $\Gamma \cap Q = \emptyset$ (this last one is not necessarily required).
So

$$\begin{aligned}
\Sigma = \{\alpha_1, \dots, \alpha_k\} &\implies \langle\Sigma\rangle = (\alpha_1)_b \# (\alpha_2)_b \# \dots (\alpha_k)_b \#. \\
\Gamma = \{\alpha_1, \dots, \alpha_n\} &\implies \langle\Gamma\rangle = \dots \\
\sqcup = \alpha_7 \implies \langle\sqcup\rangle = (\alpha_7)_b \\
Q = \{q_1, \dots, q_m\} &\implies \langle Q\rangle = (q_1)_b \# \dots \\
q_0, F &\implies \dots
\end{aligned}$$

We will soon discuss δ. So we may see that $< M >= \langle\Sigma\rangle\langle\Gamma\rangle \mid \langle\sqcup\rangle \mid \ldots \mid \langle\delta\rangle$.

δ: If $\delta(q,\alpha) = (q',\beta,D)$ then we will say that $(q,\alpha) \overset{\delta}{to} (q',\beta,D)$ is the transition. We will represent δ through representations of its transitions:

$$\#\#(q)_b\#(\alpha)_b\#(q')_b\#(\beta)_b\#(D)_B$$

We can additionally convert the input w to a string $\langle w\rangle = (w_1)_b\#(w_2)_b\ldots$, and so $\langle M,w\rangle = \langle M\rangle\langle w\rangle$.

Definition 1.2 (Universal TM)**.** *A TM U is called **universal** if given $\langle M,w\rangle$:*

1. *If M does not stop running on w, then U will also not stop*
2. *If M stops on w, then U will stop, and*
$$U(\langle M,w\rangle) = \langle M,w\rangle$$
3. *If the final states of M are $q_{acc} = 1,\ q_{rej} = 0$, then U will stop in the same state as M.*

Theorem 1. *There exists a universal Turing machine U*

Proof . This proof is incomplete.
The machine U will run in the following manner:

1. Input $\langle M,w\rangle$
2. U will write on the tape $\langle M\rangle(q_0)_b\#\langle w\rangle$ (Note that the elements after $\langle M\rangle$ are C_0)
3. $\langle M\rangle\langle C_i\rangle \to \langle M\rangle\langle C_{i+1}\rangle$
4. When we arrive to the final configuration:
   (a) We will delete $\langle M\rangle$
   (b) We will delete $(q)_b$
   (c) If $q \in \{0,1\}$, then finish in q, otherwise we will finish in q_F

□

1.3 Does M stop on w?

Hey look! It's the halting problem!

Let us define our language $HALT = HALT_{TM} = \{\langle M,w\rangle : \text{M stops when running on } w\}$.

Definition 1.3 (Decision TM)**.** *A TM M is called a decision machine if the set of final states is $F = \{q_{acc}, q_{rej}\}$.*

Definition 1.4 (Language of a DTM)**.** *The language of a decision TM*

$$L(M) = \{w : M \textit{ stops in state } q_{acc} \textit{ when run on } w\}$$

In this case we will say that M accepts w. If M stops on w in state q_{rej} we will say that M rejects w.

Definition 1.5 (Recognition)**.** *If $L(M) = L$, then we will say that M recognises L.*

Theorem 2. *There exists a TM that recognises HALT.*

Definition 1.6.

$$RE = \{L : \textit{There exists a TM M that recognises } L\}$$

So we may now instead write the above theorem as

Theorem 3.

$$HALT \in RE$$

Proof . UTM. We will convert every final state of the UTM to an accepting state. Therefore, if the simulation ends, then the language is accepted in RE, and otherwise it is not. □

Definition 1.7. *The TM M **decides** the language L if $L(M) = L$, and also stops on **every** input*

Theorem 4 (Halting problem)**.** *There is no TM that **decides** HALT:*

Definition 1.8.

$$R = \{L : \textit{There exists a TM that decides } L\}$$

And so:

Theorem 5 (Halting problem)**.**

$$HALT \notin R$$

. We will assume w.l.o.g that there exists a TM D that decides $HALT$. We will use it to build the TM E, which given an input $\langle M\rangle$, runs D on it:

1. E will copy the input: $M \to \langle M, \langle M\rangle\rangle$
2. E will run D as a procedure
3. If D returns "yes", then E will enter an infinite loop, and if D returns "no", then E will accept (in short, stops).

We will run the TM E on the input $\langle E\rangle$. Will E stop on the input $\langle E\rangle$? If E stops, then D has returned that E stops, and then E will enter an infinite loop, and will not stop, which is a contradiction. On the other hand, if E does not stop, then D will return that E has not stopped, but then E **will** stop, which is once again, a contradiction.
This is a contradiction, and therefore the assumption that D exists is incorrect. □

2 Reductions

Let us define

$$HALT_\varepsilon = \{\langle M\rangle : \text{M stops on every input } \varepsilon\}$$

Theorem 6.

$$HALT_\varepsilon \notin R$$

Proof . We may construct the TM red (contraction of reduction) which operates as follows on the input $\langle M, w\rangle$: Red will create a description of a TM M', where given an empty input, it prints w, and then runs like M. This is to say $\langle M, w\rangle \to \langle M'\rangle$.

Let us assume the contradiction that D is the TM that decides $HALT_\varepsilon$. Given $\langle M, w\rangle$, we will give them as input to Red, which will then return $\langle M'\rangle$. This will be given as input to D (as described in halting problem), and returns yes or no. Therefore, the first 3 steps decides $HALT$, which is a contradiction, since there is no machine that decides $HALT$. □