

CS 483 - Introduction to the Theory of Computation

Lecture 12

Encoding Turing Machines

How is $A_{\text{TM}} = \{(M, w) \mid \text{Turing machine } M \text{ accepts } w\}$ a language?

Encoding Turing Machines

How is $A_{\text{TM}} = \{(M, w) \mid \text{Turing machine } M \text{ accepts } w\}$ a language?

A Turing machine $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{acc}}, q_{\text{rej}})$ can be written as a string in $(Q \cup \Gamma \cup \{(,), ", \leftarrow, \rightarrow\})^*$. In particular, the transition function is encoded as its **lookup table**, which is a list of tuples:

$$(q, s_1, \dots, s_k, q', s'_1, \dots, s'_k, D_1, \dots, D_k)$$

where $\delta(q, s_1, \dots, s_k) = (q', s'_1, \dots, s'_k, D_1, \dots, D_k)$.

Encoding Turing Machines

How is $A_{\text{TM}} = \{(M, w) \mid \text{Turing machine } M \text{ accepts } w\}$ a language?

A Turing machine $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{acc}}, q_{\text{rej}})$ can be written as a string in $(Q \cup \Gamma \cup \{(,), ", \leftarrow, \rightarrow\})^*$. In particular, the transition function is encoded as its **lookup table**, which is a list of tuples:

$$(q, s_1, \dots, s_k, q', s'_1, \dots, s'_k, D_1, \dots, D_k)$$

where $\delta(q, s_1, \dots, s_k) = (q', s'_1, \dots, s'_k, D_1, \dots, D_k)$.

However, it does **NOT** result in a fixed alphabet.

Encoding Turing Machines

We use binary encoding for elements in Q and Γ :

$$Q = \{q_0, q_1, q_{\text{acc}}, q_{\text{rej}}\}$$

$$\Gamma = \{a, b, c, \#, \sqcup\}$$



0000 0001 0010 0011

0100 0101 0110 0111 1000

Encoding Turing Machines

We use binary encoding for elements in Q and Γ :

$$Q = \{q_0, q_1, q_{\text{acc}}, q_{\text{rej}}\} \quad \Gamma = \{a, b, c, \#, _ \}$$



0000 0001 0010 0011

0100 0101 0110 0111 1000

Furthermore, we can even encode $\{ (,), ", \leftarrow, \rightarrow \}$ with binary:

() , $\leftarrow$ $\rightarrow$



1110 1111 1101 1100 1001

We know the block length since the recoding always starts with (.

Encoding Turing Machines

Under the binary encoding, A_{TM} is a language over the alphabet $\{0,1\}$:

$$A_{\text{TM}} = \{\text{binary encodings of } (M, w) \mid \text{Turing machine } M \text{ accepts } w\}$$

- (Formally, the binary encodings of M and (M, w) are denoted $\langle M \rangle$ and $\langle M, w \rangle$, but we will omit the brackets as it is clear from context.)

Universal Turing Machine

Theorem. There is a Turing machine U (called **universal Turing machine**) such that for every Turing machine M and its input w ,

$$U(M, w) = M(w)$$

This is true for both 0-1 outputs (accept and reject) and string outputs (which are encoded in binary).

A_{TM} is exactly the language recognized by U .

Universal Turing Machine

Description of $U(M, x)$:

- Turn M into the encoding of an equivalent single-tape M' ;
- Copy w to an empty tape;
- Simulate every step of M by looking up δ :
 - accepts if M accepts;
 - rejects if M rejects.



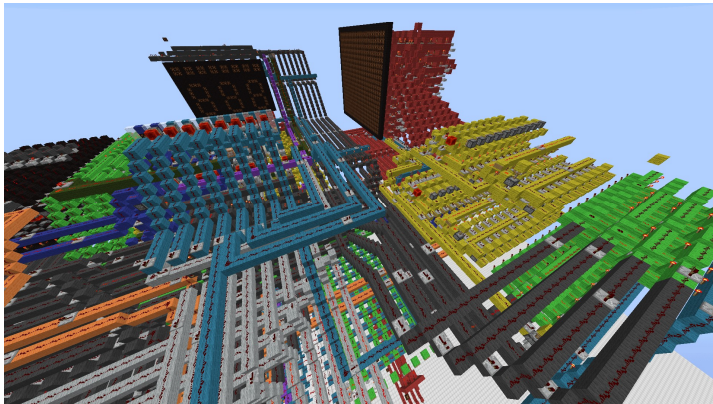
Turing Completeness

A computation model is **Turing complete**, if it is equivalent to Turing machines in computational power.

Turing Completeness

A computation model is **Turing complete**, if it is equivalent to Turing machines in computational power.

Assuming Church-Turing thesis, an equivalent definition is that the model can simulate a universal Turing machine.



Enumerator

An enumerator is a Turing machine M with a special q_{print} state. Whenever it reaches q_{print} state, it prints the content on its last tape.

The language **enumerated** by the M consists of all the strings printed during the execution of $M(\varepsilon)$.

Note that the enumerator does not necessarily halt.

Enumerator

Example: Enumerate $\{0,1\}^*$:

- For $n = 0, 1, 2, \dots$
 - For each $x \in \{0,1\}^n$, print x

Enumerator

Example: Enumerate $\{M \in \{0,1\}^* \mid M \text{ is a Turing machine}\}$:

- For $n = 0, 1, 2, \dots$
 - For each $x \in \{0,1\}^n$:
 - Check if x encodes a Turing machine; if yes then print x

Enumerator

Example: Enumerate $A_{\text{TM}} = \{ (M, w) \mid \text{Turing machine } M \text{ accepts } w \}$:

- For $n = 0, 1, 2, \dots$
 - For each $x \in \{0,1\}^n$:
 - Check if x encodes (M, w) ;
 - If yes, run $U(M, w)$;
 - print x if it accepted...?

Enumerator

Example: Enumerate $A_{\text{TM}} = \{ (M, w) \mid \text{Turing machine } M \text{ accepts } w \}$:

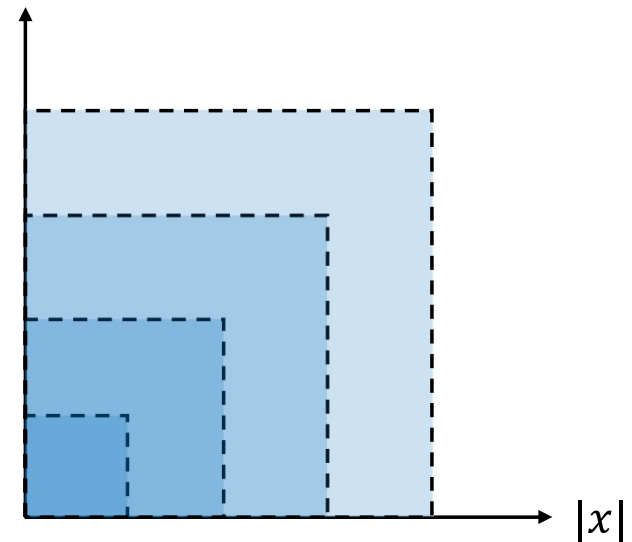
- For $n = 0, 1, 2, \dots$
 - For each $x \in \{0,1\}^{\leq n}$:
 - Check if x encodes (M, w) ;
 - If yes, run $U(M, w)$ for n steps;
 - print x if it accepted.

Enumerator

Example: Enumerate $A_{\text{TM}} = \{ (M, w) \mid \text{Turing machine } M \text{ accepts } w \}$:

- For $n = 0, 1, 2, \dots$
 - For each $x \in \{0,1\}^{\leq n}$:
 - Check if x encodes (M, w) ;
 - If yes, run $U(M, w)$ **for n steps**;
 - print x if it accepted.

Steps for $U(M, w)$
to accept



Enumerator

Theorem. A language is recognizable if and only if it can be enumerated.

Enumerator

Theorem. A language is recognizable if and only if it can be enumerated.

“If” direction: Assume M enumerates L .

Description of $M'(w)$ that recognize L :

- Run $M(\varepsilon)$:
 - Whenever $M(\varepsilon)$ prints x , check if $x = w$; if so, accept.

Enumerator

Theorem. A language is recognizable if and only if it can be enumerated.

“Only if” direction: Assume M recognizes L .

Description of $M'(\varepsilon)$ that enumerates L :

- For $n = 0, 1, 2, \dots$
 - For each $x \in \{0,1\}^{\leq n}$:
 - Run $M(x)$ for n steps;
 - print x if it accepted.

Application of Enumerator

Claim. If L_1 and L_2 are recognizable, then $L_1 \cup L_2$ is also recognizable.

Assume M_1 enumerates L_1 and M_2 enumerates L_2 .

Description of $M(\varepsilon)$ that enumerates $L_1 \cup L_2$:

- For $t = 0, 1, 2, \dots$
 - Run $M_1(\varepsilon)$ for t steps;
 - Run $M_2(\varepsilon)$ for t steps.

Application of Enumerator

Theorem. A language L is decidable if and only if both L and $\bar{L}$ are recognizable.

“Only if” direction: If L is decidable, so is $\bar{L}$. Decidability implies recognizability by definition.

Application of Enumerator

Theorem. A language L is decidable if and only if both L and $\bar{L}$ are recognizable.

“Only if” direction: If L is decidable, so is $\bar{L}$. Decidability implies recognizability by definition.

“If” direction: Assume M_1 enumerates L and M_2 enumerates $\bar{L}$.

On input w , run $M_1(\varepsilon)$ and $M_2(\varepsilon)$ in parallel: accept when M_1 prints w , and reject when M_2 prints w .

Application of Enumerator

Theorem. A language L is decidable if and only if both L and $\bar{L}$ are recognizable.

As a result, if we want to prove that $\overline{A_{\text{TM}}}$ is not recognizable, it suffices to prove that A_{TM} is not decidable.

R and RE

On a fixed alphabet (usually $\Sigma = \{0,1\}$):

- The set of all languages is denoted **ALL**.
- The set of all decidable languages is denoted **R** (as in **R**ecursive).
- The set of all recognizable languages is denoted **RE** (as in **R**ecursively **E**numerable)

R and RE

On a fixed alphabet (usually $\Sigma = \{0,1\}$):

- The set of all languages is denoted **ALL**.
- The set of all decidable languages is denoted **R** (as in **R**ecursive).
- The set of all recognizable languages is denoted **RE** (as in **R**ecursively **E**numerable)
- The set of all languages whose complement is recognizable is denoted **coRE**.

R and RE

We will show that the following Venn diagram is proper:

