Turing Computability

CSCI 2210

2023-10-30 — 2023-11-06

Language Recognized by a Turing Machine

For a Turing machine M with input alphabet Σ, the language **recognized** by M is the set of strings on which M ceases computation in its accepting state q_h.

$$\mathrm{L}(\mathrm{M}) \;:=\; \{w \in \Sigma^* \mid \mathrm{M} \text{ accepts } w\}$$

A string language is called **recursively enumerable** if it is recognized by some TM.

Non-Recursively Enumerable Languages

It is natural to ask, are all string languages recursively enumerable?

The answer is no.

Theorem

There are string languages that are not recognized by any Turing machine.

Proof.

Each Turing machine recognizes a single language.

The set of *Turing machines* over a finite alphabet Σ is **countably infinite**:

- each TM is described by a finite string of symbols (its description by components) over a finite alphabet (say, Unicode $\cup\, \Sigma$).
- we can *enumerate* these strings (say, first by length, then alphabetically).

The set of *languages* over a finite alphabet Σ is **uncountable**:

- The set of languages over Σ is the **powerset** of the set of strings over Σ.
- The result follows by *diagonalization*.

$\square$

Diagonalization

Theorem

For any countably infinite set S, the powerset $\wp(\mathrm{S})$ is strictly larger than S.

Proof.

We assume the contrary to reach a contradiction.
Since S is countable, we can enumerate its elements: $\mathrm{S} = \{x_0, x_1, x_2, \cdots\}$.
If $\wp(\mathrm{S})$ were countable, we could enumerate its elements: $\wp(\mathrm{S}) = \{\mathrm{S}_0, \mathrm{S}_1, \mathrm{S}_2, \cdots\}$.
For each natural number i, either $x_i \in \mathrm{S}_i$ or $x_i \notin \mathrm{S}_i$:

e.g.
$$
\begin{array}{c|cccc}
 & x_0 & x_1 & x_2 & \cdots \\
\hline
\mathrm{S}_0 & \notin & \notin & \in & \cdots \\
\mathrm{S}_1 & \notin & \in & \in & \cdots \\
\mathrm{S}_2 & \notin & \notin & \notin & \cdots \\
\vdots & \vdots & \vdots & \vdots &
\end{array}
$$

Consider the subset $\mathrm{S}_d \subseteq \mathrm{S}$ defined as $\mathrm{S}_d := \{x_i \in \mathrm{S} \mid x_i \notin \mathrm{S}_i\}$.
Then $\forall\, i \in \mathbb{N} \,.\, \mathrm{S}_d \neq \mathrm{S}_i$ because $x_i \in \mathrm{S}_d \iff x_i \notin \mathrm{S}_i$,
so $\mathrm{S}_d \subseteq \mathrm{S}$, yet it does not occur in the alleged enumeration of $\wp(\mathrm{S})$. $\square$

Termination

How can we tell if a string w is in the language recognized by TM M?

We can start M running on input w and see what happens.

M might eventually:

- cease computation and accept,
- cease computation and reject,
- never cease computation.

The third possibility means that we may not be able to tell whether $w \in \mathrm{L}(\mathrm{M})$.

Language Decided by a Turing Machine

If a TM M eventually ceases computation on every input, and thus either accepts or rejects every string, we say that M **decides** $\mathrm{L(M)}$.

A string language is called **decidable** if it is decided by some TM.

Because there are string languages that are not recognized by any TM,
there are string languages that are not decidable.

Deciding Language Membership

For some languages we *can* decide whether a a string is a member.

One way to do this is to encode an algorithm that decides it as a TM and then run that TM on a given input.

Deciding a Regular Language by Simulating a DFA

We know that a *regular* language over a finite alphabet Σ can be decided by a DFA.

- It is possible to encode the *specification of a DFA* as a (say, binary) string.
- It is possible to encode a *string over* Σ as a (say, binary) string.
- It is possible to encode the *ordered pair* of two strings as a (say binary) string.

Thus, for a DFA D and string $w \in \Sigma^*$ we can produce a *binary string* $\langle \mathrm{D} \, , w \rangle$ that encodes the *ordered pair* $(\mathrm{D} \, , w)$.

There is a TM $\mathrm{M}_{\mathrm{DFA}}$ that given input string $\langle \mathrm{D} \, , w \rangle$ **simulates** the operation of DFA D on input string w.

Thus, the TM $\mathrm{M}_{\mathrm{DFA}}$ accepts the string $\langle \mathrm{D} \, , w \rangle$ just in case the DFA D accepts the string w.

Because a DFA halts on every input, $\mathrm{M}_{\mathrm{DFA}}$ *decides* the regular language $\mathrm{L}(\mathrm{D})$.

Deciding Whether a Regular Language is Empty

We can use a TM to decide whether or not the language of a DFA is empty.

The idea is to apply the following algorithm to the state transition graph of a DFA:

- Enumerate the edges of the graph.
- Initialize a list of *accessible* states to $[q_0]$.
- Repeat the following steps until a fixed-point is reached:
  - for edge $e : q_x \to q_y$, if q_x is accessible remove e from the edge enumeration,
  - if q_y is not accessible, add it to the list of accessible states.
- This process will terminate with the edge enumeration containing only inaccessible source states (possibly because it is empty).
- Search the list of accessible states for an accept state, accept just in case one is found.

Universal Turing Machines

We've seen that it is possible to encode a DFA D as a string, and to define a TM $\mathrm{M}_{\mathrm{DFA}}$ that simulates its operation on an input string w when given as input $\langle \mathrm{D} , w \rangle$.

In 1936 Turing described a TM "U" such that given as input $\langle \mathrm{M} , w \rangle$, the encoding of a TM M and a string w, U simulates the operation of machine M on input w.

Such a TM U is called a **universal Turing machine**.

The language recognized by a UTM is: $\mathrm{L}(\mathrm{U}) = \{\langle \mathrm{M} , w \rangle \mid w \in \mathrm{L}(\mathrm{M})\}$.

So the operation of running a given TM on a given input is something that a TM can do.

For a DFA D and string w we can use simulation to decide whether $w \in \mathrm{L}(\mathrm{D})$.

The Acceptance Problem

The **acceptance problem** for TMs asks, given TM M and string w, is $w \in \mathrm{L(M)}$?

This time simulation cannot decide this question because:

- if M halts and accepts w then U halts and accepts $\langle \mathrm{M}\ , w \rangle$, as desired,
- if M halts and rejects w then U halts and rejects $\langle \mathrm{M}\ , w \rangle$, as desired,
- if M does not halt on w then U does not halt $\langle \mathrm{M}\ , w \rangle$, and we don't get an answer to the question.

But maybe there's some clever way to answer the question that doesn't involve running M, like the way we decided whether the language of a DFA is empty.

Alas, there is not.

Undecidability of the Acceptance Problem

Theorem

The acceptance problem for TMs is not decidable by any Turing machine;
i.e., $\mathrm{L}(\mathrm{U})$ is not a decidable language.

To show this we derive a contradiction from the premise that $\mathrm{L}(\mathrm{U})$ is decidable.

Suppose TM M_{A} could decide the acceptance problem:

- if M halts and accepts w then M_{A} halts and accepts input $\langle \mathrm{M}\,, w \rangle$,
- if M halts and rejects w then M_{A} halts and rejects input $\langle \mathrm{M}\,, w \rangle$,
- if M does not halt on input w then M_{A} halts and rejects input $\langle \mathrm{M}\,, w \rangle$.

Note that M_{A} must halt on every input because it is a *decider*.

Diagonalizing Acceptance

We show that no such machine M_{A} can exist by diagonalization.

Consider the TM D that takes as input the encoding of a TM M and:

1. runs M_{A} with input $\langle \mathrm{M}\,, \langle \mathrm{M} \rangle \rangle$,
   this decides whether machine M accepts its own encoding as input,
2. negates the result.

So D accepts $\langle \mathrm{M} \rangle$ just in case M_{A} rejects $\langle \mathrm{M}\,, \langle \mathrm{M} \rangle \rangle$,

which happens just in case M does not accept as input $\langle \mathrm{M} \rangle$,

either because it rejects or diverges.

Feeding the Diagonalizer its own Encoding

We can see how each TM behaves given the encoding of another TM as input.

e.g.

U	$\langle M_0\rangle$	$\langle M_1\rangle$	$\langle M_2\rangle$	$\cdots$
M_0	✓	$\uparrow$	$\times$	$\cdots$
M_1	$\uparrow$	✓	✓	$\cdots$
M_2	$\times$	✓	$\uparrow$	$\cdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	

M_A	$\langle M_0\rangle$	$\langle M_1\rangle$	$\langle M_2\rangle$	$\cdots$
M_0	✓	$\times$	$\times$	$\cdots$
M_1	$\times$	✓	✓	$\cdots$
M_2	$\times$	✓	$\times$	$\cdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	

Somehow M_A is able to tell when a computation will diverge and reject in those cases.

Somewhere in the enumeration of TMs is D, which accepts input $\langle M\rangle$ just in case M_A rejects input $\langle M, \langle M\rangle\rangle$:

U	$\langle M_0\rangle$	$\langle M_1\rangle$	$\langle M_2\rangle$	$\cdots$	$\langle D\rangle$	$\cdots$
D	$\times$	$\times$	✓	$\cdots$	?	$\cdots$

How does machine D behave on input $\langle D\rangle$?

Reaching a Contradiction

Machine D either accepts input $\langle \mathrm{D} \rangle$ or it does not.

- If D accepts $\langle \mathrm{D} \rangle$ then $\mathrm{M_A}$ accepts $\langle \mathrm{D}\ , \langle \mathrm{D} \rangle \rangle$, so D must reject $\langle \mathrm{D} \rangle$.
- If D does not accept $\langle \mathrm{D} \rangle$ then $\mathrm{M_A}$ rejects $\langle \mathrm{D}\ , \langle \mathrm{D} \rangle \rangle$, so D must accept $\langle \mathrm{D} \rangle$.

Both cases are impossible, so the premise that a machine $\mathrm{M_A}$ exists must be false. $\square$

Negating TMs

In the previous proof we negated the result of running a TM on an input.

For any TM M there is a **negated TM** $\bar{\mathrm{M}}$ such that:

- if M halts and accepts input w then $\bar{\mathrm{M}}$ halts and rejects w,
- if M halts and rejects input w then $\bar{\mathrm{M}}$ halts and accepts w,
- if M doesn't halt then $\bar{\mathrm{M}}$ doesn't halt either.

The Halting Problem

The **halting problem** for TMs asks, given TM M and string w, does M eventually halt in input w, either by accepting or rejecting it?

Theorem

The halting problem is equivalent to the acceptance problem:
if we could decide one then we could decide the other.

Proof.

$\Rightarrow$ If TM $\mathrm{M_H}$ could decide the halting problem, then we could use it to decide the acceptance problem:

- we run $\mathrm{M_H}$ on input $\langle \mathrm{M}\ , w\rangle$,
- if $\mathrm{M_H}$ accepts then M halts on input w so we just run M on input w,
- if $\mathrm{M_H}$ rejects then M does not halt on input w, so M does not accept w, so we reject.

$\Leftarrow$ If TM $\mathrm{M_A}$ could decide the acceptance problem, then we could use it to decide the halting problem:

- we run $\mathrm{M_A}$ on both input $\langle \mathrm{M}\ , w\rangle$ and input $\langle \bar{\mathrm{M}}\ , w\rangle$,
- if $\mathrm{M_A}$ accepts $\langle \mathrm{M}\ , w\rangle$ then M accepts input w, so M halts on w, so we accept.
- if $\mathrm{M_A}$ accepts $\langle \bar{\mathrm{M}}\ , w\rangle$ then M rejects input w, so M halts on w, so we accept.
- if $\mathrm{M_A}$ rejects both $\langle \mathrm{M}\ , w\rangle$ and $\langle \bar{\mathrm{M}}\ , w\rangle$ then M neither accepts nor rejects w, so M does not halt on w, so we reject.

$\square$

Undecidability of the Halting Problem

Corollary

The halting problem for TMs is not decidable by any Turing machine.

Complement of a Language

The **complement** of a language $\mathrm{L} \subseteq \Sigma^*$ is the set of strings over the same alphabet that are not contained in L:

$$\bar{\mathrm{L}} := \{w \in \Sigma^* \mid w \notin \mathrm{L}\}$$

Complement of a Decidable Language

Theorem

The complement of a decidable language is decidable.

Proof.

If language L is decidable then there is a TM M that halts on all inputs with $\mathrm{L}(\mathrm{M}) = \mathrm{L}$.

The negated TM $\bar{\mathrm{M}}$ accepts exactly the strings that M rejects, and vice-versa.

Because M halts on all inputs, so does $\bar{\mathrm{M}}$.

Thus $\mathrm{L}(\bar{\mathrm{M}}) = \bar{\mathrm{L}}$. $\square$

Complement of a Recursively Enumerable Language

Theorem

If languages L and $\bar{\mathrm{L}}$ are both recursively enumerable then L is decidable.

Proof.

For language $\mathrm{L} \subseteq \Sigma^*$ and string $w \in \Sigma^*$, either $w \in \mathrm{L}$ or $w \in \bar{\mathrm{L}}$.

Each recursively enumerable language has a TM that recognizes it.
Say $\mathrm{L}(\mathrm{M}_{\mathrm{L}}) = \mathrm{L}$ and $\mathrm{L}(\mathrm{M}_{\bar{\mathrm{L}}} = \bar{\mathrm{L}})$.

To decide whether $w \in \mathrm{L}$, run both M_{L} and $\mathrm{M}_{\bar{\mathrm{L}}}$ in parallel on w.

Exactly one must eventually accept (why?).

If M_{L} accepts then $w \in \mathrm{L}$ and if $\mathrm{M}_{\bar{\mathrm{L}}}$ accepts then $w \notin \mathrm{L}$. $\square$

Hierarchy of Computable Languages

Looking back on the models of computation we have studied, we can identify the following classes of string languages: