Theory of Computation Part II - Turing Machines

Ethan Graham

22nd of June 2023

Lecture 4 - Introduction to Turing Machines

Formal Definition of Turing Machines

So far with automata we have had no memory. We, ideally, would like to be able to remember things while maintaining a fixed program size.

We implement this by the means of an infinite-length tape that is read by a head. At any given point, the head can only see the current symbol "under" it.

Formal Definition Turing Machine: A Turing Machine is a 7-tuple $(Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$

- Q : Set of states
- Σ : The input alphabet **not** containing blank symbol $\sqcup$
- Γ : The tape alphabet, where $\Sigma \cup \{\sqcup\} \subseteq \Gamma$ (*i.e. we include blank symbol at least, but there could be more tape symbols not present in input alphabet*)
- $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$: The transition function
- $q_0 \in Q$: The start state
- $q_{accept} \in Q$: The accept state
- $q_{reject} \in Q$: The reject state

Formal Definition TM Configuration:

We write uqv where $u, v \in \Gamma^*$ and $q \in Q$

- q is the current state
- uv is the current tape content
- v's first symbol is the current head location

Given a configuration uaq_ibv where $a, b \in \Gamma$, $u, v \in \Gamma^*$ and $q_i \in Q$, we move to:

- uq_jacv if $\delta(q_i, b) = (q_j, c, L)$
- $uacq_jv$ if $\delta(q_i, b) = (q_j, c, R)$

Intuitively: if in uaq_ibv, b is the current head symbol, it will be overwritten by c. We then move either left or right (respectively moving ac to right/left of the head)

We start at some configuration $C_1 = q_0w$, $w \in \Sigma^*$. Every step we obtain a new configuration.

The machine accepts/rejects if we halt at state q_{accept}/q_{reject}

Lecture 5 - Decidability and Undecidability, the Halting Problem

Definition Turing-Recognizable:

A language L is Turing-Recognizable if there exists a Turing Machine M which recognizes L. In other words: if $w \in L$, M halts and accepts. If $w \notin L$, M halts and rejects or loops forever.

Definition Turing-Decidable:

A language L is Turing-Recognizable if there exists a Turing Machine M which decides L. In other words: For all $w \in L$, M halts and accepts. For all $w \notin L$, M halts and rejects.

Decidability is stronger than recognizability because it requires that TM M halts on every input.

Church-Turing Thesis

Church and Turing had two different definitions of what an algorithm is. Church's was λ-calculus, whereas Turing's was his machines. We can show that these two definitions are equivalent. This means that anything that can be written as a program, can be made into a Turing machine. Therefore we can use the abstraction of programs/algorithms instead of formally defining Turing Machines.

Definition: Encoding

Let S be some object. We can write its binary encoding $\langle S \rangle$

Uncountability of $\mathbb{R}$

Theorem: $|\mathbb{R}| \neq |\mathbb{N}|$

Proof: Using Cantor's Diagonalization

intuition: Let's say that I have a table with all of the real numbers $x_1, ..., x_n$ listed where $n = |\mathbb{N}|$ I choose to construct the new number $y = e_1, ..., e_j, ..., e_n$ where e_i is different from x_i in position i.

Since y is real, it must be in the table. Let's assume that it is in row j, i.e. $y = x_j$ However, we have said that y is different in position j from x_j. i.e. $e_j \neq x_j^{(j)}$ We have now implied that $y \neq y$, since $y = x_j$, which is a contradiction. $\square$

Theorem: There exists a language which is undecidable

Proof: We once again use the diagonalization argument.

We make a table containing all possible TMs with M_i as the rows and $\langle M_i \rangle$ as the columns, where $i = 1, ..., n$ and $n = |\mathbb{N}|$ (since Turing Machines are countable)

We construct the following language $DIAG = \{\langle M_i \rangle | M_i \textit{ does not accept } \langle M_i \rangle\}$ which we will show is undecidable.

Suppose towards contradiction $\exists M$ that decides $DIAG$ i.e. $L(M) = DIAG$

Since we have listed all possible Turing Machines, we know that $M = M_j$ for some j.

We can't have $\langle M_j \rangle \in L(M_j)$ since that would imply that the machine accepts itself and thus $M_j \notin DIAG \rightarrow$ if $M \in L(M) = DIAG$ then $M \notin DIAG$.

Similarly, we can't have $\langle M_j \rangle \notin L(M_j)$ since that would inply that this machine does not accept itself, and thus by definition of $DIAG$ it implies that $\langle M_j \rangle \in DIAG \rightarrow$ if $M \notin L(M) = DIAG$ then $M \in DIAG$ Thus we have a contradiction. Showing us that $DIAG$ is undecidable.

Halting Problem

Theorem: The following language is undecidable $HALT = \{\langle M, w \rangle : M \textit{ is a TM that halts on } w\}$

Proof:

Intuition: We make a decider for $DIAG$ (which is undecidable) using a decider for $HALT$.

Assume we have a decider H for $HALT$. We construct the following Turing Machine D

```
D on input <M>:
    if H(<M, <M>>):
        return !M(<M>)
    else
        return accepted
```

Concretely: if M halts on input $\langle M \rangle$, then return the opposite of its result. Otherwise (if M doesn't halt on $\langle M \rangle$) we return `accepted`. We notice here that D halts for all inputs. If M accepts $\langle M \rangle$, then we reject. If M rejects $\langle M \rangle$ or loops forever (= rejection), we accept. Thus by definition of $DIAG$ we see that it is decided by D. Which is a contradiction because we have shown that $DIAG$ is undecidable.

Theorem: Let L be a language. Then

$$L \text{ is decidable } \Leftrightarrow L \text{ is recognizable and } \overline{L} \text{ is recognizable}$$

Corollary $\overline{HALT}$ is recognizable. Since $HALT$ is undecidable and recognizable, by the above theorem it follows that $\overline{HALT}$ must be unrecognizable.

Lecture 6 - Reductions

Now we attack more abstract proof methods using reductions. For example, let's say we have the following language: $NE_{DFA} = \{\langle D \rangle : L(D) = \emptyset\}$ Consider now the following language: $L_{path} = \{\langle G, u, v \rangle : G \textit{ is a graph and there exists a path from } u \textit{ to } v\}$ We know that a DFA D has a non-empty language ***iff*** there exists a path from q_0 to an accepting state. Thus we have reduced NE_{DFA} to L_{path} with $f(\langle D \rangle) = \langle G, u, v \rangle$

Definition: Computable

A function $f : \Sigma^* \to \Sigma^*$ is computable if $\exists M$ a TM such that on every w, M halts with just $f(w)$ on its tape.

Definition: Mapping reducible

We write $A \leq_m B$ if there exists a computable function f s.t. for every $w \in \Sigma^*$ we have $w \in A \Leftrightarrow f(w) \in B$

f does not need to be bijective, but for every element in A we should be able to land on an element in B and every element in $\overline{A}$ must land on an element in $\overline{B}$

Theorem: If $A \leq_m B$ and B is decidable, then A is decidable.

Proof B is decidable, therefore we have a decider D for it. We also have a reduction f from A to B. We can now define a turing machine M that decides A.

```
M on input w:
    run D on input f(w)
    output result
```

By definition of reduction: $w \in A \Leftrightarrow w \in B$ and $w \notin A \Leftrightarrow w \notin B$ Therefore M accepts all words inside A and rejects all words in $\overline{A}$

Corollary: if $A \leq_m B$ and A is undecidable, then B is undecidable.

Some Reduction Examples

Example 1: $A_{TM} \leq_m HALT$ and thus $HALT$ is undecidable.

To show these, we make a reduction f from A_{TM} to $HALT$

```
f on input <M, w>:
    construct a turing Machine M' such that:
        if M(w) accepts, M' accepts
        if M(w) rejects, M' loops forever
```

We can see that $\langle M, w\rangle \in A_{TM} \Leftrightarrow \langle M', w\rangle \in HALT$

There are other examples in the course that I will omit here for simplicity.

More Theorems

if $A \leq_m B$ and B is recognizable, then A is recognizable.

contrapositive if $A \leq_m B$ and A is unrecognizable, then B is unrecognizable.

Example 2:

let $EQ_{TM} = \{\langle M_1, M_2\rangle : L(M_1) = L(M_2)\}$ and we show that $\overline{A_{TM}} \leq_m EQ_{TM}$ and thus that EQ_{TM} is unrecoginzable.

Proof

we define the following reduction f that given input $\langle M, w\rangle$ outputs $\langle M_1, M_2\rangle$ such that these machines share the same language ***iff*** $\langle M, w\rangle \in A_{TM}$ i.e. $\langle M, w\rangle \notin \overline{A_{TM}}$

```
f on input <M, w>:
    construct M_2 on any input:
        reject
    construct M_1 on any input:
        run M on w.
        accept if this accepts, otherwise enter an infinite loop

    return <M_1, M_2>
```

We notice that if $\langle M, w\rangle \in \overline{A_{TM}}$ then M_1 loops on all inputs thus $L(M_1) = \emptyset = L(M_2) \Rightarrow f(\langle M, w\rangle) \in EQ_{TM}$

If $\langle M, w\rangle \notin \overline{A_{TM}}$ then $L(M_1) = \Sigma^* \neq \emptyset = L(M_2) \Rightarrow f(\langle M, w\rangle) \notin EQ_{TM}\square$