CMSC 28100

Introduction to Complexity Theory

Spring 2024
Instructor: William Hoza

Python script ⇒ Turing machine

- Basic idea:
  - Variable ⇒ Tape (assuming the variable holds a list of bits)
  - List index ⇒ Head
  - Line of code ⇒ State

Turing machines as a programming language

- You can think of the Turing machine model as a primitive programming language
- From a programming perspective, the model is extremely inconvenient and annoying, because it has so few features!
- However, our goal is to prove impossibility results
- The model has few features, which will make our lives easier, not harder

The Church-Turing Thesis

- Let L be a language

Church-Turing Thesis:

There exists an "algorithm" / "procedure" for figuring out whether a given string is in L if and only if there exists a Turing machine that decides L.

← Intuitive notion

← Mathematically precise notion

Turing machines vs. your laptop

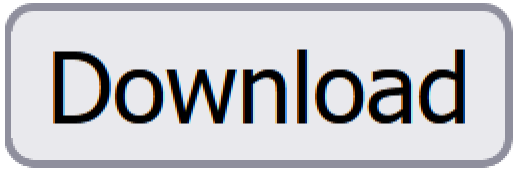
- **OBJECTION:**
  - "My laptop is a single device that can run arbitrary computations.
  - I don't use one laptop for email, a second laptop for Zoom, a third laptop for Tetris, and a fourth laptop for photo editing. I just use one laptop for everything.
  - In contrast, a single Turing machine only solves one problem.
  - If M decides one language, then it can't also decide a different language.
  - Therefore, Turing machines don't properly model my laptop."

Code as data

- The response to this objection is based on the principle of viewing "code as data"
- A Turing machine M can be encoded as a string $\langle M \rangle$

Encoding a Turing machine as a string

- Example: Problem set 1, problem 4



⇓

Symbols

States	>	_	0	1	#	$
a		(o, _, R)	(b, _, R)	(c, _, R)	(d, _, R)	
b		(o, 0, R)	(b, 0, R)	(c, 0, R)	(d, 0, R)	
c		(o, 1, R)	(b, 1, R)	(c, 1, R)	(d, 1, R)	
d		(o, #, R)	(b, #, R)	(c, #, R)	(d, #, R)	
e						
f						
g						
h						
i						
j						

`turing-machine.json`

```
… {"a": {">": null, "_": ["o", "_",
"R"], "0": ["b", "_", "R"], "1":
["c", "_", "R"], "#": ["d", "_",
"R"], "$": null, "&": null, "%":
null}, "b": {">": null, "_": ["o",
"0", "R"], "0": ["b", "0", "R"],
"1": ["c", "0", "R"], …
```

↰ A text file (string) that encodes a Turing machine

Encoding a Turing machine as a string

- For a Turing machine $M = (Q, \Sigma, \Gamma, \Diamond, \sqcup, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$, we could define $\langle M \rangle \in \{0, 1, \#, \&, \$, \%\}^*$ as follows
  - Assume WLOG that $Q = \{0, 1, 2, \ldots, m\}$ and $\Gamma = \{m + 1, \ldots, m + k\}$
  - Assume WLOG that $q_0 = 0$; $q_{\text{accept}} = m - 1$; and $q_{\text{reject}} = m$
  - Assume WLOG that $\Diamond = m + 1$; $\sqcup = m + 2$; and $\Sigma = \{m + 3, m + 4, \ldots, m + 2 + r\}$
  - We let $\langle M \rangle = \langle m \rangle \# \langle r \rangle \# \langle k \rangle \# \langle \delta \rangle$, where $\langle \delta \rangle$ is the list of all entries in the transition table, where rows are separated by & symbols, cells within a row are separated by $ symbols, and the individual components of each entry are separated by % symbols

Analyzing a given Turing machine

- Given the encoding $\langle M \rangle$ of a Turing machine M, one can try to answer various questions about M
  - How many states does M have?
  - How big is the tape alphabet of M?
  - Does M accept ###11 within 10000 steps?

Analyzing TMs

```
@weight(0.5)
@number("3")
def test3(self):
    """Run the machine on input ###11"""
    val = simulate(self.transition, "###11", 10000)
    self.assertEqual(val, "Accept")
```

- Example: The autograder for problem set 1, problem 4

```
def simulate(transition, input, steps):
    SYMBOLS = [">", "_", "0", "1", "#", "$", "&", "%"]
    STATES = ["a", "b", "c", "d", "e", "f", "g", "h", "i", "j", "k", "l", "m", "n", "o", "p"]

    state = STATES[0]
    tape = [SYMBOLS[0]] + list(input)
    headPosition = 1

    for i in range(steps):
        if (headPosition >= len(tape)):
            tape.append(SYMBOLS[1])

        symb = tape[headPosition]
        arr = transition[state][symb]
        if arr == None:
            return "No transition available"

        state = arr[0]
        tape[headPosition] = arr[1]
        headPosition = headPosition + 1 if arr[2] == "R" else headPosition - 1

        ⋮
```

Simulating one step

- For every Turing machine M and configuration C of M, define

$$\mathrm{STEP}(\langle M, C \rangle) = \langle M, \mathrm{NEXT}(C) \rangle$$

Lemma: There exists a Turing machine S that computes STEP. That is, given $\langle M, C \rangle$ as input, the machine S halts, and its final configuration is $\Diamond q_{\text{accept}} \mathrm{STEP}(\langle M, C \rangle)$, possibly followed by some number of ⊔ symbols.

- (Proof left as an exercise)

Universal Turing

What is the universal Turing machine's input alphabet?

A: A fixed, constant-size alphabet that doesn't depend on anything

B: Whatever M's input alphabet is

C: The union of M's input alphabet and the alphabet for encoding M

D: The union of all possible alphabets

Respond at PollEv.com/whoza or text "whoza" to 22333

Theorem: There exists a T

machine M and every inpu

- If M accepts w, then U accepts $\langle M, w \rangle$.
- If M rejects w, then U rejects $\langle M, w \rangle$.
- If M loops on w, then U loops on $\langle M, w \rangle$.

- **Proof sketch:** (1) Construct $C = \Diamond q_0 w$. (2) Alternate between updating $C \leftarrow \text{NEXT}(C)$ and checking whether C is a halting configuration

Universal Turing machines

- A universal Turing machine can be "programmed" to do anything that is computationally possible
- This is why you don't need separate laptops for separate computational tasks
- If you are stranded on an alien planet and you are trying to build a computer, your job is to build a universal Turing machine

The Church-Turing Thesis

- Let L be a language

Church-Turing Thesis:

There exists an "algorithm" / "procedure" for figuring out whether a given string is in L if and only if there exists a Turing machine that decides L.

← Intuitive notion

← Mathematically precise notion

Note on standards of rigor

- Going forward, when we want to construct a Turing machine (e.g., for an existence proof), we will simply describe what it does in plain English, as if we are giving instructions to a human being
  - Each plain English description can be formalized as a Turing machine, but this is tedious
  - You should follow this convention on problem set 3 and beyond
- Nevertheless, the Turing machine model is extremely valuable for us, because it tells us what an arbitrary algorithm looks like!

Which problems

can be solved

through computation?

What are Turing machines capable of?

What are Turing machines NOT capable of?

Decidable and undecidable

- Let L be a language
- We say that L is decidable if there exists a Turing machine M that decides L
- Otherwise, we say that L is undecidable

Computability vs. Complexity

- For now, we don't care how long it takes to decide L
  - "Computability Theory." Possible vs. Impossible
  - As long as M has a finite running time on every input, we're satisfied
- Later, we will study what happens when we do care how long it takes
  - "Complexity Theory." Tractable vs. Intractable
  - We will also consider other computational resources besides time

Which languages are decidable?