Turing Machines (TM)

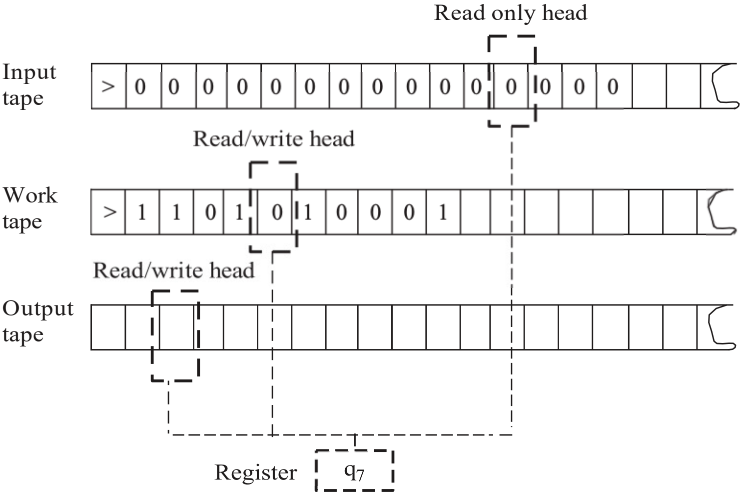


Figure: A 3-tape Turing Machine

Turing Machines (TM)

- The running time $T(n)$ on input of size n is the number of basic operations.
- Making the machine more powerful (e.g., bigger alphabet, more states, etc.) can be simulated by the simpler TM with only polynomial slowdown, i.e., $T(n)$ algorithm on stronger TM runs in $O(T(n)^c)$ time in weaker TM, for some constant $c > 0$.
- Any TM can be described by a fixed-format string (e.g., (Alphabet, States, Transition rules)). A numerical encoding of this string is a unique numerical ID for the TM in our encoding scheme.
  $\Rightarrow$ If α is the encoding of a TM, we will denote this TM as M_α. $\Rightarrow$ TMs can get the encoding α of TM M_α as input.
  $\Rightarrow$ We can built a universal TM (let's call it UTM) that can simulate any other TM M_α it gets in its input. $\Rightarrow$ **SOFTWARE!**
- If $M_\alpha(x)$ runs in time $T(|x|)$, $UTM(\alpha, x)$ runs in time $O(T(|x|) \log T(|x|))$, i.e., we lose only a logarithmic factor.

Computability

- **Decision problem:** Let $f : \{0,1\}^* \to \{0,1\}$ be a binary function. The computation of f is called a decision problem. The set $L_f = \{x : f(x) = 1\}$ is the language defined by f.
- **Algorithm:** A mechanical "recipe" for solving a decision problem that (i) always terminate (ii) with the correct answer.
- If decision problem L has algorithm then it is called decidable. Otherwise it is undecidable.
- If L has a TM that always halts with 1 for all $x \in L_f$, but may not halt when $x \notin L_f$ (and outputs 0 if it halts) is called recursive enumerable.

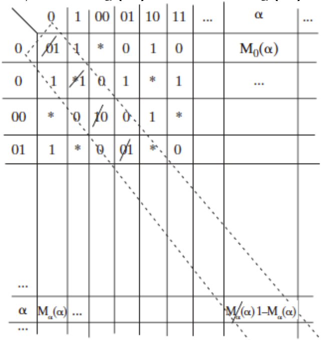
Computability

Theorem 1

There exists function $UC : \{0,1\}^* \to \{0,1\}$ that is not computable by any TM (i.e., UC is undecidable).

Proof: By diagonalization. We define for every string $a \in \{0,1\}^*$

$$UC(a) = \begin{cases} 0, & \text{if } M_a(a) = 1 \\ 1, & \text{if } M_a(a) = 0 \text{ or } M_a(a) \text{ doesn't halt} \end{cases}$$



Computability

Definition: $HALT(a,x) = 1$ iff $M_a(x)$ halts (halting problem).

Theorem 2

$HALT$ is undecidable.

Proof: By reduction $UC \leq HALT$. **Assume** M_{HALT} decides $HALT$.
$M_{UC}(a) :=$ If $M_{HALT}(a,a) = 0$ then return 1 else return $\neg M_a(a)$.

- $M_{HALT}(a,a) = 0 \Rightarrow M_a(a)$ doesn't halt $\Rightarrow UC(a) = 1$
- $M_{HALT}(a,a) = 1 \Rightarrow M_a(a)$ halts $\Rightarrow UC(a) = \neg M_a(a)$

$\Rightarrow M_{UC}(a)$ decides $UC \Rightarrow$ **contradiction** of Thm. 1. $\square$

- Gödel's theorem and Decidability

The class P

Definition: A complexity class is a set of functions computable within given resource bounds.

Definition: Language $L \in DTIME(T(n))$ iff there is TM that decides L in time $c \cdot T(n)$ for some constant $c > 0$.

Note: $DTIME(T(n))$ is defined for *decision problems* (languages).

Definition: $P = \bigcup_{c \geq 0} DTIME(n^c)$

Note 1: n is the size of writing the input in bits (on the TM tape), **not** the value of the input. E.g., algorithm that solves equation $Ax = 1$ in time $O(\log^3 A)$ is polynomial, algorithm that runs in time $O(A^2)$ is pseudo-polynomial.

Note 2: P is the only class closed under composition, i.e., algorithm with poly-time work and polynomially many calls to subroutines in P is still in P!

Computational model may not matter

Church-Turing thesis: Any model of computation can be simulated by a TM.

Church-Turing thesis (strong form): Any model of computation can be efficiently (i.e., with polynomial overhead) simulated by a TM.

Note: What about quantum computers?

Comments on the definition of P

- Worst-case analysis and exact computation
- Precision (or $\mathbb{R}$ vs. $\mathbb{N}$)
- Use of randomness
- Use of quantum mechanics or other exotic physics
- Decision problems are too restrictive

Read Edmond's quote (from someone who defined P when P didn't yet exist...) Read the **Chapter notes and history**.