Theory of Computation
Computable functions and Self Reference

Turing Machines

- $M = (Q, \Sigma, \Gamma, \vdash, B, \delta, s, t, r)$
- States set Q, Input alphabet Σ, Tape alphabet Γ
- left marker $\vdash$, blank symbol B
- transition function δ
- start state s, unique accept state t, unique reject state r.
- Parts of the machine: Input tape, Work tape, tape-head, Finite control

Turing Machine algorithms

- In FLAT, we looked at decision problems with Turing machine algorithms.
- Problem 1: Is the input number n divisible by 2?
- Problem 2: Are two given numbers n, m given as $0^n\#0^m$ coprime?
- Problem 3: Is the input number n a prime number?

Problem 1: Algorithm Sketch

Is the input number n divisible by 2?

- Input: 0^n, otherwise reject.
  Input tape looks like: $\vdash 00\ldots0BB\ldots$, with n $0's$.
- Good practice to put right end marker:
  $\vdash 0^n \dashv BB\ldots$.
- Mark every alternate 0: $\vdash 0\hat{0}0\hat{0}\ldots\dashv BBB$.
  One way: Remember every alternate 0 by using states.
  More general way: Copy 00 in some portion of the work tape, mark the alternate 0 by using some sort of matching of 00 with portions of 0^n.
- Even: if you do not try to mark $\dashv$ as $\hat{\dashv}$. Stop after crossing $\dashv$.
- Go to accept state t if n is even, otherwise go to reject state r.

Problem 2: Algorithm Sketch

Are two given numbers n, m given as $0^n\#0^m$ coprime?

- Input: $0^n\#0^m$, otherwise reject.
  Input tape looks like: $\vdash 0^n\#0^m \dashv BB\ldots$.
- Coprime: Need to determine if $\gcd(n, m) = 1$.
- Euclidean Algorithm:
  $\gcd(n, m)$:
  if $m = 0$ then return n;
  else return $\gcd(m, n \mod m)$.

Problem 2: Algorithm Sketch contd.

Are two given numbers n, m given as $0^n\#0^m$ coprime?

- From $\vdash 0^n\#0^m \dashv BB\ldots$ you need to:
  Check if the symbol after the $\#$ is $\dashv$: then $m = 0$ and $\gcd(n, 0) = n$.
  Else, write down $\vdash 0^{n \bmod m}\#0^m \dashv$ (preferably on the work tape).
  Then, reverse the string: You should have $\vdash 0^m\#0^{n \bmod m} \dashv$, preferably on the work tape.
  Continue with the Euclidean algorithm till you get an answer.
- Go to accept state t if $\gcd(n, m) = 1$, otherwise go to reject state r.

Problem 3: Algorithm Sketch

Is the input number n a prime number?

- Input $\vdash 0^n BB\ldots$.
  With right end marker: $\vdash 0^n \dashv BB\ldots$.
- Algorithm for determining primes: sieve of Eratosthenes;
  Write down all numbers $2, 3, \ldots, n$.
  Take the smallest number on the list that is not crossed off.
  Cross of all its multiples.
  Continue till you can: if all numbers except for n has been crossed off, then n is a prime number.
- Main steps of implementation:
  (i) Finding the smallest number that has not been marked off;
  (ii) Finding multiples of that number.
- Go to accept state t if n is prime, otherwise to reject state r.

Computable Functions

- A decision problem can be thought of as a binary function $f : \Sigma^* \to \{0,1\}$.
- Eg: Is the input number n divisible by 2?
  $f(0^n) = 1$ if n is even.
  For any other $x \in \{0,1\}^*$, $f(x) = 0$.
- The functions that have corresponding Turing machine algorithms are called *computable functions*.

More computable functions

- A Turing machine can also be thought as a computer of functions from positive integers to positive integers.
- Input: If the function f has one argument, say i_1 then it is represented in unary as 0^{i_1}
- Input: If the function has multiple arguments, say k arguments $\{i_1, i_2, \ldots, i_k\}$ then it is represented as $0^{i_1}10^{i_2}1\ldots10^{i_k}$ - unary representation of arguments separated by 1's
- Note that padding by more 1s is allowed - we only care about the maximal blocks of consecutive 0s.
- Output: if $f(i_1, i_2, \ldots, i_k) = m$ then the output should be 0^m - unary representation of m.

Integer Computable functions

- We can design the Turing machines such that all k parameters of a function need not be defined.
- We can also define the transition function δ such that depending on the number of parameters in the input, a different integral function (upto constantly many) is computed: if input has one argument then f_1 is computed, two arguments then f_2 is computed and so on for a constant number of functions.

Integer Computable functions contd.

- $f(i_1, i_2, \ldots, i_k)$ is defined for all $i_1, i_2, \ldots, i_k$ and has a Turing machine computing it - total recursive function. Correspond to recursive languages.
- $f(i_1, i_2, \ldots, i_k)$ not defined for all $i_1, i_2, \ldots, i_k$ and has a Turing machine computing it on the defined values (will loop on the undefined values) - partial recursive function. Correspond to recursively enumerable languages.

Total recursive functions

Try the following functions:

- $f(n) = 2^{2^n}$
- $f(m, n) = m - n$

Self Reference

- A Turing machine can do a lot more!
- **Can design a TM SELF that can ignore the given input and print out a copy of its own description.**
- Usually we think of a machine being produced by something more powerful than itself - but this is contradicted if we can construct SELF.
- Similar example in programming languages: A program that outputs a copy of itself.
- Similar example in English language: Print out this sentence

Designing SELF: Auxiliary computable function

Lemma: There is a computable function $q : \Sigma^* \to \Sigma^*$ where, for any string w, $q(w)$ is the description of a Turing machine P_w that prints out w and then halts.
Proof Sketch:

- Construct a machine M_q that does the following:
- Takes input w
- Constructs P_w such that on any input P_w will erase that input and write w on its worktape and halt.
- Outputs $< P_w >$, which is the encoding of P_w.

Constructing SELF

- We construct two subroutines A, B such that $SELF = AB$. SELF should output encoding $< SELF >=< AB >$
- $A = P_{<B>}$. So A on any input outputs $< B >$. Thus $< A >=< P_{<B>} >= q(< B >)$.
- B on input $< M >$ where M is a portion of a Turing Machine, computes $q(< M >)$ first by using M_q as a subroutine.
- Then B concatenates the resultant string and $< M >$ to make a complete TM description.
- Finally B prints this description in the worktape and halts.

Constructing SELF contd.

- First A runs and prints $< B >$ on the tape of $SELF$.
- Then B starts. It looks at the tape of $SELF$ and finds its input $< B >$.
- B calculates $q(< B >) =< A >$ and concatenates in front of $< B >$ to obtain the TM description $< SELF >=< AB >$.
- B prints this description and halts.