Computability – Key Points

- Church-Turing thesis and its importance
- definitions of *recursively enumerable* and *recursive* with respect to languages
- intuition for the equivalence of Turing machines to other models of computation, including real computers

Church-Turing Thesis

- an *effective method* is one which
  - can be expressed by a finite number of instructions, each involving a finite number of symbols
  - always terminates in a finite number of steps, and always produces a correct answer
  - can, at least in principle, be carried out by a human with only pencil and paper
  - requires no ingenuity, only rote following of the instructions
- the Church-Turing thesis states that

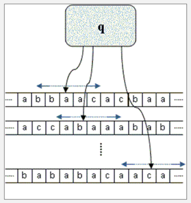
  a function on the natural numbers can be calculated by an effective method if and only if it is computable by a Turing machine

Alonzo Church

- American mathematician, 1903-1995
- known for
  - major contributes to mathematical logic
  - foundational contributions to theoretical computer science
  - λ calculus
    - underlies functional programming languages such as Scheme as well as Java lambda expressions
  - Church's theorem
    - proved the unsolvability of the Entscheidungsproblem (decision problem) by showing that there is no computable function which decides if two given λ-calculus expressions are equivalent
  - Church-Turing thesis

https://en.wikipedia.org/wiki/Alonzo_Church

Two (Multi)-Tape Turing Machines



- two (or more) tapes, each with separate read/write heads
- idea
  - let *M* be a two-tape Turing machine and *K* be a one-tape Turing machine
  - introduce two new symbols (e.g. @, $) to mark the position of the read/write heads and separate tape 1's contents from tape 2
  - *K*'s tape will contain the contents of *M*'s tape 1 followed by the contents of *M*'s tape 2, with @ inserted to the left of the head position on each tape and $ at the beginning, end, and between the two tapes
    - e.g. if *M*'s tapes contain `abb##cca` and `01#111#001`, respectively, with the heads on the underlined symbols, then *K*'s tape will contain `$a@bb##cca$01#111#00@1$`
  - to simulate *M*'s operation, *K* scans tape to find symbols to the right of the @ symbols, then updates its state and the tape content accordingly

https://gyires.inf.unideb.hu/GyBITT/26/ch04s06.html

Recursively Enumerable

- a *recursively enumerable language* is one for which there is a program whose output is exactly the strings in the language
- a *recursively enumerable language* is one whose strings can be output on the second tape of a two-tape Turing machine
  - no requirement as to order, and repeats are allowed

Theorem 5.1. *Let Σ be an alphabet and let L be a language over Σ. Then the following are equivalent:*

1. *There is a Turing machine that accepts L.*
2. *There is a two-tape Turing machine that runs forever, making a list of strings on its second tape, such that a string w is in the list if and only if $w \in L$.*
3. *There is a Turing-computable function $f\colon \{a\}^* \to \Sigma^*$ such that L is the range of the function f.*

- idea of proof
  - property 3 → property 2
    - let L be a language that satisfies property 3
    - construct a two-tape Turing machine that, for each $n \geq 0$, uses tape 1 to generate a^n and compute $f(a^n)$, then copies $f(a^n)$ to tape 2
  - property 2 → property 3
    - let M be a machine that lists L
    - define g to be the function where $g(a^n)$ is the $(n+1)^{th}$ item in the list produced by M
    - g is Turing-computable because $g(a^n)$ can be produced by running M until the $(n+1)^{th}$ item is produced, then halting with that item as the output

Theorem 5.1. *Let Σ be an alphabet and let L be a language over Σ. Then the following are equivalent:*

1. *There is a Turing machine that accepts L.*
2. *There is a two-tape Turing machine that runs forever, making a list of strings on its second tape, such that a string w is in the list if and only if $w \in L$.*
3. *There is a Turing-computable function $f\colon \{a\}^* \to \Sigma^*$ such that L is the range of the function f.*

- idea of proof
  - property 2 → property 1
    - let L be a language that satisfies property 2
    - let T be a two-tape Turing machine that lists the elements of L
    - construct M which, given an input w, simulates the computation of T – when T produces a string in the list, M compares the string to w and halts if they are the same
    - if $w \in L$, T will eventually produce it and M will halt → M accepts L

Theorem 5.1. *Let Σ be an alphabet and let L be a language over Σ. Then the following are equivalent:*

1. *There is a Turing machine that accepts L.*
2. *There is a two-tape Turing machine that runs forever, making a list of strings on its second tape, such that a string w is in the list if and only if $w \in L$.*
3. *There is a Turing-computable function $f\colon \{a\}^* \to \Sigma^*$ such that L is the range of the function f.*

- idea of proof
  - property 1 → property 2
    - let L be Turing-acceptable and M be a machine that accepts L
    - cannot build a two-tape machine T by generating each of the elements of Σ^* in turn, checking to see if M accepts each because M only halts if $w \in L$
    - T must instead simulate M on all of the elements of L at once – it repeatedly generates the next element in Σ^* and then advances M one step on all of the current elements, writing the corresponding input to tape 2 whenever a computation halts
    - T eventually goes through all elements of Σ^*, and simulation of M will eventually end for all $w \in L$, so T will eventually produce all $w \in L$

Theorem 5.2. *A language L is Turing acceptable (equivalently, recursively enumerable) if and only if there is a general grammar that generates L.*

- idea of proof
  - grammar $\rightarrow$ Turing acceptable
    - M generates every string derivable from the start symbol S –
      - start with `w$S` on the tape
      - repeatedly
        - for each string on the tape and each production $x \rightarrow y$, if x occurs in the string, append `$` to the end of the tape and copy the string, replacing x with y
        - compare the new string to w, halting if they match
    - if $w \in L$, eventually M will produce it and halt