Algorithms: (Informal for now - Later: formal definition)

Informal notion - good enough to prove algorithms exist.

Formal notion - needed to prove impossibility of algorithms

Informal definition of algorithm

- Given by a finite set of unambiguous instructions.
- Describe step-by-step computations.
- Only access a finite amount of data once.
- There is a fixed upper bound on how much data can be accessed.
- No appeal to intuition, to human judgement.

Cannot add arbitrary precision integers in a single step.

Usually: algorithms work with symbols, strings of symbols.
Integers can be represented in base 2, or base 10 a unary

Combiningfd objects: can coded as a string of symbols.
Efficiency is not a concern

"Effective" - algorithm is allowed to use arbitrary time and space

vs "Feasible" - algorithms that run fast enough or efficiently enough to be usable.

Church-Turing thesis:
The informal notion of algorithm corresponds to the formal definitions that will be given later.

A consequence

There is a universal algorithm - i.e. that simulate any other algorithm.
Make compilers & interpreters.

"Meta-algorithm" - algorithm that describes or algorithms as input.
Forming algorithms based on the informal notion of algorithm (for now)

Conventions

Defn A alphabet $\Sigma$ is a finite set of symbols.

Example $\Sigma = \{0, 1\}$ or $\Sigma = \{0, 1, \ldots, 9\}$ or $\Sigma = \{a, b, c\}$

Defn $k$-ary function $f : (\Sigma^*)^k \rightarrow \Sigma^*$

$\Sigma^*$ - set of strings of symbols from $\Sigma$ of length $\geq 0$

$\emptyset$ - empty string $|\emptyset| = 0$

If $w \in \Sigma^*$, $|w|$ = length + # of symbols in $w$

Example Concatenation function $f(w_1, w_2) = w_1w_2$ $w_1, w_2 \in \Sigma^*$

Suppose $w_1 = a_1a_2 \ldots a_e$ $a_i \in \Sigma$ $w_2 = b_1 \ldots b_k$ $b_i \in \Sigma$

$w_1w_2 = a_1a_2 \ldots a_e b_1b_2 \ldots b_k$

Example $f(w) = |w|$ encoded in base 2 $\Sigma = \{0, 1\}$

$f(101101100) = 1001$
$$\Sigma^* = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup \Sigma^3 \cup \ldots$$

$$\Sigma^k = \{ \omega \in \Sigma^* : |\omega| = k \}$$

**Defn** A *k-ary function* $f : (\Sigma^*)^k \rightarrow \Sigma^*$ is **computable** if there is an algorithm $M$ such that if $\omega_1, \ldots, \omega_k \in \Sigma^*$, $M$ on input $\omega_1 \ldots \omega_k$ eventually outputs $\omega = f(\omega_1 \ldots \omega_k)$.

**Example** *Concatenation.*

The length function example is computable.

**Defn** A *k-ary relation* $R$ is a subset of $(\Sigma^*)^k$.

If $\omega_1, \ldots, \omega_k \in \Sigma^*$, $R(\omega_1 \ldots \omega_k)$ is either true or false, i.e., $<\omega_1 \ldots \omega_k>$ is either in $R$ or not in $R$.

**Defn** A relation is **decidable** if there is an algorithm $M$ such that for all $\omega_1, \ldots, \omega_k \in \Sigma^*$, if $R(\omega_1 \ldots \omega_k)$, then $M$ on input $\omega_1 \ldots \omega_k$ produces "Accept" ("yes")

if $\neg R(\omega_1 \ldots \omega_k)$, then $M$ on input $\omega_1 \ldots \omega_k$ produces "Reject" ("no").
\( M(w_1 \cdots w_k) \) denotes the action of \( M \) on input \( w_1 \cdots w_k \).

For \( M \) to decide \( R \), \( M(w_1 \cdots w_k) \) must produce an answer (must "halt" on all inputs).

\( M(w_1 \cdots w_k) \) accepts if \( R(w_1 \cdots w_k) \) holds.

\( M(w_1 \cdots w_k) \) rejects if \( R(w_1 \cdots w_k) \) does not hold.

**Example**  Set of palindromes: \( \{ \omega \in \Sigma^* : \omega^R = \omega \} \).

\( \omega^R \) is the reversal of \( \omega \) as \( \omega \) written backward.

\((a, \cdots a)^R = a a a \cdots a, \quad a, \in \Sigma.\)

This is decidable.

**Example** \( f: \mathbb{N} \to \{0, 1, 2\}^* \quad f(i) - an \approximation\) to \( \pi \) accurate to within \( \frac{1}{i} \).

is (becomes) a computable function.
Let \( R \) be a \( k \)-ary relation of \( N \)
(viewing \( N \) as binary representations).

The characteristic function \( X_R \) of \( R \) is the \( k \)-ary function

\[
X_R(n_1, \ldots, n_k) = \begin{cases} 
1 & \text{if } R(n_1, \ldots, n_k) \text{ is true} \\
0 & \text{if } R(n_1, \ldots, n_k) \text{ is false}
\end{cases}
\]

Thus, \( X_R \) is computable if \( R \) is decidable.

**Pf:** Assume \( M \) decides \( R \)

Input \( n_1, \ldots, n_k \)

Algorithm

Run \( M \) on \( \text{input } n_1, \ldots, n_k \)

If \( M \) accepts, output \( 1 \) (and halt)
If \( M \) rejects, output \( 0 \) (and halt).