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

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 ~~to~~ 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 or unary

Combinatorial 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 & interpreter.

"Meta-algorith" - algorithm that descriptions of algorithms as input.

Formalizing algorithms based on the informal notion of algorithm (for now)

Conventions

Defn A alphabet Σ is a finite set of symbols.

Example $\Sigma = \{0,1\}$ or $\Sigma = \{0,1,\dots 9\}$ or $\Sigma = \{a\}$

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

Σ^* - set of strings of symbols from Σ of lengths ≥ 0

ε - empty string $\quad |\varepsilon| = 0$

"w" "word" i.e "string"

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

Example Concatenation function $f(w_1, w_2) = w_1 w = w_1 \circ w_2$

Suppose $w_1 = a_1 a_2 \dots a_\ell \quad a_i \in \Sigma \quad w_1 \in \Sigma^*$

$w_2 = b_1 \dots b_k \quad b_i \in \Sigma \quad w_2 \in \Sigma^*$

$w_1 w_2 = a_1 a_2 \dots a_\ell b_1 b_2 \dots b_k$

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

$f(10110110) = 1001$

$$\Sigma^* = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup \Sigma^3 \cup \cdots$$

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

computable

__Def'n__ A k-ary function $f : (\Sigma^*)^k \to \Sigma^*$ is __computable__ if there is an algorithm M s.t. if $w_1, \ldots, w_k \in \Sigma^*$ M on input $w_1 \ldots w_k$ eventually outputs $w = f(w_1 \ldots w_k)$

__Example__ Concatenation.

The length function example is computable.

__Def'n__ A k-ary relation R is a subset of $(\Sigma^*)^k$

If $w_1 \ldots w_k \in \Sigma^*$, $R(w_1 \ldots w_k)$ is either true or false

i.e. $\langle w_1 \ldots w_k \rangle$ is either in R or not in R.

__Defn__ A relation is __decidable__ if there is an algorithm M s.t. for all $w_1 \ldots w_k \in \Sigma^*$

if $R(w_1 \ldots w_k)$, then M on input $w_1 \ldots w_k$ produces "Accept" ("Yes")

if $\neg R(w_1 \ldots w_k)$, then M on input $w_1 \ldots w_k$ 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 iff $R(w_1 \cdots w_k)$ holds

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

Example Set of palindromes. $= \{w \in \Sigma^* : w^R = w\}$.

w^R - reversal of w is w written backwards

$(a_1 \cdots a_\ell)^R = a_\ell a_{\ell-1} \cdots a_1 \qquad a_i \in \Sigma$.

This is decidable.

Example $f: \mathbb{N} \to \{0,1\}^*$ $\qquad f(i)$ - an approximation to π accurate to within $1/i$.

is (becomes) a computable function.

Defn Let R be a k-ary relation of $\mathbb{N}$

(viewing $\mathbb{N}$ as binary representations)

The characteristic function χ_R of R is the

k-ary function

$$\chi_R(n_1 \ldots n_k) = \begin{cases} 1 & \text{if } R(n_1 \ldots n_k) \text{ is true} \\ 0 & \text{if } R(n_1 \ldots n_k) \text{ is false} \end{cases}$$

Thm χ_R is computable iff R is decidable.

Pf: Assumption: M decides R

Input $n_1 \ldots n_k$

Algorithm

Run M on input $n_1 \ldots n_k$

If M accepts, output 1 (and halts)

If M rejects, output 0 (and halt).