What is an algorithm?

Yiannis N. Moschovakis
UCLA and University of Athens

CSLI, May 31, 2014

36,500,000 Google hits and no definition

- Wikipedia: *An algorithm is a step-by-step procedure for calculations*
- Knuth: *A computational method is . . .* (*computation model*)
  An algorithm is a computational method which terminates in finitely many steps for all [inputs]
- Common: *An algorithm is a program* or *An algorithm or program . . .*

This approach takes algorithms to be syntactic objects

> *Frege: You cannot forbid the use of an arbitrarily produced process or object as a sign for something else*

- Common: *Algorithms are Turing machines*
  or *processes which can be simulated by Turing machines*

Turing machines do not express faithfully low complexity algorithms
van Emde Boas: simulation . . . is hard to define as a mathematical object

How can there be no definition?

when algorithms have been studied extensively, deeply and rigorously for (at least) eighty years?

- The Justice Potter Stewart argument: *I know one when I see it*

  For rigorous analysis, algorithms are specified precisely by a computation model or a recursive procedure

- In particular, complexity theory and (especially) the derivation of lower bounds with respect to various complexity measures, is always developed relative to fixed computation models

- Compare to: probability theory, which was developed intensively (and rigorously) for some 300 years before random variables were defined precisely in full generality (Kolmogorov 1933)

Outline of the talk

- Three classical algorithms (4 slides)
- Least fixed point recursion (2 slides)
- Recursors, the set-theoretic objects which model algorithms (5 slides)
- Algorithms from specified primitives (2 slides)
- One application (if time permits) (2 slides)

Some references, all posted in www.math.ucla.edu/$\sim$ynm

- The formal language of recursion (1989)
- A mathematical modeling of pure, recursive algorithms
- On founding the theory of algorithms (1998)
- What is an algorithm? (2001)
- Is the Euclidean algorithm optimal among its peers?, with Lou van den Dries (2004)
- Elementary algorithms and their implementations, with Vassilis Paschalis (2008)

The Euclidean algorithm

For $x, y \in \mathbb{N},\ x \geq y \geq 1$,

$$(*) \quad \boxed{\mathrm{gcd}(x, y) = \text{if } (\mathrm{rem}(x, y) = 0) \text{ then } y \text{ else } \mathrm{gcd}(y, \mathrm{rem}(x, y))}$$

where $\mathrm{rem}(x, y)$ is the remainder of the division of x by y,

- $(*)$ expresses an algorithm ε from $\mathrm{rem}, =_0$ which computes $\mathrm{gcd}(x, y)$

$$\begin{aligned} \mathrm{calls}_{\varepsilon}^{\mathrm{rem}}(x, y) = &\text{ the number of calls to rem} \\ &\text{ required to compute } \mathrm{gcd}(x, y) \text{ by } \varepsilon \\ \leq &\ 2\log(y) \qquad (x \geq y \geq 2) \end{aligned}$$

Conjecture (open): *For every algorithm α which computes* $\mathrm{gcd}(x, y)$ $(x, y \in \mathbb{N}, x \geq y > 0)$ *from* rem *and* $=_0$,

there is a sequence $(x_n \geq y_n)_n$, such that $y_n \to \infty$

$$\text{and } \mathrm{calls}_{\varepsilon}^{\mathrm{rem}}(x_n, y_n) \leq \mathrm{calls}_{\alpha}^{\mathrm{rem}}(x_n, y_n)$$

- It assumes that "algorithm from $\mathrm{rem}, =_0$" and $\mathrm{calls}_{\alpha}^{\mathrm{rem}}$ are defined

The color of leaves

A (binary, colored) forest is a structure

$$\boxed{\mathbf{F} = (F, s, d, \mathrm{Leaf}, \mathrm{Red}, =)} \text{ where } \mathrm{Leaf}, \mathrm{Red} \subseteq F \text{ and } s, d : F \rightarrowtail F$$

A (maximal) *path from x* is any sequence $p = (x_0, \ldots)$ of length $|p| \leq \infty$ such that

$$i < |p| \implies [\neg \mathrm{Leaf}(x_i) \;\&\; x_{i+1} \in \{s(x_i), d(x_i)\}]$$

- $\mathbf{F}$ is grounded if it has no infinite paths. On grounded $\mathbf{F}$ let

$$R(x) \iff \text{every path from } x \text{ ends in a red leaf}$$

$$(*) \quad \boxed{R(x) \iff \text{if } \mathrm{Leaf}(x) \text{ then } \mathrm{Red}(x) \text{ else } [R(s(x) \;\&\; R(d(x))]}$$

- $(*)$ expresses a recursive algorithm ρ which decides $R(x)$ on $\mathbf{F}$
  - The Euclidean can be expressed by a while program from rem, $=_0$
  - (Tiuryn 1989) *On some grounded forest, no algorithm expressed by a while program of* $\mathbf{F}$ *decides* $R(x)$

($\mathbf{F}$ is the disjoint union of all finite, binary, colored trees)

The sieve of Eratosthenes

$$\text{Primes} = p(u_0) \text{ where}$$

$$\Big\{ u_0 = (2, 3, 4, 5, \ldots),$$
$$p(u) = \text{Print}(\text{head}(u))\frown p(\textit{sieve}(\text{head}(u), \text{tail}(u))),$$
$$\textit{sieve}(x, v) = \text{if } (x \mid \text{head}(v)) \text{ then } \textit{sieve}(x, \text{tail}(v))$$
$$\text{else head}(v)\frown \textit{sieve}(x, \text{tail}(v)) \Big\}$$

$(S = (\mathbb{N} \to \mathbb{N}), u_0, u, v \in S, p : S \to S, x \in \mathbb{N}, \textit{sieve} : \mathbb{N} \times S \to S)$

- A system of mutual recursive equations which expresses an algorithm σ from head, tail, $\mid$, $\frown$ and (the act) Print
- $\textit{sieve}(x, v)$ removes from v all numbers divisible by x
- $p(u)$ prints head(u) and then calls itself on $\textit{sieve}(\text{head}(u), \text{tail}(u))$
- σ computes successively
  $u_0 = (2, 3, 4, \ldots)$, $u_1 = (3, 5, 7, \ldots)$, $u_2 = (5, 7, 11, \ldots), \ldots$
  and (as a side effect) "prints" the heads of these sequences
- Does the recursive system above "specify" σ completely?

The basic, practical problem — too many notions!

- It seems like the basic notion should be that of algorithm from (given) primitives

- Too many notions associated with an algorithm: calls, recursive definitions, complexity functions, side effects (and interaction, which is more complex), simulation, implementability, ...

For specific algorithms many of these are natural and simple, but a general theory might be excessively complex

- The lesson from probability theory: it is even more complex, but there is a useful and fairly simple basic notion:

A random variable is a measurable function $X : M \to \mathbb{R}$ on a sample space (a measure space of total measure 1)

- *For algorithms, the background mathematical theory is fixed point recursion on complete posets*

Complete posets

- A poset is a pair $(D, \leq_D)$ where $\leq_D$ is a partial ordering of D
- A poset D is (directed or chain) complete if every linearly ordered subset $X \subseteq D$ (a chain) has a least upper bound $\sup(X)$. Every complete poset has a least element, $\sup(\emptyset) = \perp$
- A set A is identified with the flat poset $A_\perp = A \cup \{\perp\}$, where
$$x \leq_{A_\perp} y \iff x = \perp \vee x = y$$
- The (naturally defined, cartesian) product $D_1 \times \cdots \times D_n$ of complete posets is complete
- A function $f : D \to E$ is monotone if
$$x \leq_D y \implies f(x) \leq_E f(y),$$
and strict if in addition $f(x) \neq \perp \implies x$ is total (maximal) in D
- $\mathrm{Mon}(D, E), \mathrm{Strict}(D, E)$ are the posets of monotone and strict functions ordered pointwise. They are complete, if D, E are complete

Least fixed point recursion

- A function $f : D \to E$ on complete posets is (Scott) continuous if

$$\sup_E\{f(x) \mid x \in D\} = f(\sup_D X) \quad \text{(for every chain } X \subseteq D)$$

- The poset $\mathrm{Cont}(D, E)$ of all continuous functions is complete and

$$\mathrm{Strict}(D, E) \subseteq \mathrm{Cont}(D, E) \subseteq \mathrm{Mon}(D, E)$$

Theorem (classical)

Every monotone function $f : D \to D$ on a complete poset has a least fixed point $\overline{d} = \min(d \in D)[f(d) = d]$, characterized by

$$f(\overline{d}) = \overline{d}, \quad (\forall d)[f(d) \leq d \implies \overline{d} \leq d]$$

Moreover: if $f : X \times D \to D$ is monotone, then the function

$$g(x) = \min(d \in D)[f(x, d) = d] \quad (x \in X)$$

is also monotone, and if f is continuous, then so is g

⋆ (Monotone) recursors

- A recursor $\alpha : X \rightsquigarrow W$ on one complete poset to another is a tuple

$$\alpha = (\alpha_0, \alpha_1, \ldots, \alpha_k),$$

such that for suitable, complete posets $D_1, \ldots, D_k$:

(1) Each part $\alpha_i : X \times D_1 \times \cdots D_k \to D_i$, $(i = 1, \ldots, k)$ is monotone

(2) The output part $\alpha_0 : X \times D_1 \times \cdots \times D_k \to W$ is also monotone

- α_0 is the head of α; $(\alpha_1, \ldots, \alpha_k)$ its body; $D_\alpha = D_1 \times \cdots \times D_k$ is its solution poset, and its transition mapping $\tau_\alpha : X \times D_\alpha \to D_\alpha$ is

$$\tau_\alpha(x, d) = (\alpha_1(x, d), \ldots, \alpha_n(x, d)) \quad (x \in X, d \in D_\alpha)$$

- The function $\overline{\alpha} : X \to W$ computed by α is

$$\overline{\alpha}(x) = \alpha_0(x, \overline{d}_x), \text{ where } \overline{d}_x = \min(d \in D_\alpha)[\tau_\alpha(x, d) = d]$$

- We express all this succinctly by writing

$$\alpha(x) = \alpha_0(x, d) \text{ where } \{d = \tau_\alpha(x, d)\}, \qquad \text{(recursor)}$$

$$\text{(function)} \qquad \overline{\alpha}(x) = \alpha_0(x, d)\, \overline{\text{where}}\, \{d = \tau_\alpha(x, d)\}$$

The importance of the solution poset

$\alpha(x) = \alpha_0(x, d)$ where $\{d = \tau_\alpha(x, d)\}$, $(x \in X, d \in D_\alpha = D_1 \times \cdots \times D_k)$

- The Morris example (Manna 1975)

$$\boxed{p(s,t) = \text{if } (s = 0) \text{ then } 0 \text{ else } p(s-1, p(s,t))} \quad (s, t \in \mathbb{N})$$

- The "official" associated recursor is

$\alpha(s,t) = p(s,t)$

where $\{p = \lambda(s,t)(\text{if } (s = 0) \text{ then } 0 \text{ else } p(s-1, p(s,t)))\}$

Solutions of the Morris recursive equation:

- If p varies over $\mathrm{Strict}(\mathbb{N}^2, \mathbb{N})$ (call by value),

$$\overline{p}(s,t) = \text{if } (s = 0) \text{ then } 0 \text{ else } \perp$$

- If p varies over $\mathrm{Cont}(\mathbb{N}^2, \mathbb{N})$ or $\mathrm{Mon}(\mathbb{N}^2, \mathbb{N})$ (call by name), $\overline{p}(s,t) = 0$
- In the recursor representing the sieve of Eratosthenes we should use streams and continuous function spaces to insure "implementability"

⋆ Recursor isomorphism (identity)

Suppose $\alpha, \beta : X \rightsquigarrow W$ are recursors

$$\alpha(x) = \alpha_0(x, d) \text{ where } \{d = \tau_\alpha(x, d)\}, \quad D = D_1 \times \cdots, D_k$$
$$\beta(x) = \beta_0(x, e) \text{ where } \{e = \tau_\beta(x, e)\}, \quad E = E_1 \times \cdots \times E_l$$

• *A recursor does not change if we replace its posets by isomorphic copies and permute the order of the parts in its body*

We say that α is naturally isomorphic (equal) with β, $\alpha \cong \beta$, if

• $k = l$, i.e., α and β have the same number of parts

• There is a permutation $\pi : \{1, \dots, n\} \rightarrowtail\!\!\!\!\!\twoheadrightarrow \{1, \dots, n\}$ and for each $i = 1, \dots, k$, a poset isomorphism $\rho_i : D_i \rightarrowtail\!\!\!\!\!\twoheadrightarrow E_{\pi(i)}$, such that the induced isomorphism $\rho_\pi : D \rightarrowtail\!\!\!\!\!\twoheadrightarrow E$ preserves the parts, i.e.,

$$\alpha_0(x, d) = \beta_0(x, \rho_\pi(d)),$$
$$\rho_i(\alpha_i(x, d)) = \beta_i(x, \rho_\pi(d)) \quad (i = 1, \dots, k)$$

• Natural recursor isomorphism is a very fine notion—perhaps too fine

Operations on recursors, I

- Degenerate recursors. Each function $f : X \to W$ can be viewed as a degenerate recursor (f) with empty body,

$$\delta_f(x) = f(x) \text{ where } \{ \ \}$$

- Composition of a recursor with a function. For $\beta : Y \rightsquigarrow W$ and $g : X \to Y$ a monotone function, put

$$\alpha(x) = \beta(g(x)) = \beta_0(g(x), d) \text{ where } \{d = \tau_\beta(g(x), d)\};$$

then $\boxed{\overline{\alpha}(x) = \overline{\beta}(g(x))}$

- Recursor composition. For $\gamma : X \rightsquigarrow V$ and $\beta : V \times Z \rightsquigarrow W$, put

$$\alpha(x, z) = \beta(\gamma(x), z)$$
$$= \beta_0(v, z, d) \text{ where } \{v = \gamma_0(x, e), e = \tau_\gamma(x, e), d = \tau_\beta(v, z, d)\};$$

then $\boxed{\overline{\alpha}(x, z) = \overline{\beta}(\overline{\gamma}(x), z)}$

- $\delta_f(g(x)) = f(g(x)) \text{ where } \{\,\} \neq \delta_f(\delta_g(x)) = f(v) \text{ where } \{v = g(x)\}$

Operations on recursors, II

- λ-substitution. For given β, γ, put

$$\alpha(x) = \beta(\lambda u \gamma(x,u)) = \beta_0(r,e) \text{ where } \{e = \tau_\beta(r,e),$$
$$r = \lambda u \gamma_0(x,u,d(u)), d = \lambda u \tau_\gamma(x,u,d(u))\};$$

then $\boxed{\overline{\alpha}(x) = \overline{\beta}(\lambda u \overline{\gamma}(x,u))}$

- Recursor recursion. For given recursors $\beta^0, \ldots, \beta^k$, put

$$\begin{aligned}
\alpha(x) = \quad & \beta^0(x,d) \text{ where } \{d_1 = \beta^1(x,d), \ldots, d_k = \beta^k(x,d)\} \\
= \quad & \beta^0_0(x,d,e^0) \text{ where } \{d_1 = \beta^1_0(x,d,e^1), \ldots, d_k = \beta^k_0(x,d,e^k), \\
& \qquad e^0 = \tau_{\beta^0}(x,d,e^0), \\
& \qquad e^1 = \tau_{\beta^1}(x,d,e^1), \\
& \qquad \vdots \\
& \qquad e^k = \tau_{\beta^k}(x,d,e^k)\};
\end{aligned}$$

then $\boxed{\overline{\alpha}(x) = \overline{\beta}^0(x,d) \,\overline{\text{where}}\, \{d_1 = \overline{\beta}^1(x,d), \ldots, d_k = \overline{\beta}^k(x,d)\}}$

Monotone and strict structures

- A monotone structure is a pair $\mathbf{M} = (\mathcal{D}, \mathcal{F})$, where

(1) $\mathcal{D}$ is a set of complete posets which contains the (flat) Boolean poset $\{\mathrm{tt}, \mathrm{ff}\}_\perp$

(2) Each $f \in \mathcal{F}$ is a monotone function

$$f : D_1 \times \cdots \times D_n \to D \quad (D_1, \ldots, D_k, D \in \mathcal{D})$$

- With each (first order) structure

$$\mathbf{A} = (A, \{\phi^{\mathbf{A}}\}_{\phi \in \Phi}) \quad (\Phi \text{ a set of function and relation symbols})$$

we associate the strict monotone (in fact continuous) structure

$$\mathbf{A}_s = (\{A_\perp, \{\mathrm{tt}, \mathrm{ff}\}_\perp\}, \{\phi_s^{\mathbf{A}}\}_{\phi \in \Phi}), \text{ where}$$

if ϕ is a k-ary function symbol, then $\phi_s^{\mathbf{A}} : A_\perp^k \to A_\perp$ is the strict extension of $\phi^{\mathbf{A}}$,
and if ϕ is a k-ary relation symbol, then $\phi_s : A_\perp^k \to \{\mathrm{tt}, \mathrm{ff}\}_\perp$ is the strict extension of the *characteristic function* of $\phi^{\mathbf{A}}$

- We do not assume that $=_A$ is one of the primitives of $\mathbf{A}$

⋆ Algorithms from specified primitives

Slogan: an algorithm of **M** *is a recursor which is explicitly definable from the primitives of* **M**

- An algorithm of a monotone structure $\mathbf{M} = (\mathcal{D}, \mathcal{F})$ (or from $\mathcal{F}$) is a recursor which belongs to every collection of recursors $\mathcal{R}$ with the following properties:

(A1) $\mathcal{R}$ contains all the (degenerate) recursors δ_f with $f \in \mathcal{F}$

(A2) $\mathcal{R}$ contains δ_f for every "relevant" call or conditional function

$$\mathrm{ev}^{D,W}(p, x) = p(x) \quad (p : D_1 \times \cdots \times D_n \to E \text{ with } D_i, W \in \mathcal{D})$$

$$\mathrm{cond}^{W}(r, x, y) = \text{if } r \text{ then } x \text{ else } y \quad (r \in \{\mathrm{tt}, \mathrm{ff}\}_\perp, x, y \in W \in \mathcal{D})$$

(A3) $\mathcal{R}$ is closed under composition with the functions $\mathrm{ev}^{D,W}$ and every projection $\pi_i(x_1, \ldots, x_n) = x_i \quad (1 \leq i \leq n)$

(A4) $\mathcal{R}$ is closed under recursor composition, λ-substitution and recursor recursion

- For a first order structure **A**, the algorithms of **A** (i.e., $\mathbf{A}_s$) compute the (call-by-value, McCarthy) **A**-recursive partial functions on A

The most important thing missing is an account of implementations and the connection between an implementable algorithm and its implementations. A very little of this has been worked out.

Intrinsic complexities in a structure $\mathbf{A} = (A, \{\phi^{\mathbf{A}}\}_{\phi \in \Phi})$

• Calls complexity. For each algorithm $\alpha : A^n \rightsquigarrow \{\mathrm{tt}, \mathrm{ff}\}_\perp$ of $\mathbf{A}$ and each $\Phi_0 \subseteq \Phi$, we can define

$$\begin{aligned}\mathrm{calls}_\alpha^{\Phi_0}(x) = {} & \text{the number of calls to primitives } \phi^{\mathbf{A}} \text{ with } \phi \in \Phi_0 \\ & \text{made by } \alpha \text{ in the computation of } \overline{\alpha}(x) \quad (\overline{\alpha}(x)\downarrow)\end{aligned}$$

This agrees with the usual calls-complexity for "concrete algorithms"

Theorem

With each relation $R \subseteq A^n$ which is decidable by some algorithm of $\mathbf{A}$ and each $\Phi_0 \subseteq \Phi$, there is a function $\mathrm{calls}_R^{\Phi_0} : A^n \to \mathbb{N}$ such that for every algorithm α of $\mathbf{A}$ which decides R,

$$\mathrm{calls}_R^{\Phi_0}(\vec{x}) \leq \mathrm{calls}_\alpha^{\Phi_0}(\vec{x}) \quad (\vec{x} \in A^N)$$

• The intrinsic calls-complexity $\mathrm{calls}_R^{\Phi_0}$ is usually not trivial (next page)

• There are similar results for a large variety of complexity measures which can be defined for $\mathbf{A}$-algorithms

An example

- Coprimeness, $x \perp\!\!\!\perp y \iff \mathrm{gcd}(x, y) = 1$

Theorem (van den Dries, ynm)

Suppose $\mathbf{A} = (\mathbb{N}, \{\phi^{\mathbf{A}}\}_{\phi \in \Phi})$ *is a structure on* $\mathbb{N}$ *whose primitives are Presburger (piecewise linear) functions and relations. There is a rational* $r > 0$, *such that*

$$\mathrm{calls}^{\Phi}_{\perp\!\!\!\perp}(a, a^2 - 1) \geq r \log(a) \quad (a > 2)$$

In particular, the binary (Stein) algorithm is "suboptimal" up to a multiplicative constant for deciding coprimeness from Presburger primitives.