Recursive algorithms from specified primitives

Yiannis N. Moschovakis
UCLA and University of Athens

Federal University of Ceará, Fortaleza, 24 September, 2018

What is an algorithm?

- **Algorithms are not just Turing machines!** (later)
  and there is no generally accepted definition of what they are

- Many classical notions were defined after centuries of study:
  The natural numbers $\mathbb{N} = \{0, 1, \ldots\}$, ? – 1870s (Dedekind)
  The real numbers $\mathbb{R}$, ? – 1870s (Dedekind, Cantor)
  Random variables, 1700s – 1931 (Kolmogorov)

- A precise definition of a notion should make it possible to give rigorous proofs of results which have been proved intuitively; and so I will start with an elementary discussion of some classical algorithms

- This material is (mostly) from the monograph

  Abstract recursion and intrinsic complexity (ARIC)

  forthcoming in the *Lecture Notes in Logic* series published by the Association for Symbolic Logic and Cambridge University Press

The Euclidean algorithm ε (with division) for $\gcd(x, y)$

- The Division Theorem for $\mathbb{N} = \{0, 1, \ldots\}$: For $x, y \in \mathbb{N}$ with $y > 0$, there are unique numbers $q = \mathrm{iq}(x, y), r = \mathrm{rem}(x, y)$ such that

$$x = yq + r, \quad 0 \leq r < y$$

$$\boxed{\gcd(x, y) = \max\{t \mid \mathrm{rem}(x, t) = \mathrm{rem}(y, t) = 0\}} \quad (x, y \geq 1)$$

- Specification of ε by a `while program`: given $x, y \in \mathbb{N}$:

(ε) `while` $y \neq 0$ $\{x := y;\ \ y := \mathrm{rem}(x, y)\}$ `return` x

▶ **Fact**. *If $y = 0$, then ε returns x, and if $y \neq 0$, then ε returns* $\gcd(x, y)$

- Equivalent specification of ε by a recursive equation:

▶ **Fact**. *The recursive equation (in the function variable p)*

(ε) $\quad p(x, y) = \mathrm{if}\ (y = 0)\ \mathrm{then}\ x\ \mathrm{else}\ p(y, \mathrm{rem}(x, y))$

has a unique (total) solution $\overline{p}(x, y)$, and

$$\overline{p}(x, y) = \mathrm{if}\ (y = 0)\ \mathrm{then}\ x\ \mathrm{else}\ \overline{p}(x, y) = \gcd(x, y)$$

The complexity of the Euclidean

- $\boxed{c_\varepsilon(x,y) = \text{the number of calls to rem that } \varepsilon \text{ makes on the input } x, y}$

  (We do not count calls to $\mathrm{eq}_0(y) \iff y = 0$—we could)

- **Fact**: *If* $x \geq y \geq 2$, *then* $\boxed{c_\varepsilon(x,y) \leq 2\log y \leq 2\log x}$ $\quad (\log = \log_2)$

- **Basic question**: *Is the Euclidean optimal* (in some natural sense), *on some infinite set of inputs?*

- **Main Conjecture:** *For every* $\boxed{\text{algorithm } \alpha \text{ from rem and } \mathrm{eq}_0}$ *which computes* $\gcd(x,y)$ *when* $x, y \geq 1$, *there is a number* $\delta > 0$, *such that for infinitely many pairs* (x,y) *with* $x > y \geq 1$,

$$\boxed{c_\alpha(x,y) = \text{the number of calls } \alpha \text{ makes to rem} \geq \delta \log x}$$

- *The Main Conjecture is not about Turing machines with oracles, which can compute* $\gcd(x,y)$ *with no oracle calls at all*

- **Fact**. *For the Fibonacci numbers* $F_0 = 0,\ F_1 = 1,\ F_{k+2} = F_k + F_{k+1}$,

$$\boxed{c_\varepsilon(F_{k+1}, F_k) = k - 1 \geq (1/2)\varphi \log F_{k+1}} \quad (\varphi = (1/2)(1+\sqrt{5}),\ k \geq 2)$$

Variations of the Euclidean

- **Coprimeness**: $x \perp\!\!\!\perp y \iff x, y \geq 1 \;\&\; \gcd(x, y) = 1$

  $(\varepsilon_{\perp\!\!\!\perp})$ $\texttt{while}\ y \neq 0 \left\{x := y;\ y := \mathrm{rem}(x, y)\right\}$

  $\texttt{return}\Big(\text{if } (x = 1) \text{ then tt else ff}\Big)$

- **Fact**. *If* $x, y \geq 1$, *then* $\varepsilon_{\perp\!\!\!\perp}$ *decides* $x \perp\!\!\!\perp y$

- $\varepsilon_{\perp\!\!\!\perp}$ can also be specified by a system of two recursive equations

- $K(X) =$ the *ring* of polynomials in the indeterminate X over the *field* K

- **Fact**. *The natural version of the Euclidean on* $K(X)$ *computes the* (monic) *polynomial* $\gcd(P(x), Q(X))$

- **Fact**. *The Sturm algorithm is a minor variation of the Euclidean on* $\mathbb{N}, \mathbb{R}$, *and* $\mathbb{R}(X)$ *which, given* $P(X) \in \mathbb{R}(X)$ *and an open interval* (α, β) *in* $\mathbb{R}$, *computes the number of real roots of* $P(X)$ *in* (α, β)

- These algorithms act on complex inputs and use primitives which are not "computable" in any intuitive way—e.g., $\alpha = 0$ on $\mathbb{R}$;
  — their "implementations" for all inputs bring in numerical analysis

The mergesort algorithm

- Suppose L is a set with a fixed (total) ordering $\leq$ on it.
- A string $v = (v_0, \ldots, v_{n-1}) = v_0 v_1 \cdots v_{n-1} \in L^*$ is sorted if $v_0 \leq v_1 \leq \cdots \leq v_{n-1}$, and for each $u \in L^*$,

$$\mathrm{sort}(u) = \text{ the unique sorted "rearrangement" of } u$$

- **Fact**. *The two recursive equations (on L, L^*, with variables p, merge)*

$$p(u) = \text{ if } |u| \leq 1 \text{ then } u \text{ else } \mathrm{merge}(p(\mathrm{half}_1(u)), p(\mathrm{half}_2(u)))$$

$$\mathrm{merge}(w, v) = \text{ if } |w| = 0 \text{ then } v \text{ else if } |v| = 0 \text{ then } w$$
$$\text{else if } (\mathrm{head}(w) \leq \mathrm{head}(v)) \text{ then } \mathrm{cons}(\mathrm{head}(w), \mathrm{merge}(\mathrm{tail}(w), v))$$
$$\text{else } \mathit{cons}(\mathrm{head}(v), \mathrm{merge}(w, \mathrm{tail}(v)))$$

has unique (total) solutions $\overline{p}, \overline{\mathrm{merge}}$, and $\boxed{\mathrm{sort}(u) = \overline{p}(u)}$

- There is a natural, more detailed system of four equations which expresses the mergesort from $\leq$ and the primitives of `Lisp`

$$\mathrm{nil} = (\ ),\ \ \mathrm{eq}_{\mathrm{nil}}(u),\ \ \mathrm{head}(u) = u_0,\ \ \mathrm{tail}(u),\ \ \mathrm{cons}(s, u) = s u_0 \cdots u_{n-1}$$

The complexity of sorting

- There is no single `while` program which expresses faithfully

  the algorithm from the primitives of `Lisp` and $\leq$

  defined by the mergesort

  — because to compute merge$(p(\text{half}_1(u)), p(\text{half}_2(u)))$ you have to decide whether to compute the two values $p(\text{half}_1(u))$ and $p(\text{half}_2(u)))$ together, in parallel or in sequence, in one of two orders

- **Fact** (proved in every class on algorithms). *For the mergesort μ, if*

$$\boxed{c_\mu(u) = \text{ the number of calls to } \leq \text{ that } \mu \text{ makes on the input } u},$$

  then $\boxed{c_\mu(u) \leq |u| \log |u|}$ (Proved "intuitively" by induction on $|u|$)

- **Fact** (proved in every class on complexity). *If α is any sorting algorithm that uses comparisons, then* $\boxed{(\exists u)[c_\alpha(u) \geq |u| \log |u|]}$

- A rigorous formulation and proof can be given if α is specified by a `while` program from $\leq$ and the `Lisp` primitives

Partial functions and (partial) structures

- A partial function $f : X \rightharpoonup W$ is a (total) function $f : D_f \to W$ on some set $D_f \subseteq X$, its domain of convergence
- $$f(x)\downarrow \iff x \in D_f, \quad f(x)\uparrow \iff x \notin D_f,$$
$$f(x) = g(x) \iff [f(x)\uparrow \ \& \ g(x)\uparrow] \vee (\exists w \in W)[f(x) = g(x) = w],$$
$$f \sqsubseteq g \iff (\forall x)[x \in D_f \implies f(x) = g(x)],$$
$$f(g(x), h(x)) = w$$
$$\iff (\exists u, v)[g(x) = u \ \& \ h(x) = v \ \& \ f(u, v) = w]$$
- Unified notation for n-ary partial functions and relations on a set A:

$$\boxed{f : A^n \rightharpoonup A_s} \quad (s \in \{\mathtt{ind}, \mathtt{boole}\}, A_{\mathtt{ind}} = A, A_{\mathtt{boole}} = \{\mathtt{tt}, \mathtt{ff}\})$$

- A vocabulary is a finite set $\Phi = \{\varphi_1, \ldots, \varphi_m\}$ of function symbols, each with an assigned sort s and arity n_i
- A (partial) Φ-structure is a tuple $\boxed{\mathbf{A} = (A, \mathbf{\Phi}) = (A, \varphi_1^{\mathbf{A}}, \ldots, \varphi_m^{\mathbf{A}})}$, where for each i, $\boxed{\varphi_i^{\mathbf{A}} : A^{n_i} \rightharpoonup A_s}$
- (Structures with many sorts (data types) are disjoint unions of these)

Problem: define algorithms of $\mathbf{A} = (A, \mathbf{\Phi})$ so that:

(1) *An algorithm of* $\mathbf{A}$ *computes a partial function* $f : A^n \rightharpoonup A_s$

(2) *The Euclidean(s) and the mergesort are algorithms*

(3) *The* **Facts** *proved intuitively can be proved rigorously and the Main Conjecture can be made precise*

- How do we define (or model faithfully) mathematical objects in set theory?
- Def. A real number is a sequence $x : \mathbb{N} \to \mathbb{Q}$ which has the Cauchy proper

$$\mathbb{R} = \{x : \mathbb{N} \to \mathbb{Q} \mid x \text{ is Cauchy}\};$$

and two real numbers x, y are equal (or equivalent) if "they get arbitrarily close together", i.e.,

$$x \cong y \iff \lim_{n\to\infty} |x(n) - y(n)| = 0$$

- We could take the reals to be the set of equivalence classes of this set $\mathbb{R}$ under $\cong$

Iterative and **recursive** algorithms of a structure **A**

- An algorithm of **A** is iterative if it is specified by a `while` program which is explicitly defined on A from the primitives $\mathbf{\Phi}$ of **A** (known)
- An algorithm of **A** is recursive if it is specified by a recursive program of **A** (needs rigorous definition)
- It can be argued that if we accept **(1)** - **(3)**, then the problem of defining the algorithms of **A** comes down to identifying them with either the iterative or the recursive algorithms of **A**
- The first choice—that

algorithms are (ultimately) specified by `while` programs

  is the standard view: it is (explicitly or implicitly) accepted by almost all computer scientists, including Knuth; it covers all familiar models of computation—Turing machines, . . . , RAMS, . . .); and it has been developed extensively by Gurevich and his co-workers
- — but it does not cover recursive algorithms like the mergesort

Recursive algorithms: two more examples

- Primitive recursion, $\boxed{f(0,y) = g(y), \quad f(x+1,y) = h(f(x),x,y)}$

▶ **Fact**. *f is the least fixed point $\overline{p}$ in $\mathbf{N}_{g,h} = (\mathbb{N}, \mathrm{Pd}, \mathrm{eq}_0, g, h)$ of*

$(*)$ $\boxed{p(x,y) = \text{if } (x = 0) \text{ then } g(y) \text{ else } h(p(\mathrm{Pd}(x),y),\mathrm{Pd}(x),y)}$

- f can be computed by a `while` program of $(\mathbf{N}_{g,h}, S)$, but the recursive algorithm specified by $(*)$ is not obviously the algorithm specified by any `while` program of $\mathbf{N}_{g,h}$

▶ **Fact** (Bezout's Lemma). *For all $x, y \geq 1$, there are $\alpha, \beta \in \mathbb{Z}$ such that*

$(**)$ $$\gcd(x,y) = \alpha x + \beta y;$$

and $(**)$ holds with $\alpha = \overline{p}(x,y), \beta = \overline{q}(x,y)$, the least fixed points of

$$\boxed{\begin{array}{rcl} p(x,y) & = & \text{if } (\mathrm{rem}(x,y) = 0) \text{ then } 0 \text{ else } q(y,\mathrm{rem}(x,y)), \\ q(x,y) & = & \text{if } (\mathrm{rem}(x,y) = 0) \text{ then } 1 \\ & & \text{else } p(y,\mathrm{rem}(x,y)) - \mathrm{iq}(x,y)q(y,\mathrm{rem}(x,y)) \end{array}}$$

Recursive Φ-programs and their fixed-point semantics

- A recursive (McCarthy) Φ-program is a syntactic expression

$$E(\vec{u}) \equiv \underbrace{E_0(\vec{u})}_{\text{head}} \text{ where } \Big\{ \underbrace{\mathsf{p}_1(\vec{x}_1) = E_1(\vec{x}_1), \ldots, \mathsf{p}_K(\vec{x}_K) = E_K(\vec{x}_K)}_{\text{body, all variables in it are bound}} \Big\}$$

which satisfies the following conditions:

(1) $\mathsf{p}_1, \ldots, \mathsf{p}_K$ are distinct (fresh) partial function variables

(2) With $\vec{x}_0 \equiv \vec{u}$, each $E_i(\vec{x}_i)$ is an explicit term in the vocabulary $\Phi \cup \{\mathsf{p}_1, \ldots, \mathsf{p}_K\}$ whose variables are all in the list $\vec{x}_i$

(3) The arities and sorts of the variables p_i and the terms $E_i(\vec{x}_i)$ "fit", so that the equations in the body of E are well-formed

- For a Φ-structure $\mathbf{A}$ and $i \leq K$, we set $\boxed{\alpha_i(\vec{x}_i, \vec{p}) = \mathrm{den}((\mathbf{A}, \vec{p}), E_i(\vec{x}_i))}$
- $\boxed{\mathrm{den}(\mathbf{A}, E(\vec{u})) = \alpha_0(\vec{u}, \overline{p}_1, \ldots, \overline{p}_K)}$ where $\overline{p}_1, \ldots, \overline{p}_K$ are the least fixed points of the system $\boxed{p_i(\vec{x}_i) = \alpha_i(\vec{x}_i, p_1, \ldots, p_K), \quad i = 1, \ldots, K}$

Fixed-point definitions of complexity functions

- Fix a Φ-structure $\mathbf{A}$ and a Φ-program

$$E(\vec{u}) \equiv E_0(\vec{u}) \text{ where } \left\{ p_1(\vec{x}_1) = E_1(\vec{x}_1), \ldots, p_K(\vec{x}_K) = E_K(\vec{x}_K) \right\}$$

- To compute $\mathrm{den}(\mathbf{A}, E(\vec{u}))$ for some $\vec{u} \in A^n$:
  (1) Compute (in any way) the least fixed points $\overline{p}_1, \ldots, \overline{p}_K$ of the system of recursive equations in the body of $E(\vec{u})$;
  (2) Set $\mathrm{den}(\mathbf{A}, E(\vec{u})) = \mathrm{den}(\mathbf{A}^{\overline{p}}, E_0(\vec{u}))$ where $\mathbf{A}^{\overline{p}} = (\mathbf{A}, \overline{p}_1, \ldots, \overline{p}_K)$, is the expansion of $\mathbf{A}$ by these fixed points

▶ *For each* $\mathbf{A}, E(\vec{u}), \vec{u} \in A^n$ *and* $\Phi_0 \subseteq \Phi$, *we can define*

$$(*) \quad c_{E(\vec{u})}(\Phi_0)(\mathbf{A}, \vec{u}) = \text{ the number of calls to } \varphi_i \in \Phi_0 \text{ in the definition of } \mathrm{den}(\mathbf{A}, E(\vec{u}))$$

by a direct least-fixed-point recursion, so $(*)$ *is plausible; it is a theorem if* $E(\vec{u})$ *is* (the recursive representation of) *a* `while` *program; and it validates the proofs above about the mergesort*

- The same can be done about many other complexity functions

Algorithmic equivalence of recursive programs

- Assume we have a reasonable definition of the algorithm expressed by a Φ-program on a Φ-structure and set

  $E(\vec{u}) \cong F(\vec{u}) \iff_{\mathrm{df}} E(\vec{u})$ and $F(\vec{u})$ express the same algorithm in $\mathbf{A}$

  For $\varphi, \psi, c \in \Phi$ of suitable arities and sorts, TRUE or FALSE:

  (1) $\varphi(c) \cong \varphi(p)$ where $\{p = c\}$ TRUE

  (2) $\psi(c, c) \cong \psi(p, p)$ where $\{p = c\}$ FALSE

  (3) $\psi(c, c) \cong \psi(p, q)$ where $\{q = c, p = c\}$ TRUE

  (4) $\psi(p_1(u), p_2(v))$ where $\{p_1(u) = E(u, p_2), p_2(s) = F(s, p_1, p_2)\}$

  $\cong \psi(r_1(u), r_2(v))$ where $\{r_2(t) = F(t, r_1, r_2), r_1(t) = E(t, r_2)\}$ TRUE

- Def. Congruence of programs: $E(\vec{u}) \equiv_c F(\vec{u})$ if F can be constructed from E by renaming the bound variables and reordering the parts in the body

Program reduction

- We define a one-step reduction relation $\boxed{E(\vec{u}) \Rightarrow_1 F(\vec{u})}$ on Φ-programs and set

  $E(\vec{u}) \Rightarrow F(\vec{u}) \iff E(\vec{u}) \equiv_c F(\vec{u})$

  or there exists a sequence $(F_1(\vec{u}), \dots F_k(\vec{u}))$ such that

  $E(\vec{u}) \Rightarrow_1 F_1(\vec{u}) \Rightarrow_1 \cdots \Rightarrow F_k(\vec{u}) \equiv F(\vec{u})]$

- $E(\vec{u})$ is irreducible $\iff (\forall F(\vec{u}))\Big[E(\vec{u}) \Rightarrow F(\vec{u}) \implies E(\vec{u}) \equiv_c F(\vec{u})\Big]$

$\psi(c,d) \Rightarrow_1 \psi(p,d)$ where $\Big\{p = c\Big\} \Rightarrow_1 \psi(p,r)$ where $\Big\{r = d, p = c\Big\}$.

$\psi(c,c) \Rightarrow_1? \psi(p,p)$ where $\Big\{p = c\Big\}$ FALSE

$\psi(c,c) \Rightarrow_1 \psi(p,c)$ where $\Big\{p = c\Big\} \Rightarrow_1 \psi(p,r)$ where $\Big\{r = c, p = c\Big\}$.

$\psi(p)$ where $\Big\{p = \varphi(c)\Big\} \Rightarrow_1 \psi(p)$ where $\Big\{p = \varphi(q), q = c\Big\}$.

- Reduction models (one step at a time, innermost first) compilation

The strict recursive algorithms of a Φ-structure $\mathbf{A}$

- *If* $E(\vec{u}) \Rightarrow F(\vec{u})$, *then for every* Φ*-structure* $\mathbf{A}$:
  (1) $\mathrm{den}(\mathbf{A}, E(\vec{u})) = \mathrm{den}(\mathbf{A}, F(\vec{u}))$
  (2) *For every* $\Phi_0 \subseteq \Pi$, $c_{E(\vec{u})}(\Phi_0)(\mathbf{A}, \vec{u}) = c_{F(\vec{u})}(\Phi_0)(\mathbf{A}, \vec{u})$
  (and the same is true for many other complexity measures)

Canonical Form Theorem
Every recursive Φ*-program* $E(\vec{u})$ *is effectively reducible to a* **unique up to congruence** *irreducible program, its canonical form* $\mathrm{cf}(E(\vec{u}))$

- Def. The strict algorithm (intension) expressed by a Φ-recursive program $E(\vec{u})$ in a Φ-structure $\mathbf{A}$ is $E(\vec{u})$, $\boxed{\mathrm{int}_s(\mathbf{A}, E(\vec{u})) = E(\vec{u})}$
- Def. $E(\vec{u}) \cong_s F(\vec{u}) \iff \mathrm{cf}(E(\vec{u})) \equiv_c \mathrm{cf}(F(\vec{u}))$

- *The identity relation between recursive programs is decidable* (NP) *and at least as difficult as the problem of isomorphism for finite graphs*

The denotational algorithms of a Φ-structure $\mathbf{A}$

- Suppose $\mathbf{A} = (A, \varphi, \psi)$ and $\varphi = \psi$
  If $E(\mathrm{u}) \equiv \varphi(\mathrm{u})$ where $\{\ \}$, then $c_{E(\mathrm{u})}(\varphi)(u) = 1 \neq c_{E(\mathrm{u})}(\psi)(u) = 0$

For $\boxed{E(\vec{\mathrm{u}}) \equiv E_0(\vec{\mathrm{u}}) \text{ where } \left\{\mathrm{p}_1(\vec{\mathrm{x}}_1) = E_1(\vec{\mathrm{x}}_1), \ldots, \mathrm{p}_K(\vec{\mathrm{x}}_K) = E_K(\vec{\mathrm{x}}_K)\right\}}$:

- Def. $\mathfrak{r}(\mathbf{A}, E(\vec{u})) = (\alpha_0, \ldots, \alpha_K)$ with $\alpha_i(\vec{x}_i, \vec{p}) = \mathrm{den}((\mathbf{A}, \vec{p}), E_i(\vec{x}_i))$
- Def. $\mathrm{int}_d(\mathbf{A}, E(\vec{u})) = \mathfrak{r}(\mathbf{A}, (\mathrm{cf}(E(\vec{u}))))$
- Def. $E(\vec{u}) \cong_{d,\mathbf{A}} F(\vec{u}) \iff \mathrm{int}_d(\mathbf{A}, E(\vec{u})) = \mathrm{int}_d(\mathbf{A}, F'(\vec{u}))$
  for some $F' \equiv_c F$

Decidability of identity of denotational algorithms *For every infinite* $\mathbf{A}$*, the relation* $\cong_{d_{\mathbf{A}}}$ *on recursive programs is decidable*

- One of the strict or denotational algorithms of a program may be more appropriate for a particular problem—as for random variables

Comments

- Specification of algorithms by recursive rather than imperative programs simplifies greatly proofs of correctness which come down to showing that the function which we want our algorithm to compute (together with some auxiliary functions) satisfy a system

  This separates the proof of correctness for the algorithm from the proof of correctness of (some) implementation of it—which is surely needed at some time
- The most interesting contribution of this work is the direct definition of complexity measures to recursive problems partly because the lower bound results established for a recursive program hold for all its implementations.