What is an algorithm?

Yiannis N. Moschovakis
UCLA and University of Athens

Chiemsee, July 25 2014

Some publications, all posted in www.math.ucla.edu/∼ynm

On the general theory

- The formal language of recursion (1989)
- A mathematical modeling of pure, recursive algorithms (1989)
- On founding the theory of algorithms (1998)
- What is an algorithm? (2001)
- Elementary algorithms and their implementations, with Vasilis Paschalis (2008)

Applications

- Is the Euclidean algorithm optimal among its peers?, with Lou van den Dries (2004)
- Arithmetic complexity with Lou van den Dries (2009)
- A logical calculus of meaning and synonymy (2006)

Outline

36,500,000 Google hits and no formal definition

- Wikipedia: *An algorithm is a step-by-step procedure for calculations*
- 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 very hard to define as a mathematical object

★ Knuth: *A computational method* [computation model] is ...
An algorithm is a computational method which terminates in finitely many steps for all [inputs]

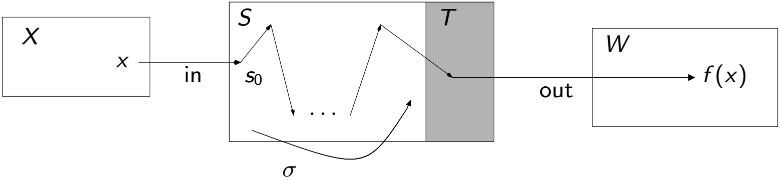
- Girard (and others): An algorithm is expressed by a constructive proof of a statement of the form $(\forall x \in A)(\exists y \in B)P(x, y)$

We need to make precise

- The mathematical structure of algorithms
- The way in which algorithms are effective (constructive, definable)

These two aspects of algorithms are related but separate

Structure: computation models (machines, while programs)



A computation model $\mathfrak{m} : X \rightsquigarrow W$ is a tuple $(S, \mathrm{in}, \sigma, T, \mathrm{out})$ such that

(1) S is a non-empty set (of states)

(2) X is a set and $\mathrm{in} : X \to S$ is the input function

(3) $\sigma : S \to S$ is the transition function

(4) T is the set of terminal states, $T \subseteq S$

(5) W is a set and $\mathrm{out} : T \to W$ is the output function

- $\overline{\mathfrak{m}}(x) = \mathrm{out}(\sigma^n(\mathrm{in}(x)))$ where $n =$ least such that $\sigma^n(\mathrm{in}(x)) \in T$
- $\boxed{s := \mathrm{in}(x); \mathrm{while}(s \notin T)\{s := \sigma(s)\}; \mathrm{return\ out}(s)}$

- $\mathfrak{m}$ computes the partial function $\overline{\mathfrak{m}} : X \rightharpoonup W$

Definability in $\mathbf{M} = (\{M_i\}_{i \in I}, \{\varphi^{\mathbf{M}} \mid \varphi \in \Phi\})$

- Each M_i is a set of sort i including $M_{\texttt{boole}} = \{\mathrm{tt}, \mathrm{ff}\}$
- Each $\varphi \in \Phi$ has a type $(\langle i_1, \ldots, i_{n-1} \rangle, j)$ (with $i_k \neq \texttt{boole}$) and $\varphi^{\mathbf{M}} : M_{i_1} \times \cdots \times M_{i_{n-1}} \rightharpoonup M_j$ is a strict partial function

$\mathbf{M}$ is total if every $\varphi^{\mathbf{M}}$ is total

- A (usual) first-order structure $\mathbf{M} = (M, f_1, \ldots, f_{k-1}, R_1, \ldots, R_{l-1})$, with M and $\{\mathrm{tt}, \mathrm{ff}\}$ as the basic universes and the relations represented by their characteristic functions

- Unary arithmetic, $\mathbf{N}_1 = (\mathbb{N}, 0, S, \mathrm{Pd}, =_0)$

- Binary arithmetic, $\mathbf{N}_2 = (\mathbb{N}, 0, 1, \mathrm{iq}_2, \mathrm{rem}_2, \mathrm{em}_2, \mathrm{om}_2, =_0)$, with $\mathrm{iq}_2(x) = \mathrm{iq}(x, 2), \mathrm{rem}_2(x) = \mathrm{rem}(x, 2), \mathrm{em}_2(x) = 2x, \mathrm{om}_2(x) = 2x + 1$

- An $\mathbf{M}$-machine for a first-order $\mathbf{M} = (M, \{\varphi^{\mathbf{M}} \mid \varphi \in \Phi\})$ is a computation model $(S, \mathrm{in}, \sigma, T, \mathrm{out}) : M^n \rightsquigarrow M_j$ in which $S = M^k$ for some k and $\mathrm{in}, \sigma, T, \mathrm{out}$ are definable by explicit Φ-terms with branching, $\texttt{if } A \texttt{ then } B \texttt{ else } C$

- *Turing machines on k symbols are $\mathbf{N}_k$-machines* (k-ary arithmetic)

The (almost) standard view

★ *Algorithms are computation models*

★ Algorithms from given functions and relations are **M**-machines, where the primitives of **M** include the given functions and relations and some additional absolutely computable operations

E.g., a Turing machine from $R \subset \mathbb{N}^2$ has an oracle for R but operates on the set of strings Σ^* from some alphabet, uses the basic operations on them and assumes a specific representation of numbers by strings (typically unary or binary)

- These principles are implicitly assumed in much of complexity theory and defended (with specific extra operations) by Gurevich and others

- *There is no general agreement on which primitives of computation are absolute*—one of the problems with this view

- I will discuss a broader view, by which algorithms are specified by systems of recursive equations and computation models are implementations of elementary algorithms

The Euclidean algorithm

For $x, y \in \mathbb{N}^+ = \{n \in \mathbb{N} \mid n > 0\}$, with set of states $S = \mathbb{N}^2$

$$(*) \quad \boxed{s := x; t := y; \text{while}(\text{rem}(s,t) \neq 0)[(s,t) := (t, \text{rem}(s,t))]; \text{return } t}$$

where $\text{rem}(s,t)$ is the remainder of the division of s by t,

- $(*)$ defines an $(\mathbb{N}, \text{rem}, =_0)$-machine ε which computes $\text{gcd}(x,y)$

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

Conjecture (open): *For every algorithm α which computes the* gcd *function from* rem *and* $=_0$:

for x, y *with arbitrarily large* $\min(x,y)$,

$$\text{calls}_{\varepsilon}^{\text{rem}}(x,y) \leq \text{calls}_{\alpha}^{\text{rem}}(x,y)$$

- It assumes that "algorithm from rem, $=_0$" and $\text{calls}_{\alpha}^{\text{rem}}$ are defined and is trivial for computation models with more primitives (e.g., TMs)

The extended Euclidean (as a recursive algorithm)

Bezout's Lemma. *There are functions* $\alpha, \beta : \mathbb{N}^+ \times \mathbb{N}^+ \to \mathbb{Z}$ *such that*

$$(*) \quad \text{if } x, y \in \mathbb{N}^+, \text{ then} \quad \gcd(x, y) = \alpha(x, y)x + \beta(x, y)y$$

It is easy to check that $(*)$ holds if $\alpha(x, y), \beta(x, y)$ satisfy the system

$$\begin{aligned}
\alpha(x, y) &= \text{if } (\text{rem}(x, y) = 0) \text{ then } 0 \text{ else } \beta(y, \text{rem}(x, y)),\\
\beta(x, y) &= \text{if } (\text{rem}(x, y) = 0) \text{ then } 1\\
&\qquad\qquad\qquad \text{else } \alpha(y, \text{rem}(x, y)) - \text{iq}(x, y)\beta(y, \text{rem}(x, y))
\end{aligned}$$

where $\text{iq}(x, y)$ is the integer quotient of x by y;

and this expresses a recursive algorithm which computes suitable functions $\alpha, \beta : \mathbb{N}^+ \times \mathbb{N}^+ \to \mathbb{Z}$ from the primitives $0, 1, \text{rem}, \text{iq}, -, =_0$

- The corresponding recursive equation expressing the Euclidean is

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

The color of leaves

A (binary, colored) forest is a structure

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

A *path from* x_0 is any sequence $p = (x_0, \ldots)$ of length $|p| \leq \infty$ s.t.

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

- **F** is grounded if it has no infinite paths, and on such **F** we set

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

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

- $(*)$ expresses a recursive algorithm ρ which decides $R(x)$ on **F** and there are many such divide-and-conquer algorithms (e.g., the mergesort)

- (Tiuryn 1989) *On some grounded forest, no algorithm expressed by an* **F***-machine decides* $R(x)$

★ ρ can be implemented by many machines on $F' \supset F$ with more primitives

The sieve of Eratosthenes

Primes $= p(u_0)$ where

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

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

- A system of recursive equations which expresses an algorithm σ on S from $\mathrm{head}, \mathrm{tail}, \mid, \frown$ and (the act) Print
- $\mathit{sieve}(x, v)$ removes from v all numbers divisible by x
- $p(u)$ prints $\mathrm{head}(u)$ and then calls itself on $\mathit{sieve}(\mathrm{head}(u), \mathrm{tail}(u))$

- 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

- σ operates on completed infinite objects and never terminates

• *The basic notion is that of algorithm from primitives* with a very broad understanding of "primitives"

• Problem: too many notions are associated with an algorithm: calls to the primitives, recursive definitions, complexity functions, termination, side effects (and interaction, which is more complex), simulation, implementability, . . .

For specific algorithms many of these are simple and naturally defined, 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)

• We look for a similar solution, which takes the basic notions of the theory of algorithms from an existing mathematical theory

• Claim: *For deterministic algorithms, the background theory is least fixed point recursion on complete posets*

Basic poset theory, I

- A poset is a pair $(X, \leq_X)$ where $\leq_X$ is a partial ordering of X
- A poset D is (directed- or chain-) complete if every linearly ordered subset (chain) $C \subseteq D$ has a least upper bound $\sup(C)$.

Every complete poset has a least element, $\sup(\emptyset) = \perp$

- A poset D is complete if and only if every directed subset $C \subseteq D$ has a least upper bound. (The proof requires the Axiom of Choice)
- A set X can be viewed as a discrete poset or represented by the complete flat poset $X_\perp = X \cup \{\perp\}$, where

$$s \leq_{X_\perp} t \iff s = \perp \vee s = t$$

- The (naturally defined, cartesian) product $D_1 \times \cdots \times D_n$ of complete posets is complete

Basic poset theory, II

- A function $f : X \to W$ is monotone if

$$x \leq_X y \implies f(x) \leq_W f(y),$$

and strict if $f(x) \neq \perp \implies x$ is total (maximal) in X

- A function $f : X \to W$ is (Scott) continuous if

$$\sup_X(C) = \overline{x} \implies \sup_W\{f(x) \mid x \in C\} = f(\overline{x})$$

for every chain $C \subseteq X$ or (equivalently) for every directed subset $C \subseteq X$

- $\mathrm{Strict}(X, W), \mathrm{Cont}(X, W)$ and $\mathrm{Mon}(X, W)$ are complete if W is complete, and

$$\mathrm{Strict}(X, W) \subseteq \mathrm{Cont}(X, W) \subseteq \mathrm{Mon}(X, W)$$

- For sets X, Y, $\mathrm{Strict}(X_\perp, Y_\perp) \cong (X \rightharpoonup Y)$
  $=$ the poset of all partial functions on X to Y ordered under inclusion

Least fixed point recursion

Theorem (Least Fixed Point Theorem, LFP, 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

- The proof is "constructive", i.e., $\overline{d}$ is built up by iterating f,

$$\overline{d}^0 = f(\perp), \overline{d}^1 = f(\overline{d}^0), \ldots, \overline{d} = \sup_{\xi \in \mathrm{Ords}} \overline{d}_\xi$$

- In many applications, $D = D_1 \times \cdots \times D_k$ is a product poset

Definition by mutual recursion

The examples we gave were all definitions of the form

$$(*) \quad f(x) = f_0(x,p) \text{ where } \Big\{ p_1(u_1) = f_1(u_1,x,p), \dots, p_k(u_k) = f_k(u_k,x,p) \Big\}$$

$$= f_0(x,p) \text{ where } \Big\{ p_1 = \lambda(u_1) f_1(u_1,x,p), \dots, p_k = \lambda(u_k) f_k(u_k,x,p) \Big\}$$

with $p = (p_1, \dots, p_k)$;

the value $f(x)$ is determined by computing the least solution tuple $\overline{p}_x = (\overline{p}_{1,x}, \dots, \overline{p}_{k,x})$ of the system within the braces and then setting

$$f(x) = f_0(\overline{p}_x) = f_0(x, \overline{p}_{1,x}, \dots, \overline{p}_{k,x})$$

- We argued in each case that $(*)$ expresses an algorithm for computing f from $f_0, f_1, \dots, f_k$

- Key idea: *An algorithm is*
the semantic content of a definition by simultaneous recursion

⋆ (Monotone) recursors $\alpha = (\alpha_0, \ldots, \alpha_k) : X \rightsquigarrow W$

- A recursor $\alpha : X \rightsquigarrow W$ on a poset X to a complete poset W 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 head part $\alpha_0 : X \times D_1 \times \cdots \times D_k \to W$ is also monotone

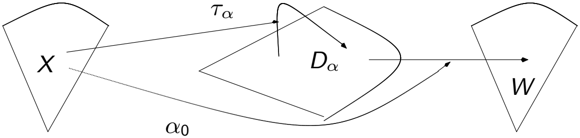
- $(\alpha_1, \ldots, \alpha_k)$ is the body of α;
$D_\alpha = D_1 \times \cdots \times D_k$ is its solution space;
and its transition mapping $\tau_\alpha : X \times D_\alpha \to D_\alpha$ is

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

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

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

⋆ (Monotone) recursors $\alpha = (\alpha_0, \dots, \alpha_k) : X \rightsquigarrow W$



(1) X, W are posets and W is complete

(2) The solution space $D_\alpha = D_1 \times \cdots \times D_\alpha$ of α is the (complete) product of complete posets $D_1, \dots, D_k$

(3) Each part $\alpha_i : X \times D_\alpha \to D_i$ $(i = 1, \dots, k)$ is monotone

(4) The head $\alpha_0 : X \times D_\alpha \to W$ is monotone

(5) The transition map $\tau_\alpha : X \times D_\alpha \to D_\alpha$ is given by $\tau_\alpha(x, d) = (\alpha_1(x, d), \dots, \alpha_k(x, d))$

- 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 space

$\alpha(x) = \alpha_0(x, d)$ 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 (with a head) is

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

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

- If p varies over $\text{Strict}(\mathbb{N}_\perp^2, \mathbb{N}_\perp) \cong (\mathbb{N}^2 \rightharpoonup \mathbb{N})$ (call by value), then

$$\overline{\alpha}(s,t) = \overline{p}(s,t) = \text{if } (s = 0) \text{ then } 0 \text{ else } \perp \quad (s, t \in \mathbb{N})$$

- If p varies over $\text{Cont}(\mathbb{N}_\perp \times \mathbb{N}_\perp, \mathbb{N}_\perp)$ (call by name), then

$$\overline{\alpha}(s,t) = \overline{p}(s,t) = 0 \quad (s, t \in \mathbb{N})$$

- A variant of the sieve of Eratosthenes can be expressed by a continuous recursor on the poset $\text{Streams}(\mathbb{N}, \mathbb{N})$ of all streams on $\mathbb{N}$

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

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

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

Operations on recursors, I

- Trivial recursors. Each monotone function $f : X \to W$ can be viewed as a degenerate recursor $\delta_f = (f)$ 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, define $\alpha : X \rightsquigarrow W$ by

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

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

Operations on recursors, II

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

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

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

- *Composition with a function is not the same as composition with the trivial recursor representing it*: e.g., if $\beta = \delta_f$ and $\gamma = \delta_g$,

$$\beta(g(x)) = \delta_f(g(x)) = f(g(x)) \text{ where } \{\,\}$$
$$\not\cong \beta(\delta_g(x)) = \delta_f(\delta_g(x)) = f(v) \text{ where } \{v = g(x)\}$$

(Both of these recursors compute the same function $x \mapsto f(g(x))$)

Operations on recursors, III

- Recursor combination. For given recursors $\beta^0, \dots, \beta^k$, put

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

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

- This operation combines in parallel $k+1$ recursive definitions
- Application: Proof of Kleene's First Recursion Theorem

Operations on recursors, IV, λ-substitution

Given $\gamma : X \times U \rightsquigarrow V$, $\beta : P \times Y \rightsquigarrow W$ with $P \subseteq \mathrm{Mon}(U, V)$ complete, we want to define

$$(*) \qquad \alpha(x, y) = \beta(\lambda u \gamma(x, u), y)$$

so that

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

The obvious necessary hypothesis is that

$$\text{for all } x \in X,\ \lambda u \overline{\gamma}(x, u) \in P$$

and when this holds, we set (with p ranging over P)

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

which insures $(**)$

- Typically $P = \mathrm{Strict}(X, W)$ in first order or higher type recursion

The recursor representation of computation models

If $\mathfrak{m} : X \rightsquigarrow W$ is a computation model expressed by the program

$$s := \mathrm{in}(x); \mathrm{while}(s \notin T)\{s := \sigma(s)\}; \mathrm{return\ out}(s),$$

we associate with $\mathfrak{m}$ the tail recursor

$$\boxed{\alpha_{\mathfrak{m}}(x) = p(\mathrm{in}(x)) \text{ where } \{p(s) = \text{if } (s \in T) \text{ then out}(s) \text{ else } p(\sigma(s))\}}$$

where p ranges over the poset of strict partial functions $(S \rightharpoonup W)$

Theorem (ynm, Paschalis)

For any two computation models $\mathfrak{m}, \mathfrak{m}' : X \rightsquigarrow W$ and with the natural notion of machine isomorphism,

$$\mathfrak{m} \cong \mathfrak{m}' \iff \alpha_{\mathfrak{m}} \cong \alpha_{\mathfrak{m}'}$$

- This result (with the specific definition of $\alpha_{\mathfrak{m}}$) insure that the recursor representation $\alpha_{\mathfrak{m}}$ of a computation model codes faithfully all the combinatorial and complexity properties of $\mathfrak{m}$

Implementations (sketch)

- For $\alpha, \beta : X \rightsquigarrow W$, a reduction (or direct simulation) of α to β is any monotone $\pi : X \times D_\alpha \to D_\beta$ such that for all $x, \in X, d \in D_\alpha$,

(R1) $\tau_\beta(x, \pi(x, d)) \leq_{D_\beta} \pi(x, \tau_\alpha(x, d))$

(R2) $\beta_0(x, \pi(x, d)) \leq_W \alpha_0(x, d)$

(R3) $\overline{\alpha}(x) = \overline{\beta}(x)$

- $\alpha \leq_r \beta \iff$ *there is a reduction* $\pi : X \times D_\alpha \to D_\beta$

- An implementation of $\alpha : X \rightsquigarrow W$ is any computation model $\mathfrak{m} : X \rightsquigarrow W$ such that $\alpha \leq_r \alpha_{\mathfrak{m}}$, and α is implementable if it has an implementation

- Most standard "implementations" of recursive algorithms satisfy this definition, but very little beyond this is known about this notion which is central to our

★ Main view: *Algorithms are recursors and computation models implement them*—when they are implementable

⋆ Algorithms are recursors from given primitives

$$\alpha(x) = \alpha_0(x, d) \text{ where } \{d_1 = \alpha_1(x, d), \ldots, d_k = \alpha_k(x, d)\}$$

is an algorithm from $(\alpha_0, \alpha_1, \ldots, \alpha_k)$; obvious but not very useful

- Example. Duplication on $\mathbb{N}$ from $0, S, \mathrm{Pd}$ and $=_0$:

$$\alpha(x) = p(x, x) \text{ where } \Big\{p(x, y) = \boxed{\text{if } (y = 0) \text{ then } x \text{ else } S(p(x, \mathrm{Pd}(y))} \Big\}$$

$$\overline{p}(x, y) = x + y, \quad \overline{\alpha}(x) = \overline{p}(x, x) = 2x$$

- α is an algorithm from the function defined in the box

$$\beta(x) = p(x, x) \text{ where } \Big\{p(x, y) = \text{if } q_1(x, y) \text{ then } x \text{ else } q_2(x, y),$$
$$q_1(x, y) = \chi_{=_0}(y), \quad q_2(x, y) = S(q_3(x, y)),$$
$$q_3(x, y) = p(x, q_4(x, y)), \quad q_4(x, y) = \mathrm{Pd}(y)\Big\}$$

- Again $\overline{\beta}(x) = 2x$, and β is an algorithm from $0, S, \mathrm{Pd}, =_0$, because *its parts are direct calls to these primitives and the conditional*

$$\alpha = (\alpha_0, \alpha_1, \ldots, \alpha_k) : X \rightsquigarrow W$$

$$\alpha(x) = \alpha_0(x, d) \text{ where } \{d_1 = \alpha_1(x, d), \ldots, d_k = \alpha_k(x, d)\}$$

- What does it mean to say that

$(*)$ α is from $\mathbf{\Phi}$

  - $\mathbf{\Phi}$ must include the complete posets in the solution space $D = D_1 \times \cdots \times D_k$ of α
    (to distinguish e.g., call-by-value from call-by-name)
  - $\mathbf{\Phi}$ may include "given" functions and operations on various sets and posets (like $0, S, \mathrm{Pd}, =_0$ in the example)
  - The conditional and recursive calls should be allowed "for free", together (perhaps) with other logical operations

- A definition of $(*)$ in the most general case is difficult to formulate (and in any case I don't have one I like)

- I will develop here the simplest and most useful case of first-order primitives and then discuss some of its extensions

The term language $\mathsf{L}(\Phi)$, $\mathbf{M} = (\{M_i\}_{i \in I}, \{\varphi^{\mathbf{M}} \mid \varphi \in \Phi\})$

- Individual variables $v_0^i, v_1^i, \ldots$ for each sort $i \in I$
- Function variables $p_0^s, p_1^s, \ldots$ for each type $s = (\langle i_1, \ldots, i_{n-1}\rangle, j)$
- Constants φ for each function symbol $\varphi \in \Phi$

- Terms and their sorts and free and bound variable occurrences are defined recursively, subject to the natural restrictions:

$$\begin{aligned} A :\equiv \mathrm{tt} \mid \mathrm{ff} \mid v \mid p(A_1, \ldots, A_n) \mid \varphi(A_1, \ldots, A_n) \\ \mid \texttt{if } A \texttt{ then } B \texttt{ else } C \\ \mid A_0 \texttt{ where } \{p_1(u_1) = A_1, \ldots, p_k(u_k) = A_k\} \end{aligned}$$

- In the recursive term $A \equiv A_0 \texttt{ where } \{p_1(u_1) = A_1, \ldots, p_k(u_k) = A_k\}$:
  - Each u_i is a list of individual variables which are bound in all their occurrences in the equation $p_i(u_i) = A_i$ (equivalent to $p_i = \lambda u_i A_i$)
  - Each function variable p_i is bound in all its occurrences in A
  - $\mathrm{sort}(A) = \mathrm{sort}(A_0)$

Denotational semantics of $\mathsf{L}(\Phi)$, $\mathbf{M} = (\{M_i\}_{i \in I}, \{\varphi^{\mathbf{M}} \mid \varphi \in \Phi\})$

$$
\begin{aligned}
A :\equiv\ & \mathrm{tt} \mid \mathrm{ff} \mid v \mid p(A_1, \ldots, A_n) \mid \varphi(A_1, \ldots, A_n) \\
& \mid \texttt{if } A \texttt{ then } B \texttt{ else } C \\
& \mid A_0 \texttt{ where } \{p_1(u_1) = A_1, \ldots, p_k(u_k) = A_k\}
\end{aligned}
$$

- If $\mathrm{type}(p) = (\langle i_1, \ldots, i_{n-1}\rangle, j)$, then $p : M_{i_1} \times \cdots \times M_{i_{n-1}} \rightharpoonup M_j$
- For each term A, each sequence x of distinct individual and function variables which includes all the free variables of A, and each sequence x of objects of $\mathbf{M}$ with matching sorts and types

$$\boxed{\mathrm{den}(A)\{\mathsf{x} := x\} = \text{the denotation of } A \text{ when } \mathsf{x} = x}$$

- Standard definition, using least-fixed-points for recursion
- A partial function or functional $f : X \rightharpoonup M_j$ is recursive in $\mathbf{M}$ if

$$f(x) = \mathrm{den}(A)\{\mathsf{x} := x\} \quad \text{for some term } A \text{ and variables } \mathsf{x}$$

- $\mathrm{rec}(\mathbf{N}_1) = \mathrm{rec}(\mathbf{N}_2) =$the classical recursive partial functionals on $\mathbb{N}$
- John McCarthy's elegant, deterministic definition of recursion on $\mathbb{N}$

Elementary algorithms, $\mathbf{M} = (\{M_i\}_{i \in I}, \{\varphi^{\mathbf{M}} \mid \varphi \in \Phi\})$

$$
\begin{aligned}
A :\equiv \mathrm{tt} \mid \mathrm{ff} \mid v \mid p(A_1, \ldots, A_n) \mid \varphi(A_1, \ldots, A_n) \\
\mid \texttt{if } A \texttt{ then } B \texttt{ else } C \\
\mid A_0 \texttt{ where } \{p_1(u_1) = A_1, \ldots, p_k(u_k) = A_k\}
\end{aligned}
$$

- Immediate: $Z :\equiv u \mid p(u_1, \ldots, u_m)$
- Exp-irreducible: $T :\equiv Z \mid \mathrm{tt} \mid \mathrm{ff} \mid \varphi(Z_1, \ldots, Z_n) \mid \text{if } Z_1 \text{ then } Z_2 \text{ else } Z_3$
- An **M**-algorithm or algorithm from $\{\varphi^{\mathbf{M}} \mid \varphi \in \Phi\}$ is any recursor

$$
(*) \quad \alpha(x) = \alpha_0(x, \vec{p}) \text{ where } \Big\{p_1(u_1) = \alpha_1(u_1, x, \vec{p}), \ldots, p_k(u_k) = \alpha_k(u_k, x, \vec{p})\Big\}
$$

in which every α_i is defined in **M** by an explicit irreducible term

- Key idea: the (absolute) primitives of first-order computation are
  - the constants tt, ff,
  - random access to function variables $p(u_1, \ldots, u_m)$,
  - calls to the given primitives and the conditional,
  - mutual recursion

Referential intensions, $\mathbf{M} = (\{M_i\}_{i \in I}, \{\varphi^{\mathbf{M}} \mid \varphi \in \Phi\})$

$$
\begin{aligned}
A :\equiv \mathrm{tt} \mid \mathrm{ff} \mid v \mid p(A_1, \ldots, A_n) \mid \varphi(A_1, \ldots, A_n) \\
\mid \texttt{if } A \texttt{ then } B \texttt{ else } C \\
\mid A_0 \texttt{ where } \{p_1(u_1) = A_1, \ldots, p_k(u_k) = A_k\}
\end{aligned}
$$

- With each term A and list x of distinct variables which includes all the free variables of A, we can associate its referential intension

$$
\mathrm{int}(A)(x) = \alpha_0(x, \vec{p}) \text{ where } \Big\{p_1(u_1) = \alpha_1(u_1, x, \vec{p}), \ldots, p_k(u_k) = \alpha_k(u_k, x, \vec{p})\Big\},
$$

an algorithm of $\mathbf{M}$ which computes $\mathrm{den}(A)\{x := x\}$

- This is done by a (careful) recursion on A, which takes into account which subterms of A are immediate
  - Immediate terms $Z \equiv u \mid p(u_1, \ldots u_m)$ are assigned functions, not recursors; they denote immediately
  - Exp-irreducible terms $Z \mid \mathrm{tt} \mid \mathrm{ff} \mid \varphi(Z_1, \ldots, Z_n) \mid \text{if } Z_1 \text{ then } Z_2 \text{ else } Z_3$ are assigned trivial recursors; they denote directly

$$\mathbf{M} = (M, \{\varphi^{\mathbf{M}} \mid \varphi \in \Phi\}), \quad \mathrm{alg}(\mathbf{M}) = \text{the algorithms of } \mathbf{M}$$

- The algorithms of **M** compute the **M**-recursive partial functions and relations on M
- They can be easily specified using Φ-recursive programs, i.e., terms of $\mathrm{L}(\Phi)$
- alg(**M**) is closed under many operations on recursors, including composition and recursion combination
- There is a small, interesting collection of facts about these objects

★ The basic definitions and results can be extended easily to structures whose universes are the sets in the higher types over given basic sets $\{M_i\}_{i \in I}$ and whose primitives are higher type objects of various kinds
Basic example: the Gentzen cut elimination algorithm on extensions of arithmetic with the ω-rule (cf. Schwichtenberg's article in the Handbook of Logic)

Applications to complexity in arithmetic

• Calls complexity. For each algorithm $\alpha : M^n \rightsquigarrow M_j$ of a structure $\mathbf{M} = (M, \{\varphi^{\mathbf{M}} \mid \varphi \in \Phi\})$ and each $\Phi_0 \subseteq \Phi$, we can define

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

This agrees with the usual calls-complexity for "concrete algorithms" defined by computation models

• Many more "natural" complexity functions including $\mathrm{time}_\alpha(x)$ and

$$\begin{aligned}\mathrm{size}_\alpha(x) = {} & \text{the size of the smallest set } M_x \subseteq M\\ & \text{that } \alpha \text{ must see to compute } \overline{\alpha}(x) \le \mathrm{calls}_\alpha(x) = \mathrm{calls}_\alpha^{\Phi}(x)\end{aligned}$$

• Obvious: $(\mathbb{N}, 0, S, \mathrm{Pd}, =_0) = \mathrm{rec}(\mathbb{N}, 0, 1, \mathrm{iq}_2, \mathrm{rem}_2, \mathrm{em}_2, \mathrm{om}_2, =_0)$ but $\mathrm{alg}(\mathbb{N}, 0, S, \mathrm{Pd}, =_0) \neq \mathrm{alg}(\mathbb{N}, 0, 1, \mathrm{iq}_2, \mathrm{rem}_2, \mathrm{em}_2, \mathrm{om}_2, =_0)$

• *Complexity theory for elementary algorithms depends heavily and essentially on the primitives included in the structure*

The weak optimality of Stein's algorithm for coprimeness

x is coprime with $y \iff \gcd(x,y) = 1 \quad (x, y \in \mathbb{N}^+)$

- The structure of Stein: $\mathbf{N}_s = (\mathbb{N}, 0, 1, +, \dot{-}, \mathrm{iq}_2, \mathrm{rem}_2, <, =)$

The primitives are Presburger (piecewise linear) functions and relations

- Stein's algorithm $\sigma : \mathbb{N}^+ \times \mathbb{N}^+ \rightsquigarrow \mathbb{N}$ computes $\gcd(x,y)$ and so decides coprimeness in $\mathbf{N}_s$ with $\mathrm{calls}_\sigma(x,y) \le C \log \max(x,y)$

Theorem (van den Dries, ynm, 2004, 2009)

If an algorithm α from finitely many Presburger primitives decides coprimeness on $\mathbb{N}$, then for some $r > 0$ and all $a > 2$,

$$\mathrm{calls}_\alpha(a, a^2 - 1) > r \log_2(a^2 - 1)$$

It follows that Stein's algorithm is optimal (up to a multiplicative constant) from Presburger primitives on infinitely many inputs

- Similar lower bounds for Presburger algorithms hold for *primality*, *being a perfect square*, *having no square factors* and several more relations in arithmetic and algebra

A lower bound for algorithms from division with remainder

Theorem (van den Dries, ynm, 2004, 2009)

Suppose α *is an algorithm from finitely many Presburger primitives,* iq *and* rem, *and* $\xi > 1$ *is a quadratic irrational; there is some* $r > 0$ *such that for all sufficiently large coprime* $a, b \in \mathbb{N}$

$$\left|\xi - \frac{a}{b}\right| < \frac{1}{b^2} \implies \mathrm{calls}_\alpha(a, b) > r \log_2 \log_2 a$$

In particular, this holds

- *for positive Pell pairs* (a, b) *satisfying* $a^2 = 2b^2 + 1 \quad (\xi = \sqrt{2})$
- *For Fibonacci pairs* (F_{k+1}, F_k) *with* $k \geq 3 \quad (\xi = \frac{1}{2}(1 + \sqrt{5}))$

- This is one log short of the conjecture about the Euclidean and the method of proof cannot prove it (Vaughan Pratt)

- The discovery of the results in this and the preceding slide depend essentially on the notion of elementary recursive algorithm

Frege's sense and denotation as algorithm and value

- $1 + 1 = 2$ vs. *there are infinitely many primes*
  Same denotation (truth value) but different meaning

- Frege: *Each closed well-formed "term" (including every sentence) has a sense (meaning) which determines its denotation*,

$$\boxed{A \mapsto \mathrm{sense}(A) \mapsto \mathrm{den}(A)}$$

- [The sense of a sign]
– *may be the common property of many people*,
– *is grasped by everyone who is sufficiently familiar with the language*,
– [is preserved by faithful translation]

- *The sense contains the mode of presentation of the denotation*

- Common view: sense(A) is a "process" which computes den(A)

- With a precise notion of (abstract) "algorithm" replacing "process", it is possible to turn this view into an interesting mathematical theory of meaning and synonymy

Adding meaning to Montague semantics

- Language: The *typed λ-calculus with acyclic recursion* $\mathsf{L}^{\lambda}_{\mathsf{ar}}$, an extension of Richard Montague's *language of intensional logic* which can "express" substantial fragments of natural language

- Interpretation: In every *higher type structure* **M** over basic sets of *entities*, *truth values* and *states*, each *closed term* A of $\mathsf{L}^{\lambda}_{\mathsf{ar}}$ is assigned

$$\boxed{\text{a value } \mathrm{den}(A) \quad \text{and} \quad \text{a referential intension } \mathrm{int}(A)}$$

$\mathrm{int}(A)$ is an **M**-algorithm, the meaning of A, and it computes $\mathrm{den}(A)$

- $\mathrm{int}(A)$ and the synonymy relation $\mathrm{int}(A) \cong \mathrm{int}(B)$ are definable in $\mathsf{L}^{\lambda}_{\mathsf{ar}}$

- The theory yields defensible accounts of some standard puzzles about global and local synonymy in the philosophy of language, e.g.,

$\boxed{\text{he is the thief} \not\cong_a \text{George is the thief}}$ in a state a where $\mathrm{he}(a) = \mathrm{George}$

- It can express quantification and anaphora into propositional attitudes

George claims that someone stole his laptop

Machines vs. recursors on total $\mathbf{M} = (\{M_i\}_{i \in I}, \{\varphi^{\mathbf{M}} \mid \varphi \in \Phi\})$

- An **M**-machine is a computation model $(S, \mathrm{in}, \sigma, T, \mathrm{out}) : M^n \rightsquigarrow M_j$ in which $S = M^k$ for some k and $\mathrm{in}, \sigma, T, \mathrm{out}$ are definable by explicit Φ-terms

- Tiuryn's Theorem: *There is a structure* **M** *and an* **M***-recursive relation which cannot be decided by any machine of* **M**

- Fact: *Every algorithm α of a structure* **M** *can be implemented by a machine of some* $\mathbf{M}^*$*, an expansion of an extension of* **M**

- Typically M^* is the set of all finite sequences from the M_i's and the additional primitives manipulate these strings—but there are many choices for $\mathbf{M}^*$ and no principled way to choose one among them

- Divide-and-conquer recursive algorithms (such as that of Tiuryn) include the mergesort, the (finitary) Gentzen cut-elimination algorithm, etc. Most (arguably all) their important properties can be derived directly from their recursive specifications

What are the absolute primitives of elementary algorithms?

- In modeling elementary algorithms by recursors, we assumed "for free"

(1) random access to function variables,
(2) calls to the primitives,
(3) branching, and
(4) mutual recursion

- In defining computation models which do not have function variables, (1) is typically replaced by assignments

$$x := A \quad \text{(with an explicit term } A\text{)}$$

- *Random Access Machines* (RAMs) over $\mathbf{M} = (M, \{\varphi^{\mathbf{M}} \mid \varphi \in \Phi\})$ allow a partial function $F : M \rightharpoonup M$ and assignments $F(a) := b$

- *If* $0, 1$ *and* $=_0$ *are among the primitives of* $\mathbf{M}$ *for some* $0, 1 \in M$, *then equality on* M *is decided by the* RAM *program*

$$F(x) := 1; F(y) := 0; \text{if } (F(x) = 0) \text{ return tt else return ff}$$

- This can be managed for $\mathbf{N}_1$ or $\mathbf{N}_2$, but not in the abstract case