The fixed point theory of complexity

Yiannis N. Moschovakis

Department of Mathematics University of California, Los Angeles, CA, USA
and Department of Mathematics, University of Athens, Greece
`ynm@math.ucla.edu`

Starting with [2], Scott and his students and followers developed a general method which assigns to every program E its *denotation* $\mathrm{den}(E)$, the object computed by E, typically a function of some sort. The method uses *least-fixed-point theorems* in various complete posets and has evolved into a rich mathematical theory.

Denotational semantics are useful because they provide a precise *criterion of correctness* for a program E, which should compute what we wanted it to compute; but $\mathrm{den}(E)$ tells us nothing about the *complexity* of E, which is what we want to know next.

As it turns out, Scott's fixed-point-methods can be easily extended to extract from a program E many of its intensional properties, including some natural, implementation-independent *measures of complexity* of E. We will focus on programs which compute functions (or decide relations) on a set A, for which complexity theory is most developed; and for these, the two, key moves that are needed are to identify *the primitives* (functions and relations on A) that E uses as oracles and to translate E into a *recursive* (McCarthy) *program* from these primitives, which can be done using routine, well-understood methods.

1 Partial functions and partial structures

A *partial function* $f : X \rightharpoonup W$ is a (total) function $f : D_f \to W$ on some $D_f \subseteq X = \{x : f(x)\downarrow\}$, its *domain of convergence*, and $f \sqsubseteq g \iff (\forall x \in X)[f(x)\downarrow \implies f(x) = g(x)]$.

A *vocabulary* is a finite set Φ of function symbols, each with a specified *arity* $n_\phi = 0, 1, \ldots$, and *sort* $s_\phi \in \{\mathtt{ind}, \mathtt{boole}\}$; and a (partial) Φ-*structure* is a pair

$$\mathbf{A} = (A, \Phi^{\mathbf{A}}) = (A, \{\phi^{\mathbf{A}}\}_{\phi\in\Phi}),$$

where $\boxed{\phi^{\mathbf{A}} : A^{n_\phi} \rightharpoonup A_{s_\phi}}$ with $A_{\mathtt{ind}} = A$ and $A_{\mathtt{boole}} = \mathbb{B} = \{\mathrm{tt}, \mathrm{ff}\}$. For example,

$$\mathbf{N} = (\mathbb{N}, 0, 1, +, \cdot, =) \text{ and } \mathbf{N}_u = (\mathbb{N}, 0, 1, S, \mathrm{Pd}, \mathrm{eq}_0)$$

are the *standard* and the *unary* structures on $\mathbb{N} = \{0, 1, \ldots\}$.

2 Explicit Φ-terms, A-terms and functionals

The explicit **A**-terms (with pf variables) are defined and assigned sorts by *structural recursion*,

$$E :\equiv \mathrm{tt} \mid \mathrm{ff} \mid x\ (x \in A) \mid \mathsf{v}_i \mid \mathsf{q}_i^{s,n}(E_1, \ldots, E_n) \mid \phi(E_1, \ldots, E_{n_\phi}) \mid \text{if } E_0 \text{ then } E_1 \text{ else } E_2,$$

where $\mathsf{v}_0, \mathsf{v}_1, \ldots$ is a fixed sequence of formal individual variables (of sort $\mathtt{ind}$); for each sort $s \in \{\mathtt{ind}, \mathtt{boole}\}$ and each n, $\mathsf{q}_0^{s,n}, \mathsf{q}_1^{s,n}, \ldots$ is a fixed sequence of formal partial function (pf)

The results in this paper are from Part I of [1].

variables of sort s and arity n; and we assume the natural restrictions on arities and sorts so that the clauses of the definition make sense. E is a Φ-*term* if no constants from A occur in it.

Explicit $\mathbf{A}$-terms are interpreted in expansions $(\mathbf{A}, p_1, \dots, p_K)$ of $\mathbf{A}$ by partial functions $p_1, \dots, p_K$ whose sorts and arities match those of the pf variables $\mathsf{p}_1, \dots, \mathsf{p}_K$ which occur in them; and if all the individual variables which occur in E are in the list $\vec{\mathsf{x}} = (\mathsf{x}_1, \dots, \mathsf{x}_n)$ and $\vec{x} = (x_1, \dots, x_n) \in A^n$, we set[1]

$$\mathrm{den}((\mathbf{A}, \vec{p}), E(\vec{x})) := \text{ the value of } E(\vec{x}) \text{ with } \vec{p} = (\overline{p}_1, \dots, \overline{p}_K)$$

by the usual recursion on the definition of terms, so $\mathrm{den}((\mathbf{A}, \vec{p}), E(\vec{x})) \in A_{\mathrm{sort}(E)}$.

A *functional* $f(\vec{x}, \vec{p})$ on A is *explicit* in $\mathbf{A}$ if for some $\mathbf{A}$-explicit term E,

$$f(\vec{x}, \vec{p}) = \mathrm{den}((\mathbf{A}, \vec{p}), E(\vec{x})) \quad (\vec{x} \in A^n).$$

3 Recursive (McCarthy) programs

A *recursive* Φ*-program* is a syntactic expression

$$E \equiv E_0 \ \mathsf{where} \ \Big\{ \mathsf{p}_1(\vec{\mathsf{x}}_1) = E_1 \quad \dots, \quad \mathsf{p}_K(\vec{\mathsf{x}}_K) = E_K \Big\} \qquad (\Phi\text{-programs})$$

where each E_i is an explicit Φ-term whose pf variables are in the list $\mathsf{p}_1, \dots, \mathsf{p}_K$ (the *recursive variables* of E), and for $i = 1, \dots, K$, $\mathrm{sort}(\mathsf{p}_i) = \mathrm{sort}(E_i)$ and the individual variables which occur in E_i are in the list $\vec{\mathsf{x}}_i$, so that the system of equations within the braces is well-formed.

To interpret recursive programs, we use the following, simplest, classical

Fixed Point Theorem 1. *Every well-formed system of* $\mathbf{A}$*-explicit equations*

$$\Big\{ p_1(\vec{x}_1) = f_1(\vec{x}_1, p_1, \dots, p_K), \quad \dots, \quad p_K(\vec{x}_K) = f_K(\vec{x}_K, p_1, \dots, p_K) \Big\}$$

has a $\sqsubseteq$*-least* (canonical) *solution tuple* $\overline{p}_1, \dots, \overline{p}_K$ *characterized by*

$$\overline{p}_i(\vec{x}_i) = f_i(\vec{x}_i, \overline{p}_1, \dots \overline{p}_K) \quad (\textit{all } \vec{x}_i, i = 1, \dots, K), \qquad \text{(FP)}$$

$$(\textit{for all } i, \vec{x}_i)\Big(f_i(x_i, q_1, \dots, q_K)\!\downarrow \quad \Longrightarrow \quad f_i(x_i, q_1, \dots, q_K) = q_i(x_i) \Big)$$
$$\Longrightarrow \ \overline{p}_1 \sqsubseteq q_1, \dots, \overline{p}_K \sqsubseteq q_K. \qquad \text{(MIN)}$$

If all the individual variables which occur in the *head* E_0 of E are in the list $\vec{\mathsf{x}}$ and $(\overline{p}_1, \dots, \overline{p}_K)$ is the canonical solution tuple of the system

$$\Big\{ p_1(\vec{x}_1) = \mathrm{den}((\mathbf{A}, \vec{p}), E_1(\vec{x}_1)), \dots, p_K(\vec{x}_K) = \mathrm{den}((\mathbf{A}, \vec{p}), E_1(\vec{x}_K)) \Big\}$$

in its *body*, we set

$$f_E(\vec{x}) = \mathrm{den}(\mathbf{A}, E)(\vec{x}) =_{\mathrm{df}} \mathrm{den}((\mathbf{A}, \overline{p}_1, \dots, \overline{p}_K), E_0(\vec{x})). \qquad (*)$$

Consider the following two, classical examples:

---

[1] Here $E(\vec{x}) \equiv E\{\vec{\mathsf{x}}_i :\equiv \vec{x}_i\}$, the result of replacing in E each variable x_i by x_i.

Lemma 2 (The Euclidean algorithm ε for $\perp\!\!\!\perp$)**.** *The recursive program*

$$\mathrm{eq}_1(p(x,y)) \text{ where } \Big\{p(x,y) = \text{if } \mathrm{eq}_0(y) \text{ then } x \text{ else } p(y, \mathrm{rem}(x,y))\Big\}$$

of the structure[2] $\mathbf{N}_\varepsilon = (\mathbb{N}, \mathrm{rem}, \mathrm{eq}_0, \mathrm{eq}_1)$ *decides coprimeness for* $x, y \geq 1$*,*

Lemma 3 (The merge-sort algorithm ms)**.** *If* $\leq$ *orders* L*, then the recursive program*

$$\begin{aligned} p(u) \text{ where } \Big\{p(u) = {} & \text{ if } (\mathrm{tail}(u) = \mathrm{nil}) \text{ then } u \text{ else } q(p(\mathrm{half}_1(u)), p(\mathrm{half}_2(u))),\\ q(w,v) = {} & \text{ if } (w = \mathrm{nil}) \text{ then } v \text{ else } \text{ if } (v = \mathrm{nil}) \text{ then } w\\ & \text{else } \text{ if } (\mathrm{head}(w) \leq \mathrm{head}(v)) \text{ then } \mathrm{cons}(\mathrm{head}(w), q(\mathrm{tail}(w), v))\\ & \text{else } \mathrm{cons}(\mathrm{head}(v), q(w, \mathrm{tail}(v)))\Big\} \end{aligned}$$

of the structure $\mathbf{L}^*{}_{\mathsf{ms}} = (\mathbf{L}^*, \mathrm{half}_1, \mathrm{half}_2, \leq)$ *computes for each sequence* $u \in L^*$ *its ordered rearrangement* $\mathrm{sort}(u)$ *relative to* $\leq$*.*

4 Complexity theory for recursive programs

Fix a Φ-structure $\mathbf{A}$ and a recursive Φ-program E, let $\overline{p}_1, \ldots, \overline{p}_K$ be as in Section 3 and set

$$\begin{aligned} \mathrm{Conv}(\mathbf{A}, E) = \{M : M \equiv N\{\mathsf{y}_1 :\equiv y_1, \ldots, \mathsf{y}_m :\equiv y_m\} & \text{ where}\\ N(\mathsf{y}_1, \ldots, \mathsf{y}_m) & \text{ is a subterm of some } E_i,\\ \vec{y} \in A^m & \text{ and } \overline{M} = \mathrm{den}((\mathbf{A}, \overline{p}_1, \ldots, \overline{p}_K), M)\!\downarrow\}; \end{aligned}$$

for example, if $f_E(\vec{x})$ is the partial function computed by E as in (*), then

$$f_E(\vec{x})\!\downarrow \implies \Big(E_0(\vec{x}) \in \mathrm{Conv}(\mathbf{A}, E) \;\&\; f_E(\vec{x}) = \overline{E_0(\vec{x})}\Big).$$

We will use these *convergent* $(\mathbf{A}, E)$*-terms* to define several natural complexity measures of recursive programs, starting with the following most-basic one:

Lemma 4 (**Tree-depth complexity**)**.** *For fixed* $\mathbf{A}$ *and* E*, there is exactly one function* $D = D_E^{\mathbf{A}} : \mathrm{Conv}(\mathbf{A}, E) \to \mathbb{N}$ *such that:*

(D1) $D(\mathrm{tt}) = D(\mathrm{ff}) = D(x) = D(\phi) = 0$ (if $\mathrm{arity}(\phi) = 0$ and $\phi^{\mathbf{A}}\!\downarrow$).

(D2) $D(\phi(M_1, \ldots, M_m)) = \max\{D(M_1), \ldots, D(M_m)\} + 1$.

(D3) *If* $M \equiv \text{if } M_0 \text{ then } M_1 \text{ else } M_2$*, then*

$$D(M) = \begin{cases} \max\{D(M_0), D(M_1)\} + 1, & \text{if } \overline{M}_0 = \mathrm{tt},\\ \max\{D(M_0), D(M_2)\} + 1, & \text{if } \overline{M}_0 = \mathrm{ff}. \end{cases}$$

---

[2] Some notation: $S(x) = x + 1, \quad x \dot{-} y = \max(x - y, 0); \quad \mathrm{Pd}(x) = x \dot{-} 1, \quad \mathrm{eq}_x(y) \iff y = x$
for $y > 0, \mathrm{rem}(x,y)$ and $\mathrm{iq}(x,y)$ are the unique $r, q \in \mathbb{N}$ s.t. $x = yq + r \;\&\; r < y$,
$\gcd(x,y) =$ the greatest common divisor of x and y, $\quad x \perp\!\!\!\perp y \iff \gcd(x,y) = 1 \quad (x, y \geq 1)$,
$\mathbf{L}^* = (L^*, \mathrm{nil}, \mathrm{eq}_{\mathrm{nil}}, \mathrm{head}, \mathrm{tail}, \mathrm{cons})$ is the LISP structure over a set L, $\quad |u| =$ the length of $u \in L^*$,
for $u \in L^*, \mathrm{half}_1(u) =$ the first half and $\mathrm{half}_2(u) =$ the second half of u.

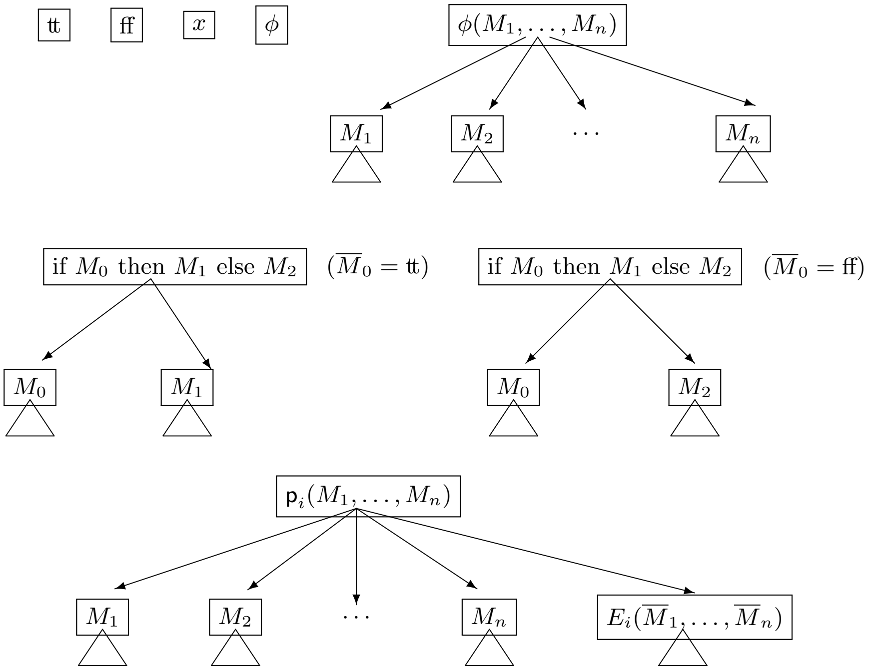

Figure 1: The computation tree.

(D4) *If* p_i *is a recursive variable of* E *of arity* m, *then*

$$D(\mathsf{p}_i(M_1,\dots,M_m)) = \max\{D(M_1),\dots,D(M_m), D(E_i(\overline{M}_1,\dots,\overline{M}_m))\} + 1.$$

The tree-depth complexity of E *is that of its head,*

$$d_E(\vec{x}) = d(\mathbf{A}, E(\vec{x})) =_{\mathrm{df}} d(E_0(\vec{x})) \quad (\mathrm{den}^{\mathbf{A}}_E(\vec{x})\downarrow).$$

This is proved by analysing the proof of Theorem 1.

5 The computation tree $\mathcal{T}(M)$

Using recursion on $D(M)$, we can associate with each $M \in \mathrm{Conv}(\mathbf{A}, E)$ a grounded tree $\mathcal{T}(M)$ whose *depth* is exactly $D(M)$ and which can be viewed as an "ideal (parallel) computation" of $\overline{M}$. We can also define in this way several natural *complexity measures* on recursive programs:

5.1 The sequential logical complexity $L^s(M)$ (time)

Define

$$L^s(M) = L^s(\mathbf{A}, E, M) \quad (M \in \mathrm{Conv}(\mathbf{A}, E))$$

for a Φ-structure $\mathbf{A}$ and a Φ-program E by the following recursion on $D(M)$:

(L^s1) $L^s(\mathrm{tt}) = L^s(\mathrm{ff}) = L^s(x) = 0$, and $L^s(\phi) = 1$ if $\mathrm{arity}(\phi) = 0$ and $\phi^{\mathbf{A}}\downarrow$.

(L^s2) $L^s(\phi(M_1,\dots,M_n)) = L^s(M_1) + L^s(M_2) + \cdots + L^s(M_n) + 1$.

(L^s3) If $M \equiv$ if M_0 then M_1 else M_2, then

$$L^s(M) = \begin{cases} L^s(M_0) + L^s(M_1) + 1 & \text{if } \overline{M}_0 = \mathrm{tt}, \\ L^s(M_0) + L^s(M_2) + 1 & \text{if } \overline{M}_0 = \mathrm{ff}. \end{cases}$$

(L^s4) $L^s(\mathsf{p}_i(M_1, \dots, M_n)) = L^s(M_1) + \cdots + L^s(M_n) + L^s(E_i(\overline{M}_1, \dots, \overline{M}_n)) + 1$,

and set $\mathsf{time}_E(\vec{x}) = l^s(\mathbf{A}, E(\vec{x})) =_{\mathrm{df}} L^s(E_0(\vec{x})) \quad (\mathrm{den}_E^{\mathbf{A}}(\vec{x})\downarrow)$.

Intuitively, $\mathsf{time}_E(\vec{x})$ counts the number of steps required for the computation of $f_E(\vec{x})$ using "the algorithm expressed by" E.

5.2 The number-of-calls complexity $C^s(\Phi_0)(M)$ (**calls**)

Define

$$C^s(\Phi_0)(M) = C^s(\Phi_0)(\mathbf{A}, E(\vec{\mathsf{x}}), M) \qquad (\Phi_0 \subseteq \Phi, M \in \mathrm{Conv}(\mathbf{A}, E))$$

for a Φ-structure $\mathbf{A}$, a Φ-program E and $\Phi_0 \subseteq \Phi$, by the following recursion on $D(M)$:

(C^s1) $C^s(\Phi_0)(\mathrm{tt}) = C^s(\Phi_0)(\mathrm{ff}) = C^s(\Phi_0)(x) = 0 \quad (x \in A)$; and if $\mathrm{arity}(\phi) = 0$ and $\phi\downarrow$, $C^s(\Phi_0)(\phi) = 0$ if $\phi \notin \Phi_0$, and $C^s(\Phi_0)(\phi) = 1$ if $\phi \in \Phi_0$.

(C^s2) If $M \equiv \phi(M_1, \dots, M_n)$, then

$$C^s(\Phi_0)(M) = \begin{cases} C^s(\Phi_0)(M_1) + \cdots + C^s(\Phi_0)(M_n) + 1, & \text{if } \phi \in \Phi_0, \\ C^s(\Phi_0)(M_1) + \cdots + C^s(\Phi_0)(M_n), & \text{otherwise.} \end{cases}$$

(C^s3) If $M \equiv$ if M_0 then M_1 else M_2, then

$$C^s(\Phi_0)(M) = \begin{cases} C^s(\Phi_0)(M_0) + C^s(\Phi_0)(M_1), & \text{if } \overline{M}_0 = \mathrm{tt}, \\ C^s(\Phi_0)(M_0) + C^s(\Phi_0)(M_2), & \text{if } \overline{M}_0 = \mathrm{ff}. \end{cases}$$

(C^s4) If $M \equiv \mathsf{p}_i(M_1, \dots, M_n)$ with p_i a recursive variable of E, then

$$C^s(\Phi_0)(M) = C^s(\Phi_0)(M_1) + \cdots + C^s(\Phi_0)(M_n) + C^s(\Phi_0)(E_i(\overline{M}_1, \dots, \overline{M}_n)).$$

The *number of* Φ_0*-calls complexity* in $\mathbf{A}$ of $E(\vec{\mathsf{x}})$ is that of its head term,

$$\mathsf{calls}(\Phi_0)_E(\vec{x}) = \mathsf{calls}(\Phi_0)(\mathbf{A}, E(\vec{x})) =_{\mathrm{df}} C^s(\Phi_0)(\mathbf{A}, E(\vec{\mathsf{x}}), E_0(\vec{x})),$$

and it is defined exactly when $\mathrm{den}_E^{\mathbf{A}}(\vec{x})\downarrow$.

This is a very natural complexity measure: $C^s(\Phi_0)(M)$ counts *the number of calls to the primitives in* Φ_0 which are needed to compute $\overline{M}$ using "the algorithm expressed" by the program E and disregarding the "logical steps" (branching and recursive calls) as well as calls to primitives not in Φ_0. Classical examples:

Lemma 5. *With the notation of Lemmas 2 and 3* (and $F_0, F_1, \dots$ the *Fibonacci numbers*) :

$$\begin{aligned} \mathsf{calls}(\mathrm{rem})(\mathbf{N}_\varepsilon, \perp\!\!\!\perp(x, y)) &\le 2\log y && (1 \le x \le y, y \ge 2) \\ \mathsf{calls}(\mathrm{rem})(\mathbf{N}_\varepsilon, \perp\!\!\!\perp(F_{k+1}, F_k)) &= k - 1 \ge r \log F_{k+1} && (\text{fixed } r, \text{ all } k \ge 2) \\ \mathsf{calls}(\le)(\mathbf{L}^*_{\mathsf{ms}}, \mathsf{ms}(u)) &\le |u| \log |u| && (\text{all } u \in L^*, u \ne \mathrm{nil}) \end{aligned}$$

We skip the similar definitions of *parallel versions* of these complexity functions, $\mathsf{p\text{-}time}_E(\vec{x})$ (parallel time) and $\mathsf{p\text{-}calls}(\Phi_0)(\mathbf{A}, E(\vec{x}))$ (depth-of-calls).

6 Complexity inequalities

For a fixed Φ-structure **A** and a recursive Φ-program E, easily

$$\begin{tikzcd}
 & \mathsf{calls}(\vec{x}) \arrow[dr, phantom, sloped, "\leq"] & & (\ell+1)^{\mathsf{p\text{-}calls}(\vec{x})} \\
\mathsf{p\text{-}calls}(\vec{x}) \arrow[ur, phantom, sloped, "\leq"] & & \mathsf{time}(\vec{x}) \arrow[ur, phantom, sloped, "\leq"] \arrow[dr, phantom, sloped, "\leq"] & \\
d(\vec{x}) \arrow[u, phantom, sloped, "\leq"] \arrow[r, phantom, "\leq"] & \mathsf{p\text{-}time}(\vec{x}) \arrow[ur, phantom, sloped, "\leq"] & & (\ell+1)^{d(\vec{x})+1}.
\end{tikzcd}$$

where ℓ is the largest arity of any primitive or pf variable which occurs in E.

Much more significant (and substantially more difficult to prove) is

Theorem 6 (**Tserunyan's inequalities**, [3]). *For every recursive* Φ*-program* E*, there are constants* K_s, K_p *such that for every* Φ*-structure* **A** *and every* $\vec{x} \in A^n$*, if* $\mathrm{den}(\mathbf{A}, E(\vec{x}))\downarrow$*, then*

$$\text{(a)}\ \ \mathsf{time}_E(\vec{x}) \leq K_s + K_s \mathsf{calls}(\Phi)_E(\vec{x}), \quad \text{(b)}\ \ \mathsf{p\text{-}time}_E(\vec{x}) \leq K_p + K_p \mathsf{p\text{-}calls}(\Phi)_E(\vec{x}).$$

(a) says that the large $\mathsf{time}_E(\vec{x})$ needed by any recursive Φ-program E to compute $f(x)$ in **A** is not caused by the large number of logical operations that E must do—"the high logical complexity of the algorithm expressed by E"—but by the large number $\mathsf{calls}(\Phi)_E(\vec{x})$ of necessary calls to the primitives of **A**, up to a linear factor which is independent of the structure **A**; ditto for the parallel complexity $\mathsf{p\text{-}time}_E(\vec{x})$ and its calls-counting (depth) counterpart $\mathsf{p\text{-}calls}_E(\vec{x})$. Taken together, the Tserunyan inequalities provide some explanation why lower bound results (which limit a large variety of algorithms) are most often proved by counting calls to the primitives, which is well known and little understood.

7 Concluding remark

There is no generally accepted definition of *what an algorithm is*, and so complexity functions are defined and studied on *models of computations* (finite register machines, decision trees, Turing machines, RAMs, etc.) here viewed as *implementations of algorithms*; but then we do not have a generally accepted definition of *what an implementation of the merge-sort in Lemma 3 is*, even though the study of these implementations is a rich (and rigorous) area of research.

We have outlined a simple theory of *complexity of algorithms* for those algorithms which are expressed by recursive programs. It implies many of the classical complexity results because all the standard computation models are *faithfully represented* by recursive programs using routine, well-understood methods; and it can be argued that *every algorithm which computes a function on a set A from given functions and relations on A is faithfully expressed by a recursive program.*

References

[1] Yiannis N. Moschovakis. *Abstract recursion and intrinsic complexity*, volume 48 of *ASL Lecture Notes in Logic*. Cambridge University Press, 2019. posted in ynm's homepage.

[2] D. S. Scott and C. Strachey. Towards a mathematical semantics for computer languages. In J. Fox, editor, *Proceedings of the Symposium on computers and automata*, pages 19–46, New York, 1971. Polytechnic Institute of Brooklyn Press.

[3] Anush Tserunyan. (1) *Finite generators for countable group actions;* (2) *Finite index pairs of equivalence relations;* (3) *Complexity measures for recursive programs.* Ph.D. Thesis, University of California, Los Angeles, 2013. Kechris, A. and Neeman, I., supervisors.