A mathematical modeling of pure, recursive algorithms

Yiannis. N. Moschovakis*
Department of Mathematics, UCLA
ynm@math.ucla.edu

February 9, 1989

Abstract

This paper follows previous work on the *Formal Language of Recursion* FLR and develops *intensional* (algorithmic) *semantics* for it: the *intension* of a term t on a structure $\mathbf{A}$ is a *recursor*, a set–theoretic object which represents the (abstract, recursive) algorithm defined by t on $\mathbf{A}$. Main results are the soundness of the reduction calculus of FLR (which models faithful, algorithm–preserving compilation) for this semantics, and the robustness of the class of algorithms assigned to a structure under algorithm adjunction.

This is the second in a sequence of papers begun with [16] in which we develop a foundation for the theory of computation based on a mathematical modeling of *recursive algorithms*. The general features, aims and methodological assumptions of this program were discussed and illustrated by examples in the preliminary, expository report [15]. In [16] we studied the formal language of recursion FLR which is the main technical tool for this work, we developed several alternative denotational semantics for it and we established a key *unique termination theorem* for a *reduction calculus* which models faithful (algorithm–preserving) compilation. Here we will define the *intensional semantics* of FLR for structures with given (pure) *recursors*, the set–theoretic objects we use to model pure (side–effect–free) algorithms: the *intension* of a term t on each structure $\mathbf{A}$ is a recursor which models the algorithm expressed by t on $\mathbf{A}$. Basic results of the paper

*During the preparation of this paper the author was partially supported by an NSF Grant.

include the *soundness* of the FLR reduction calculus for this semantics[1] and the *persistence under algorithm adjunction* of the class of algorithms thus assigned to a given structure. This last result is an intensional version of the central result in abstract recursion theory, which in turn generalizes the classical (first) Recursion Theorem of Kleene. It provides strong evidence of the *robustness* of our choice of representation of algorithms.

The paper naturally depends heavily on [15] and [16]. I have attempted, however, to introduce enough examples and comments so that the casual reader can get the gist of what I am trying to do.[2]

1 Recursors

As an illustration of the modeling of algorithms we will adopt, consider the familiar algorithm for computing x^2, but on the structure

$$\mathbf{A} = (N, 0, zero, pred, plus),$$

i.e. when we take as "given" on N the constant 0, the predecessor and addition functions $pred$, $plus$ and the Boolean–valued identity with 0, $zero(x) =$ if $(x = 0)$ then `1` else `0`:

$$\begin{aligned} sq(x) = times(x, x) \text{ where} \qquad (1)\\ times(u, v) = \text{if } zero(v) \text{ then } 0\\ \text{else } plus(times(u, pred(v)), u). \end{aligned}$$

We view (1) as the definition not of the square function—which is assumed known—but of a specific *recursive algorithm* for computing it; and we take the point of view that this recursive algorithm is specified completely by equation (1), so that there is no need to explain further (for example) how the recursion in this definition will be implemented. As a first approximation then, we might model this algorithm by the pair of *functionals*

$$\begin{aligned} f_0(times, x) &\simeq times(x, x),\\ f_1(u, v, times, x) &\simeq \text{if } zero(v) \text{ then } 0 \text{ else } plus(times(u, pred(v)), u), \end{aligned}$$

[1] FLR reduction is also *complete* for glogal equality of intensions, but we will not prove this here.

[2] I have omitted historical comments and general references to past work relating abstract recursion and computation. The most important papers which have influenced this work are (in chronological order) Kleene [7], McCarthy [10], Kleene [8], Landin [9], Platek [19], Gandy [4], Moschovakis [11], [12], Barwise–Gandy–Moschovakis [2], Scott–Strachey [20], Moschovakis [13], [14], Kechris–Moschovakis [5], Feferman [3], Normann [17], Backus [1]. Some additional discussion of sources is included in [15].

where *times* varies over partial, binary functions on N; the *abstract computation* of x^2 determined by this algorithm involves computing (in some unspecified way) the least fixed point $\overline{times}$ of the equation

$$times(u, v) \simeq f_1(u, v, times)$$

and then setting $x^2 = \overline{times}(x, x)$. This approach assumes that each value of the functional f_1 can be computed *in one step* on the structure $\mathbf{A}$, i.e. it does not analyze the required computation of such values *directly* in terms of the givens of the structure $\mathbf{A}$. To model the situation more accurately, we will "expand" the definition (1) in the form

$$\begin{array}{l} sq(x) = \ times(x, x) \text{ where} \\ \quad \{ \\ \quad times(u, v) \ \simeq \ \text{if } test(v) \text{ then } initvalue() \text{ else } loop(u, v), \\ \quad test(v) \quad\quad \simeq zero(v), \\ \quad initvalue() \ \simeq 0, \\ \quad loop(u, v) \quad \simeq plus(loop_1(u, v), u), \\ \quad loop_1(u, v) \ \simeq times(u, loop_2(v)), \\ \quad loop_2(v) \quad \simeq pred(v), \\ \quad \} \end{array} \tag{2}$$

and represent *the algorithm intended by* (1) *on* $\mathbf{A}$ by the tuple of functionals

$$\mathbf{sq} = [g_0, g_1, \ldots, g_6], \tag{3}$$

where $g_0 = f_0$ and the remaining six g_i's are defined by the fixed point equations in (2), e.g.

$$g_4(u, v, times, test, initvalue, loop, loop_1, loop_2) \simeq plus(loop_1(u, v), u).$$

Now each g_i is defined immediately in terms of the givens of $\mathbf{A}$, so that it is natural to assume that its values can be computed in "one step" on $\mathbf{A}$.

This object **sq** is a typical *pure recursor* and it will be *the intension on* $\mathbf{A}$ of the formal FLR version of (1). It is a semantic (set–theoretic) object which not only abstracts from the particular symbols used in (1), but also incorporates "semantic facts" about $\mathbf{A}$ not specified by the notation: for example, the expression

$$\begin{array}{l} sq'(x) = times(x, x) \text{ where} \\ \qquad\quad times(u, v) \simeq \text{if } zero(v) \text{ then } 0 \\ \qquad\qquad\qquad\quad \text{else } plus(u, times(u, pred(v))) \end{array}$$

(which differs from (1) in the order of the arguments to *plus*) will be assigned the same intension **sq**, simply because *plus* is commutative in **A**. On the other hand, the expression

$$\begin{aligned} sq''(x) = time(x) \text{ where} \\ time(v) \simeq \text{if } zero(v) \text{ then } 0 \\ \text{else } plus(time(pred(v)), x) \end{aligned}$$

is assigned by the same process a different intension: it expresses the alternative algorithm for computing x^2, where $x \cdot v$ is defined by recursion on v (with x held constant) and then x is substituted in for v.

Implicit in this analysis is the claim that the set theoretic object **g** in (3) codes precisely *the essential, mathematical and implementation-independent properties of the "algorithm"* intended by definition (1) on the structure **A**. As a first attempt to justify this claim, we analyzed several similar examples in [15], and we will consider some more in the sequel. We will look for additional justification of this approach to founding computation theory in the "naturalness" and usefulness of the intensional semantics for the language FLR.

Recall from section 1B of [16] that a (many sorted) *universe* is a family of *basic sets*

$$\mathcal{U} = \{\mathcal{U}(i) \mid i \in B\}$$

indexed by a set of basic types B, including "bool". A space of *individuals* U is a product of basic sets, $P(U, V)$ is the *pf space* of all partial functions from U to V and among the *product spaces* we admit the product **I** of "no factors". A *functional* is any *monotone* map on a product space to a basic set, $f : X \rightharpoonup W$. (These objects are all assigned types in the usual way.)

DEFINITION **1.1** *A (pure)* **recursor** *on a universe* $\mathcal{U}$ *is a tuple of functionals*

$$\mathbf{f} = [f_0, f_1, \ldots, f_n] : X \hookrightarrow W, \tag{4}$$

such that for suitable individual spaces $U_0 = \mathbf{I}$, $W_0 = W$, $U_1, W_1, \ldots, U_n, W_n$,

$$f_i : U_i \times P(U_1, W_1) \times \cdots \times P(U_n, W_n) \times X \rightharpoonup W_i.$$

We call f_0 *the* **head part** *and* $f_1, \ldots, f_n$ *the* **recursion parts** *of* **f**. *The* **denotation** *of* **f** *is a functional*

$$\overline{\mathbf{f}} : X \rightharpoonup W,$$

defined by

$$\overline{\mathbf{f}}(x) \simeq f_0(\overline{p}_{1,x}, \ldots, \overline{p}_{n,x}, x),$$

where for each x, $\overline{p}_{1,x}, \ldots, \overline{p}_{n,x}$ *are the simultaneous least fixed points of the equations*

$$p_i(u_i) \simeq f_i(u_i, p_1, \ldots, p_n, x) \quad (1 \leq i \leq n).$$

The restrictions on the types of the functionals of a recursor insure that the least–fixed–point equations in the definition of the denotation make sense.

With each recursor $\mathbf{f}$ as in (4) above and each $x \in X$, we can associate the *parameter–free recursor*

$$\mathbf{f}(x) = [f_{0,x}, f_{1,x}, \ldots, f_{n,x}] : \mathbf{I} \hookrightarrow W$$

by fixing x in the functionals of $\mathbf{f}$,

$$f_{i,x}(u_i, p_1, \ldots, p_n) \simeq f_i(u_i, p_1, \ldots, p_n, x).$$

Clearly, for every $x \in X$,

$$\overline{\mathbf{f}}(x) \simeq \overline{\mathbf{f}(x)}(\ ),$$

(where the empty space in $(\ )$ stands for the sole member of $\mathbf{I}$) and $\mathbf{f}$ is completely determined by the operation $x \mapsto \mathbf{f}(x)$. In fact, most often we will define a recursor $\mathbf{f}$ by giving an expression for $\mathbf{f}(x)$, using informally the symbolism of FLR and beginning with the obvious representations

$$\begin{aligned} \mathbf{f}(x) &= \mathrm{rec}(u_1, \ldots, p_n)[f_0(\vec{p}, x), \ldots, f_n(u_n, \vec{p}, x)] \\ &= f_0(\vec{p}, x) \text{ where } [p_i(u_i) = f_i(u_i, \vec{p}, x) \ : \ 1 \leq i \leq n], \end{aligned}$$

where $\vec{p} = p_1, \ldots, p_n$. For example, if

$$\mathbf{f}(s, q, x) = \mathrm{rec}(u, p)[f_0(p, s, q, x), f_1(u, p, s, q, x)]$$

and g, h are *functionals* of the appropriate types, we can define *the substitution of* g, h *in* $\mathbf{f}$ by

$$\begin{aligned} \mathbf{f}'(x) &= \mathbf{f}(g(x), \lambda(v)h(v, x), x) \qquad (5) \\ &= \mathrm{rec}(u, p)[f_0(p, g(x), \lambda(v)h(v, x), x), f_1(u, p, g(x), \lambda(v)h(v, x), x)]. \end{aligned}$$

With each functional $f : X \rightharpoonup W$ we associate the "degenerate recursor" $[f]$, whose denotation is obviously f,

$$\overline{[f]}(x) = f(x).$$

In the rec notation, $[f](x) = \ \mathrm{rec}(\ )[f(x)]$.

The order in which the parts of a recursor are listed has no bearing on the algorithm coded by the recursor, so (for example) we should have

$$\mathrm{rec}(u,p,v,q)[f_0(p,q), f_1(u,p,q), f_2(v,p,q)] \\ = \mathrm{rec}(v,q,u,p)[f_0(p,q), f_2(v,p,q), f_1(u,p,q)].$$

The general definition of equality for recursors is a bit messier to state.

DEFINITION **1.2** *Two recursors*

$$\mathbf{f} = [f_0, \ldots, f_n], \ \mathbf{g} = [g_0, \ldots, g_m] : X \hookrightarrow W$$

on the same parameter space are **strongly equivalent** *(or just* **equal***) if they have the same number of parts n and there exists a permutation σ on $\{0, 1, \ldots, n\}$ with inverse ρ and $\sigma(0) = 0$, such that for $i = 0, \ldots, n$ and all $u_i, x, p_1, \ldots, p_n$ in the appropriate spaces:*

$$f_i(u_i, p_1, \ldots, p_n, x) \simeq g_{\rho(i)}(u_i, p_{\sigma(1)}, \ldots, p_{\sigma(n)}, x).$$

In the example above we take $\sigma(1) = 2, \sigma(2) = 1$. It is quite trivial to check that *equal recursors have the same denotations.*

2 Unraveling nested recursions

Suppose

$$\mathbf{g} = [g_0, g_1] : U \times P(U, W) \times X \hookrightarrow W$$

is a recursor with denotation $\overline{\mathbf{g}} : U \times P(U, W) \times X \rightharpoonup W$ and set

$$\mathbf{h}_1(x) = \ \mathrm{rec}(u,p)[f(p,x), \overline{\mathbf{g}}(u,p,x)], \tag{6}$$

where f is some given functional. The algorithm represented by this recursor $\mathbf{h}_1$ assumes both f and $\overline{\mathbf{g}}$ as "given", and it is natural to ask if we cannot replace it by another algorithm which computes the same function directly in terms of f and the functionals g_0, g_1 which determine $\mathbf{g}$. Substituting formally (at first) $\mathbf{g}$ for $\overline{\mathbf{g}}$ in (6), we get

$$\mathbf{h}(x) = \ \mathrm{rec}(u,p)[f(p,x), \ \mathrm{rec}(v,q)[g_0(q,u,p,x), g_1(v,q,u,p,x)]] \tag{7}$$

and the problem is to understand the algorithmic meaning of this "nested recursion" and then to model it by a recursor. One basic result is that the recursor $\mathbf{h}$ *has the same denotation* as

$$\mathbf{h}'(x) = \ \mathrm{rec}(u,p,u,v,r)[f(p,x), \qquad\qquad (8)$$
$$g_0(r(u,\cdot),u,p,x),\ g_1(v,r(u,\cdot),u,p,x)],$$

where $r(u,\cdot) = \lambda(v)r(u,v)$. Park [18] refers to this as the *Bekič–Scott principle* and it has been rediscovered in various formulations by several people.[3] Here we will take it as a fundamental principle that *the recursor* $\mathbf{h}'$ *represents the algorithm intended by the nested recursion* $\mathbf{h}$, so that in fact we have the *algorithmic identity*

$$\mathrm{rec}(u,p)[f(p,x),\ \mathrm{rec}(v,q)[g_0(q,u,p,x), g_1(v,q,u,p,x)]] \qquad (9)$$
$$= \ \mathrm{rec}(u,p,u,v,r)[f(p,x),\ g_0(r(u,\cdot),u,p,x),\ g_1(v,r(u,\cdot),u,p,x)].$$

Part of the justification for (9) comes from an analysis of all the plausible *natural implementations* of these two functionals: they should turn out to be (essentially) the same. We will not elaborate on this here. From the technical point of view, however, this identification amounts to a definition, since we already know the meaning of (8) but we do not have a precise meaning for (7). We will need this definition in its full generality.

DEFINITION **2.1** *Suppose that we are given recursors*

$$\mathbf{f}_i = [f_{i,0},\ldots,f_{i,m(i)}] : U_i \times P(U_1,W_1) \times \cdots \times P(U_n,W_n) \times X \hookrightarrow W_i \quad (i \leq n)$$

where $U_0 = \mathbf{I}$ *and the remaining individual spaces match as indicated. We define the* **recursor combination** *of* $\mathbf{f}_0, \ldots, \mathbf{f}_n$ *by:*

$$\mathrm{rc}[\mathbf{f}_0,\mathbf{f}_1,\ldots,\mathbf{f}_n] \ =_{def} \ [g_{0,0},\ldots,g_{0,m(0)}, \qquad (10)$$
$$g_{1,0},\ldots,g_{1,m(1)},$$
$$\cdots$$
$$g_{n,0},\ldots,g_{n,m(n)}],$$

where for $i \leq n,\ j \leq m(i)$,

$$g_{i,j}(u_i,v_{i,j},r_{0,1},\ldots,r_{0,m(0)},r_{1,0},\ldots,r_{1,m(1)},\ldots,r_{n,0},\ldots,r_{n,m(n)},x)$$
$$\simeq f_{i,j}(v_{i,j},r_{i,1}(u_i,\cdot),\ldots,r_{i,m(i)}(u_i,\cdot),u_i,r_{1,0},r_{2,0},\ldots,r_{n,0},x).$$

We will also use the alternative rec *notation for this operation,*

$$\mathrm{rec}(u_1,p_1,\ldots,u_n,p_n)[\mathbf{f}_0(\vec{p},x),\ldots,\mathbf{f}_n(u_n,\vec{p},x)] = \ \mathrm{rc}[\mathbf{f}_0,\ldots,\mathbf{f}_n].$$

The operation rc is the semantic version of the syntactic operation `rc` defined in (2C.2) of [16] in the following sense.

[3] Cf. the discussion on the recursion theorem in section 4.

FACT **2.2** *If for* $i = 0, \ldots, n$

$$\begin{aligned}\mathbf{f}_i(u_i, \vec{p}, x) = rec(&v_{i,1}, q_{i,1}, \ldots, v_{i,m(i)}, q_{i,m(i)}) \\ &[f_{i,0}(\vec{q_i}, u_i, \vec{p}, x), \ldots, f_{i,m(i)}(v_{i,m(i)}, \vec{q_i}, u_i, \vec{p}, x)],\end{aligned}$$

then the syntactic operation `rc` *applied (formally) to the doubly recursive expression*

$$\mathrm{rec}(u_1, p_1, \ldots, u_n, p_n)[\mathbf{f}_0(\vec{p}, x), \ldots, \mathbf{f}_n(u_n, \vec{p}, x)]$$

yields exactly the rec *representation of* $\mathrm{rc}[\mathbf{f}_0, \ldots, \mathbf{f}_n]$. ⊣

Now (9) above is a theorem, very easy to check once the notation is penetrated. Another simple exercise which helps clarify the notation is the following.

FACT **2.3** *If* $\mathbf{f} = [f_0, \ldots, f_n]$ *is a recursor and for each* i, $[f_i]$ *is the recursor representation of the functional* f_i*, then*

$$\mathrm{rc}[[f_0], \ldots, [f_n]] = [f_0, \ldots, f_n],$$

i.e. in rec *notation,*

$$\begin{aligned}\mathrm{rec}(u_1, p_1, \ldots, u_n, p_n)&[\mathrm{rec}(\ )[f_0(\vec{p}, x)], \ldots, \mathrm{rec}(\ )[f_n(u_n, \vec{p}, x)]] \\ &= \mathrm{rec}(u_1, p_1, \ldots, u_n, p_n)[f_0(\vec{p}, x), \ldots, f_n(u_n, \vec{p}, x)].\end{aligned}$$ ⊣

The basic fact about the operation of recursor combination is that it commutes with the taking of denotations. This can be proved directly by a least–fixed–point argument, but it also follows from (2B.4) in [16].

THEOREM **2.4** *If* $\mathbf{f}_0, \ldots, \mathbf{f}_n$ *are recursors so that their recursor combination is defined, and if*

$$\begin{aligned}\mathbf{f}(x) &= \mathrm{rec}(u_1, p_1, \ldots, u_n, p_n)[\mathbf{f}_0(\vec{p}, x), \ldots, \mathbf{f}_n(u_n, x)] \\ \mathbf{g}(x) &= \mathrm{rec}(u_1, p_1, \ldots, u_n, p_n)[\overline{\mathbf{f}}_0(\vec{p}, x), \ldots, \overline{\mathbf{f}}_n(u_n, x)],\end{aligned}$$

then $\overline{\mathbf{f}}(x) = \overline{\mathbf{g}}(x)$. ⊣

The recursor combination operation can be used to define many natural operations on recursors, including *substitution*, as follows.

DEFINITION **2.5** *If* $\mathbf{g} : X \hookrightarrow U$, $\mathbf{h} : V \times X \hookrightarrow V'$ *and* $\mathbf{f} : U \times P(V, V') \times X \hookrightarrow W$, *set*

$$\mathbf{f}(\mathbf{g}(x), \lambda(v)\mathbf{h}(v, x), x) = \text{ rec}(r, v, p)[\mathbf{f}(r(), p, x),\ \mathbf{g}(x),\ \mathbf{h}(v, x)], \tag{11}$$

where $r : \mathbf{I} \rightharpoonup U$ *varies over partial functions "with no arguments". As a special case, when* $\mathbf{f}$ *is the conditional,*

$$\begin{aligned} &\text{if } \mathbf{c}(x) \text{ then } \mathbf{g}(x) \text{ else } \mathbf{h}(x) \\ &\quad = \text{ rec}(p, q, r)[\text{if } p() \text{ then } q() \text{ else } r(), \mathbf{c}(x), \mathbf{g}(x), \mathbf{h}(x)]. \end{aligned} \tag{12}$$

Notice that if $\mathbf{g} = [g]$, $\mathbf{h} = [h]$ are the recursor representations of functionals, then (11) is different from the functional substitution of g and h into $\mathbf{f}$ defined in (5).

To illustrate these definitions, suppose we define the identity on N by the recursion

$$\mathbf{id}(x) = \text{ rec}(n, id)[id(x),\ \text{if } \mathbf{zero}(n) \text{ then } \mathbf{0} \text{ else } \mathbf{succ}(id(\mathbf{pred}(n)))], \tag{13}$$

where we have assumed as given $0, zero, succ$ and $pred$, by their representing recursors. To compute the recursor which models **id**, we start with the definition of the conditional in (12) and use the recursor combination formulas,

$$\begin{aligned} \mathbf{id}(x) = \text{rec}(n, id)[&id(x), \\ &\text{rec}(p, q, r)[\text{ if } p() \text{ then } q() \text{ else } r(), \\ &\quad \mathbf{zero}(\text{n}), \\ &\quad \mathbf{0}, \\ &\quad \mathbf{succ}(id(\mathbf{pred}(n)))] \\ = \text{rec}(n, id, n, p, n, q, n, r)[&id(x), \\ &\quad \text{if } p(n) \text{ then } q(n) \text{ else } r(n), \\ &\quad \mathbf{zero}(n), \\ &\quad \mathbf{0}, \\ &\quad \mathbf{succ}(id(\mathbf{pred}(n)))]. \end{aligned} \tag{14}$$

Next we should reduce the compositions in the last expression and apply recursor combination again; when we are done we will have seven functionals in **id**, all of them "immediately" computable in terms of the given recursors.

3 Intensions

Recall from [16] that a **structure signature** is a triple $\tau = (B, S, d)$ with B a set of types, S a set of function(al) symbols and d a function which assigns to each function symbol $\mathtt{f} \in S$ a function type $d(\mathtt{f})$ over B; a **functional structure** of signature τ is a pair $\mathbf{A} = (\mathcal{U}, \mathcal{F})$ where $\mathcal{U}$ is a universe over B and $\mathcal{F}$ assigns a functional $\mathcal{F}(\mathtt{f})$ of type $d(\mathtt{f})$ on $\mathcal{U}$ to every function symbol in S. For our purposes here it is natural and useful to allow for richer structures which may have arbitrary recursors among their givens.

DEFINITION **3.1** *A* **recursor structure** *of type* $\tau = (B, S, d)$ *is a pair*

$$\mathbf{A} = (\mathcal{U}, \mathcal{F}),$$

where $\mathcal{U}$ *is a universe on* B *and*

$$\mathcal{F} = \{\mathcal{F}(\mathtt{f}) \mid \mathtt{f} \in S\}$$

is a family of recursors, such that for each $\mathtt{f} \in S$*, the type of the denotation* $\overline{\mathcal{F}(\mathtt{f})}$ *is* $d(\mathtt{f})$*. The* **associated functional structure** *is*

$$\overline{\mathbf{A}} = (\mathcal{U}, \overline{\mathcal{F}}),$$

of the same signature and on the same universe, where for each function symbol $\mathtt{f}$*,*

$$\overline{\mathcal{F}}(\mathtt{f}) = \overline{\mathcal{F}(\mathtt{f})}.$$

Denotations on $\mathbf{A}$ are computed in $\overline{\mathbf{A}}$; i.e. for each list of variables $\vec{\mathtt{x}}$ which includes all the free variables of a term t, we set

$$den(\vec{\mathtt{x}}, t) \text{ on } \mathbf{A} = den(\vec{\mathtt{x}}, t) \text{ on } \overline{\mathbf{A}}, \tag{15}$$

following definition (1D.3) of [16]. We think of $\mathbf{A}$ as an "algorithmic refinement" of $\overline{\mathbf{A}}$, whose recursors represent algorithms that compute the given functionals of $\overline{\mathbf{A}}$.

Functional structures are special cases of recursor structures with degenerate recursors. Structures with non–degenerate recursors arise naturally as expansions, e.g. we may wish to add to the simplest structure for arithmetic

$$\mathbf{N} = (N, 0, succ, pred, zero) \tag{16}$$

the recursor **id** of (13) which computes the identity function. The expansion $(\mathbf{N}, \mathbf{id})$ has the same signature but is different from its associated functional structure $(\mathbf{N}, id)$, where we take the identity *function* on N as immediately computable. These two structures, however, have the same denotations.

Recall from [16] that a **context** is a set E of basic and pf variables and an expression is **immediate in a context** E if it is congruent to a basic variable, $p(v_1, \ldots, v_n)$ with $p, v_1, \ldots, v_n \in E$ or $\lambda(u^1, \ldots, u^m)p(v_1, \ldots, v_n)$ with $p, v_1, \ldots, v_n \in E \cup \{u^1, \ldots, u^m\}$.[4]

In the remainder of this section we will prove the following main result of the paper.

THEOREM **3.2 Main Result**. *Fix a recursor structure* $\mathbf{A}$.

(a) *There is a* **unique** *way to associate with each term* t*, each context* E *and each list of variables* $\vec{\mathtt{x}} \equiv \mathtt{x}_1, \ldots, \mathtt{x}_k$ *which includes all the free variables of* t*, a recursor*

$$\mathrm{int}(\vec{\mathtt{x}}, E)t : X \hookrightarrow W,$$

where X *is the space of the same type as the list* $\vec{\mathtt{x}}$ *and* W *is the basic set of the value type of* t*, so that the following conditions are satisfied.*

1. $\mathrm{int}(\mathtt{x}_1, \ldots, \mathtt{x}_k, E)\mathtt{x}_i = [\lambda(x_1, \ldots, x_k)x_i]$.
2. *If* $t_1, \ldots, t_n$ *are immediate in* E*, then*

$$\mathrm{int}(\vec{\mathtt{x}}, E)p(t_1, \ldots, t_n) = [den(\vec{\mathtt{x}}, p(t_1, \ldots, t_n))].$$

3. *If* $t_1, \ldots, t_n$ *are immediate in* E *and* $\mathtt{f}$ *is interpreted on* $\mathbf{A}$ *by the recursor* $\mathbf{f}$*, then*

$$\mathrm{int}(\vec{\mathtt{x}}, E)\mathtt{f}[t_1, \ldots, t_n](x) = \mathbf{f}(den(\vec{\mathtt{x}}, t_1)(x), \ldots, den(\vec{\mathtt{x}}, t_n)(x)).$$

4. *If* t *is not immediate in* E *and* $\overline{a}, \overline{b}$ *are sequences of terms, then*

$$\mathrm{int}(\vec{\mathtt{x}}, E)p(\overline{a}, t, \overline{b}) = \ \mathrm{rec}(r)[\mathrm{int}(r, \vec{\mathtt{x}}, E \cup \{r\})p(\overline{a}, r(), \overline{b}),\ \mathrm{int}(r, \vec{\mathtt{x}}, E)t].$$

[4] One preprint version of [16] unfortunately used a simplified notion of context and immediacy, reasonable for the results of that paper but inadequate for what we will do with it here. This is the correct definition as it will appear in the published version of the paper.

5. *If t is not immediate in E and $\overline{a}, \overline{b}$ are sequences of terms or λ-terms, then*

$$\mathrm{int}(\vec{\mathtt{x}}, E)\mathtt{f}[\overline{a}, t, \overline{b}] = \ \mathrm{rec}(r)[\mathrm{int}(r, \vec{\mathtt{x}}, E \cup \{r\})\mathtt{f}[\overline{a}, r(), \overline{b}],\ \mathrm{int}(r, \vec{\mathtt{x}}, E)t].$$

6. *If $\lambda(u)t$ is not immediate in E and $\overline{a}, \overline{b}$ are sequences of terms or λ-terms, then*

$$\begin{aligned}&\mathrm{int}(\vec{\mathtt{x}}, E)\mathtt{f}[\overline{a}, \lambda(u)t, \overline{b}]\\ &\qquad = \ \mathrm{rec}(u, r)[\mathrm{int}(r, \vec{\mathtt{x}}, E \cup \{r\})\mathtt{f}[\overline{a}, r, \overline{b}],\ \mathrm{int}(u, r, \vec{\mathtt{x}}, E)t].\end{aligned}$$

(b) *If $\vec{p} = p_1, \ldots, p_n$ and for $i = 0, \ldots, n$,*

$$\mathbf{f}_i = \mathrm{int}(u_i, \vec{p}, \vec{\mathtt{x}}, E \cup \{u_i, \vec{p}\})t_i,$$

then

$$\mathrm{int}(\vec{\mathtt{x}}, E)\mathtt{rec}(u_1, p_1, \ldots, u_n, p_n)[t_0,\ t_1, \ldots,\ t_n] = \ \mathrm{rc}[\mathbf{f}_0,\ \mathbf{f}_1, \ldots, \mathbf{f}_n].$$

(c) *The intension of a term computes its denotation, i.e. for any E,*

$$\overline{\mathrm{int}(\vec{\mathtt{x}}, E)t}(x) \simeq \mathit{den}(\vec{\mathtt{x}}, t)(x).$$

(d) *The* FLR *reduction calculus is sound for intensions, i.e.*

$$s \sim_E t \Longrightarrow \mathrm{int}(\vec{\mathtt{x}}, E)s = \mathrm{int}(\vec{\mathtt{x}}, E)t. \qquad \dashv$$

The rules for manipulating intensions in (a), (b) are the semantic versions of the syntactic rules of the FLR calculus, and it may be argued directly that "they preserve the algorithm". We used all of them in the examples, to transform (1) to (2) and (13) to (14). They imply uniqueness of intensions, and in fact they "overdetermine" the notion, in the sense that it is not obvious how to define intensions so that (a) and (b) hold. It is possible to reformulate these conditions so that they become a definition, but then (d) is not that trivial; to prove it we must redo "the semantic version" of the *unique termination property* for the FLR reduction calculus, the main result of [16]. We will adopt an indirect approach, which defines intensions using normal forms of terms and then uses the syntactic results of [16] to prove the Main Result.

The key idea is that *in a functional structure*, if we reduce a term t to normal form

$$\mathtt{nf}(t, E) \equiv \mathtt{rec}(u_1, \ldots, p_n)[t_0, \ldots, t_n],$$

then we should obviously set

$$\mathrm{int}(\vec{\mathtt{x}}, E)t = [den(\vec{p}, \vec{\mathtt{x}}, t_0), \ldots, den(u_n, \vec{p}, \vec{\mathtt{x}}, t_n)].$$

To deal with the general case, we will first expand each recursor structure $\mathbf{A}$ into a functional structure $\mathbf{A}^\circ$, we will define a natural translation of the language of $\mathbf{A}$ into the language of $\mathbf{A}^\circ$ and we will compute intensions there.

DEFINITION **3.3** *With each recursor structure* $\mathbf{A}$ *we associate its* **functional expansion** $\mathbf{A}^\circ$ *by replacing each recursor* $\mathbf{f} = [f_0, f_1, \ldots, f_n]$ *in* $\mathbf{A}$ *with the functionals* $f_0, f_1, \ldots, f_n$*. The signature* $(B, S, d)^\circ$ *of* $\mathbf{A}^\circ$ *is an expansion of the signature of* $\mathbf{A}$*; we keep the symbol* $\mathtt{f}$ *which named* $\mathbf{f}$ *to name* f_0 *and we introduce new symbols* $\mathtt{f}_1, \ldots, \mathtt{f}_n$ *for the recursion parts (if any) of* $\mathbf{f}$*.*

Notice that if $\mathbf{A}$ is a functional structure of signature τ, then $\tau^\circ = \tau$ and $\mathbf{A}^\circ = \mathbf{A}$.

DEFINITION **3.4** *Fix a structure* $\mathbf{A}$ *of signature* τ*. With each term or* λ*-term* t *of* $\mathrm{FLR}(\tau)$ *and each context* E*, we associate the* **translation** $\mathtt{tr}(t, E)$*, an expression of the same category as* t *in* $\mathrm{FLR}(\tau^\circ)$*. The definition of* $\mathtt{tr}(t, E)$ *is by recursion on* t*.*

Tr 1. If t is a variable, then $\mathtt{tr}(t, E) \equiv t$.

Tr 2. $\mathtt{tr}(p(t_1, \ldots, t_n), E) \equiv p(\mathtt{tr}(t_1, E), \ldots, \mathtt{tr}(t_n, E))$.

Tr 3. $\mathtt{tr}(\lambda(u)t, E) \equiv \lambda(u)\mathtt{tr}(t, E \cup \{u\})$.

Tr 4. As a typical example of this case, suppose

$$t \equiv \mathtt{f}[w, s, \lambda(u)t^*]$$

where w is immediate in E and $s, \lambda(u)t^*$ are not immediate in E. Suppose $\mathbf{f}$ has $n+1$ parts in $\mathbf{A}$ ($n = 0$ is possible), so that we have function symbols $\mathtt{f}, \mathtt{f}_1, \ldots, \mathtt{f}_n$ in $\mathrm{FLR}(\tau^\circ)$. We set

$$\begin{aligned}&\mathtt{tr}(\mathtt{f}[w,s,\lambda(u)t^*],E)\\ &\quad\equiv\ \mathtt{rec}(v_1,q_1,\ldots,v_n,q_n,r,u,p)\\ &\qquad[\mathtt{f}[\vec{q},w,r(),p],\ \mathtt{f}_1[v_1,\vec{q},w,r(),p],\ldots,\ \mathtt{f}_n[v_n,\vec{q},w,r(),p],\\ &\qquad\ \mathtt{tr}(s,E),\ \mathtt{tr}(t^*,E\cup\{u\})],\end{aligned}$$

where r, p and the v_i, q_i are fresh variables. In the general case we translate in this way those arguments of $\mathtt{f}$ which are not immediate in E and leave the immediate ones alone.

Tr 5. If $t \equiv\ \mathtt{rec}(u_1,\ldots,p_n)[t_0,\ldots,t_n]$, compute first for each i the translation $t_i^* \equiv \mathtt{tr}(t_i, E\cup\{u_i,p_1,\ldots,p_n\})$ and set

$$\mathtt{tr}(t,E) \equiv\ \mathtt{rec}(u_1,\ldots,p_n)[t_0^*,\ldots,t_n^*].$$

The basic fact about this translation operation is that it preserves the syntactic, intensional equivalence relation of [16].

THEOREM **3.5** *Under the hypotheses of definition* (3.4)*,*

$$s \sim_E t \Longrightarrow \mathtt{tr}(s,E) \sim_E \mathtt{tr}(t,E),$$

and if **A** *is a functional structure, then for every* t, E*,*

$$t \sim_E \mathtt{tr}(t,E).$$

PROOF of the main assertion is by induction on the definition of $\rightarrow_E$ and $\sim_E$ and most of the cases are trivial. The interesting cases are **R2** and **R3**. We consider a special case of **R2** which illustrates the method. Suppose

$$\mathtt{f}[w,t,s] \rightarrow_E \mathtt{rec}(r)[\mathtt{f}[w,r(),s],\ t] \qquad (t \text{ not immediate in } E)$$

where w is immediate in E but s is not. Assuming for simplicity that the recursor interpreting $\mathtt{f}$ has just two parts, the translations of the two sides of this reduction are:

$$\begin{aligned}L\ &\equiv\ \mathtt{rec}(u,q,r_t,r_s)[\mathtt{f}[q,w,r_t(),r_s()],\mathtt{f}_1[u,q,w,r_t(),r_s()],t^\circ,s^\circ],\\ R\ &\equiv\ \mathtt{rec}(r)[\mathtt{rec}(u,q,r_s)[\mathtt{f}[q,w,r(),r_s()],\mathtt{f}_1[u,q,w,r(),r_s()],s^\circ],t^\circ],\end{aligned}$$

where t°, s° are the translation of t and s in E. Now

$$R \rightarrow_E \mathtt{rec}(u,q,r_s,r)[\mathtt{f}[q,w,r(),r_s()],\mathtt{f}_1[u,q,w,r(),r_s()],s^\circ,t^\circ] \qquad (17)$$

by an application of the reduction rule **R4**, and the right hand side in the reduction (17) is congruent with L.

To prove the last assertion of the theorem, check first (trivially) that if $\mathbf{A}$ is a functional structure and t is a recursive term, irreducible in E, then $t \equiv_c \mathtt{tr}(t, E)$; hence for any t, $\mathtt{tr}(\mathtt{nf}(t, E), E) \equiv_c \mathtt{nf}(t, E)$ and by the first part $\mathtt{tr}(t, E) \sim_E \mathtt{tr}(\mathtt{nf}(t, E), E)$, so $\mathtt{tr}(t, E) \sim_E \mathtt{nf}(t, E) \sim_E t$. ⊣

DEFINITION **3.6 Definition of intensions.** *Fix a structure* $\mathbf{A}$*, let* $\mathbf{A}^\circ$ *be its functional expansion, and suppose that* $\vec{\mathtt{x}}$ *is a list of variables which includes all the free variables of some term* t *and* E *is a context. If*

$$\mathtt{nf}(\mathtt{tr}(t, E), E) \equiv \mathtt{rec}(u_1, p_1, \ldots, u_n, p_n)[t_0, t_1, \ldots, t_n]$$

is the E*–normal form of the translation of* t*, then we set*

$$\mathrm{int}(\vec{\mathtt{x}}, E)t = [den(\vec{\mathtt{x}}, t_0), den(\vec{\mathtt{x}}, t_1), \ldots, den(\vec{\mathtt{x}}, t_n)].$$

Because of Theorem **3.5**, we have immediately:

FACT **3.7** *If* $\mathbf{A}$ *is a functional structure and*

$$\mathtt{nf}(t, , E) \equiv \mathtt{rec}(u_1, p_1, \ldots, u_n, p_n)[t_0, t_1, \ldots, t_n],$$

then $\mathrm{int}(\vec{\mathtt{x}}, E)t = [den(\vec{\mathtt{x}}, t_0), den(\vec{\mathtt{x}}, t_1), \ldots, den(\mathtt{x}, t_n)]$. ⊣

Proof of the Main Result. Part (d) follows immediately from Theorem **3.5**.

Part (c) is also routine, if a bit tedious: check first that the operation $\mathtt{tr}$ preserves denotations as does intensional equivalence $\sim_E$ (for any E) by Theorem (2B.4) of [16], and then appeal to the definitions of denotations of recursive terms in [16] and of recursors in section 1.

In part (a), (1)–(3) are trivial and (4)–(6) follow directly from (b) and (d).

Proof of part (b). Suppose first that $\mathbf{A}$ is a functional structure and for $i = 0, \ldots, n$ let

$$\begin{aligned} t_i^* &\equiv \mathtt{nf}(t_i, E \cup \{u_i, \vec{p}\}), \\ \mathbf{f}_i(u_i, \vec{p}, x) &= \mathrm{int}(u_i, \vec{p}, x, E)t_i^*. \end{aligned}$$

By the definition of normal forms and the fact that intensions are preserved when we pass to normal form, the value of the left–hand–side of (b) is

$$L = \mathrm{int}(\vec{\mathtt{x}}, E)\ \mathtt{rc}\ \mathtt{rec}(u_1, \ldots, p_n)[t_0^*, \ldots, t_n^*].$$

and the value of the right–hand–side of (b) is

$$R = \ \mathrm{rc}[\mathbf{f}_0, \ldots, \mathbf{f}_n].$$

Now each t_i^* is an irreducible recursive term and its intension $\mathbf{f}_i$ is just the sequence of denotations of its parts; using this and the fact that rc "unravels" doubly recursive functional expressions exactly as `rc` unravels doubly recursive terms, we can verify $L = R$ by a direct computation.

If $\mathbf{A}$ is an arbitrary recursor structure and if t_i° is the translation of t_i in the appropriate context, then

$$\begin{aligned}\mathrm{int}(\vec{\mathtt{x}}, E)&\mathtt{rec}(u_1,\ldots, p_n)[t_0, \ldots, t_n)\\ &= \mathrm{int}(\vec{\mathtt{x}}, E)\mathtt{rec}(u_1, \ldots, p_n)[t_0^\circ, \ldots, t_n^\circ)\end{aligned}$$

and we can apply the result on the functional structure $\overline{\mathbf{A}}$ and (once more) the fact that a term has the same intension as its translation. ⊣

4 Algorithms

In choosing basic notions for a theory of computation, one must decide at the outset what will be taken "for granted": in particular, if we admit a certain set W as a the domain of some computable function, should we necessarily also admit the identity function id_W and/or the (binary) identity relation $=_W$ on W as given? The question has some merit in theories which allow only special (structured) domains of computation, for which it may argued (from the assumed structure) that an obvious algorithm which computes one or the other of these operations is automatically available. In the present approach, however, we allow arbitrary basic sets in universes and there is no justification for such assumptions. We take as primitives only the Boolean constants $\mathtt{0}, \mathtt{1}$, the conditional, the operations of substitution and λ-substitution and (most significantly) recursion.[5] As a practical matter, our approach is clearly the least restrictive, since the general theory applies

[5]These are the primitives for the theory of *pure algorithms*, with no "state dependence" or "side effects" with which we are concerned in this paper.

to structures which include all the identity functions and relations among the givens.

Since a single basic variable $\mathtt{v}$ is a term of FLR which naturally defines the identity function on the basic set of its type, we will use only the intensions of **special terms** to model the algorithms of a structure. These are defined in (1E.4) of [16]. What we need here is the characterization, that *a term t is special exactly when its normal form in any context E is an irreducible recursive term*

$$\mathtt{nf}(t, E) \equiv \mathtt{rec}(u_1, \ldots, p_n)[t_0, \ldots, t_n]$$

such that none of the irreducible explicit parts t_i is a single variable—i.e. when each t_i is in one of the simple forms

$$p(s_1, \ldots, s_m), \qquad \mathtt{f}[s_1, \ldots, s_m],$$

with the s_j's immediate in $E \cup \{u_i, \vec{\mathtt{p}}\}$.

DEFINITION **4.1** *An* **algorithm** *of a recursor structure* $\mathbf{A}$ *is any recursor*

$$\mathbf{f} : X \hookrightarrow W$$

on the universe of $\mathbf{A}$*, such that for some list of variables* $\vec{\mathtt{x}}$ *with type that of* X *and for some special term* $t(\vec{\mathtt{x}})$ *whose free variables are all included in* $\vec{\mathtt{x}}$*,*

$$\mathbf{f} = \mathrm{int}(\vec{\mathtt{x}}, \emptyset)t(\vec{\mathtt{x}}).$$

If $\mathcal{K}$ is a class of recursors of the same signature τ, then a (global) $\mathcal{K}$– **algorithm** *is an operation which assigns to each $\mathbf{A} \in \mathcal{K}$ a recursor $\mathbf{f}_\mathbf{A}$, so that for a fixed special $t(\vec{\mathtt{x}})$ as above in the common language,*

$$\mathbf{f}_\mathbf{A} = \mathrm{int}_\mathbf{A}(\vec{\mathtt{x}}, \emptyset)t(\vec{\mathtt{x}}).$$

The $\mathbf{A}$– (or $\mathcal{K}$–) **recursive functionals** *are the denotations (or global denotations) of the algorithms of $\mathbf{A}$ (or $\mathcal{K}$).*

Let us note immediately two trivial consequences of the definitions and (c) in the Main Result **3.2**:

FACT **4.2** (a) *If $\mathbf{A}$ and $\mathbf{B}$ have the same algorithms, then they have the same recursive functionals.*

(b) *The recursive functionals of a structure $\mathbf{A}$ are precisely the denotations of special terms on $\mathbf{A}$, and hence $\mathbf{A}$ and the associated functional structure $\overline{\mathbf{A}}$ have the same recursive functionals.* ⊣

On the other hand, $\mathbf{A}^\circ$ may have more denotations that $\mathbf{A}$, e.g. if some recursor of $\mathbf{A}$ is defined by

$$\mathbf{f}(x) = \mathrm{rec}(u, p)[f(x),\ g(x)]$$

so that it "never calls" the functional g, which is then recursive in $\mathbf{A}^\circ$ but not in $\mathbf{A}$. Similarly, the associated functional structure $\overline{\mathbf{A}}$ nay have more or fewer algorithms than $\mathbf{A}$.

Technically it is a definition, but **4.1** is clearly meant to express a claim: that *the intuitive notion of an abstract recursive algorithm can be modeled faithfully by the technical notion of an absolute intension of a special term of* FLR. This proposed **Church's Thesis for algorithms** was discussed briefly in section 2G of [15]. Here we will provide some evidence for it by the next result, which suggests that our modeling of algorithms is strongly robust.

THEOREM **4.3 The intensional recursion theorem**. *If* $\mathbf{f}$ *is an algorithm of a recursor structure* $\mathbf{A}$*, then the expansion* $(\mathbf{A},\ \mathbf{f})$ *has exactly the same algorithms as* $\mathbf{A}$*.*

More specifically, if $t(\vec{\mathtt{x}})$ *is a* $\emptyset$*–irreducible term and*

$$\mathbf{f} = \mathrm{int}(\vec{\mathtt{x}}, \emptyset)t(\vec{\mathtt{x}}), \tag{18}$$

and if $s(\mathtt{f})$ *is a term in the language of the expanded structure* $(\mathbf{A},\ \mathbf{f})$ *which is irreducible in some context* E*, then for any list of variables* $\vec{\mathtt{y}}$ *which includes all the free variables of* $s(\mathtt{f})$*,*

$$\mathrm{int}(\vec{\mathtt{y}}, E)s(\mathtt{f}) = \mathrm{int}(\vec{\mathtt{y}}, E)s(\lambda(\vec{\mathtt{x}})t(\vec{\mathtt{x}})). \tag{19}$$

To understand the intensional recursion theorem, compare it with the following extensional version, which follows immediately from it by **4.2**.

THEOREM **4.4 The extensional recursion theorem**. *If* f *is a recursive functional of a structure* $\mathbf{A}$*, then the expansion* $(\mathbf{A},\ f)$ *has the same recursive functionals as* $\mathbf{A}$*.* ⊣

For the structure $\mathbf{N}$ of arithmetic, **4.4** is a version of Kleene's classical (first) Recursion Theorem, XXVI in [6]. Kleene also proved the generalization to recursion in higher types, but (unfortunately) he buried it in the

impenetrable and practically unread section 10 of [8]. (The DISCUSSION at the end of the same section contains also the key idea for developing an indexing–free theory of abstract recursion on arbitrary functional structures.) Essentially as stated here, the extensional recursion theorem was announced in 4.4 of [14], extending the earlier 6B.4 of [13], and a proof of it was published in 3.3 of [5].

Kleene viewed his first recursion theorem as providing strong evidence for Church's Thesis, as it shows that it is impossible to escape from the class of recursive partial functions by some form of "diagonalization", without introducing entirely new principles of computation; cf. the discussion in §66 of [6]. One may argue similarly that **4.3** supports Church's Thesis for algorithms as formulated above: granting our basic framework, the theorem guarrantees that we cannot escape from the class of algorithms we assigned to a structure by any form of explicit or implicit definition, short of introducing new algorithmic principles not included among the primitives of FLR.

Incidentally, it is quite obvious that every classical Herbrand–Gödel–Kleene system of equations determines an algorithm in our sense, and we get all algorithms of $\mathbf{N}$ this way. On the other hand, it does not seem possible to define a Turing machine which "represents faithfully" the algorithm for computing x^2 "intended" by (1)—there is no way to code into a Turing machine "the recursion" which is the essence of (1) without getting bogged down into the details of its implementation.

We will come back to a fuller discussion of Church's Thesis for algorithms in a future paper in this sequence.

It is necessary to assume in the detailed statement of the theorem that the term $s(\mathtt{f})$ is irreducible. For example, suppose that over a structure with some given binary function g,

$$\mathbf{f}(x) = \mathrm{int}(\mathtt{x}, \emptyset)\mathtt{g}[\mathtt{x}, \mathtt{x}],$$

and in the expansion by $\mathbf{f}$, take

$$s(\mathtt{f}) \equiv \mathtt{f}[\mathtt{c}],$$

where $\mathtt{c}$ names a given constant. Now, clearly

$$\begin{aligned} \mathtt{nf}(s(\mathtt{f}), \emptyset) &\equiv \mathtt{rec}(r)[\mathtt{f}[r()],\ \mathtt{c}], \\ \mathtt{nf}(s(\lambda(\mathtt{x})\mathtt{g}[\mathtt{x}, \mathtt{x}]), \emptyset) &\equiv \mathtt{rec}(r_1, r_2)[\mathtt{g}[r_1(), r_2()],\ \mathtt{c},\ \mathtt{c}], \end{aligned}$$

and the intensions of these two terms cannot be the same since the first has two parts while the second has three. To apply the theorem in such a case, we first compute the normal form of $s(\mathtt{f})$ in the expanded structure,

$$s^*(\mathtt{f}) \equiv \mathtt{rec}(r)[\mathtt{f}[r()],\ \mathtt{c}];$$

now $s^*(\mathtt{f})$ and $s(\mathtt{f})$ have the same intensions and $s^*(\mathtt{f})$ easily has the same intension as its substitution

$$s^*(\lambda(\mathtt{x})\mathtt{g}[\mathtt{x},\mathtt{x}]) \equiv \mathtt{rec}(r)[\mathtt{g}[r(),r()],\ \mathtt{c}].$$

We need a technical lemma.

LEMMA **4.5** *Suppose $t(\vec{\mathtt{x}})$ is simplified, irreducible in the context J, where no variable in the list $\vec{\mathtt{x}} = \mathtt{x}_1, \ldots, \mathtt{x}_n$ is in J, and let*

$$\mathbf{f} = \mathrm{int}(\vec{\mathtt{x}}, J)t(\vec{\mathtt{x}}); \tag{20}$$

suppose further that the expressions $z_1(\vec{\mathtt{y}}), \ldots, z_n(\vec{\mathtt{y}})$ are immediate in some context $E \supseteq J$; then

$$\mathrm{int}(\vec{\mathtt{y}}, E)t(z_1(\vec{\mathtt{y}}), \ldots, z_n(\vec{\mathtt{y}})) = \lambda(y)\mathbf{f}(z_1(y), \ldots, z_n(y)). \tag{21}$$

PROOF. "Simplified" is defined in (2B.13) of [16] and simply means that there are no vacuous $\mathtt{rec}(\ )$'s in $t(\vec{\mathtt{x}})$, and by $z_i(y)$ we obviously mean the value of the expression $z_i(\vec{\mathtt{y}})$ when $\vec{\mathtt{y}} = y$.

Suppose first that $t(\vec{\mathtt{x}}) \equiv p(w_1(\vec{\mathtt{x}}), \ldots, w_m(\vec{\mathtt{x}}))$. Since $t(\vec{\mathtt{x}})$ is irreducible in J, each $w_j(\vec{\mathtt{x}})$ is immediate in J—which contains none of the $\mathtt{x}_i$'s—and hence if some basic variable $\mathtt{x}_i$ occurs in $w_j(\vec{\mathtt{x}})$ we must have $w_j(\vec{\mathtt{x}}) \equiv \mathtt{x}_i$; it follows that $p(w_1(\vec{\mathtt{z}}(\vec{\mathtt{y}})), \ldots, w_m(\vec{\mathtt{z}}(\vec{\mathtt{y}})))$ is irreducible in E, using the fact that immediacy in J implies immediacy in the larger E. Hence the intension of $t(\vec{\mathtt{z}}(\vec{\mathtt{y}}))$ in E is just its denotation and (21) follows from (20) by the definitions.

The argument is the same for the other cases of explicit $t(\vec{\mathtt{x}})$.

If $t(\vec{\mathtt{x}}) \equiv \mathtt{rec}(u_1, \ldots, p_n)[t_0, \ldots, t_n]$ is recursive and irreducible in J, then each t_i is explicit, irreducible in $J \cup \{u_i, \vec{p}\}$. The result follows in this case by a simple computation using the explicit case just proved and (b) of the Main Result **3.2**. ⊣

Proof of the Intensional Recursion Theorem. Assume (18) and suppose first that $s(\mathtt{f})$ is explicit, irreducible in E and (without loss of

generality) simplified. There is nothing to prove if $\mathtt{f}$ does not occur in $s(\mathtt{f})$, and if it does, then

$$\begin{aligned} s(\mathtt{f}) &\equiv \mathtt{f}[z_1(\vec{\mathtt{y}}), \ldots, z_n(\vec{\mathtt{y}})], \\ s(\lambda(\vec{\mathtt{x}})t(\vec{\mathtt{x}})) &\equiv t(z_1(\vec{\mathtt{y}}), \ldots, z_n(\vec{\mathtt{y}})) \end{aligned}$$

with $z_1(\vec{\mathtt{y}}), \ldots, z_n(\vec{\mathtt{y}})$ immediate in E, so that the lemma yields (19).

The case when $s(\mathtt{f})$ is recursive can be verified by an easy computation, using the explicit case and (b) of **3.2**. ⊣

References

[1] J. Backus, *Can programming be liberated from the von Neumann style? A functional style and its algebra of programs*, Comm. of the ACM, 21 (1978), 613–641.

[2] J. Barwise J., R. O. Gandy and Y. N. Moschovakis, *The next admissible set*, J. of Symbolic Logic, 36 (1971), 108–120.

[3] S. Feferman, *Inductive schemata and recursively continuous functionals*, in: Colloquium '76, R. O. Gandy, J. M. E. Hyland eds., Studies in Logic, North Holland, Amsterdam (1977), 373–392.

[4] R. O. Gandy, *General recursive functionals of finite type and hierarchies of functionals*, Ann. Fac. Sci. Univ. Clermont–Ferrand, 35 (1967), 5–24.

[5] A. S. Kechris and Y. N. Moschovakis, *Recursion in higher types*, in: Handbook of Logic, J. Barwise ed., Studies in Logic, North Holland, Amsterdam (1976), 681–737.

[6] S. C. Kleene, Introduction to metamathematics, van Nostrand, Princeton (1952).

[7] S. C. Kleene, *Recursive functionals and quantifiers of finite types, I*, Trans. Amer. Math. Soc., 91 (1959), 1–52.

[8] S. C. Kleene, *Recursive functionals and quantifiers of finite types, II*, Trans. Amer. Math. Soc., 108 (1963), 106–142.

[9] P. J. Landin, *The mechanical evaluation of expressions*, Computer J., 6 (1964), 308–320.

[10] J. McCarthy, *Recursive functions of symbolic expressions and their computation by machine, Part I*, Comm. of the ACM, 3 (1960), 184–195.

[11] Y. N. Moschovakis, *Abstract first order computability I, II*, Trans. Amer. Math. Soc., 138 (1969), 427–504.

[12] Y. N. Moschovakis, *Axioms for computation theories–first draft*, in: Logic Colloquium '69, R. Gandy, C. E. M. Yates eds., Studies in Logic, North Holland, Amsterdam (1971), 199–255.

[13] Y. N. Moschovakis, Elementary induction on Abstract Structures, Studies in Logic, North Holland, Amsterdam (1974).

[14] Y. N. Moschovakis, *On the basic notions in the theory of induction*, in: Logic, Foundations of Mathematics and Computability, R. E. Butts, J. Hintikka eds., Reidel, Dordrecht–Boston (1977), 207–236.

[15] Y. N. Moschovakis, *Abstract recursion as a foundation of the theory of algorithms*, in: Computation and Proof Theory, M. M. Richter et al eds., Lecture Notes in Mathematics (1104), Springer, Berlin (1984), 289–364.

[16] Y. N. Moschovakis, *The formal language of recursion*, to appear in J. Symbolic Logic.

[17] D. Normann, *Set recursion*, in: Generalized Recursion Theory II, J. E. Fenstad, R. O. Gandy, G. E. Sacks eds., Studies in Logic, North Holland, Amsterdam (1978), 303–320.

[18] D. Park, *On the semantics of fair parallelism*, Proc. Copenhagen Winter School, Lecture Notes in Computer Science (86), Springer (1980), 504–526.

[19] R. Platek, Foundations of recursion theory, Ph. D. Thesis, Stanford Univ., 1966.

[20] D. S. Scott and C. Strachey, *Towards a mathematical semantics for computer languages*, in: Proc. of the Symposium on Computers and Automata, Polytechnic Institute of Brooklyn Press, New York (1971), 19–46.