THE JOURNAL OF SYMBOLIC LOGIC
Volume 54, Number 4, Dec. 1989

THE FORMAL LANGUAGE OF RECURSION

YIANNIS N. MOSCHOVAKIS[1,2]

Contents

INTRODUCTION

This is the first of a sequence of papers in which we will develop a foundation for the theory of computation based on a precise, mathematical notion of *abstract algorithm*. To understand the aim of this program, one should keep in mind clearly the distinction between an algorithm and the object (typically a function) computed by that algorithm. The theory of computable functions (on the integers and on abstract structures) is obviously relevant to this work, but we will focus on making rigorous and identifying the mathematical properties of the finer (intensional) notion of algorithm.

Received April 14, 1988; revised September 1, 1988.

[1] During the preparation of this paper the author was partially supported by an NSF Grant.

[2] I thank the referee for the many errors found, and for numerous critical comments which prompted me to simplify and clarify the text in several places.

0022-4812/89/5404-0006/$04.70

It is characteristic of this approach that we take *recursion* to be a fundamental (primitive) process for constructing algorithms, not a derived notion which must be reduced to others—e.g. iteration or application and abstraction, as in the classical λ-calculus. We will model algorithms by *recursors*, the set-theoretic objects one would naturally choose to represent (syntactically described) recursive definitions. Explicit and iterative algorithms are modelled by (appropriately degenerate) recursors.

The main technical tool we will use is the *formal language of recursion*, FLR, a language of terms with two kinds of semantics: on each suitable structure, the *denotation* of a term t of FLR is a function, while the *intension* of t is a recursor (i.e. an algorithm) which *computes* the denotation of t. FLR is meant to be *intensionally complete*, in the sense that every (intuitively understood) "algorithm" should "be" (faithfully modelled, in all its essential properties by) the intension of some term of FLR on a suitably chosen structure.

In this paper we will define FLR and study its basic *syntactic properties*, specifically a *calculus of reduction and equivalence* for terms. Each term can be reduced to another which is (formally) *irreducible*, by a sequence of applications of some simple *reduction rules*, and the main technical result of the paper is the *uniqueness* (up to congruence) of the resulting *normal form* of the term. Read as equivalences, the reduction rules "axiomatize" *the fundamental transformations of recursive definitions which preserve the algorithm* expressed by a term t; the normal form of t describes then this algorithm directly and immediately in terms of the "givens" of the structure. In programming terms, the reduction calculus for FLR formalizes (faithful—algorithm preserving) *compilation* of programs, in contrast to reduction in the λ-calculus which formalizes computation of values.

In fact, the terms of FLR can be taken to denote quite general kinds of "functions", which may depend on a *state*, for example, or have *side effects*; and we can also understand the fundamental recursion construct in various ways—e.g. to denote either "call by value" or "call by name" recursion. As a result, we can interpret naturally in FLR most familiar programming languages, and then use such interpretations to derive denotational and intensional (algorithmic) semantics for programming languages. Without going into detail, we will supply sufficiently many comments on this so that a reader familiar with the theory of programming languages will see clearly how to interpret them in FLR.

In Part 3 we will provide some justification for the reduction calculus by showing that it preserves the values of terms in the several distinct denotational semantics for FLR to which we alluded above. In [Moschovakis alg] we develop the (more important for our program) intensional semantics of FLR for structures with *pure recursors* which model pure (side-effect-free) algorithms; and in a subsequent paper of this sequence we will extend the theory to cover sequential side effects and concurrent algorithms.

Many languages and specification systems syntactically similar to FLR have been studied in the literature, beginning with the classical Herbrand-Gödel-Kleene *systems of equations* [Kleene 1952] and including *Pure Lisp* [McCarthy 1960], *recursion schemes* [Greibach 1975] and the CCS calculus of [Milner 1980]. FLR can be viewed naturally as a variant of some of these and a second-order generalization of the others, as its "function symbols" can take arbitrary "partial function

variables" for arguments. Thus the main technical result of this paper can be viewed as a traditional proof of the unique termination property for a reduction calculus on a familiar language.

A preliminary version of this research was reported in [Moschovakis 1984], which we will cite by the initials of its title ARFTA. That was a wide ranging report, which discussed (with no proofs) several aspects and potential applications of the theory. Some of the basic definitions were deliberately simplified in ARFTA for purposes of exposition, and in addition the theory has been expanded and refined considerably since ARFTA was written. The technical results of this paper can be read independently of ARFTA, but we will often refer to that paper for the many examples and the general motivation it gives.

This work may be of interest to theoretical computer scientists, but also to logicians and recursion theorists (like myself) who have worked on the connection between recursive definitions and computability. I have included enough basic, expository material so that the paper is easily accessible to both these groups.

Part 1. Basic Definitions

§1A. Notation. We will use standard set-theoretic notation and terminology, except for a few special notations which we have collected in this section.

If $x_1,\dots,x_n$ are given objects, *the string* $x_1x_2\cdots x_n$ is the same object as *the sequence* $(x_1,\dots,x_n)$. If σ and τ are strings, then $\sigma\tau$ is the *concatenation* of σ and τ, and "$\sigma \equiv \tau$" means that σ and τ are identical.

We will write $f\colon A \rightharpoonup B$ to indicate that f is a *partial function* on A to B, i.e. a total function on some subset D of A, $f\colon D \to B$. As usual:

$$\begin{aligned} f(x)\downarrow &\Leftrightarrow x \in D \qquad (f(x) \textit{ is defined}),\\ f(x)\uparrow &\Leftrightarrow x \notin D \qquad (f(x) \textit{ is undefined}),\\ f(x) \simeq w &\Leftrightarrow f(x)\downarrow \ \&\ f(x) = w,\\ f(x) \simeq g(y) &\Leftrightarrow \text{either } f(x)\uparrow \ \&\ g(y)\uparrow\\ &\qquad \text{or for some } w,\ f(x) \simeq w \ \&\ g(y) \simeq w. \end{aligned}$$

If $f\colon B_1 \times \cdots \times B_n \longrightarrow C$ and for $i = 1,\dots,n$, $g_i\colon A \longrightarrow B_i$, then *the substitution of* $g_1,\dots,g_n$ *in* f, $h\colon A \longrightarrow C$, $h(x) \simeq f(g_1(x),\dots,g_n(x))$ is defined in the obvious way, so that in particular

$$h(x)\downarrow \Leftrightarrow g_i(x)\downarrow \ \&\ \cdots \ \&\ g_n(x)\downarrow \ \&\ f(g_i(x),\dots,g_n(x))\downarrow.$$

In the case $n = 1$, this is *the composition* of g and f.

The *nowhere defined* partial function will be denoted by the empty set symbol $\varnothing$.

For abstraction, we will use both the logical λ-notation and the algebraic notation, where the bound variable is indicated by a dot; i.e. if $f\colon U \times X \longrightarrow W$, then for each $x \in X$,

$$f(\cdot, x) = \lambda(u)f(u,x)\colon U \longrightarrow W,$$

and for each $u' \in U$,

$$f(\cdot, x)(u') \simeq \lambda(u) f(u, x)(u') \simeq f(u', x).$$

We will also use the common index notation for sequences, $\langle a_i : i \leq n \rangle = a_0, \ldots, a_n$, sometimes skipping the angles in the presence of other delimiters, such as $(a_i : i \leq n) = (a_0, \ldots, a_n)$ or $[a_i : i \leq n] = [a_0, \ldots, a_n]$. Since we will need often to deal with sequences from 1 to some n, it is convenient to set $\langle a_i : i \leq' n \rangle = a_1, \ldots, a_n$.

§1B. Types and pure typed objects.

DEFINITION 1B.1. A *set of basic types* is any set B which contains the string "bool". The set of *types* $T = T(B)$ (which we will use) over B is defined by the following recursive clauses.

t1. If $\bar{u}_1, \ldots, \bar{u}_n$ are basic types in B, then the string $(\bar{u}_1, \ldots, \bar{u}_n)$ is an *individual type* in T.

t2. If $\bar{u}$ is an individual type and $\bar{w}$ is a basic type, then the string $(\bar{u} \rightharpoonup \bar{w})$ is a *partial function* or *pf type* in T.

t3. If $\bar{x}_1, \ldots, \bar{x}_n$ are basic or pf types, then the string $(\bar{x}_1, \ldots, \bar{x}_n)$ is a *product type* in T.

t4. If $\bar{x}$ is a product type and $\bar{w}$ is a basic type, then the string $(\bar{x} \rightharpoonup \bar{w})$ is a *function type* in T.

Clearly every individual type is also a product type, and every pf type is also a function type. We allow $n = 0$ in these clauses, so that ( ) is an individual type.

We have chosen names for these types which suggest the use we will make of them in the interpretation of programming languages. Individual spaces will represent the "given" sets of tuples, on which we will accept definition by recursion as fundamental; objects of function type will interpret "functions", i.e. algorithms which act on individuals, may call other algorithms and are expected to *return a value*; objects of pf type will denote partial approximations to functions.

The "pure" type structure we will define here is the natural domain for the interpretation of programs which have no dependence on the state and no side effects—e.g. the programs of *Pure Lisp*. In Part 3 we will define more complex type structures in which we can interpret less restricted programs.

DEFINITION 1B.2. A *pure* (many sorted) *universe* over a set of basic types B is any family of sets $\mathscr{U}$ indexed by B, $\mathscr{U} = \{\mathscr{U}(i) : i \in B\}$, such that $\mathscr{U}(\text{bool}) = \{0, 1\}$. The *basic objects* of $\mathscr{U}$ are the members of the basic sets of $\mathscr{U}$, and the other typed objects of $\mathscr{U}$ are defined by the following recursive clauses corresponding to **t1**–**t4**.

po1. An *individual* of type $\bar{u} = (\bar{u}_1, \ldots, \bar{u}_n)$ is any member of the cartesian product $U = \mathscr{U}(\bar{u}_1) \times \cdots \times \mathscr{U}(\bar{u}_n)$. We will consider every space of individuals U as a partially ordered set (poset), equipped with the trivial (flat) partial ordering $=$.

po2. An object of pf type $(\bar{u} \rightharpoonup \bar{w})$ is any partial function (pf) in the space $P(U, W) = \{p : U \rightharpoonup W\}$, where U and W are the individual and basic spaces respectively of type $\bar{u}$ and $\bar{w}$. Each pf space $P(U, W)$ carries the natural partial order of inclusion,

$$p \leq q \Leftrightarrow (\forall u)[p(u)\downarrow \Rightarrow p(u) \simeq q(u)] \qquad (u \in U).$$

po3. A *point* of product type $\bar{x} = (\bar{x}_1, \ldots, \bar{x}_n)$ is any member of the *product space* $X = X_1 \times \cdots \times X_n$, where each X_i is the basic set or pf space of type $\bar{x}_i$. Each

product space carries the product partial ordering of its factors,

$$(x_1,\dots,x_n) \le (x'_1,\dots,x'_n) \Leftrightarrow x_1 \le x'_1 \;\&\; \cdots \;\&\; x_n \le x'_n.$$

po4. An object of function type $(\bar{x} \rightharpoonup \bar{w})$ is any partial function(al) $f\colon X \rightharpoonup W$, on the product space of type $\bar{x}$ to the basic set of type $\bar{w}$ which is *monotone* relative to the canonical partial ordering on X:

$$[f[x] \simeq w \;\&\; x \le y] \Rightarrow f[y] \simeq w.$$

Every total or partial function $f\colon U \rightharpoonup W$ on an individual space to a basic set is a functional, as are the *application* and *conditional* functionals, for each $\bar{u}$, $\bar{w}$:

$$ap_{\bar{u},\bar{w}}[p,u] \simeq p(u) \qquad (u \in U,\ p\colon U \rightharpoonup W),$$

$$cond_{\bar{u},\bar{w}}[i,q,r,u] \simeq \text{if } i \text{ then } q(u) \text{ else } r(u) \qquad (i \in \{0,1\},\ u \in U, q, r\colon U \rightharpoonup W).$$

In abstract recursion theory the following *discontinuous* monotone functional is used to represent existential quantification over the individual space U:

$$E^{\#}_U(p) \simeq \begin{cases} 1 & \text{if } (\exists u \in U)p(u) \simeq 1, \\ 0 & \text{if } (\forall u \in U)p(u) \simeq 0. \end{cases}$$

As in ARFTA, we will reduce cartesian products of product spaces to their basic and pf components, so that if $X = X_1 \times \cdots \times X_n$ and $Y = Y_1 \times \cdots \times Y_m$ are two product spaces of respective types $\bar{x} = (\bar{x}_1,\dots,\bar{x}_n)$ and $\bar{y} = (\bar{y}_1,\dots,\bar{y}_m)$, then *by definition* their product is the space

$$X \times Y = X_1 \times \cdots \times X_n \times Y_1 \times \cdots \times Y_m$$

of type $(\bar{x},\bar{y}) \equiv (\bar{x}_1,\dots,\bar{x}_n,\bar{y}_1,\dots,\bar{y}_m)$. Similarly, if $x = (x_1,\dots,x_n)$ and $y = (y_1,\dots,y_m)$ are two points in these spaces, then $(x,y) = (x_1,\dots,x_n,y_1,\dots,y_m)$ is their concatenation in $X \times Y$. We take $\mathbf{I}$ to be the *cartesian product of no factors*, which (formally) has only one element (the empty sequence) and for each X satisfies $X \times \mathbf{I} = \mathbf{I} \times X = X$.

§1C. The syntax of FLR.

DEFINITION 1C.1. A *signature* or *similarity type* is a triple $\tau = (B,S,d)$, where B is a set of basic types, S is any set (of *function constants*) and d is a mapping which assigns to each $\mathsf{f} \in S$ a function type $d(\mathsf{f})$ over B.

For a fixed signature τ, the *formal language of recursion* $\mathrm{FLR} = \mathrm{FLR}(\tau)$ associated with τ is defined as follows.

Variables. FLR has an infinite list of *basic variables of type i*, for each basic type i, and an infinite list of *pf variables of type* $(\bar{u} \rightharpoonup \bar{w})$, for each pf type $(\bar{u} \rightharpoonup \bar{w})$.

It will be convenient to call an arbitrary string $u \equiv u^1,\dots,u^m$ of *distinct* basic variables separated by commas, an *individual variable* of type $(\bar{u}^1,\dots,\bar{u}^m)$, where $\bar{u}^1,\dots,\bar{u}^m$ are the basic types of $u^1,\dots,u^m$. This will include the empty string $\varnothing$ which has type ( ). Notice that these sequences do not behave like true syntactic objects, for example the individual variables x, y, z and y, w "overlap".

Constants. FLR has the function constants of the signature τ, and each of these has an associated function type, also given by τ. In addition, FLR has the following function constants with the indicated function types.

Truth and falsity. 1 and 0 are constants of type $((\ ) \rightarrow \text{bool})$.

Conditionals. For each basic type $\bar{w}$, there is a function constant $\mathsf{cond}_{\bar{w}}$ of type $((\text{bool}, ((\ ) \rightarrow \bar{w}), ((\ ) \rightarrow \bar{w})) \rightarrow \bar{w})$.

Expressions. There are two kinds of expressions in FLR, *terms*, of basic type and *pf terms*, of pf type; each expression has an associated list of free occurrences of variables in it. We will define these notions simultaneously by the recursive clauses **T1–T5**.

T1. Each basic or pf variable is a term or pf term respectively, with its own type, and it occurs free in itself.

T2. If each t_i is a term with type $\bar{t}_i$ $(i = 1, \ldots, n)$ and p is a pf variable of type $((\bar{t}_1, \ldots, \bar{t}_n) \rightarrow \bar{w})$, then the string $p(t_1, \ldots, t_n)$ is a term of type $\bar{w}$. The free occurrences of variables in this term are those in $t_1, \ldots, t_n$ and the new occurrence of p.

In the case $n = 0$, this clause introduces the term $p(\ )$ of type $\bar{w}$, whenever p is of type $((\ ) \rightarrow \bar{w})$.

T3. If t is a term of type $\bar{w}$ and $u \equiv u^1, \ldots, u^m$ is an individual variable of type $\bar{u}$, then the λ-term $\lambda(u)t$ is a *pf term* of type $(\bar{u} \rightarrow \bar{w})$. The free occurrences in this pf term are the free occurrences of variables in t other than $u^1, \ldots, u^m$. When $m = 0$, this allows the λ-term $\lambda(\ )t$ of type $((\ ) \rightarrow \bar{w})$, with free occurrences exactly those of t.

Pf variables and λ-terms are the only pf terms which we will use in FLR, but more general pf terms come up in extensions.

T4. If f is a function constant with type $((\bar{t}_1, \ldots, \bar{t}_n) \rightarrow \bar{w})$, and each t_i is an expression with type $\bar{t}_i$, then the string $\mathsf{f}[t_1, \ldots, t_n]$ is a term of type $\bar{w}$. The free occurrences in this term are those in $t_1, \ldots, t_n$. If $n = 0$, we will typically abbreviate $\mathsf{f}[\ ]$ by f.

We will use the obvious abbreviations in connection with the special functions we put in FLR for every signature τ,

$$1 \equiv 1[\ ] \text{ (truth)}, \qquad 0 \equiv 0[\ ] \text{ (falsity)},$$

$$\textsf{if } s \textsf{ then } t_1 \textsf{ else } t_2 \equiv \mathsf{cond}_{\bar{w}}[s, \lambda(\ )t_1, \lambda(\ )t_2] \text{ (conditional)}.$$

These abbreviations are unambiguous, because the type $\bar{w}$ needed to identify the functional $\mathsf{cond}_{\bar{w}}$ can be read off the types of the given terms.

An expression is *explicit* if it can be constructed using only **T1–T4**. The last clause introduces the *recursive terms.*

T5. If $u_1, \ldots, u_n$ are individual variables of respective individual types $\bar{u}_1, \ldots, \bar{u}_n$, if $p_1, \ldots, p_n$ are distinct pf variables of respective types $(\bar{u}_1 \rightarrow \bar{w}_1), \ldots, (\bar{u}_n \quad \bar{w}_n)$ and if $t_0, t_1, \ldots, t_n$ are terms of respective types $\bar{w}, \bar{w}_1, \ldots, \bar{w}_n$, then

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

is a term of type $\bar{w}$. We call the terms $t_0, \ldots, t_n$ the *parts* of the *recursive term* t; t_0 is the *output* or *head* part. An occurrence of a variable v is free in t if it is in some t_i and the variable v is not one of the basic variables in the sequence u_i or one of the pf variables in the sequence $p_1, \ldots, p_n$. Any one of the sequences $u_1, \ldots, u_n$ may be empty, so that the clause allows constructs of the form $\mathsf{rec}(p, x, q)[t_0, t_1, t_2]$. We also allow $n = 0$, so that for each term t, the expression $\mathsf{rec}(\ )[t] \equiv \mathsf{rec}(\ )t$ is a term.

A more familiar notation for our recursion construct would be

$$\mathsf{rec}(u_1, p_1, \ldots, u_n, p_n)[t_0, t_1, \ldots, t_n] \equiv t_0 \quad \textsf{where } \{p_1(u_1) \simeq t_1, \ldots, p_n(u_n) \simeq t_n\}.$$

The type restrictions in **T5** insure that the equations $p_i(u_i) \simeq t_i$ make sense, i.e. both sides have the same basic type. We have adopted a notation which is not very "friendly", but it follows the traditional practice of logic, with the variable-binding operator rec in the front and the occurrences of variables it binds "within its scope"; this makes it harder to use the construct but easier to argue about it—and this is what we will mostly do here.

The definition of terms was presented here in its complete, messy glory, because we will use it as a "template" for definitions and proofs by induction on the generation of these syntactical objects: we will treat cases **T1–T5** assuming each time the notation and assumptions on types established here.

Bound occurrences of variables. Occurrences of variables in expressions which are not specified to be free by the clauses above are *bound*. For example, $u^1, \dots, u^m$ are bound in all their occurrences in $\lambda(u^1, \dots, u^m)t$ by **T4**. In the recursive term of **T5**, all the occurrences of $p_1, \dots, p_n$ are bound, and the basic variables in the sequence u_i are bound in the prefix and in their occurrences in t_i.

Substitutions. If $t(v)$ is an expression, v is a basic variable or function constant of basic value type $\bar{w}$ and s a basic variable or term of type $\bar{w}$, we let

$$t(s) \equiv \text{subst}(t(v), v/s)$$

be the result of replacing every free occurrence of v in $t(v)$ by s; similarly for a *simultaneous substitution*,

$$t(s_1, \dots, s_n) \equiv \text{subst}(t(v_1, \dots, v_n), v_1/s_1, \dots, v_n/s_n).$$

Every occurrence of a constant is "free" for purposes of this definition.

Suppose $s(u) \equiv s(u^1, \dots, u^m)$ is a term of type $\bar{w}$, the basic variables $u^1, \dots, u^m$ have types $\bar{u}^1, \dots, \bar{u}^m$ and r is a function constant or a pf variable with type $(\bar{u} \rightharpoonup \bar{w})$, where $\bar{u} \equiv (\bar{u}^1, \dots, \bar{u}^m)$. We define *the λ-substitution of $\lambda(u)s(u)$ for* r *in an expression* $t(\mathsf{r})$,

$$t(\lambda(u)s(u)) \equiv \text{subst}(t(\mathsf{r}), \mathsf{r}/\lambda(u)s(u)),$$

by induction on $t(\mathsf{r})$, in the obvious way. All the cases are trivial—we "inherit" the substitution from the subexpressions, except in case **T1** when $\mathsf{r} \equiv p$ and case **T4** when $\mathsf{r} \equiv \mathsf{f}$. In these cases we do the obvious,

$$\begin{aligned}&\text{subst}(p(t_1, \dots, t_n), p/\lambda(u)s(u))\\ &\quad \equiv s(\text{subst}(t_1, p/\lambda(u)s(u)), \dots, \text{subst}(t_n, p/\lambda(u)s(u))),\end{aligned}$$

and similarly with f instead of p.

Finally, suppose $s(x) \equiv s(x_1, \dots, x_m)$, where now $x_1, \dots, x_m$ are basic or pf variables and $t(\mathsf{r})$ is an expression, where r is a function constant such that $\mathsf{r}(x_1, \dots, x_m)$ is a term. We define

$$t(\lambda(x_1, \dots, x_m)s(x)) \equiv \text{subst}(t(\mathsf{r}), \mathsf{r}/\lambda(x_1, \dots, x_m)s(x))$$

in the obvious way, by replacing each $\mathsf{r}(t_1, \dots, t_n)$ in $t(\mathsf{r})$ (recursively) by the expression $s(\text{subst}(t_1, \mathsf{r}/\lambda(x)s(x)), \dots, \text{subst}(t_n, \mathsf{r}/\lambda(x)s(x)))$.

Typically we will use simultaneous substitutions of all kinds and we will tacitly assume that they are *free*, i.e., that no free variable of an expression which is being

substituted becomes bound in the result of the substitution. By the substitution p/q we will mean $p/\lambda(u)q(u)$.

The language REC of ARFTA was somewhat more complex than FLR because it had a built-in pairing function. In addition, the recursion quantifier of REC was different, but can be easily defined in terms of rec; calling it rec′ here, we can set

$$(\mathsf{rec}'u_1, p_1, \ldots, u_n, p_n\colon u_1^*)[t_1, \ldots, t_n]$$
$$\equiv_{\mathrm{def}} \mathsf{rec}(u_1, p_1, \ldots, u_n, p_n)[p_1(u_1^*), t_1, \ldots, t_n].$$

§1D. Denotational semantics on functional structures. To illustrate the notions, we will consider in this section the most natural denotational semantics of FLR on structures with functionals.

DEFINITION 1D.1. A *functional structure* of signature $\tau = (B, S, d)$ is a pair $\mathbf{A} = (\mathscr{U}, \mathscr{F})$, where $\mathscr{U}$ is a (pure) universe on B (as in 1B.2) and $\mathscr{F} = \{\mathscr{F}(\mathsf{f})\colon \mathsf{f} \in S\}$ is a family of monotone functionals on $\mathscr{U}$, such that the type of $\mathscr{F}(\mathsf{f})$ is $d(\mathsf{f})$.

A functional structure is *first order* if every functional in it is a total mapping $\mathscr{F}(\mathsf{f})\colon U \to W$ with no pf arguments. These are the familiar (many sorted) structures of model theory, with given relations, functions and constants, where each relation R is identified with its characteristic function into $\{0, 1\}$,

$$R(u) = \begin{cases} 1 & \text{if } R(u), \\ 0 & \text{if } \neg R(u). \end{cases}$$

Most often there are finitely many objects of each kind, and the structure is of the form

$$\mathbf{A} = (A_1, \ldots, A_k, R_1, \ldots, R_l, f_1, \ldots, f_m, c_1, \ldots, c_n),$$

or even with just one basic set (other than $\{0, 1\}$) as in the case of *the structure of arithmetic*

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

here $N = \{0, 1, \ldots,\}$ is the set of natural numbers and the givens are the constant 0, the successor and predecessor functions and the unary relation of identity with 0.

The *expansion* of $\mathbf{N}$

$$(\mathbf{N}, E_N^{\#}) = (N, 0, succ, pred, zero, E_N^{\#})$$

by the functional which embodies quantification on N is the classical structure of *hyperarithmetic* theory.

Finally, there are natural *set structures* of the form

$$\mathbf{V} = (V, \varnothing, \in, pair, \cup, rep),$$

where V is a suitably closed class of sets (e.g. the hereditarily finite sets, or *all* sets), $\in$ is the membership relation (mapping into $\{0, 1\}$) and *rep* is the key *replacement functional*

$$rep(x, p) \simeq \{p(i) \mid i \in x\} \qquad (p\colon V \rightharpoonup V).$$

Functional structures were called *recursion structures* in ARFTA, which contains several more interesting examples of them.

DEFINITION 1D.2. An *assignment* into a functional structure **A** is any partial function π which is defined on *all* pf variables and *some* basic variables of FLR, and which assigns (when defined) to each variable x an object in **A** (basic or pf) of the same type as x. We partially order assignments by

$$\pi \leq \rho \Leftrightarrow (\forall x)[\pi(x)\downarrow \Rightarrow [\rho(x)\downarrow \ \& \ \pi(x) \leq \rho(x)]].$$

The use of partially defined assignments is appropriate since the terms will denote partial functions, and it will insure later that *the property of replacement* holds.

We will define a partial function

$$val\text{: Expressions} \times \text{Assignments} \rightarrow \text{Objects of } \mathbf{A}$$

so that the following conditions hold:

(c1) $val(t, \pi)$ is defined for all pf expressions t, and if $val(t, \pi) \simeq w$, then the type of w is the same as the type of the expression t.

(c2) *val* is monotone, i.e. for all terms t,

$$[val(t, \pi) \simeq w \ \& \ \pi \leq \rho] \Rightarrow val(t, \rho) \simeq w,$$

and for pf expressions t,

$$\pi \leq \rho \Rightarrow val(t, \pi) \leq val(t, \rho).$$

(c3) $val(t, \pi)$ depends only on the values of π on the variables which occur free in t.

The definition is by the following five natural recursion clauses, corresponding to the clauses **T1**–**T5** of the definition of expressions, and the proof of (c1)–(c3) in each case is easy.

V1. $val(x, \pi) \simeq \pi(x)$.

V2. $val(p(t_1, \ldots, t_n), \pi) \simeq \pi(p)(val(t_1, \pi), \ldots, val(t_n, \pi))$.

V3. $val(\lambda(\mathbf{u})t, \pi) = \lambda(u)val(t, \pi[\mathbf{u}/u])$, where $\pi[\mathbf{u}/u]$ is the assignment which agrees with π on all variables, except on the basic variables in the list $\mathbf{u}$, on which it is defined so that it assigns u to $\mathbf{u}$.

V4. $val(\mathsf{f}[t_1, \ldots, t_n], \pi) \simeq \mathscr{F}(\mathsf{f})[val(t_1, \pi), \ldots, val(t_n, \pi)]$.

V5. If $t \equiv \mathsf{rec}(\mathbf{u}_1, \mathbf{p}_1, \ldots, \mathbf{u}_n, \mathbf{p}_n)[t_0, t_1, \ldots, t_n]$, let for $i = 0, \ldots, n$

$$h_i[u_i, p_1, \ldots, p_n] \simeq val(t_i, \pi[\mathbf{u}_i/u_i, \mathbf{p}_1/p_1, \ldots, \mathbf{p}_n/p_n]),$$

where the assignment on the right is obtained from π by changing the value just on the variables $\mathbf{u}_i, \mathbf{p}_1, \ldots, \mathbf{p}_n$ to make it $u_i, p_1, \ldots, p_n$ respectively. (Notice that $\mathbf{u}_0$ is the empty individual variable.) Now by (c2) of the induction hypothesis, each h_i is a monotone functional, we can show by familiar methods that the system of equations

$$h_i[u_i, p_1, \ldots, p_n] \simeq p_i(u_i) \qquad (i = 0, 1, \ldots, n)$$

has a sequence $\bar{p}_0, \bar{p}_1, \ldots, \bar{p}_n$ of *least simultaneous solutions*, and if we set

$$\begin{aligned} val(t, \pi) &\simeq \bar{p}_0(\ ) \\ &\simeq val(t_0, \pi[\mathbf{p}_1/\bar{p}_1, \ldots, \mathbf{p}_n/\bar{p}_n]), \end{aligned}$$

then $val(t, \pi)$ is monotone in π.

DEFINITION 1D.3. Let $\mathbf{x} = \mathbf{x}_1, \ldots, \mathbf{x}_n$ be a sequence of (basic and pf) variables which includes all the free variables of a term t. The *denotation* of t, in the functional structure $\mathbf{A}$, relative to $\mathbf{x}$ is the functional

$$den(\mathbf{x}, t)\colon X \rightharpoonup W, \qquad den(\mathbf{x}, t)(x) \simeq val(t, \pi_x),$$

where X is the product space with type that of the sequence $\mathbf{x}$, W is the basic set with type that of the term t and for each $x = x_1, \ldots, x_n$, π_x is any assignment which assigns x_i to $\mathbf{x}_i$, for $i = 1, \ldots, n$.

To give just one concrete example in the structure $\mathbf{N}$, let

$$\begin{aligned}(1D.4)\qquad \mathbf{x} \cdot \mathbf{y} \equiv\ & \mathsf{rec}(i, times, j, plus)\\ & [times(\mathbf{y}),\\ & \mathsf{if\ zero}\ [i]\ \mathsf{then}\ 0\ \mathsf{else}\ plus(times(\mathsf{pred}[i])),\\ & \mathsf{if\ zero}\ [j]\ \mathsf{then}\ \mathbf{x}\ \mathsf{else}\ \mathsf{succ}[plus(\mathsf{pred}[j])]].\end{aligned}$$

The denotation of the term $\mathbf{x} \cdot \mathbf{y}$ on the variables $\mathbf{x}, \mathbf{y}$ is the function which assigns to each x, y the value of the principal (first) least fixed point of the equations

$$\begin{aligned} p_0(\ ) &\simeq times(y),\\ times(i) &\simeq \mathsf{if}\ i = 0\ \mathsf{then}\ 0\ \mathsf{else}\ plus(times(i - 1)),\\ plus(j) &\simeq \mathsf{if}\ j = 0\ \mathsf{then}\ x\ \mathsf{else}\ plus(j - 1) + 1,\end{aligned}$$

i.e. the product of x and y.

DEFINITION 1D.5. Two terms are *denotationally equivalent* if they take the same value for all assignments, on all structures,

$$s \sim_{den} t \Leftrightarrow (\forall \mathbf{A}, \pi \text{ in } \mathbf{A}) val(t, \pi) \simeq val(s, \pi).$$

Fact 1D.6 (The denotational replacement property). (1) *If $t(x)$ and w are terms so that the substitution of w for the basic variable x in $t(x)$ makes sense and is free, then for every assignment π, in every structure,*

$$val(t(w), \pi) \simeq val(t(x), \pi[x/val(w, \pi)]);$$

hence for all $t_1(x)$, $t_2(x)$, w_1, w_2:

$$[t_1(x) \sim_{den} t_2(x)\ \&\ w_1 \sim_{den} w_2] \Rightarrow t_1(w_1) \sim_{den} t_2(w_2).$$

(2) *Similarly, if the substitutions make sense and are free, then for every structure* $\mathbf{A}$,

$$\mathsf{f} = den(\mathbf{x}, w(\mathbf{x})) \Rightarrow (\forall t(\mathsf{f}), \pi)[val(t(\mathsf{f}), \pi) \simeq val(t(\lambda(x)w(x)), \pi)];$$

hence for all $t_1(\mathsf{f}), t_2(\mathsf{f}), w_1(x), w_2(x)$,

$$[t_1(\mathsf{f}) \sim_{den} t_2(\mathsf{f})\ \&\ w_1(x) \sim_{den} w_2(x)] \Rightarrow t_1(\lambda(x)w_1(x)) \sim_{den} t_2(\lambda(x)w_2(x)).$$

The proof of the first assertion in each part is by a routine induction on $t(x)$ and $t(\mathsf{f})$, and the corollary claims follow immediately.

Notice that the two terms

$$t_1(x) \equiv \mathsf{if}\ x\ \mathsf{then}\ 1\ \mathsf{else}\ 1, \qquad t_2(x) \equiv 1,$$

take the same value (1) for all assignments *which are defined on the Boolean variable x*, but if $val(w,\pi)\uparrow$, then $val(t_1(w),\pi)\uparrow$, while $val(t_2(w),\pi) = 1$; it follows that we must allow partially defined assignments if we want the replacement property to hold.

§1E. Congruence and special terms. In languages with variable-binding operators, expressions which differ only by an alphabetic change in their bound variables are equivalent in all respects. Here we must introduce a coarser notion of congruence between terms because of the operator rec; it is natural to disregard the vacuous quantifer rec( ), and to identify recursive terms which differ only by the order of their parts (other than the output part).

DEFINITION 1E.1. *Congruence* is the least equivalence relation $\equiv_c$ on the class of expressions which satisfies the following:

C1. For every pf variable p, $p \equiv_c \lambda(u)p(u)$.

C2. $[t_1 \equiv_c t'_1, \ldots, t_n \equiv_c t'_n] \Rightarrow p(t_1,\ldots,t_n) \equiv_c p(t'_1,\ldots,t'_n)$.

C3. If $z^1,\ldots,z^m$ are fresh basic variables (which do not occur in t or t'), then

$$\begin{aligned}&\mathrm{subst}(t,u^1/z^1,\ldots,u^m/z^m) \equiv_c \mathrm{subst}(t',v^1/z^1,\ldots,v^m/z^m)\\ &\quad\Rightarrow \lambda(u^1,\ldots,u^m)t \equiv_c \lambda(v^1,\ldots,v^m)t'.\end{aligned}$$

C4. $[t_1 \equiv_c t'_1,\ldots,t_n \equiv_c t'_n] \Rightarrow \mathsf{f}[t_1,\ldots,t_n] \equiv_c \mathsf{f}[t'_1,\ldots,t'_n]$.

C5a. $\mathrm{rec}(\ )[t] \equiv_c t$.

C5b. $\mathrm{rec}(u_1,\ldots,u_n)[t_0,t_1,\ldots,t_n] \equiv_c \mathrm{rec}(v_1,q_1,\ldots,v_n,q_n)[t'_0,t'_1,\ldots,t'_n]$ if there exists a permutation σ on $\{0,1,\ldots,n\}$, with inverse ρ and $\sigma(0) = 0$, such that the following two conditions hold:

(a) For $i = 1,\ldots,n$, the pf variables $p_{\sigma(i)}$ and q_i have the same type.

This implies by the type restrictions on clause **T4** that the individual variables $u_{\sigma(i)}$ and v_i also have the same type.

(b) If $r_1,\ldots,r_n$ are fresh pf variables and $z_1,\ldots,z_n$ fresh sequences of basic variables with types so that the substitutions below make sense, then for $i = 0,1,\ldots,n$, the term

$$\mathrm{subst}(t'_{\rho(i)}, v_{\rho(i)}/z_{\rho(i)}, q_1/r_1,\ldots,q_n/r_n)$$

is congruent with the term

$$\mathrm{subst}(t_i, u_i/z_{\rho(i)}, p_{\sigma(1)}/r_1,\ldots,p_{\sigma(n)}/r_n).$$

For example,

$$\mathrm{rec}(u_1,p_1,u_2,p_2,u_3,p_3)[t_0,t_1,t_2,t_3] \equiv_c \mathrm{rec}(u_2,p_2,u_3,p_3,u_1,p_1)[t_0,t_2,t_3,t_1],$$

taking $\sigma(1) = 2$, $\sigma(2) = 3$, $\sigma(3) = 1$.

Fact 1E.2. *If $t \equiv_c s$, then every pf or basic variable x has the same number of free occurences in t and in s.*

Fact 1E.3. *If $t \equiv_c s$, then, on every functional structure* **A** *and for every assignment π into* **A**, $val(t,\pi) \simeq val(s,\pi)$.

The proof is by induction on the definition of terms, a bit messy in case **T4**. ⊣

Almost all the properties of expressions we will study are invariant under congruence, and it is sometimes convenient to consider congruent expressions as identical.

In §1D.4 of ARFTA we defined terms for REC in a restricted way, so that basic variables were not terms. As a consequence (in particular) the *identity function* on each basic set was not (automatically) defined by a term. For FLR we have stayed closer to familiar terminology by calling "terms" the wider class of syntactical objects which includes the basic variables. Still, it is important to identify the subclass of terms which will define algorithms, and we will call these here "special".

1E.4. *Special terms.* A term t is *special* if it is of the form $p(t_1,\dots,t_n)$, $\mathsf{f}[t_1,\dots,t_n]$, or (recursively) $\mathsf{rec}(u_1,p_1,\dots,u_n,p_n)[s_0,\dots,s_n]$, where $s_0,\dots,s_n$ are all special.

It is easy to assign (by recursion on **T1**–**T5**) to each term t a sequence $v_1,\dots,v_n$ of specific occurrences of basic variables which "prevent" t from being special; we can then obtain special terms from t by substituting special terms $t_1,\dots,t_n$ for $v_1,\dots,v_n$. For example, we can obtain a special term defining multiplication in **N** by replacing the single offending occurrence of the free variable **x** in the term of (1D.4) by the following recursive special term which defines the identity on **N**:

(1E.5) $\mathsf{id}[\mathbf{x}] \equiv \mathsf{rec}(i,p)[p(\mathbf{x}), \mathsf{if\ zero}\ [i]\ \mathsf{then}\ 0\ \mathsf{else}\ \mathsf{succ}[p(\mathsf{pred}[i])]]$.

Fact 1E.6. *If* $s \equiv_c t$ *and* s *is special, then so is* t.

The question whether the identity should be assumed automatically as defining (directly and uniquely) an algorithm on every functional structure was discussed briefly in §2C of ARFTA. We will come back to it in [Moschovakis alg].

§1F. Recursive functionals.

DEFINITION 1F.1. A functional $f\colon X \rightharpoonup W$ on the universe of a functional structure **A** is **A**-*recursive* if it is the denotation of a special term of FLR, relative to some sequence of variables.

On the structure **N** these are the classical recursive functionals of [Kleene 1952]. The study of these functionals in various structures, or (globally) on classes of structures of the same signature, is the proper subject of *generalized* or *abstract recursion theory*, which has been developed extensively, both by logicians (mostly on infinite structures) and by theoretical computer scientists (mostly on classes of finite structures). It should be pointed out that there are many different approaches to abstract recursion theory in the literature, and it is not entirely trivial to show that the various notions of "recursive function(al)s" which have been introduced all fall naturally (by choosing **A** carefully) under the present concept. Some results of this type are mentioned in ARFTA.

It is easy to find examples of functions which are denotations of terms but are not recursive, because they cannot be defined by a special term. More interesting are the cases of recursive functions which can only be defined nontrivially, by a special term which expresses some of their natural intensional properties, as in the case of the identity on **N** above. A similar example is that of the identity on the structure **V** of sets, which is defined by the special term

$$\mathsf{id}[x] \equiv \mathsf{rec}(i,p)[p(x), \mathsf{rep}[i,p]].$$

The intensional theory associates with each term and argument where its denotation is defined an *ordinal stage* which represents "the length of the computation" expressed by the term. In this case, as one might expect, *stage*(x) = (the set-theoretic rank of x) + 1.

2. Reduction in FLR

§2A. Introduction. In this (main) part of the paper, we will define a calculus of *reduction* and *intensional equivalence* for FLR, and we will prove that every term t can be reduced to a unique (up to congruence) equivalent term t^*, which is *irreducible*. The intended interpretation is that

(2A.1) $t \sim s \Leftrightarrow t$ and s define the same algorithm,

(2A.2) $t \to s \Leftrightarrow s$ defines "more directly" the algorithm of t,

so that irreducible (special) terms define algorithms directly in terms of the givens.

To collect evidence for these claims, we must produce a precise, mathematical definition (a modelling) for the notion of algorithm, argue that it expresses adequately our intuitive understanding of the concept, develop an *intensional semantics* for FLR which associates an algorithm with each term, and then prove outright at least (2A.1)—the second claim (2A.2) presumably being obvious. This we will take up in [Moschovakis alg] and subsequent papers of this sequence.

The results of this part are purely syntactical and can be understood (formally) without a precise knowledge of the intended intensional models. To motivate the definitions, however, it is worth discussing here briefly and intuitively an important issue which comes up in any attempt to interpret terms by algorithms. For the example we will assume a relatively clear intuition of *pure algorithms* which have no side effects, no dependence on a "state" and can be combined freely in parallel with each other. The phenomenon illustrated, however, is quite general.

Consider the two terms $f[t]$ and $\mathsf{rec}(p)[f[p(\,)], t]$, suppose we have an algorithm $\hat{f}$ which computes the given unary function f, and suppose we have already defined an algorithm $\hat{t}$ which computes the value of the term t. Assuming that these algorithms may be combined in parallel, there is a natural way to compute the value of the term $f[t]$: *start both f and $\hat{t}$; if f needs the value of t before $\hat{t}$ has computed it, have $\hat{f}$ wait; if (and when) $\hat{t}$ produces a value pass it to $\hat{f}$; if $\hat{f}$ produces a value give it as the output.* Similarly, there is a natural way to compute the value of $\mathsf{rec}(p)[f[p(\,)], t]$, assuming again that recursive definitions may be run concurrently: *start both $\hat{f}$ and $\hat{t}$; if $\hat{f}$ needs the value of $p(\,)$ before $\hat{t}$ has computed a value, have $\hat{f}$ wait; if (and when) $\hat{t}$ produces a value, call it $p(\,)$ and pass it to $\hat{f}$; if $\hat{f}$ produces a value give it as the output.*

On the basis of this analysis, one may argue that these two terms are computed by (essentially) the same algorithm, in symbols $f[t] \sim \mathsf{rec}(p)[f[p(\,)], t]$; taking the special case when $t \equiv r(\,)$, we then have

(2A.3) $$f[r(\,)] \sim \mathsf{rec}(p)[f[p(\,)], r(\,)].$$

But there is a problem with (2A.3). Assume (for simplicity) that it takes just one *step* (timeunit) to compute the value of each $f[x]$—and of course $r(\,)$—and apply (2A.3) to $f[p(\,)]$: if we replace this term by its alleged equivalent within the scope of rec, we get

(2A.4) $$f[r(\,)] \sim \mathsf{rec}(p)[\mathsf{rec}(q)[f[q(\,)], p(\,)], r(\,)]$$

which is not true, even though the two sides obviously have the same value. One

intensional difference between these two terms is that the natural algorithm we would assign (by the same analysis) to the right-hand side will need three steps to produce a value, while the one for the left-hand side will produce a value in two steps.

The problem is that the algorithm assigned to the recursive term on the right of (2A.3) already assumes the identification $p(\ ) \simeq r(\ )$ and does not assign any "cost for the call" involved in this identification; while the heart of the matter in the algorithmic reading of $f[t]$ is that a timeunit is assumed as "the cost of the call" for the value of t.

Put another way, one intensional difference between $f[r(\ )]$ on the left of (2A.3) and $f[p(\ )]$ on the right is that the latter lies within the scope of the variable-binding operator $\mathsf{rec}(p)$ and is being computed in a different *context* than $f[r(\ )]$, specifically the context where p is a recursion variable. The equivalence (2A.3) was "justified" by an intensional reading of its two sides *in the empty context.*

§2B. The reduction calculus. According to this analysis, a term may express different algorithms depending on where it occurs within a larger term—and specifically on which of its free pf variables are within the scope of a rec quantifier in the larger term.

DEFINITION 2B.1. A *context* is any set E of variables; *the full context* is the set of all variables.

For example, in order to specify *in the empty context* the algorithm defined by a recursive term

$$\mathsf{rec}(u_1, p_1, \ldots, u_n, p_n)[t_0, \ldots, t_n],$$

we will need the algorithm defined by each t_i in the nontrivial context $E_i = \{u_i, p_1, \ldots, p_n\}$.

DEFINITION 2B.2. 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 $p, v_1, \ldots, v_n \in E \cup \{u^1, \ldots, u^m\}$.

The motivation is that if we are computing a function f in a context E, then the pf variables in E name other functions which are being computed concurrently with f in a joint recursion, and the basic variables in E are the *local variables* of the recursion.

DEFINITION 2B.3. *Reduction in* E, $\rightarrow_E$, and *equivalence in* E, $\sim_E$, are defined by the recursion clauses **R1**–**R11** in the table, where s, t are terms, E is a context, $\bar{a}$, $\bar{b}$ are (possibly empty) sequences of expressions, the obvious typing restrictions are assumed so that the expressions on either side of $\rightarrow_E$ and $\sim_E$ are terms, and $r, r_1, \ldots, r_m$ are "fresh" variables in **R1**–**R5**.

R1–**R3** reduce explicit computation to (trivial) recursion, and could be motivated further by an elaboration of the discussion in the preceeding section; **R6** makes $\rightarrow_E$ coarser than $\equiv_c$; **R7** makes $\rightarrow_E$ transitive; **R9**–**R11** (with **R6**) define $\sim_E$ as the least equivalence relation which is coarser than $\rightarrow_E$.

The key clause **R8** allows for the reduction of nontrivial recursive terms, given reducts of their parts in the relevant, enlarged context.

TABLE. THE REDUCTION AND EQUIVALENCE RULES

R1 $p(\bar{a}, t, \bar{b}) \to_E \mathsf{rec}(r)[p(\bar{a}, r(\ ), \bar{b}), t]$ (t not imm. in E).

R2 $\mathsf{f}[\bar{a}, t, \bar{b}] \to_E \mathsf{rec}(r)[\mathsf{f}[\bar{a}, r(\ ), \bar{b}], t]$ (t not imm. in E).

R3 $\mathsf{f}[\bar{a}, \lambda(u)t, \bar{b}] \to_E \mathsf{rec}(u, r)[\mathsf{f}[\bar{a}, r, \bar{b}], t]$ ($\lambda(u)t$ not imm. in E).

R4 $\mathsf{rec}(u_1, \ldots, p_n)[\mathsf{rec}(v_1, \ldots, q_m)[s_0(q_1, \ldots, q_m), \ldots, s_m(q_1, \ldots, q_m)], t_1, \ldots, t_n]$
$\to_E \mathsf{rec}(v_1, r_1, v_2, r_2, \ldots, v_m, r_m, u_1, p_1, \ldots, p_n)$
$[s_0(r_1, \ldots, r_m), \ldots, s_m(r_1, \ldots, r_m), t_1, t_2, \ldots, t_n]$.

R5 $\mathsf{rec}(u_1, \ldots, p_n)[t_0, \mathsf{rec}(v_1, \ldots, q_m)[s_0(q_1, \ldots, q_m), \ldots, s_m(q_1, \ldots, q_m)], t_2, \ldots, t_n]$
$\to_E \mathsf{rec}(u_1, p_1, u_1, v_1, r_1, u_1, v_2, r_2, \ldots, u_1, v_m, r_m, u_2, p_2, \ldots, p_n)$
$[t_0, s_0(r_1(u_1, \cdot), \ldots, r_m(u_1, \cdot)), \ldots, s_m(r_1(u_1, \cdot), \ldots, r_m(u_1, \cdot)), t_2, \ldots, t_n]$.

R6 $s_1 \equiv_c s_2 \Rightarrow s_1 \to_E s_2$.

R7 $s_1 \to_E s_2, s_2 \to_E s_3 \Rightarrow s_1 \to_E s_3$.

R8 If $J(i) = E \cup \{u_i, p_1, \ldots, p_n\}$, then
$s_0 \to_{J(0)} t_0, \ldots, s_n \to_{J(n)} t_n$
$\Rightarrow \mathsf{rec}(u_1, \ldots, p_n)[s_0, \ldots, s_n] \to_E \mathsf{rec}(u_1, \ldots, p_n)[t_0, \ldots, t_n]$.

R9 $s_1 \to_E s_2 \Rightarrow s_1 \sim_E s_2$.

R10 $s_1 \sim_E s_2 \Rightarrow s_2 \sim_E s_1$.

R11 $s_1 \sim_E s_2, s_2 \sim_E s_3 \Rightarrow s_1 \sim_E s_3$.

Finally, the most crucial clauses are **R4** and **R5**, which reduce the nesting of the rec quantifier in terms. It is a bit easier to decipher their special cases with $n = m = 1$, which have fewer dots:

(a) $\mathsf{rec}(u, p)[\mathsf{rec}(v, q)[s_0(q), s_1(q)], t_1] \to_E \mathsf{rec}(v, r, u, p)[s_0(r), s_1(r), t_1]$,

(b) $\mathsf{rec}(u, p)[t_0, \mathsf{rec}(v, q)[s_0(q), s_1(q)]] \to_E \mathsf{rec}(u, p, u, v, r)[t_0, s_0(r(u, \cdot)), s_1(r(u, \cdot))]$,

where $r(u, \cdot) \equiv \lambda(v)r(u, \cdot)$. One may give a pretty convincing, intuitive justification for these reductions based on an informal notion of algorithm, but we will not take the space to do this here. On the other hand, it is important to point out that they—and all the other reductions—preserve values.

THEOREM 2B.4. *For each functional structure* **A**, *assignment* π *into* **A** *and context* E,

$$s \sim_E t \Rightarrow val(s, \pi) \simeq val(t, \pi).$$

PROOF. We show by induction on **R1**–**R11** that both $\to_E$ and $\sim_E$ preserve values, the only nontrivial case being **R5**. We will outline the argument for the special case (b) of **R5** above, which is well known (in various guises) as the reduction of simultaneous to iterated least-fixed-point-recursion: it is called the *Bekič-Scott principle* in [Park 1980].

Showing all the variables which are relevant, the two sides of (b) look as follows:

$$LHS \equiv \mathsf{rec}(u, p)[t_0(p), \mathsf{rec}(v, q)[s_0(u, p, q), s_1(u, p, v, q)]],$$
$$RHS \equiv \mathsf{rec}(u, p, u, v, r)[t_0(p), s_0(u, p, r(u, \cdot)), s_1(u, p, v, r(u, \cdot))].$$

We fix the values of any other free variables which may occur in these terms and use the same symbols $t_0, s_0, \ldots$ to name the (functional) denotations of the terms.

Define the functional

$$F(u,p) = \mu q[(\forall v)s_1(u,p,v,q) \simeq q(v)] \tag{2B.5}$$
$$= \text{the least pf } q \text{ such that}$$
$$(\forall v,w)[s_1(u,p,v,q) \simeq w \Rightarrow q(v) \simeq w];$$

then (easily) *F is a monotone functional*, we can set

$$\bar{p} = \mu p[(\forall u)s_0(u,p,F(u,p)) \simeq p(u)], \tag{2B.6}$$

and by the definition $val(LHS) \simeq t_0(\bar{p})$. Similarly, if $\tilde{p},\tilde{r}$ are the least partial functions which satisfy for all u,v the equations

$$s_0(u,p,r(u,\cdot)) \simeq p(u), \tag{2B.7}$$

$$s_1(u,p,v,r(u,\cdot)) \simeq r(u,v), \tag{2B.8}$$

then $val(RHS) \simeq t_0(\tilde{p})$, so that we only need prove that $\bar{p} = \tilde{p}$.

Let $\bar{r}(u,v) \simeq F(u,\bar{p})(v)$ and notice that from (2B.6) and (2B.5) we have

$$s_0(u,\bar{p},\bar{r}(u,\cdot)) \simeq \bar{p}(u), \qquad s_1(u,\bar{p},v,\bar{r}(u,\cdot)) \simeq \bar{r}(u,v),$$

so that by the characterization of $\tilde{p}$ and $\tilde{r}$ as the least fixed points of (2B.7) and (2B.8), we get $\tilde{p} \subseteq \bar{p}$ and $\tilde{r} \subseteq \bar{r}$.

To prove the converse, for each u let

$$\tilde{q}_u(v) \simeq \tilde{r}(u,v) \simeq s_1(u,\tilde{p},v,\tilde{q}_u),$$

with the second equation coming from (2B.8); hence, by the definition of F,

$$F(u,\tilde{p}) \subseteq \tilde{q}_u, \tag{2B.9}$$

and, using the monotonicity of s_0,

$$\begin{aligned} s_0(u,\tilde{p},F(u,\tilde{p})) &\simeq w \\ &\Rightarrow s_0(u,\tilde{p},\tilde{q}_u) \simeq w \\ &\Rightarrow s_0(u,\tilde{p},\tilde{r}(u,\cdot)) \simeq w \\ &\Rightarrow \tilde{p}(u) \simeq w \quad \text{(by (2B.7))}, \end{aligned}$$

so that by the characterization of $\bar{p}$ we have $\bar{p} \subseteq \tilde{p}$ and the argument is complete. ⊣

We will end this section with some simple results which relate $\rightarrow_E$ and $\sim_E$ for varying contexts E. We let $\rightarrow_f$ denote *reduction in the full context*,

$$s \rightarrow_f t \Leftrightarrow s \rightarrow_{\{\text{all variables}\}} t, \tag{2B.10}$$

and similarly for *equivalence in the full context*, $\sim_f$.

Fact 2B.11. *Let s, t be terms and E, F contexts.*

(1) $[s \rightarrow_E t \ \& \ E \supseteq F] \Rightarrow s \rightarrow_F t$, *so that in particular, for all E,*

$$s \rightarrow_f t \Rightarrow s \rightarrow_E t \Rightarrow s \rightarrow_\varnothing t.$$

(2) *There are s, t, E such that $s \rightarrow_\varnothing t$ but not $s \rightarrow_E t$.*

(3) *If the variable z does not occur free in s, then for every t,*

$$s \to_E t \Rightarrow s \to_{E \cup \{z\}} t.$$

(4) *If E contains all the free variables of s and $s \to_E t$, then E contains all the free variables of t and $s \to_f t$.*

PROOF. (1) If $E \supseteq F$, then every expression which is not immediate in E is also not immediate in F, so that (directly from the definitions) every basic reduction valid in E is also valid in F.

(2) $p(q(\ )) \to_\varnothing \mathsf{rec}(r)[p(r(\ )), q(\ )]$, but if $q \in E$, then no reduction applies to $p(q(\ ))$ relative to E.

(3) Use induction on $\to_E$, noticing that if z does not occur in s, then every subexpression of s which is not immediate in E is also not immediate in $E \cup \{z\}$.

(4) Check first by an easy induction on $\to_E$ that if $s \to_E t$, then every variable has the same number of free occurrences in s and in t. Another easy induction verifies that if E contains all the free variables of s, $E \subseteq F$ and $s \to_E t$, then $s \to_F t$. Using this and (1), we then get, for arbitrary F,

$$s \to_E t \Rightarrow s \to_{E \cup F} t \Rightarrow s \to_F t. \qquad \dashv$$

DEFINITION 2B.12. A term s is *irreducible in the context E*—or plain *irreducible* when $E = \{\text{all variables}\}$—if, for every t, $s \to_E t \Rightarrow s \equiv_c t$.

Recall that a *recursive term* is one which begins with the quantifier rec while an *explicit expression* is one in which the quantifier rec does not occur at all. The next definition helps describe easily the class of irreducible terms.

DEFINITION 2B.13. An expression t is *simplified* if either the vacuous recursion quantifier rec( ) does not occur in t, or $t \equiv \mathsf{rec}(\ )s$, and rec( ) does not occur in s.

Fact 2B.14. (1) *If a term t is irreducible in a context F, it is irreducible in every larger context $E \supseteq F$.*

(2) *Every term t is congruent to a simplified term, and to a recursive simplified term.*

(3) *A simplified nonrecursive term t is irreducible in E if and only if it is a basic variable or in one of the forms $p(t_1, \ldots, t_n)$ or $\mathsf{f}[t_1, \ldots, t_n]$, where $t_1, \ldots, t_n$ are immediate in E and simplified, so that in particular t is explicit. (These forms include* 1, 0, *and* if s then t_1 else t_2 *with s, t_1, t_2 immediate in E and simplified.)*

(4) *A simplified recursive term*

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

is irreducible in E if and only if each t_i is a simplified explicit term, irreducible in $E \cup \{u_i, p_1, \ldots, p_n\}$.

Proof. (1) is immediate from (2B.11) and (2) is trivial: simply delete all vacuous rec( ) from a term to get a congruent, simplified term, and if this is not recursive, add a rec( ) in the front. (3) and (4) follow directly from an examination of the reductions. $\dashv$

It is not hard to see directly that every term can be reduced to some simplified, irreducible term. This, however, will be an immediate consequence of the proof we will give in the next section of the uniqueness of the final reduct, so we will not give a separate argument here.

§2C. Normal forms. In this section we will define a syntactic transformation on terms

$$t \mapsto \mathsf{nf}(t, E) \equiv \mathsf{rec}(u_1, p_1, \ldots, u_n, p_n)[t_0, t_1, \ldots, t_n], \tag{2C.1}$$

such that $\mathsf{nf}(t, E)$ is a simplified, recursive term, irreducible in E, and $t \to_E \mathsf{nf}(t, E)$. We will then prove in the next section that (up to congruence) $\mathsf{nf}(t, E)$ is the unique irreducible in E term t^* such that $t \to_E t^*$.

The definition of $\mathsf{nf}(t, E)$ is naturally by recursion on t, and its most interesting case is when t is a recursive term, when we must combine several nested recursions into one. For this we will use a syntactic transformation which combines several applications of the reduction rules **R4** and **R5**.

Using the notation conventions of 1A for sequences, we can specify an arbitrary recursive term by

$$t \equiv \mathsf{rec}(u_i, p_i\colon i \leq' n)[t_i\colon i \leq n],$$

and rule **R5** can be expressed by

$$\begin{aligned}
&\mathsf{rec}(u_i, p_i\colon i \leq' n)[t_0(p_1, \ldots, p_n),\\
&\qquad \mathsf{rec}(v_j, q_j\colon j \leq' m)[s_j(q_1, \ldots, q_m, p_1, \ldots, p_n)\colon j \leq m],\\
&\qquad\qquad \langle t_i(p_1, \ldots, p_n)\colon 2 \leq i \leq n\rangle]\\
&\to_E \mathsf{rec}(\langle u_1, v_j, r_j\colon j \leq m\rangle, \langle u_i, p_i\colon 2 \leq i \leq n\rangle)\\
&\qquad [t_0(r_0, p_2, \ldots, p_n), \langle s_j(r_1(u_1, \cdot), \ldots, r_m(u_1, \cdot), r_0, p_2, \ldots, p_n)\colon j \leq m\rangle,\\
&\qquad\qquad \langle t_i(r_0, p_2, \ldots, p_n)\colon 2 \leq i \leq n\rangle].
\end{aligned}$$

Notice that the pf variable r_0 occurs in the prefix of the right-hand side and has been substituted for p_1 in the parts; it "names" s_0.

DEFINITION 2C.2. *The operation* rc. Suppose t is a *doubly recursive term*, i.e.

$$t \equiv \mathsf{rec}(u_i, p_i\colon i \leq' n)[t_i(p_1, \ldots, p_n)\colon i \leq n], \tag{2C.3}$$

and for $i \leq n$,

$$\begin{aligned}
t_i(p_1, \ldots, p_n) \equiv \mathsf{rec}&(v_{ij}, q_{ij}\colon j \leq' m(i))\\
&[t_{ij}(q_{i1}, \ldots, q_{im(i)}, p_1, \ldots, p_n)\colon j \leq m(i)].
\end{aligned} \tag{2C.4}$$

The *recursive combination* of t is the recursive term

$$\begin{aligned}
\mathsf{rc}(t) \equiv \mathsf{rc}\, t \equiv \mathsf{rec}&(u_i, v_{ij}, r_{ij}\colon i \leq n, j \leq m(i), (i, j) \neq (0, 0))\\
&[t_{ij}(r_{i1}(u_i, \cdot), \ldots, r_{im(i)}(u_i, \cdot), r_{10}, \ldots, r_{n0})\colon i \leq n, j \leq m(i)],
\end{aligned}$$

where the pf variables r_{ij} are fresh and

$$r_{ij}(u_i, \cdot) \equiv \lambda(v_{ij}) r_{ij}(u_i, v_{ij}).$$

We must check that $\mathsf{rc}(t)$ is indeed a term, i.e. that the indicated substitutions respect types. Notice that the prefix $\mathsf{rec}(u_i, v_{ij}, r_{ij}\colon i \leq n, j \leq m(i), (i, j) \neq (0, 0))$ implies that the types of the variables match so that $r_{ij}(u_i, v_{ij})$ makes sense; thus $r_{ij}(u_i, \cdot)$ is a λ-term which can be substituted for q_{ij} in $t_{ij}(q_{i1}, \ldots, q_{im(i)}, p_1, \ldots, p_n)$, since the prefix $\mathsf{rec}(v_{ij}, q_{ij})$ in (2C.4) implies that $q_{ij}(v_{ij})$ is well-formed. A similar

argument justifies the substitution of r_{j0} for p_j ($j > 0$), since $v_{j0} = \varnothing$ and hence $r_{j0}(u_j, v_{j0}) \equiv r_{j0}(u_j)$ is well-formed, just as $p_j(u_j)$ is well-formed.

The basic property of the operation rc is that it combines several applications of the reductions **R4**, **R5** and **R6**.

Fact 2C.5. *For each doubly recursive term t as in* (2C.3), (2C.4) *and each context E,*

$$t \to_E \mathsf{rc}\, t.$$

Proof. Suppose t is as in (2C.3)–(2C.4) and apply **R5** to it in the formulation above, using the hypothesis that $u_0 = v_{io} = \varnothing$ for all $i \leq' n$, to get:

$$t \to_E \mathsf{rec}(\langle u_1, v_{1j}, r_{1j} : j \leq m(1)\rangle, \langle u_i, p_i : 2 \leq i \leq n\rangle)$$
$$[t_0, \langle t_{1j}(r_{10}(u_1, \cdot), \ldots, r_{1m(1)}(u_1, \cdot), r_{10}, p_2, \ldots, p_n) : j \leq m(1)\rangle, t_2, \ldots, t_n].$$

Now apply **R6** to get a congruent term where t_2 is the first part after the head, apply **R5** in the same way and then use congruence again to get

$$t \to_E \mathsf{rec}(\langle u_i, v_{ij}, r_{ij} : i \leq' 2, j \leq m(i)\rangle, \langle u_i, p_i : 3 \leq i \leq n\rangle)$$
$$[t_0, \langle t_{ij}(r_{i0}(u_i, \cdot), \ldots, r_{im(i)}(u_i, \cdot), r_{10}, r_{20}, p_3, \ldots, p_n) : i \leq' 2, j \leq m(i)\rangle, t_3, \ldots, t_n].$$

The result follows by doing this n times and then applying **R4** once to reduce t_0. ⊣

DEFINITION 2C.6. *Normal forms.* For each term t and each context E, the simplified recursive term $\mathsf{nf}(t, E)$—*the normal form of t in E*—is defined by recursion on t. *The (absolute) normal form of t* is the normal form of t in the empty context,

$$\mathsf{nf}(t) \equiv \mathsf{nf}(t, \varnothing), \tag{2C.7}$$

and *the full normal form of t* is the normal form of t in the full context,

$$\mathsf{fnf}(t) \equiv \mathsf{nf}(t, \{\text{all variables}\}). \tag{2C.8}$$

NF1. For a basic variable v, $\mathsf{nf}(v, E) \equiv \mathsf{rec}(\ )[v] \equiv \mathsf{rec}(\ )v$.

NF2. As a typical example of this case, suppose $t \equiv p(w, t^*)$, where w is immediate in E and t^* is not, and we have computed

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

where we may have $n = 0$ so that $\mathsf{nf}(t^*, E) \equiv \mathsf{rec}(\ )[t_0^*]$. We set

$$\mathsf{nf}(t, E) \equiv \mathsf{rec}(r, u_1, p_1, \ldots, u_n, p_n)[p(w, r(\ )), t_0, \ldots, t_n].$$

In the general case, where $t \equiv p(t_1, \ldots, t_n)$, we want to reduce those t_i's which are not immediate in E as we did with t^*, and leave those which are immediate in E as arguments. Notice that if $t_1, \ldots, t_n$ are all immediate in E, then

$$\mathsf{nf}(p(t_1, \ldots, t_n), E) \equiv \mathsf{rec}(\ )p(t_1, \ldots, t_n) \equiv_c p(t_1, \ldots, t_n).$$

NF4. As a typical example again, suppose

$$t \equiv \mathsf{f}[w, s, \lambda(u^1, \ldots, u^k)t^*],$$

where w is immediate in E, s is not immediate in E and $\lambda(u^1, \ldots, u^k)t^*$ is not

immediate in E. Here w may be a term or a λ-term. Suppose that

$$\mathsf{nf}(s, E) \equiv \mathsf{rec}(v_1, q_1, \ldots, v_m, q_m)[s_0, \ldots, s_m],$$
$$\mathsf{nf}(t^*, E \cup \{u^1, \ldots, u^k\}) \equiv \mathsf{rec}(z_1, p_1, \ldots, z_n, p_n)[t_0(p_1, \ldots, p_n), \ldots, t_n(p_1, \ldots, p_n)]$$

(where again some rec may be vacuous) and set

$$\mathsf{nf}(t, E) \equiv \mathsf{rec}(r_s, v_1, \ldots, q_m, r_t, u, z_1, r_1, u, z_2, \ldots, u, z_n, r_n)$$
$$[\mathsf{f}[w, r_s(\ ), r_t], s_0, \ldots, s_m, t_0', \ldots, t_n'],$$

where $u \equiv u^1, \ldots, u^k$ and, for $i = 1, \ldots, n$, $t_i' \equiv t_i(r_1(u, \cdot), \ldots, r_n(u, \cdot))$. In the general case $t \equiv \mathsf{f}[t_1, \ldots, t_n]$, we reduce in E those among the expressions $t_1, \ldots, t_n$ which are not immediate in E (like s and $\lambda(u)t^*$ above) and we construct the normal form as above, by plugging into f the immediate expressions and the "outputs" of the expressions which are reduced. Notice the introduction of the additional occurrences of the recursion variables u in the prefix above. Again, if all $t_1, \ldots, t_n$ are immediate in E, this definition gives

$$\mathsf{nf}(\mathsf{f}[t_1, \ldots, t_n], E) \equiv \mathsf{rec}(\ )\mathsf{f}[t_1, \ldots, t_n] \equiv_c \mathsf{f}[t_1, \ldots, t_n].$$

An interesting special case of **NF4** is when f is the conditional functional. To put down one example of this, when w is immediate in E and s, t are not,

$$\mathsf{nf}(s, E) \equiv \mathsf{rec}(v_1, q_1, \ldots, v_m, q_m)[s_0, \ldots, s_m],$$
$$\mathsf{nf}(t, E) \equiv \mathsf{rec}(u_1, p_1, \ldots, u_n, p_n)[t_0, \ldots, t_n],$$

we have

$$\mathsf{nf}(\mathsf{if}\ w\ \mathsf{then}\ s\ \mathsf{else}\ t, E)$$
$$\equiv \mathsf{rec}(r_s, v_1, \ldots, q_m, r_t, u_1, \ldots, p_n)$$
$$[\mathsf{if}\ w\ \mathsf{then}\ r_s(\ )\ \mathsf{else}\ r_t(\ ), s_0, \ldots, s_m, t_0, \ldots, t_n].$$

NF5. Of course this is the most interesting and complicated case, but we set up the notation for it in defining the operation rc. If

$$t \equiv \mathsf{rec}(u_1, p_1, \ldots, u_n, p_n)[t_0, \ldots, t_n],$$

set for $i = 0, \ldots, n$

$$t_i^* \equiv \mathsf{nf}(t_i, E \cup \{u_i, p_1, \ldots, p_n\}),$$
$$t^* \equiv \mathsf{rec}(u_1, p_1, \ldots, u_n, p_n)[t_0^*, \ldots, t_n^*];$$

now t^* is a doubly recursive term and we can set $\mathsf{nf}(t, E) = \mathsf{rc}\, t^*$. It is important in this case that each part t_i of the recursive term t is reduced in the enlarged context $E \cup \{u_i, p_1, \ldots, p_n\}$.

This completes the definition of normal form. We will conclude the section with a list of their most elementary syntactic properties, mostly skipping the proofs (which are easy).

THEOREM 2C.9. *For each term t and context E,* $\mathsf{nf}(t, E)$ *is a simplified recursive term, irreducible in E, and* $t \to_E \mathsf{nf}(t, E)$.

The proof is direct, by induction on the definition, using (2B.14) and (2C.5). ⊣

Fact 2C.10. (1) *A basic or pf variable v has n free occurrences in a term t if and only if it has n free occurrences in* $\mathsf{nf}(t, E)$.

(2) *If t is a special term, then so is* $\mathsf{nf}(t, E)$.

(3) *If t and t' are congruent, then* $\mathsf{nf}(t, E)$ *and* $\mathsf{nf}(t', E)$ *are congruent.*

Proof. (1) and (2) are trivial. The proof of (3) is by induction on t, and it is again completely trivial in the explicit cases. We outline the proof in Case **T5**, which is also easy but notationally messy.

Now

$$t \equiv \mathsf{rec}(u_1, p_1, \ldots, u_n, p_n)[t_0, \ldots, t_n], \qquad t' \equiv \mathsf{rec}(v_1, q_1, \ldots, v_n, q_n)[t'_0, \ldots, t'_n]$$

and there is a permutation $\rho\colon \{0, \ldots, n\} \to \{0, \ldots, n\}$ with $\rho(0) = 0$ such that s_i and $s'_{\rho(i)}$ are congruent, where s_i and $s'_{\rho(i)}$ are obtained trivially by substitutions into t_i and t'_i. By induction hypothesis then, the two terms

$$\mathsf{nf}(s_i, E) \equiv \mathsf{rec}(\cdots)[t_{i0}, \ldots, t_{im(i)}],$$
$$\mathsf{nf}(s'_{\rho(i)}, E) \equiv \mathsf{rec}(\cdots)[t'_{\rho(i)0}, \ldots, t'_{\rho(i)m(i)}]$$

are congruent, so for each i, there is a permutation $\rho_i\colon \{0, \ldots, m(i)\} \to \{0, \ldots, m(i)\}$ with $\rho_i(0) = 0$, such that s_{ij} and $s'_{\rho(i)\rho_i(j)}$ are congruent, where again these are obtained by trivial substitutions from t_{ij} and $t'_{\rho(i)\rho_i(j)}$. To show the congruence of $\mathsf{nf}(t, E)$ with $\mathsf{nf}(t', E)$, take the permutation $\rho(i, j) = (\rho(i), \rho_i(j))$ on $\{(i, j)\colon i \leq n, j \leq m(i)\}$ ordered lexicographically (so that $\rho(0, 0) = (\rho(0), \rho_0(0)) = (0, 0)$), and plug into the formula to show that the corresponding parts are congruent. ⊣

§2D. Uniqueness of normal form. The definition of the normal form of a term determines a "strategy" for applying the reduction rules in a specific order ("innermost first"), until an irreducible term is obtained. Here we will show that for every two terms t, s and context E,

$$s \to_E t \Rightarrow \mathsf{nf}(s, E) \equiv_c \mathsf{nf}(t, E),$$

which means that no matter which order we choose to apply the reduction rules, we will always reach the same irreducible term—up to congruence. The heart of the proof is a simple combinatorial lemma about iterates of the operation rc, which we will prove by brute force, for lack of a better method.

Fact 2D.1 (Lemma on recursion combinations). *Suppose*

$$s \equiv \mathsf{rec}(u_1, p_1, \ldots, u_n, p_n)[s_0, \ldots, s_n] \tag{2D.2}$$

is a triply recursive term, i.e. each s_i is a doubly recursive term. Then

$$(\mathsf{rc})^2 s \equiv \mathsf{rc}\{\mathsf{rec}(u_1, \ldots, p_n)[\mathsf{rc}\, s_0, \ldots, \mathsf{rc}\, s_n]\}. \tag{2D.3}$$

Proof. Rewriting the definition of rc of (2C.2) in more uniform notation, if

$$\begin{aligned} t \equiv\ & \mathsf{rec}(v_i, q_i\colon i \leq' n) \\ & [\mathsf{rec}(v_{ij}, q_{ij}\colon j \leq' n(i))[t_{ij}(q_{i1}, \ldots, q_{in(i)}, q_1, \ldots, q_n)\colon j \leq n(i)]\colon i \leq n], \end{aligned} \tag{2D.4}$$

then

$$\begin{aligned} \mathsf{rc}(t) \equiv\ & \mathsf{rec}(v_i, v_{ij}, \tilde{q}_{ij}\colon i \leq n, j \leq n(i), (i, j) \neq (0, 0)) \\ & [t_{ij}(\tilde{q}_{i1}(v_i, \cdot), \ldots, \tilde{q}_{in(i)}(v_i, \cdot), \tilde{q}_{10}, \ldots, \tilde{q}_{n0})\colon i \leq n, j \leq n(i)], \end{aligned} \tag{2D.5}$$

where $v_0 = \varnothing$, $v_{i0} = \varnothing$ for $i \leq n$, and we are using the abbreviated λ-notation

$$\tilde{q}_{ij}(v_i, \cdot) \equiv \lambda(v_{ij})\tilde{q}_{ij}(v_i, v_{ij}).$$

Suppose now that the given term is

$$s \equiv \mathsf{rec}(u_\alpha, p_\alpha\colon \alpha \leq' N)[s_\alpha(p_1, \ldots, p_N)\colon \alpha \leq N], \tag{2D.6}$$

where for each $\alpha \leq N$,

$$\begin{aligned} s_\alpha(p_1, \ldots, p_N) \equiv \mathsf{rec}(&u_{\alpha\beta},\ p_{\alpha\beta}\colon \beta \leq' N(\alpha)) \\ &[s_{\alpha\beta}(p_{\alpha 1}, \ldots, p_{\alpha N(\alpha)}, p_1, \ldots, p_N)\colon \beta \leq N(\alpha)], \end{aligned} \tag{2D.7}$$

and for each α and $\beta \leq N(\alpha)$,

$$\begin{aligned} &s_{\alpha\beta}(p_{\alpha 1}, \ldots, p_{\alpha N(\alpha)}, p_1, \ldots, p_N) \\ &\quad \equiv \mathsf{rec}(u_{\alpha\beta\gamma}, p_{\alpha\beta\gamma}\colon \gamma \leq' N(\alpha, \beta)) \\ &\qquad [s_{\alpha\beta\gamma}(p_{\alpha\beta 1}, \ldots, p_{\alpha\beta N(\alpha,\beta)}, p_{\alpha 1}, \ldots, p_{\alpha N(\alpha)}, p_1, \ldots, p_N)\colon \gamma \leq N(\alpha, \beta)]. \end{aligned} \tag{2D.8}$$

Computation of the right-hand side of (2D.3). Fix α and apply (2D.5) with $t \equiv s_\alpha$ and $t_{ij} \equiv s_{\alpha\beta\gamma}$. We set $i \equiv \beta$, $v_i \equiv u_{\alpha\beta}$, $q_i \equiv p_{\alpha\beta}$, $j \equiv \gamma$, $v_{ij} \equiv u_{\alpha\beta\gamma}$ and $q_{ij} \equiv p_{\alpha\beta\gamma}$, so that β varies from 0 to $N(\alpha)$, and for each β, γ varies from 0 to $N(\alpha, \beta)$. Then

$$\begin{aligned} &\mathsf{rc}\, s_\alpha(p_1, \ldots, p_N) \\ &\quad \equiv \mathsf{rec}(u_{\alpha\beta}, u_{\alpha\beta\gamma}, \tilde{p}_{\alpha\beta\gamma}\colon \beta \leq N(\alpha), \gamma \leq N(\alpha, \beta), (\beta, \gamma) \neq (0, 0)) \\ &\qquad [s_{\alpha\beta\gamma}(\tilde{p}_{\alpha\beta 1}(u_{\alpha\beta}, \cdot), \ldots, \tilde{p}_{\alpha\beta N(\alpha,\beta)}(u_{\alpha\beta}, \cdot), \tilde{p}_{\alpha 10}, \ldots, \tilde{p}_{\alpha N(\alpha)0}, \\ &\qquad\qquad p_1, \ldots, p_N)\colon \beta \leq N(\alpha), \gamma \leq N(\alpha, \beta)]. \end{aligned} \tag{2D.9}$$

To compute the right-hand side, we must reapply rc once more to the doubly recursive term

$$\mathsf{rec}(u_\alpha, p_\alpha\colon \alpha \leq' N)[\mathsf{rc}\, s_\alpha(p_1, \ldots, p_N)\colon \alpha \leq N].$$

Let us first make an alphabetic change of variables in (2D.9) which removes all the tildes from the subscripted variables p, and then compute, setting this time $i \equiv \alpha$, $v_i \equiv u_\alpha$, $q_i \equiv p_\alpha$, $j \equiv \beta\gamma$, $v_{ij} \equiv u_{\alpha\beta\gamma}$ and $q_{ij} \equiv p_{\alpha\beta\gamma}$:

$$\begin{aligned} RHS \equiv \mathsf{rec}(&u_\alpha, u_{\alpha\beta}, u_{\alpha\beta\gamma}, \tilde{p}_{\alpha\beta\gamma}\colon \alpha \leq N, \beta \leq N(\alpha), \gamma \leq N(\alpha, \beta), (\alpha, \beta, \gamma) \neq (0, 0, 0)) \\ &[s_{\alpha\beta\gamma}(\tilde{p}_{\alpha\beta 1}(u_\alpha, u_{\alpha\beta}, \cdot), \ldots, \tilde{p}_{\alpha\beta N(\alpha,\beta)}(u_\alpha, u_{\alpha\beta}, \cdot), \\ &\qquad \tilde{p}_{\alpha 10}(u_\alpha, \cdot), \ldots, \tilde{p}_{\alpha N(\alpha)0}(u_\alpha, \cdot), \\ &\qquad \tilde{p}_{100}, \ldots, \tilde{p}_{N00})\colon \alpha \leq N, \beta \leq N(\alpha), \gamma \leq N(\alpha, \beta)]. \end{aligned} \tag{2D.10}$$

Again, we can simplify this formula by removing the tildes, through an alphabetic change of bound variables.

Notice that the formula for rc calls for the following replacement of terms in the matrix: $q_{ij}(v_{ij}) \to \tilde{q}_{ij}(v_i, v_{ij})$. In the first application here with $i \equiv \beta$ and $j \equiv \gamma$ (and after we rename the variables to remove the tildes) this gives

$$p_{\alpha\beta\gamma}(u_{\alpha\beta\gamma}) \to p_{\alpha\beta\gamma}(u_{\alpha\beta}, u_{\alpha\beta\gamma}),$$

and in the second application with $i \equiv \alpha$ and $j \equiv \beta\gamma$ this gives the further

replacement

$$p_{\alpha\beta\gamma}(u_{\alpha\beta}, u_{\alpha\beta\gamma}) \to p_{\alpha\beta\gamma}(u_\alpha, u_{\alpha\beta}, u_{\alpha\beta\gamma}),$$

which is what we get by the symbolic functional substitution

$$p_{\alpha\beta\gamma} \to p_{\alpha\beta\gamma}(u_\alpha, u_{\alpha\beta}, \cdot).$$

Computation of the left-hand side of (2D.3). We must apply rc twice to the term

$$s \equiv \mathsf{rec}(u_\alpha, p_\alpha\colon \alpha <' N)[s_\alpha(p_1, \ldots, p_N)\colon \alpha \leq N],$$

where the s_α's are defined above. In the first application we are setting $i \equiv \alpha$, $v_\alpha \equiv u_\alpha$, $j \equiv \beta$ and $v_{ij} \equiv u_{\alpha\beta}$, and we get

$$\begin{aligned}(2D.11)\qquad \mathsf{rc}\, s \equiv \mathsf{rec}(u_\alpha, u_{\alpha\beta}, \tilde{p}_{\alpha\beta}\colon \alpha \leq N, \beta \leq N(\alpha), (\alpha,\beta) \neq (0,0))\\ [s_{\alpha\beta}(\tilde{p}_{\alpha 1}(u_\alpha, \cdot), \ldots, \tilde{p}_{\alpha N(\alpha)}(u_\alpha, \cdot), \tilde{p}_{10}, \ldots, \tilde{p}_{N0})\colon \alpha \leq N, \beta \leq N(\alpha)].\end{aligned}$$

To get the left-hand side, we must reapply rc once more to this doubly recursive term, where the indexing of the parts is on the set $\{(\alpha,\beta)\colon \alpha \leq N, \beta \leq N(\alpha)\}$, lexicographically ordered. We set $i = \alpha\beta$, $v_i \equiv u_\alpha$, $u_{\alpha\beta}$, $j = \gamma$ and $v_j \equiv p_{\alpha\beta\gamma}$, and a careful plugging in gives for LHS exactly the same expression as the RHS in (2D.10), which completes the proof. ⊣

THEOREM 2D.12. *For each context E and all terms s, t,*

$$s \to_E t \Rightarrow \mathsf{nf}(s, E) \equiv_c \mathsf{nf}(t, E),$$

and hence

$$\begin{aligned} s \sim_E t &\Leftrightarrow \mathsf{nf}(s, E) \equiv_c \mathsf{nf}(t, E),\\ &\Leftrightarrow \textit{for some term } w,\ s \to_E w\ \&\ t \to_E w.\end{aligned}$$

PROOF. We must verify that each application of a reduction rule in the list **R1**–**R8** preserves the normal form. Let us call "part (*i*)" the proof for **R***i*, and to simplify notation, set $s \mapsto_E t \Leftrightarrow \mathsf{nf}(s, E) \equiv t$.

Part (8). Check first (easily) that for doubly recursive terms s^*, t^*,

$$s^* \equiv_c t^* \Rightarrow \mathsf{rc}\, s^* \equiv_c \mathsf{rc}\, t^*.$$

Now if $s_i \mapsto_{J(i)} s_i^*$, $t_i \mapsto_{J(i)} t_i^*$ for $i \leq n$, then by the induction hypothesis $s_i^* \equiv_c t_i^*$ $(i \leq n)$, and

$$\begin{aligned}\mathsf{nf}(\mathsf{rec}(u_1, \ldots, p_n)[s_0, \ldots, s_n], E) &\equiv \mathsf{rc}\ \mathsf{rec}(u_1, \ldots, p_n)[s_0^*, \ldots, s_n^*]\\ &\equiv_c \mathsf{rc}\ \mathsf{rec}(u_1, \ldots, p_n)[t_0^*, \ldots, t_n^*] \equiv \mathsf{nf}(\mathsf{rec}(u_1, \ldots, p_n)[t_0, \ldots, t_n], E).\end{aligned}$$

Part (6) is a restatement of (2C.10), and part (7) is trivial.

Part (5) follows from the following special case of the definition of rc:

$$\begin{aligned}&\mathsf{rc}\ \mathsf{rec}(u_1, \ldots, p_n)[\mathsf{rec}(\ )t_0^*, \mathsf{rec}(v_1, \ldots, q_m)[s_0^*(q_1, \ldots, q_m), \ldots, s_m^*(q_1, \ldots, q_m)],\\ &\qquad\qquad \mathsf{rec}(\ )t_2^*, \ldots, \mathsf{rec}(\ )t_n^*]\\ &\equiv_c \mathsf{rec}(u_1, p_1, u_1, v_1, \tilde{q}_1, \ldots, u_1, v_m, \tilde{q}_m, u_2, p_2, \ldots, u_n, p_n)\\ &\qquad [t_0^*, s_0^*(\tilde{q}_1(u_1, \cdot), \ldots, \tilde{q}_m(u_1, \cdot)), \ldots, s_m^*(\tilde{q}_1(u_1, \cdot), \ldots, \tilde{q}_m(u_1, \cdot)), t_2^*, \ldots, t_n^*].\end{aligned}$$

To see how to use this, let $J(j) = \{v_j, u_1, p_1, \ldots, p_n, q_1, \ldots, q_m\}$, suppose

$$s_j(q_1, \ldots, q_m) \mapsto_{J(j)} s_j^*(q_1, \ldots, q_m) \qquad (j \leq m),$$

let $K(i) = E \cup \{u_i, p_1, \ldots, p_n\}$ and suppose $t_i \mapsto_{K(i)} t_i^*$ $(i = 0 \text{ or } 2 \leq i \leq n)$. Notice that by **R6** we also have $\mathsf{rec}(\ )t_i \mapsto_{K(i)} t_i^*$. By part (8) then, the left-hand side of **R5** has the same normal form as

$$\begin{aligned} w \equiv \mathsf{rec}(u_1, \ldots, p_n)[&\mathsf{rec}(\ )t_0, \\ &\mathsf{rec}(v_1, \ldots, q_m)[s_0(q_1, \ldots, q_m), \ldots, s_m(q_1, \ldots, q_m)] \\ &\mathsf{rec}(\ )t_2, \ldots, \mathsf{rec}(\ )t_n], \end{aligned}$$

and by definition and the lemma on recursive combinations (2D.1)

$$\begin{aligned} w \mapsto_E \mathsf{rc}\{&\mathsf{rec}(u_1, \ldots, p_n)[\mathsf{rc}\,\mathsf{rec}(\ )t_0^*, \\ &\mathsf{rc}\,\mathsf{rec}(v_1, \ldots, q_m)[s_0^*(q_1, \ldots, q_m), \ldots, s_m^*(q_1, \ldots, q_m)], \\ &\mathsf{rc}\,\mathsf{rec}(\ )t_2^*, \ldots, \mathsf{rc}\,\mathsf{rec}(\ )t_n^*]\} \\ \equiv (\mathsf{rc})^2\mathsf{rec}(u_1, \ldots, p_n)[&\mathsf{rec}(\ )t_0^*, \\ &\mathsf{rec}(v_1, \ldots, q_m)[s_0^*(q_1, \ldots, q_m), \ldots, \ldots, s_m^*(q_1, \ldots, q_m)], \\ &\mathsf{rec}(\ )t_2^*, \ldots, \mathsf{rec}(\ )t_n^*]. \end{aligned}$$

By the special case of the formula for rc above then, this last term is

$$\begin{aligned} \equiv_c \mathsf{rc}\,&\mathsf{rec}(u_1, p_1, u_1, v_1, \tilde{q}_1, u_1, v_2, \tilde{q}_2, \ldots, u_1, v_m, \tilde{q}_m, u_2, \ldots, p_n) \\ &[t_0^*, s_0^*(\tilde{q}_1(u_1, \cdot), \ldots, \tilde{q}_m(u_1, \cdot)), \ldots, s_m^*(\tilde{q}_1(u_1, \cdot), \ldots, \tilde{q}_m(u_1, \cdot)), t_2^*, \ldots, t_n^*]. \end{aligned}$$

To complete the proof, it is enough to verify that for each j

$$s_j(\tilde{q}_1(u_1, \cdot), \ldots, \tilde{q}_m(u_1, \cdot)) \mapsto_{J(j)} s_j^*(\tilde{q}_1(u_1, \cdot), \ldots, \tilde{q}_m(u_1, \cdot)) \tag{2D.13}$$

since this identifies the last term precisely as the normal form of the right-hand side of **R5**. The proof of (2D.13) is easy, using only the fact that the expressions $\tilde{q}_l(u_1, \cdot)$ are immediate in $\tilde{J}(j) = \{u_1, v_j, \tilde{q}_1, \ldots, \tilde{q}_m, p_1, \ldots, p_n\}$.

Part (4) is proved by a similar argument which we will omit.

Part (1). Assuming for simplicity that $\bar{a}$, $\bar{b}$ are single terms, that a is immediate in E and that

$$\begin{aligned} b &\mapsto_E \mathsf{rec}(v_1, \ldots, q_m)[b_0, \ldots, b_m], \\ t &\mapsto_E \mathsf{rec}(u_1, \ldots, p_n)[t_0, \ldots, t_n], \end{aligned}$$

we have, by definition (using the hypothesis that t is not immediate in E)

$$\begin{aligned} p(a, t, b) \mapsto_E \mathsf{rec}&(r_t, u_1, \ldots, p_n, r_b, v_1, \ldots, q_m) \\ &[p(a, r_t(\ ), r_b(\ )), t_0, \ldots, t_n, b_0, \ldots, b_m]. \end{aligned} \tag{2D.14}$$

Similarly, with r a new variable (not in b or t),

$$\begin{aligned}&\mathsf{rec}(r)[p(a,r(\ ),b),t]\\&\quad\mapsto_E \mathsf{rc}\ \mathsf{rec}(r)[\mathsf{rec}(r_b,v_1,\dots,q_m)[p(a,r(\ ),r_b(\ )),b_0,\dots,b_m],\\&\qquad\qquad \mathsf{rec}(u_1,\dots,p_n)[t_0,\dots,t_n]\\&\quad\equiv \mathsf{rec}(r_b,v_1,\dots,q_m,r,u_1,\dots,p_n)[p(a,r(\ ),r_b(\ )),b_0,\dots,b_m,t_0,\dots,t_n].\end{aligned}\tag{2D.15}$$

The normal forms in (2D.14) and (2D.15) are congruent, and hence $p(a,t,b)$ has the same normal form relative to E as $\mathsf{rec}(r)[p(a,r(\ ),b),t]$.

The proofs of parts (2) and (3) are very similar, and the second assertion of the theorem follows immediately from the first and (2C.9), using induction on the definition of $\sim_E$ and taking $w \equiv \mathsf{nf}(s,E)$.

THEOREM 2D.16. *For each context E and term t, $\mathsf{nf}(t,E)$ is the unique (up to congruence) term t^* which is irreducible in E and such that $t \to_E t^*$.*

PROOF. We have already noted in (2C.9) that $\mathsf{nf}(t,E)$ is irreducible in E and $t \to_E \mathsf{nf}(t,E)$. Notice also that if s is irreducible in E, then directly from the definition of normal forms and (2B.14), $\mathsf{nf}(s,E) \equiv_c s$; from (2D.12) then, if $t \to_E s$ and s is irreducible in E, then

$$\mathsf{nf}(t,E) \equiv_c \mathsf{nf}(s,E) \equiv_c s.$$

As another application of (2D.12), let us verify that rule **R8** is valid for equivalence as well as reduction.

Fact 2D.17. *If $J(i) = E \cup \{u_i, p_1, \dots, p_n\}$ (with $u_0 = \varnothing$), then*

$$\begin{aligned}&s_0 \sim_{J(0)} t_0, \dots, s_n \sim_{J(n)} t_n\\&\quad\Rightarrow \mathsf{rec}(u_1,\dots,p_n)[s_0,\dots,s_n] \sim_E \mathsf{rec}(u_1,\dots,p_n)[t_0,\dots,t_n].\end{aligned}$$

Proof. Set $s_i^* \equiv \mathsf{nf}(s_i, J(i))$ and similarly with t_i^* and t_i, so that, by (2C.9), $s_i \to_{J(i)} s_i^*$ and $t_i \to_{J(i)} t_i^*$. By two applications of **R8** then, using (2D.12) the second time,

$$\begin{aligned}\mathsf{rec}(u_1,\dots,p_n)[s_0,\dots,s_n] &\to_E \mathsf{rec}(u_1,\dots,p_n)[s_0^*,\dots,s_n^*]\\&\to_E \mathsf{rec}(u_1,\dots,p_n)[t_0^*,\dots,t_n^*].\end{aligned}$$

But by another application of **R8** we have

$$\mathsf{rec}(u_1,\dots,p_n)[t_0,\dots,t_n] \to_E \mathsf{rec}(u_1,\dots,p_n)[t_0^*,\dots,t_n^*],$$

and hence these two terms reduce in E to the same term and they are equivalent in E. ⊣

§2E. Replacement results. We say that

$$s \text{ is intensionally equivalent to } t \text{ relative to } E \Leftrightarrow s \sim_E t,\tag{2E.1}$$

where the most interesting of these relations is the strictest $\sim_f$, intensional equivalence in the full context. It is natural to ask whether intensional equivalence is preserved under substitutions, as denotational equivalence is preserved by (1D.4). The answer is "sometimes".

THEOREM 2E.2 (Intensional replacement for terms). *If $s_1(x)$, $s_2(x)$, w_1 and w_2 are terms so that the substitution of w_i for the basic variable x in $s_i(x)$ makes sense and is free for $i = 1,2$, then*

$$[s_1(x) \sim_E s_2(x) \ \&\ w_1 \sim_f w_2] \Rightarrow s_1(w_1) \sim_E s_2(w_2).\tag{2E.3}$$

PROOF. We show by induction on the length of the term $s_1(x)$, simultaneously for all $s_2(x)$, w_1 and w_2, that if $w_1 \sim_f w_2$, then

$$[s_1(x) \to_E s_2(x) \vee s_1(x) \sim_E s_2(x)] \Rightarrow s_1(w) \sim_E s_2(w).$$

The argument is quite direct, by cases on the reduction and equivalence rules. For example, suppose that by **R1** (skipping the irrelevant side terms)

$$p(t(x)) \to_E \mathsf{rec}(r)[p(r(\ )), t(x)],$$

so that $t(x)$ is not immediate in E, and hence (easily) neither of the terms $t(w_i)$ is immediate in E. Compute:

$$\begin{aligned} p(t(w_1)) &\to_E \mathsf{rec}(r)[p(r(\ )), t(w_1)] && \text{by } \mathbf{R1} \\ &\sim_E \mathsf{rec}(r)[p(r(\ )), t(w_2)] && \text{by ind. hyp. and (2D.17)} \\ &\sim_E p(t(w_2)) && \text{by } \mathbf{R1}. \end{aligned}$$

The other cases are handled similarly, by repeated appeals to (2D.17). ⊣

The situation is more interesting for λ-substitution, which does not (in general) preserve intensional equivalence. We might expect that

$$s_1(\mathsf{f}) \sim_f s_2(\mathsf{f}) \Rightarrow s_1(\lambda(x)w(x)) \sim_f s_2(\lambda(x)w(x)), \tag{2E.4}$$

whenever the λ-substitutions make sense and are free, but there are two obstructions to this, illustrated by the following examples.

EXAMPLE 2E.5. Suppose c is a function constant of 0 arity, and let

$$s_1(\mathsf{c}) \equiv p(\mathsf{c}[\ ]), \qquad s_2(\mathsf{c}) \equiv \mathsf{rec}(r)[p(r(\ )), \mathsf{c}[\ ]].$$

Now $s_1(\mathsf{c}) \to_f s_2(\mathsf{c})$ by **R1**, but if w is a variable, then

$$p(w) \not\sim_f \mathsf{rec}(r)[p(r(\ )), w],$$

because both these terms are irreducible and they are not congruent; thus (2E.4) fails for the λ-substitution $\mathsf{c}/\lambda(\ )w$.

EXAMPLE 2E.6. Let f, g, c be function constants, and (with the obvious assumptions so that the terms are well formed) put

$$\begin{aligned} s_1(\mathsf{f}) &\equiv \mathsf{f}[\mathsf{c}], \\ s_2(\mathsf{f}) &\equiv \mathsf{fnf}(s_1(\mathsf{f})) \equiv \mathsf{rec}(r)[\mathsf{f}[r(\ )], \mathsf{c}], \\ w(x) &\equiv \mathsf{g}[x, x], \end{aligned}$$

where we have abbreviated $\mathsf{c} \equiv \mathsf{c}[\ ]$. Now $s_2(\mathsf{f}) \sim_f s_1(\mathsf{f})$, since every term is intensionally equivalent with its full normal form. On the other hand, we can compute:

$$\mathsf{fnf}(s_1(\lambda(x)w(x))) \equiv \mathsf{fnf}(\mathsf{g}[\mathsf{c}, \mathsf{c}]) \equiv \mathsf{rec}(r_1, r_2)[\mathsf{g}[r_1(\ ), r_2(\ )], \mathsf{c}, \mathsf{c}], \tag{2E.7}$$

$$\mathsf{fnf}(s_2(\lambda(x)w(x))) \equiv \mathsf{rec}(r)[\mathsf{g}[r(\ ), r(\ )], \mathsf{c}], \tag{2E.8}$$

and the λ-replacement $\mathsf{f}/\lambda(x)\mathsf{g}[x, x]$ fails, since these two normal forms are not congruent.

The intensional meaning of this failure of λ-replacement is obvious from the look of the normal forms in (2E.7) and (2E.8), which express different algorithms for

computing the same quantity $\mathsf{g}[\mathsf{c},\mathsf{c}]$: in a plausible algorithmic reading of the terms, (2E.7) would have us compute c twice and pass the two values separately to the algorithm for computing g, while the "smarter" algorithm of (2E.8) computes c only once.

Nevertheless, some interesting and natural λ-replacement theorems do hold—and in fact the counterexamples above illustrate the only obstructions to replacement. The basic idea is that the substitution

$$\mathsf{f}/\lambda(x_1,\dots,x_k)w(x_1,\dots,x_k)$$

preserves intensional equivalence if the term $w(x_1,\dots,x_k)$ is not immediate and "calls" its arguments in the same way that function constants do—basically "by value". We make precise the second notion in the next definition.

DEFINITION 2E.9. A term $w(v,x_1,\dots,x_k)$ is *balanced* in the basic variable v *relative to the variables* $x_1,\dots,x_k$, if for every sequence of expressions $t_1,\dots,t_k$ and every term s which is not immediate (in the full context)

$$w(s,t_1,\dots,t_k) \sim_f \mathsf{rec}(r)[w(r(\ ),t_1,\dots,t_k),s], \tag{2E.10}$$

where we are assuming that the indicated substitutions are correct for typing and free. Similarly, $w(p,t_1,\dots,t_k)$ is *balanced* in the pf variable p *relative to* $x_1,\dots,x_k$ if

$$w(\lambda(u)s,t_1,\dots,t_k) \sim_f \mathsf{rec}(u,p)[w(p,t_1,\dots,t_k),s], \tag{2E.11}$$

whenever the substitutions make sense and are free, and $\lambda(u)s$ is not immediate.

A term w is *balanced in* $x_1,\dots,x_k$ if it is balanced in each of these variables relative to the others.

To see how we can use this notion, suppose $w(v_1,v_2,p_3)$ is balanced in v_1, v_2, p_3, and t_1 is immediate while t_2 and $\lambda(u)t_3$ are not immediate. By applying successively the conditions above and the rules of reduction,

$$\begin{aligned} w(t_1,t_2,\lambda(u)t_3) &\sim_f \mathsf{rec}(r_2)[w(t_1,r_2(\ ),\lambda(u)t_3),t_2] \\ &\sim_f \mathsf{rec}(r_2)[\mathsf{rec}(u,r_3)[w(t_1,r_2(\ ),r_3),t_3],t_2] \\ &\sim_f \mathsf{rec}(u,r_3,r_2)[w(t_1,r_2(\ ),r_3),t_3,t_2] \\ &\sim_f \mathsf{rec}(r_2,u,r_3)[w(t_1,r_2(\ ),r_3),t_2,t_3]. \end{aligned}$$

Before proving the replacement theorem, let us verify that there is a large supply of natural balanced terms.

Fact 2E.12. (1) *Suppose* $x_1,\dots,x_k$ *are distinct variables,*

$$w \equiv p(s_1,\dots,s_n) \quad \textit{or} \quad w \equiv \mathsf{f}[s_1,\dots,s_n], \tag{2E.13}$$

and each x_i *either does not occur in* w *or it occurs exactly once and* $s_j \equiv x_i$ *for some* j; *then* w *is balanced in* $x_1,\dots,x_k$.

(2) *Suppose*

$$w \equiv \mathsf{rec}(u_1,\dots,p_n)[s_0(x_1,\dots,x_k),s_1,\dots,s_n], \tag{2E.14}$$

where $s_0(x_1,\dots,x_k)$ *is balanced in* $x_1,\dots,x_k$ *and no* x_i *occurs in* $s_1,\dots,s_n$; *then* w *is balanced in* $x_1,\dots,x_k$.

Proof. (1) is trivial, using **R1**–**R3**. To prove (2), let

$$w(x_1) \equiv \mathsf{rec}(u_1,\ldots,p_n)[s_0(x_1,t_2,\ldots,t_k),s_1,\ldots,s_n]$$
$$\equiv \mathsf{rec}(u_1,\ldots,p_n)[s_0(x_1),s_1,\ldots,s_n]$$

be a substitution instance of w, assume t_1 is not immediate and compute:

$$\begin{aligned}
w(t_1) &\equiv \mathsf{rec}(u_1,\ldots,p_n)[s_0(t_1),\ldots,s_n] \\
&\sim_f \mathsf{rec}(u_1,\ldots,p_n)[\mathsf{rec}(r)[s_0(r(\ )),t_1],\ldots,s_n] && \text{by (2D.17) and hyp.} \\
&\sim_f \mathsf{rec}(r,u_1,\ldots,p_n)[s_0(r(\ )),t_1,\ldots,s_n] && \text{by } \mathbf{R4} \\
&\sim_f \mathsf{rec}(u_1,\ldots,p_n,r)[s_0(r(\ )),\ldots,s_n,t_1] && \text{by congruence} \\
&\sim_f \mathsf{rec}(r)[\mathsf{rec}(u_1,\ldots,p_n)[s_0(r(\ )),\ldots,s_n],t_1] && \text{by } \mathbf{R4} \\
&\sim_f \mathsf{rec}(r)[w(r(\ )),t_1].
\end{aligned}$$

⊣

A similar computation works for the case of a λ-term $\lambda(u)t_1$.

It follows from this result that the term

$$(\mu(p)t)(x_1,\ldots,x_k) \equiv \mathsf{rec}(u_1,\ldots,u_k,p)[p(x_1,\ldots,x_k),t(u_1,\ldots,u_k,p)]$$

which defines naturally the "least fixed point" of t is balanced in the basic variables $x_1,\ldots,x_k$. If the variable p does not occur in t, then $(\mu(p)t)(x_1,\ldots,x_k)$ is a balanced term with exactly the same denotation as t—and very nearly the same "intension".

Fact 2E.15. *Assume that $w(x)$ is not immediate in the full context and balanced in the sequence of variables $x \equiv x_1,\ldots,x_k$, and suppose the function constant* f *does not occur in $w(x)$ and is typed so that* $\mathsf{f}[x]$ *is a term. Then for any two terms $t_1(\mathsf{f})$, $t_2(\mathsf{f})$,*

$$t_1(\mathsf{f}) \sim_E t_2(\mathsf{f}) \Rightarrow t_1(\lambda(x)w(x)) \sim_E t_2(\lambda(x)w(x)),$$

assuming that the indicated substitutions are free.

PROOF. By (2D.12), it is enough to show

$$t_1(\mathsf{f}) \to_E t_2(\mathsf{f}) \Rightarrow t_1(\lambda(x)w(x)) \sim_E t_2(\lambda(x)w(x)),$$

which we do by induction on **R1**–**R8**. We will consider some of the typical cases, labelled (i) for **Ri**. Notice that if $t(\mathsf{f})$ is not immediate in E, then $t(\lambda(x)w(x))$ is also not immediate in E—this is because in the worst case $t(\mathsf{f}) \equiv \mathsf{f}[y]$ and then $t(\lambda(x)w(x)) \equiv w(y)$ which is not immediate by hypothesis.

Case 1. Suppose

$$p(\bar{a}(\mathsf{f}),t(\mathsf{f}),\bar{b}(\mathsf{f})) \to_E \mathsf{rec}(r)[p(\bar{a}(\mathsf{f}),r(\ ),\bar{b}(\mathsf{f})),t(\mathsf{f})]$$

by **R1**, so that $t(\mathsf{f})$ is not immediate in E; but then $t(\lambda(x)w(x))$ is also not immediate in E, so **R1** gives again

$$p(\bar{a}(\lambda(x)\omega(x)),t(\lambda(x)w(x)),\bar{b}(\lambda(x)w(x)))$$
$$\to_E \mathsf{rec}(r)[p(\bar{a}(\lambda(x)w(x)),r(\ ),\bar{b}(\lambda(x)w(x))),t(\lambda(x)\omega(x))].$$

Cases 2,3. These are handled exactly as 1, except when the f in the application of the rule is the same as the constant for which we are substituting. To consider a notationally simple subcase of this, suppose

$$\mathsf{f}[a, \lambda(u)t(\mathsf{f})] \to_E \mathsf{rec}(u, r)[\mathsf{f}[a, r], t(\mathsf{f})]$$

by **R3**, so we must prove

$$w(a, \lambda(u)t(\lambda(x)w(x))) \sim_E \mathsf{rec}(u, r)[w(a, r), t(\lambda(x)w(x))];$$

this is true because w is balanced.

Case 5. Considering a notationally simple case again, suppose

$$\mathsf{rec}(u, p)[t_0(\mathsf{f}), \mathsf{rec}(v, q)[s_0(\mathsf{f}), s_1(\mathsf{f})]] \to_E \mathsf{rec}(u, p, u, v, r)[t_0(\mathsf{f}), s_0^*(\mathsf{f}), s_1^*(\mathsf{f})],$$

where the superscripts * indicate the appropriate substitutions. Assuming that the substitution $\mathsf{f}/\lambda(x)w(x)$ is free in these terms, the bound variables u, p, v, r do not occur in $w(x)$, and it is clear that the substitution produces another instance of **R5**. ⊣

It is a bit simpler to prove the corresponding result for replacement "inside".

Fact 2E.16. *Suppose* $w_1(x)$, $w_2(x)$ *are both balanced in* $x \equiv x_1, \ldots, x_k$ *and not immediate; then for every term* $t(\mathsf{f})$ *such that* $\mathsf{f}[x]$ *is well-formed and the indicated substitutions are free,*

$$w_1(x) \sim_E w_2(x) \Rightarrow t(\lambda(x)w_1(x)) \sim_E t(\lambda(x)w_2(x)).$$

The proof is by induction on the term $t(\mathsf{f})$ and we will omit it, since in the only interesting case, when $t(\mathsf{f}) \equiv f[\bar{x}]$, the argument is almost identical to that in Case 3 above.

THEOREM 2E.17 (Intensional λ-replacement). *If* $w_1(x)$, $w_2(x)$ *are both balanced in the sequence of variables* $x \equiv x_1, \ldots, x_k$ *and not immediate and if* $t_1(\mathsf{f})$, $t_2(\mathsf{f})$ *are arbitrary terms such that* $\mathsf{f}[x]$ *is well formed, and the indicated substitutions are free, then*

$$t_1(\mathsf{f}) \sim_E t_2(\mathsf{f}) \Rightarrow t_1(\lambda(x)w_1(x)) \sim_E t_2(\lambda(x)w_2(x)).$$

PROOF. By (2E.15) and (2E.16),

$$t_1(\mathsf{f}) \sim_E t_2(\mathsf{f}) \Rightarrow t_1(\lambda(x)w_1(x)) \sim_E t_2(\lambda(x)w_1(x)) \;\&\; t_2(\lambda(x)w_1(x)) \sim_E t_2(\lambda(x)w_2(x))$$

and the theorem follows immediately. ⊣

This replacement theorem makes it possible to introduce into FLR *balanced, non-immediate definitions*, consistently with the deduction calculus and the contemplated intensional interpretations. If $w(x)$ is balanced and not immediate (in the full context), we can add to the signature a new function constant f, which will be interpreted as a definition $\mathsf{f}[x] := w(x)$; now all the reductions in the extended language project to reductions in the base language under the replacement $\mathsf{f}/\lambda(x)w(x)$.

3. ALTERNATIVE DENOTATIONAL SEMANTICS

Up till now we have introduced only one kind of denotational semantics for FLR, on functional structures. In this part we will consider alternative choices, both on functional structures and on richer structures in which we can model dependence on a state and the execution of side effects. The main aim is to establish the "robustness" of the reduction calculus under a wide variety of interpretations, and also to illustrate how quite arbitrary (sequential) algorithms can be expressed in FLR.

§3A. Call-by-name recursion. In defining the denotation of terms in functional structures in §1D, we adopted without discussion the so-called (fully parallel) *call-by-value* interpretation of recursive definitions. In fact this is the most natural interpretation of recursion from the logical point of view, and in [Moschovakis alg] we will argue that it is the correct interpretation to take when we want to understand the "algorithmic meaning" of terms.

On the other hand, there are other ways to understand recursive definitions, and at least one of them—*call-by-name recursion*—has been used extensively in the interpretation of programming languages. There are many reasons for this, including the fact that call-by-name is a natural understanding of recursion in the context of "symbolic computation"; it was adopted as the "official" semantics for recursion in the Algol 60 report; and it is the most direct way to understand recursion in the λ-calculus.

In this section we will prove (in outline) that FLR reduction preserves denotation under a call-by-name interpretation of recursion on functional structures.

To illustrate the difference between call-by-value and call-by-name recursion, consider the following standard example which is attributed to Morris in [Manna 1975]; f is defined on the integers, by the recursion

$$f(n,m) := \text{if } n = 0 \text{ then } 1 \text{ else } f(pred[n], f(n,m)).$$

On our understanding of recursive definition, trivially,

$$f(n,m) \simeq \text{if } n = 0 \text{ then } 1 \text{ else undefined},$$

but the following plausible computation seems to prove that $f(1,1) \simeq 1$:

$f(1,1) \simeq$ if $1 = 0$ then 1 else $f(pred[1], f(1,1))$

$(*) \quad \simeq f(pred[1], f(1,1))$

$(**) \quad \simeq$ if $pred[1] = 0$ then 1 else $f(pred[pred[pred[1]]], f(pred[1], f(1,1))) \simeq 1.$

What's "wrong" here is that the term $f(1,1)$ on the right in $(*)$ should be evaluated before we go to $(**)$, and any attempt to compute it will lead to an infinite loop; nevertheless, there is an intrinsic logic to this computation which is captured by the definition below.

For the remainder of this section we fix FLR for a specific signature τ, and we fix a functional structure $\mathbf{A}$ of signature τ.

DEFINITION 3A.1. An *environment* is any finite set α of "bindings" of the form

$$p \equiv \lambda(u)t(u), \tag{3A.2}$$

which are correctly typed, i.e. the type of the pf variable p on the left is the same as the type of the λ-term on the right; we say that *p is bound to $t(u)$ in α* by (3A.2), and we assume that no pf variable is bound to two different terms in an environment.

In the intended use of this notion, p will be bound to $t(u)$ if it is being defined recursively by $t(u)$, so that in fact p may occur free in $t(u)$—as can other pf's, which are being defined by the same simultaneous recursion. There is an obvious relationship between environments and the "contexts" we used in the reduction calculus, but these two notions are used in quite different ways and should not be

confused. In (vague) programming terms, we use the context in compiling, to know which expressions are immediate and should not be reduced any further; we use the environment in computing (using call-by-name recursion) to know which term we should evaluate when the computation calls for some value of p.

DEFINITION 3A.3. We will define a partial function

$$nval\text{: Expressions} \times \text{Assignments} \times \text{Environments} \rightharpoonup \text{Objects of } \mathbf{A},$$

such that $nval(t, \pi, \alpha)$ has the same type as the expression t, whenever it is defined—and it is always defined if t is a pf term. The definition is by recursion, so that *nval* will be the least partial function which satisfies the conditions **NV1**–**NV5** below, in the partial ordering

$$f \leq g \Leftrightarrow (\forall \text{ terms } t, \pi, \alpha)[f(t, \pi, \alpha) \simeq w \Rightarrow g(t, \pi, \alpha) \simeq w]$$
$$\& (\forall \text{ pf terms } t, \pi, \alpha)[f(t, \pi, \alpha) \subseteq g(t, \pi, \alpha)].$$

NV1. If x is a basic variable or a pf variable not bound in the environment α, then

$$nval(x, \pi, \alpha) \simeq \pi(x);$$

if x is a pf variable bound to $t(u)$ in α, then

$$nval(x, \pi, \alpha) \simeq nval(\lambda(u)t(u), \pi, \alpha).$$

NV2. If p is not bound in α, then

$$nval(p(t_1, \ldots, t_n), \pi, \alpha) \simeq \pi(p)(nval(t_1, \pi, \alpha), \ldots, nval(t_n, \pi, \alpha));$$

if p is bound to $t(u_1, \ldots, u_n)$ in α, then

$$nval(p(t_1, \ldots, t_n), \pi, \alpha) \simeq nval(t(t_1, \ldots, t_n), \pi, \alpha).$$

NV3. $nval(\lambda(\mathbf{u})t, \pi, \alpha) = \lambda(u)nval(t, \pi[\mathbf{u}/u], \alpha)$, where $\pi[\mathbf{u}/u]$ is the assignment which agrees with π on all variables, except on the basic variables in the list $\mathbf{u}$, on which it is defined so that it assigns u to $\mathbf{u}$.

NV4. $nval(\mathsf{f}[t_1, \ldots, t_n], \pi, \alpha) \simeq \mathscr{F}(\mathsf{f})[nval(t_1, \pi, \alpha), \ldots, nval(t_n, \pi, \alpha)]$.

NV5. If $t \equiv \mathsf{rec}(u_1, p_1, \ldots, u_n, p_n)[t_0, t_1, \ldots, t_n]$, then

$$nval(t, \pi, \alpha) \simeq nval(t_0, \pi, \alpha \cup \{p_1 \equiv \lambda(u_1)t_1, \ldots, p_n \equiv \lambda(u_n)t_n\}).$$

REMARK. We will refer to *proof by induction on the definition of nval* below, without making this precise. There are several well-known ways to do this, using the least-fixed point characterization of *nval* or reading **NV1**–**NV5** as the clauses of a (possibly transfinite) recursive definition (which defines this fixed point) and understanding each proof as an induction on the ordinals.

We call two expressions (call-by-name) *equivalent* if they take the same value for all assignments and in all environments,

$$s \sim_{nden} t \Leftrightarrow (\forall \mathbf{A}, \pi \text{ in } \mathbf{A}, \alpha)[nval(s, \pi, \alpha) \simeq nval(t, \pi, \alpha)]. \tag{3A.4}$$

The (absolute, call-by-name) value of an expression t relative to an assignment π is computed in the empty environment, i.e. it is set equal to $nval(t, \pi, \varnothing)$.

We need to establish first the basic replacement properties of call-by-name recursion, which are a bit more subtle than those of (1D.4) because of the interplay

between assignments and environments. Corresponding to the operation $\pi[x/a]$ which changes the assignment π just on the basic variable x by assigning a to it, we associate with each environment α, each basic variable x and each term s of the same type as x the environment $\alpha[x/s]$, by replacing each binding $p \equiv \lambda(u)t(u, x)$ in α by the binding $p \equiv \lambda(u)t(u, s)$.

THEOREM 3A.5 (Replacement properties). (1) *For each expression $s(x)$ in which the basic variable x may occur free, for each term t of the same type as x and in which x does not occur, and for each environment α,*

$$nval(s(t), \pi, \alpha[x/t]) \simeq nval(s(x), \pi[x/nval(t, \pi, \alpha[x/t])], \alpha). \tag{3A.6}$$

(2) *For each expression $s(x)$ in which the basic variable x may occur free, for each term t of the same type as x in which x does not occur, and for each environment α,*

$$nval(s(t), \pi, \alpha[x/t]) \simeq nval(s(r(\ )), \pi, \alpha[x/r(\ )] \cup \{r \equiv \lambda(\ )t\}), \tag{3A.7}$$

where r is a "fresh" pf variable (which does not occur in $s(x)$ or α).

(3) *For every three terms s_1, s_2, t, if the pf variable p is not bound in the environment α, then*

$$s_1 \sim_{nden} s_2 \Rightarrow nval(t, \alpha \cup \{p \equiv \lambda(u)s_1\}) \simeq nval(t, \alpha \cup \{p \equiv \lambda(u)s_2\}). \tag{3A.8}$$

OUTLINE OF THE PROOF. Each of the claims is proved by induction on the definition of *nval*, taking cases on the form of the term $s(x)$ in (1) and (2), and t in (3). ⊣

The technical versions of these replacement results are formulated so that their proofs are "smooth"; in practice we apply mostly simplified versions when x does not occur in $s(x)$ or in α:

$$nval(s, \pi, \alpha[x/t]) \simeq nval(s, \pi[x/nval(t, \pi, \alpha[x/t])], \alpha), \tag{3A.9}$$

$$nval(s(t), \pi, \alpha) \simeq nval(s(x), \pi[x/nval(t, \pi, \alpha)], \alpha) \quad (x \text{ not in } \alpha), \tag{3A.10}$$

$$nval(s(t), \pi, \alpha) \simeq nval(s(r(\ )), \pi, \alpha \cup \{r \equiv \lambda(\ )t\}) \quad (x \text{ not in } \alpha). \tag{3A.11}$$

THEOREM 3A.12. *For each functional structure* **A**, *assignment π into* **A**, *context E and environment α,*

$$s \sim_E t \Rightarrow s \sim_{nden} t.$$

OUTLINE OF THE PROOF. We show by induction on **R1**–**R11** that both $\rightarrow_E$ and $\sim_E$ preserve call-by-name values.

Suppose first that by **R1**

$$p(t) \rightarrow_E \mathsf{rec}(r)[p(r(\ )), t]$$

where p is bound to $s(u)$ in α. Skipping the irrelevant assignment π, we compute:

$$\begin{aligned}
nval(\mathsf{rec}(r)[p(r(\ )), t]) &\simeq nval(p(r(\ )), \alpha \cup \{r \equiv \lambda(\ )t\}) && \text{by definition} \\
&\simeq nval(s(r(\ )), \alpha \cup \{r \equiv \lambda(\ )t\}) && \text{by definition} \\
&\simeq nval(s(t), \alpha) && \text{by (3A.11)} \\
&\simeq nval(p(t), \alpha) && \text{by definition!}
\end{aligned}$$

Similar (or trivial) arguments establish the result for all the reduction rules except **R8**, **R4** and **R5**. We outline the argument for **R8** and a notationally simple subcase of **R5**.

For **R8**, compute (skipping the assignment π again),

$$\begin{aligned}
&nval(\mathsf{rec}(u_1,\dots,p_n)[s_0,s_1,\dots,s_n],\alpha)\\
&\qquad\simeq nval(s_0,\alpha\cup\{p_1\equiv\lambda(u_1)s_1,\dots\})\\
&\qquad\simeq nval(s_0,\alpha\cup\{p_1\equiv\lambda(u_1)t_1,\dots\})\\
&\qquad\quad\text{by ind. hyp. and } n \text{ applications of (3A.5), (3)}\\
&\qquad\simeq nval(t_0,\alpha\cup\{p_1\equiv\lambda(u_1)t_1,\dots\})\\
&\qquad\quad\text{by ind. hyp.}\\
&\qquad\simeq nval(\mathsf{rec}(u_1,\dots,p_n)[t_0,t_1,\dots,t_n],\alpha).
\end{aligned}$$

Finally suppose that, by **R5**,

$$\begin{aligned}
&\mathsf{rec}(u,p)[t_0,\mathsf{rec}(v,q)[s_0(u,p,q),s_1(u,p,v,q)]]\\
&\quad\to_E \mathsf{rec}(u,p,u,v,r)[t_0,s_0(u,p,r(u,\cdot)),s_1(u,p,v,r(u,\cdot))].
\end{aligned}$$

By definition, the value of the two sides of this reduction which we must prove equal are

$$\begin{aligned}
LHS &\simeq nval(t_0,\alpha\cup\{p\equiv\lambda(u)\,\mathsf{rec}(v,q)[s_0(u,p,q),s_1(u,p,v,q)]\}),\\
RHS &\simeq nval(t_0,\alpha\cup\{p\equiv\lambda(u)s_0(u,p,r(u,\cdot)),\ r\equiv\lambda(u,v)s_1(u,p,v,r(u,\cdot))\}),
\end{aligned}$$

where we have suppressed the dependence on the assignment π which does not enter the argument.

Since the term t_0 is completely arbitrary in these two expressions, the idea is to prove that *LHS* and *RHS* are equal (under the hypotheses we have accepted) for all t_0, by induction on the definition of *nval*, taking cases on the form of t_0. In this proof, the only nontrivial case is when $t_0\equiv p(w)$, where p is bound in the environments above and w is some term. For this case, we can compute (again skipping π):

$$\begin{aligned}
LHS &\simeq nval(p(w),\alpha\cup\{p\equiv\lambda(u)\,\mathsf{rec}(v,q)[s_0(u,p,q),s_1(u,p,v,q)]\})\\
&\simeq nval(\mathsf{rec}(v,q)[s_0(w,p,q),s_1(w,p,v,q)],\\
&\qquad\alpha\cup\{p\equiv\lambda(u)\mathsf{rec}(v,q)[s_0(u,p,q),s_1(u,p,v,q)]\})\\
&\simeq nval(s_0(w,p,q),\alpha\cup\{q\equiv\lambda(v)s_1(w,p,v,q),\\
&\qquad\qquad p\equiv\lambda(u)\,\mathsf{rec}(v,q)[s_0(u,p,q),s_1(u,p,v,q)]\}),\\
&\simeq nval(s_0(w,p,q),\alpha\cup\{q\equiv\lambda(v)s_1(w,p,v,q),\\
&\qquad\qquad p\equiv\lambda(u)s_0(u,p,r(u,\cdot)),\\
&\qquad\qquad r\equiv\lambda(u,v)s_1(u,p,v,r(u,\cdot))\}),
\end{aligned}$$

where the last equality is by the induction hypothesis; and for the other side,

$$\begin{aligned}
RHS &\simeq nval(p(w),\alpha\cup\{p\equiv\lambda(u)s_0(u,p,r(u,\cdot)),\ r\equiv s_1(u,p,v,r(u,\cdot)\})\\
&\simeq nval(s_0(w,p,r(w,\cdot)),\alpha\cup\{p\equiv\lambda(u)s_0(u,p,r(u,\cdot)),\ r\equiv\lambda(u,v)s_1(u,p,v,r(u,\cdot))\}).
\end{aligned}$$

Thus, the proof will be complete if we can show that for all terms $s(u,q)$ and w, for

all environments α in which p, q, r are not bound and with the obvious assumptions on the types,

$$nval(s(u,q), \alpha \cup \{q \equiv \lambda(v)s_1(u,q), p \equiv \lambda(u)s_0(u, r(u,\cdot)), r \equiv \lambda(u,v)s_1(u, r(u,\cdot))\})$$
$$\simeq nval(s(u, r(u,\cdot)), \alpha \cup \{p \equiv \lambda(u)s_0(u, r(u,\cdot)), r \equiv \lambda(u,v)s_1(u, r(u,\cdot))\}).$$

This is easily proved by induction on the definition of *nval* again, where the only nonimmediate cases when $s(u,q) \equiv q(t)$ or $s(u,q) \equiv p(t)$ can be checked directly. ⊣

The definition of *nval* suggests the classical way of reducing call-by-name recursion on a functional structure **A** to call-by-value recursion on the expansion **A***, which has the set T of terms as an additional basic set and enough added given functions so that the usual syntactic operations on terms become recursive.

§3B. States and functions with side effects. Consider the following program G(m) (in a pidgin Pascal-like programming language) which depends on the formal integer parameter m, so that it defines a function G:

```
begin;
  n:=m; w:=1;
  time( );
  while (w≠0) {w:=F(n); print(w); n:=n+1;}
  return(n);
end;
```

Here F(n) is another program which can be called with a value n for the formal integer parameter n and returns the value $F(n)$ of some (total) function on the integers; print(w) prints the integer w on the terminal in some canonical form; and time( ) is the usual "system function" which prints on the terminal the current time.

An execution of G(m) on the integer m will first print the time, and then print the successive values $F(m)$, $F(m+1)$,... of the function F until some $F(n) = 0$; it will finally "return" this n—the least root of F above m—as its value. Thus the complete *denotation* of G(m) is a function which takes an integer m and (part of) *the state of the world* (the time) as argument, and returns as value a (finite or infinite) string which begins with the object *print the time* and is followed by *changes to the state* (the *storage* of the successive values of F in the location w), *acts of communication* (the printing of the values of F) and finally (possibly) the "return" of an integer value, if in fact F has a root above m.

Now this is a fairly complex object, not easy to deal with. It turns out that it is more convenient—and more in accord with programming terminology—to think of the denotation of G(m) as basically a partial function on the integers, which however *depends on the state* and has *side effects*. Here we will use a very special case of such functions with state dependence and side effects, which is suitable for modelling *deterministic, sequential algorithms*.

DEFINITION 3B.1. A (sequential) *communicating universe* is a triple $\mathscr{C} = (\mathscr{S}, \mathscr{U}, \mathscr{A})$, where $\mathscr{U}$ is a pure universe, $\mathscr{S}$ is a nonempty set, *the set of states*, and $\mathscr{A}$ is an arbitrary set, *the set of acts*, so that each act a induces a transformation on the states

which we will denote by the same symbol $a\colon \mathscr{S} \to \mathscr{S}$. The set of (sequential, side) *effects* of $\mathscr{C}$ is $\mathscr{E}$ = all finite and infinite sequences from $\mathscr{A}$, partially ordered by the relation

$$e_1 \leq e_2 \Leftrightarrow e_1 \text{ is an initial segment of } e_2.$$

We will also consider the set of states $\mathscr{S}$ as a poset, by imposing the flat partial ordering $=$ on it.

In a toy example, we can take $\mathscr{S}$ to be the set of all pairs (v, i) of integers, where we think of v as the (integer) value stored in some "location" V and i as the number of times the system has received "input" from some string of integers; typical acts now are *store m in V* which changes (v, i) to (m, i), *getinput* which changes (v, i) to $(v, i + 1)$ and *ring the bell* which does not alter the state.

The basic objects and individuals of $\mathscr{C}$ are those of $\mathscr{U}$, as defined in (1B.2), and the other typed objects of $\mathscr{C}$ are defined by the following recursive clauses.

co2. An object of pf type $(\bar{u} \rightharpoonup \bar{w})$ in $\mathscr{C}$ is a pair $p = (p_e, p_v)$, where the *value part* of p, namely $p_v\colon \mathscr{S} \times U \rightharpoonup W$, is a partial function with $\mathscr{S}$ the set of states and U, W the individual and basic spaces of respective types $\bar{u}, \bar{w}$; the (side) *effect part* of p is a (total) function $p_e\colon \mathscr{S} \times U \to \mathscr{E}$ which assigns a sequential side effect to each state s and each $u \in U$; and the two parts are related by the condition

$$p_v(s, u)\downarrow \Rightarrow p_e(s, u) \text{ is finite.} \tag{3B.2}$$

We will call these "communicating partial functions with state dependence and sequential side effects", *cpf's* for short, and we impose the following partial order on the set $CP(U, W)$ of all of them:

$$\begin{aligned} p \leq q \Leftrightarrow (\forall s, u)[&p_e(s, u) \leq q_e(s, u) \\ &\& \; p_v(s, u)\downarrow \Rightarrow (p_e(s, u) = q_e(s, u) \;\&\; p_v(s, u) \simeq q_v(s, u))]. \end{aligned} \tag{3B.3}$$

If $p_v(s, u)\downarrow$, then by the finiteness condition (3B.2) we know that the effect

$$p_e(s, u) = (a_1, \ldots, a_n) \tag{3B.4}$$

is a finite sequence of actions which then induces a specific transition on the state,

$$p \star (s, u) \simeq a_n(a_{n-1} \cdots (a_1(s)) \cdots); \tag{3B.5}$$

by convention, this operation $p \star (s, u)$ is defined exactly when $p(s, u)$ *returns a value*—i.e. when $p_v(s, u)\downarrow$.

It is clear that in the degenerate case, where there is only one state and the set of acts is empty, then every cpf (p_e, p_v) can be identified (essentially) with the partial function p_v.

co3. As in the pure case, *points* are tuples of basic objects and objects of pf type—cpf's in this case. Each product space $X = X_1 \times \cdots \times X_n$ carries the product partial order of the canonical partial orders on its factors.

co4. An object of function type $(\bar{x} \rightharpoonup \bar{w})$ in $\mathscr{C}$ is any pair $f = (f_e, f_v)$, where the *value part* of f, $f_v\colon \mathscr{S} \times X \rightharpoonup W$, is a monotone, partial function; the (side) *effect part* of f is a monotone total function $f_e\colon \mathscr{S} \times X \to \mathscr{E}$; and the two parts are related by the condition

$$f_v(s, x)\downarrow \Rightarrow f_e(s, x) \text{ is finite.} \tag{3B.6}$$

As with cpf's, this finiteness condition insures that if $f_v(s,x)\downarrow$, then the computation of $f(s,x)$ induces a specific transition on the state, which we will denote again by

$$f \star (s,x) \simeq a_n(a_{n-1}\cdots(a_1(s))\cdots), \tag{3B.7}$$

where $f_e(s,x) \simeq a_1\cdots a_n$; and if there is only one state and no acts, then f can be identified (essentially) with f_v.

From the mathematical point of view, these functions with side effects are of course just ordinary functions which happen to take their values in somewhat complex sets. The separation of the value into a "side effect" and a (logical or mathematical) "value" is useful because it helps the intuition and makes it much easier to define various natural operations on these objects.

Suppose for example that we define a cpf p in terms of given cpf's q, r^1, r^2 by the substitution

$$p(u,v) \simeq q(r^1(u), r^2(v)); \tag{3B.8}$$

the precise version of this is

$$\begin{aligned} p_e(s,u,v) = {} & r_e^1(s,u) * r_e^2(r^1 \star (s,u), v) \\ & * q_e(r^2 \star (r^1 \star (s,u), v), r_v^1(s,u), r_v^2(r^1 \star (s,u), v)), \end{aligned} \tag{3B.9}$$

$$p_v(s,u,v) \simeq q_v(r^2 \star (r^1 \star (s,u), v), r_v^1(s,u), r_v^2(r^1 \star (s,u), v)), \tag{3B.10}$$

where $*$ denotes "non-strict" concatenation of sequences, i.e.

$$u * \text{undefined} \simeq u, \qquad u \text{ is infinite} \Rightarrow u * v \simeq u.$$

This looks messy, but it gives the obvious intuitive meaning of the *sequential* (from left to right) *computation of* $q(r^1(u), r^2(v))$ *at a given state* s: compute first $r^1(u)$ in the state s, executing all the side effects; if this computation terminates, compute $r^2(v)$ in the new state which resulted from the first computation, again executing all the side effects; if this too terminates, finally compute q on the two returned values.

DEFINITION 3B.11. A (sequential) *communicating functional structure* of signature $\tau = (B, S, d)$ is a system

$$\mathbf{A} = (\mathscr{S}, \mathscr{U}, \mathscr{A}, \mathscr{F}), \tag{3B.12}$$

where $(\mathscr{S}, \mathscr{U}, \mathscr{A})$ is a communicating universe as above and $\mathscr{F} = \{\mathscr{F}\{\mathsf{f}\}: \mathsf{f} \in S\}$ is a family of communicating functions as in **co4**, so that the type of $\mathscr{F}(\mathsf{f})$ is $d(\mathsf{f})$.

DEFINITION 3B.13. *Denotational semantics.* For a fixed structure **A** as in (3B.12) we define a total function

$$val_e: \mathscr{S} \times \text{Expressions} \times \text{Assignments} \to \mathscr{E}$$

and a partial function

$$val_v: \mathscr{S} \times \text{Expressions} \times \text{Assignments} \rightharpoonup \text{Objects of } \mathbf{A},$$

by copying almost literally the definition of the *val* partial function in (1D.2). *Assignments* are defined exactly as in (1D.2), clauses **V2** and **V4** are modified to take account of the state dependence and side effects as in (3B.9) and (3B.10) above, and the interpretation of recursion is by least fixed points, exactly as in (1D.2), this time appealing to somewhat different "familiar methods" for establishing that least fixed points exist.

In the same way, following (3A.3) we define a total function

$$nval_e\colon \mathscr{S} \times \text{Expressions} \times \text{Assignments} \times \text{Environments} \to \mathscr{E}$$

and a partial function

$$nval_v\colon \mathscr{S} \times \text{Expressions} \times \text{Assignments} \times \text{Environments} \rightharpoonup \text{Objects of } \mathbf{A},$$

by introducing the same changes in **NV2** (when p is not bound in the environment) and in **NV4**.

THEOREM 3B.14. (1) *For each communicating functional structure* **A**, *state* s *of* **A**, *assignment* π *into* **A** *and context* E,

$$t_1 \sim_E t_2 \Rightarrow val_e(s, t_1, \pi) = val_e(s, t_2, \pi) \;\&\; val_v(s, t_1, \pi) \simeq val_v(s, t_2, \pi).$$

(2) *For each communicating functional structure* **A**, *state* s *of* **A**, *assignment* π *into* **A**, *environment* α *and context* E,

$$t_1 \sim_E t_2 \Rightarrow nval_e(s, t_1, \pi, \alpha) = nval_e(s, t_2, \pi, \alpha) \;\&\; nval_v(s, t_1, \pi, \alpha) \simeq nval_v(s, t_2, \pi, \alpha).$$

The proof of (1) is by a minor modification of the proof of (2B.4), and (2) is proved by similarly modifying the proof of (3A.12). No new ideas are needed and we will omit the details. ⊣

REFERENCES

S. A. GREIBACH [1975], ***Theory of program structures: schemes, semantics, verification***, Lecture Notes in Computer Science, vol. 36, Springer-Verlag, Berlin, 1975.

S. C. KLEENE [1952], ***Introduction to metamathematics***, Van Nostrand, Princeton, New Jersey, 1952.

Z. MANNA [1975], ***Mathematical theory of computation***, McGraw-Hill, New York, 1975.

J. MCCARTHY [1960], *Recursive functions of symbolic expressions and their computation by machine.* Part I, ***Communications of the Association for Computing Machinery***, vol. 3 (1960), pp. 184–195.

R. MILNER [1980], ***A calculus of communicating systems***, Lecture Notes in Computer Science, vol. 92, Springer-Verlag, Berlin, 1980.

Y. N. MOSCHOVAKIS [1984] (ARFTA), *Abstract recursion as a foundation of the theory of algorithms*, ***Computation and proof theory***, Lecture Notes in Mathematics, vol. 1104, Springer-Verlag, Berlin, 1984, pp. 289–364.

——— [alg], *A mathematical modeling of pure, recursive algorithms*, to appear in the proceedings of the Logic at Botik '89 meeting (Pereslavl-Zalessky, July 2–9, 1989).

D. PARK [1980], *On the semantics of fair parallelism*, ***Abstract software specifications, 1979 Copenhagen winter school***, Lecture Notes in Computer Science, vol. 86, Springer-Verlag, Berlin, 1980, pp. 504–526.

DEPARTMENT OF MATHEMATICS
UNIVERSITY OF CALIFORNIA AT LOS ANGELES
LOS ANGELES, CALIFORNIA 90024