J. Functional Programming **8** (2): 97–129, March 1998. Printed in the United Kingdom

Why the constant 'undefined'? Logics of partial terms for strict and non-strict functional programming languages

ROBERT F. STÄRK*

Institute of Informatics, University of Fribourg

Rue Faucigny 2, CH–1700 Fribourg, Switzerland

(*e-mail:* robert.staerk@unifr.ch)

Abstract

In this article we explain two different operational interpretations of functional programs by two different logics. The programs are simply typed λ-terms with pairs, projections, if-then-else and least fixed point recursion. A logic for call-by-value evaluation and a logic for call-by-name evaluation are obtained as as extensions of a system which we call the basic logic of partial terms (BPT). This logic is suitable to prove properties of programs that are valid under both strict and non-strict evaluation. We use methods from denotational semantics to show that the two extensions of BPT are adequate for call-by-value and call-by-name evaluation. Neither the programs nor the logics contain the constant 'undefined'.

Capsule Review

Strict or call-by-value and non-strict or call-by-name evaluation of functional programs are central issues in modern functional programming languages. This paper gives a thorough and illuminating study of these two subjects by providing two extensions of first order predicate logic that are computationally adequate for strict and non-strict evaluation, respectively.

The paper is written in a very readable and self-contained manner and, therefore, is accessible to a broad readership. It will be of particular interest to those working on the logical foundations of functional programming languages.

1 Introduction

In developing a theory for partial computable functions it is convenient to introduce a special element $\perp$ to represent the value 'undefined'. However, must non-termination be represented by an undefined element? What is a partial function f from a set A into a set B? In mathematics, it is usually treated as a *total* function from its domain $\mathrm{dom}(f) \subseteq A$ into B. In theoretical computer science, partial functions f from A into B are often identified with *total* functions from A into the enlarged set $B \cup \{\perp\}$,

* Research partially supported by the Swiss National Science Foundation.

where $\perp$ is a new element that does not belong to B and $f(x) \coloneqq \perp$ if x is not in the domain of f.

In mathematical logic, the first view of partial functions leads to the *Logic of Partial Terms*. This is an extension of the first-order predicate calculus with a definedness predicate which is usually written as $t\downarrow$ or $E(t)$. Logics of partial terms and precursors of it have been used for the foundation of explicit and constructive mathematics (Feferman, 1975; Beeson, 1985). Troelstra and van Dalen (1988) compare the logic of partial terms with the logic of existence (Scott, 1979). The Russian constructivist school of N. A. Shanin used similar logics (Pljuvskevivcus, 1968).

The second view of partial functions leads to D. Scott's *Logic for Computable Functions* (*LCF*) which includes in its language constants $\perp$ (Gordon *et al.*, 1979). Different kinds of LCF's have been mechanized for formal proofs of properties of functional programs (Paulson, 1987).

In this article we relate the two different views of partial functions to the two different evaluation strategies that are used in modern functional programming languages, namely strict and non-strict (or lazy) evaluation, and try to explain them with two different logics. What does strict and non-strict evaluation mean?

In a strict functional programming language, the argument of a function is always evaluated before it is invoked. As a result, if the evaluation of an expression t does not terminate because it enters an infinite loop, then neither will an expression of the form $f(t)$. Scheme (Clinger and Rees, 1991) and ML (Milner *et al.*, 1990) are both examples of this.

In a non-strict language, the arguments to a function are not evaluated until their values are actually required. For example, evaluating an expression of the form $f(t)$ may still terminate, even if evaluation of t would not, if the value of the parameter is not used in the body of f. Miranda (Turner, 1986) and Haskell (Hudlak *et al.*, 1992) are examples of this approach.

The functional programs we consider in this article are simply typed λ-terms extended by pairs, projections, if-then-else, and least fixed point recursion. In order to explain the two different operational interpretations of the programs we introduce the *basic logic of partial terms* (*BPT*) and two extensions of it, VPT for call-by-value and NPT for call-by-name evaluation. The basic system, BPT, is appropriate to prove properties of programs which are valid under strict as well as non-strict evaluation. BPT is a typed subsystem of Beeson's logic of partial terms (LPT). For example, the quantifier axioms of BPT are restricted to

$$\forall x\, A(x) \to A(v) \quad \text{and} \quad A(v) \to \exists x\, A(x)$$

where v is a syntactic value (a variable, constant, pair of values, abstraction or least fixed point). Nevertheless, the system BPT is strong enough to prove useful program transformation rules like the reduction of nested as well as iterated recursion to simultaneous recursion (see the Appendix).

The logic of partial terms for call-by-value, VPT, is obtained from BPT by adding the axiom $x^{\tau}\downarrow$ which says that variables are defined for each type τ. The logic of

https://doi.org/10.1017/S0956796898002974 Published online by Cambridge University Press

partial terms for call-by-name, NPT, is obtained from BPT by adding the axiom $\exists x^{\tau} \neg x\downarrow$ which says that there exist undefined objects for each type τ. We prove that VPT is adequate for call-by-value and NPT is adequate for call-by-name evaluation. By that we mean that, (i) for any closed term t, the formula $t\downarrow$ is derivable iff the computation of t terminates under the corresponding evaluation strategy, (ii) for closed terms t of basic type and constants c, the equation $t = c$ is provable iff the computation of t stops with result c, and (iii) if $s \simeq t$ is provable, then s and t are operationally equivalent, i.e. if we replace s in a program by t then the new program behaves the same way with respect to termination and results of basic type.

The plan of the paper is at follows. After some preliminaries on CPO's in section 3, we introduce in section 4 the notion of a program and say what we mean by strict and non-strict evaluation. In section 5, we define two kinds of type structures. One is based on partial continuous functions and the other one on total continuous functions that can take the value 'undefined'. In section 6 we show that the denotational semantics of section 5 are computationally adequate for strict and non-strict evaluation. (Sections 5 and 6 have the character of a tutorial.) In section 7 we introduce the basic logic of partial terms (BPT) which proves theorems valid under call-by-value as well as call-by-name evaluation. In section 8, we extend BPT to VPT. In section 9 we consider another extension, NPT. The results of section 6 are used to show that VPT is computationally adequate for strict and NPT for non-strict evaluation. This is the main result of the article. In an appendix we show how some of Moschovakis' reduction rules of the *Formal Language of Recursion* (*FLR*) can be derived in BPT.

2 Related work

Beeson's logic of partial terms LPT (Beeson, 1985) has been used by several authors as a logical basis for functional programming. Feferman uses the logic of partial terms to provide a logical foundation for the use of type systems in functional programming and to set up logics for the termination and correctness of programs (Feferman, 1992a; Feferman, 1992b). His logics are of great expressive power and flexibility while minimal in proof-theoretic strength. Shankar has designed a logic which is simple and yet powerful enough for proving program properties which arise in practice (Shankar, 1989). It is his scheme of induction (a special case of Scott induction) that we use in our logics. A version of LPT extended by classes in the style of Feferman's theory T_0 is used in the *Program Extractor PX* of Hayashi and Nakano (1988). Since all these logics are based on Beeson's LPT they are adequate for untyped, call-by-value languages like, for example, pure Scheme.

We show in this article how Beeson's LPT can be restricted such that it is sound for call-by-name, too. The resulting system is called the basic logic of partial terms (BPT) and its theorems are true under call-by-value as well as call-by-name. While the systems just mentioned are all untyped, the programs of BPT are typed and contain explicit least fixed point recursion.

Certainly, Scott's *Logic of Computable Functions* (*LCF*) – understood as a formalization of domain theory – can be used to reason about both, strict and lazy evaluation (Gordon *et al*., 1979; Paulson, 1987). The underlying *Polymorphic Predicate λ-Calculus PPλ*, however, as it is described in Chapter 7 of Paulson's book, is doubtless a logic for *non-strict* evaluation, if we consider its typed λ-terms as programs. PPλ contains the axioms $(\lambda x\, t)\ s = t[s/x]$ which is in general not true under call-by-value if s does not terminate. So the question arises about the exact relationship between PPλ and our logic of partial terms for call-by-name (NPT). The first problem is that PPλ contains constants $\bot_\tau$ for each type τ, whereas our NPT contains a definedness predicate $t\!\downarrow$. A first attempt would be to interpret $t\!\downarrow$ as $t \neq \bot$. This approach, however, fails immediately, since in PPλ we have $\langle \bot, \bot \rangle = \bot$, whereas in NPT we have $\langle s, t \rangle\!\downarrow$, since lazy pairs are always defined.

Riecke (1991) studies *quantifier-free* logics with basic relations $t\!\downarrow$ and $s \sqsubseteq t$. He is interested in a completeness theorem for pure (recursion-free) terms with respect to what he calls the call-by-value model $\mathscr{V}$. By completeness he means that a formula $s \sqsubseteq t$ is derivable in the quantifier-free system iff it is true in $\mathscr{V}$. In our case such a completeness theorem is impossible, since any finitary system of the strength and expressiveness we consider in this paper contains total programs which are not *provably* total and totality can be expressed using the 'less defined' relation $\sqsubseteq$.

Our programs are purely functional. They do not contain assignments, side effects and destructive operations. This is somewhat a disadvantage. We hope that we can extend our logics to programs with side-effects using ideas of Mason and Talcott (1992) such as extending the formulas by *contextual assertions*.

Logics for partially defined functions are often used in (algebraic) specification and formal program development. As an example we mention the *Logic of Partial Functions* (*LPF*) for the software development method VDM (Jones and Middelburg, 1994). This logic is three-valued. The third truth value comes in, since an equation $s = t$ is considered as neither true nor false if one or both of the terms s and t are undefined. The logics we consider in this article are classical (two-valued). Our philosophy is that terms can be undefined, but the assertions (formulas) about them are either true or false. We define an equation $s = t$ to be true, if both s and t are defined and have the same value; $s = t$ is defined to be false, otherwise. The main difference to LPF is that we prove that our logics are adequate for call-by-value (call-by-name, resp.) evaluation of programs that contain nested recursion in higher types. This has not been done for LPF.

Another specification language that supports partial functions is the *Common Object-Oriented Language for Design COLD-K* (Feijs and Jonkers, 1992). This language allows descriptions at several levels of abstraction and incorporates many ideas from algebraic specification and dynamic logic. States are first-order structures with partial (strict) functions and (possibly empty) universes. Functions, however, are not treated as 'objects' in COLD-K; it does not support higher-order functions. Also it is the burden of the user to show that there exist least recursive functions which satisfy the specifications he has written down. This is the fundamental difference to the way we treat recursion here. In our logics, recursive (higher-order) functions always exist and the corresponding induction principles are built-in to the logics.

3 Preliminaries

Since we use denotational semantics as a tool to show that certain proof rules are correct, we summarize some basic definitions. Let $(A, \sqsubseteq)$ be a partial order. A subset $X \subseteq A$ is called *directed* if it is non-empty and any two elements $x, y \in X$ have an upper bound in X, i.e. there exists a $z \in Z$ such that $x \sqsubseteq z$ and $y \sqsubseteq z$. A structure $(A, \sqsubseteq, \bigsqcup)$ is called a *complete partial order* (*CPO*) if $\bigsqcup$ is a map from the power set $\mathscr{P}(A)$ into A such that $\bigsqcup X$ is the least upper bound of X for each directed subset $X \subseteq A$. A complete partial order carries its natural topology, the *Scott Topology*. A subset $X \subseteq A$ is called *open*, if

(3.1) $\forall x, y \in A\,(x \sqsubseteq y \ \&\ x \in X \implies y \in X)$,
(3.2) $\forall Y \subseteq A\,(Y \text{ directed } \&\ \bigsqcup Y \in X \implies Y \cap X \neq \emptyset)$.

This means that a subset $X \subseteq A$ is *closed*, i.e. is the complement of an open set, if

(3.3) $\forall x, y \in A\,(x \sqsubseteq y \ \&\ y \in X \implies x \in X)$,
(3.4) $\forall Y \subseteq X\,(Y \text{ directed} \implies \bigsqcup Y \in X)$.

A partial function $f : A \xrightarrow{\sim} B$ is called *continuous*, if $f^{-1}(Y)$ is open for each open set $Y \subseteq B$. The inverse image $f^{-1}(Y)$ is defined to be the set $\{x \in \mathrm{dom}(f) \mid f(x) \in Y\}$. Equivalently, we can say that $f : A \xrightarrow{\sim} B$ is continuous, if

(3.5) $\mathrm{dom}(f)$ is open,
(3.6) $\forall x, y \in \mathrm{dom}(f)\,(x \sqsubseteq y \implies f(x) \sqsubseteq f(y))$,
(3.7) $\forall X \subseteq \mathrm{dom}(f)(X \text{ directed} \implies f(\bigsqcup X) \sqsubseteq \bigsqcup f(X))$.

The set of all *partial continuous functions* from A into B is denoted by $[A \xrightarrow{\sim} B]$. It is a CPO under the following ordering. For $f, g \in [A \xrightarrow{\sim} B]$ define $f \sqsubseteq g$, if

(3.8) $\mathrm{dom}(f) \subseteq \mathrm{dom}(g)$,
(3.9) $\forall x \in \mathrm{dom}(f)(f(x) \sqsubseteq g(x))$.

For a directed set $F \subseteq [A \xrightarrow{\sim} B]$ let

(3.10) $\mathrm{dom}(\bigsqcup F) := \bigcup\{\mathrm{dom}(f) \mid f \in F\}$ and
(3.11) $(\bigsqcup F)(x) := \bigsqcup\{f(x) \mid f \in F \ \&\ x \in \mathrm{dom}(f)\}$ for $x \in \mathrm{dom}(\bigsqcup F)$.

The *product* $A \times B$ of two CPO's A and B is the set of all pairs $\langle x, y\rangle$ such that $x \in A$ and $y \in B$. It is a CPO under the following ordering:

(3.12) $\langle x, y\rangle \sqsubseteq \langle x', y'\rangle \ :\Longleftrightarrow\ x \sqsubseteq x' \ \&\ y \sqsubseteq y'$.

For a directed set $X \subseteq A \times B$ let

(3.13) $\bigsqcup X := \langle \bigsqcup\{x \mid \exists y\, \langle x, y\rangle \in X\}, \bigsqcup\{y \mid \exists x\, \langle x, y\rangle \in X\}\rangle$.

The *lift* of a CPO $(A, \sqsubseteq, \bigsqcup)$ is obtained by adding a new bottom element $\perp$ to A. Let $\perp \notin A$. Then $(A_\perp, \sqsubseteq_\perp, \bigsqcup_\perp)$ is defined as follows:

(3.14) $A_\perp := A \cup \{\perp\}$,
(3.15) $x \sqsubseteq_\perp y \ :\Longleftrightarrow\ x = \perp$ or $x \sqsubseteq y$.

For a directed set $X \subseteq A_\perp$ let

https://doi.org/10.1017/S0956796898002974 Published online by Cambridge University Press

$$(3.16)\quad \bigsqcup\nolimits_{\perp} X := \begin{cases} \bigsqcup(X \cap A), & \text{if } (X \cap A) \neq \emptyset; \\ \perp, & \text{otherwise.} \end{cases}.$$

The space of all *total continuous functions* from A into B is denoted by $[A \to B]$. If A and B are pointed (contain a least element $\perp$), then a function $f : A \to B$ is called *strict*, if $f(\perp) = \perp$. The space of all *strict, total continuous functions* is denoted by $[A \circ\!\!\to B]$. Note, that $[A \xrightarrow{\sim} B]$ is isomorphic to $[A \to B_{\perp}]$ and $[A_{\perp} \circ\!\!\to B_{\perp}]$.

Remark 3.1
There is no essential need for continuity in this article. In the construction of the function spaces one could drop condition (3.7) and work with monotone functions only. The fact that fixed points are reached after ω steps is convenient, but not important here.

4 Evaluation of programs

We consider simply typed programs. Untyped programs or polymorphic programs will be investigated in another paper. Basic types are denoted by ι, κ. Types ρ, σ, τ are built up from basic types using product $\times$ and arrow $\to$. Types are generated as follows:

$$\rho, \sigma, \tau ::= \iota \mid \sigma \times \tau \mid \sigma \to \tau.$$

A signature consists of a set of basic types and a set of constants c^{ι} and function symbols $f^{\iota_1 \times \ldots \times \iota_n \to \kappa}$. We assume that the set of basic types always includes the type *bool* and that constant and function symbols have associated types. A (partial) first-order structure for a signature has the form

$$\mathfrak{A} = (A_{\iota}, \ldots, c^{\mathfrak{A}}, \ldots, f^{\mathfrak{A}}, \ldots),$$

where A_{ι} is a non-empty set for each basic type ι, $c^{\mathfrak{A}}$ is an element of A_{ι}, if c has type ι, and $f^{\mathfrak{A}}$ is a partial function from $A_{\iota_1} \times \ldots \times A_{\iota_n}$ into A_{κ}, if f has type $\iota_1 \times \ldots \times \iota_n \to \kappa$. As an example, take the structure

$$\mathfrak{A} = (A_{bool}, A_{nat}, A_{list}, 0, succ, pred, eq, nil, cons, head, tail, null),$$

where A_{bool} is the set of truth values $\{\mathrm{tt}, \mathrm{ff}\}$, A_{nat} is the set of non-negative integers and A_{list} is the set of finite lists of non-negative integers. The constants and functions have the following types:

0^{nat}	$eq^{nat \times nat \to bool}$	$head^{list \to nat}$
$succ^{nat \to nat}$	nil^{list}	$tail^{list \to list}$
$pred^{nat \to nat}$	$cons^{nat \times list \to list}$	$null^{list \to bool}$

The constants and functions have the standard definitions: 0 is the number zero; *succ* is the successor function that adds 1 to its argument; *pred* is the predecessor functions that subtracts 1 from its argument, if it is different from zero; *eq* tests whether two numbers are equal or not; *nil* denotes the empty list; *cons* takes a number n and a list ℓ and constructs a new list, the *head* of which is n and the *tail* of which is ℓ; *null* tests whether a list is empty or not. Note, that *pred*, *head* and *tail* are partial functions here.

We need for each type τ a countably infinite set of variables x^τ, y^τ, ... of type τ. Terms (or programs) are denoted by r, s, t. We write t^τ to indicate that t is of type τ. The following list should be understood as an inductive definition. For instance, item (4) should be read as follows: if s is a term of type $\rho \to \sigma$ and t is a term of type ρ then st is a term of type σ. Terms are of the following kind:

1. variables: x^τ, y^τ, ..., $\varphi^{\rho\to\sigma}$, $\psi^{\rho\to\sigma}$,...
2. constants: c^ι
3. function constants: $f^{\iota_1\times\ldots\times\iota_n\to\kappa}$
4. applications: $(s^{\rho\to\sigma}t^\rho)^\sigma$
5. abstractions: $(\lambda x^\rho\, t^\sigma)^{\rho\to\sigma}$
6. if-then-else: $(r^{bool}\ ?\ s^\tau\ :\ t^\tau)^\tau$
7. pairs: $\langle r^\rho, s^\sigma\rangle^{\rho\times\sigma}$
8. projections: $\pi_1(t^{\rho\times\sigma})^\rho$, $\pi_2(t^{\rho\times\sigma})^\sigma$
9. least fixed points: $\mathrm{LFP}(\varphi^{\rho\to\sigma} = \lambda x^\rho\, t^\sigma)^{\rho\to\sigma}$

In the following we will omit the types unless it is really necessary to indicate them. However, all terms are typed in this article. We also make the convention that variables φ and ψ are always of function type and, if we write $\varphi\ x$, then x is of the appropriate argument type. Omitting the types, terms are of the following form:

$$r, s, t ::= x \mid c \mid f \mid s\ t \mid \lambda x\, t \mid r\ ?\ s\ :\ t \mid \langle s, t\rangle \mid \pi_i(t) \mid \mathrm{LFP}(\varphi = \lambda x\, t).$$

The conditional $(r\ ?\ s\ :\ t)$ has its usual meaning. If r is true then the result is s, otherwise the result is t. The intended interpretation of $\mathrm{LFP}(\varphi = \lambda x\, t)$ is the least function that is a solution of the equation $\varphi = \lambda x\, t$. In Scheme, for example, the term $\mathrm{LFP}(\varphi = \lambda x\, t)$, corresponds to the expression

$$(\texttt{letrec}\ ((\varphi\ (\texttt{lambda}\ (x)\ t)))\ \varphi).$$

In ML, it corresponds to the expression

$$\texttt{let fun}\ \varphi\ x = t\ \texttt{in}\ \varphi\ \texttt{end}.$$

As an example for the use of LFP, consider the following program:

$$\mathit{length} :\equiv \mathrm{LFP}(\varphi = \lambda x\,(\mathit{null}\ x\ ?\ 0\ :\ \mathit{succ}\ (\varphi\ (\mathit{tail}\ x))))$$

This program computes the length of a list. Another example is the well-known *map* functional:

$$\mathit{map} :\equiv \lambda\psi\ \mathrm{LFP}(\varphi = \lambda x\ (\mathit{null}\ x\ ?\ \mathit{nil}\ :\ \mathit{cons}\ \langle\psi\ (\mathit{head}\ x), \varphi\ (\mathit{tail}\ x)\rangle)$$

It takes a function ψ of type $\mathit{nat} \to \mathit{nat}$ and a list x and applies ψ to every element of x.

The variable φ is considered bound in the expression $\mathrm{LFP}(\varphi = \lambda x\, t)$. We denote by $t[s_1/x_1, \ldots, s_n/x_n]$ the term that is obtained from t by simultaneously substituting the term s_i for the variable x_i for $i = 1, \ldots, n$. Of course, we have to rename bound variables in t if necessary and we assume also that the term s_i has the same type as the variable x_i. We use the Greek letter Σ to denote substitutions and write $t\Sigma$ for the application of Σ to t. The set of free variables of a term t is denoted by

https://doi.org/10.1017/S0956796898002974 Published online by Cambridge University Press

Table 1. *Call-by-value evaluation*

$$\frac{}{v \xrightarrow[\mathrm{v}]{\mathrm{ev}} v} \text{ if } v \text{ is a value} \qquad \frac{s \xrightarrow[\mathrm{v}]{\mathrm{ev}} u \qquad t \xrightarrow[\mathrm{v}]{\mathrm{ev}} v \qquad u\ v \xrightarrow[\mathrm{v}]{\mathrm{ap}} w}{s\ t \xrightarrow[\mathrm{v}]{\mathrm{ev}} w}$$

$$\frac{s \xrightarrow[\mathrm{v}]{\mathrm{ev}} u \qquad t \xrightarrow[\mathrm{v}]{\mathrm{ev}} v}{\langle s, t\rangle \xrightarrow[\mathrm{v}]{\mathrm{ev}} \langle u, v\rangle} \qquad \frac{t \xrightarrow[\mathrm{v}]{\mathrm{ev}} \langle u, v\rangle}{\pi_1(t) \xrightarrow[\mathrm{v}]{\mathrm{ev}} u} \qquad \frac{t \xrightarrow[\mathrm{v}]{\mathrm{ev}} \langle u, v\rangle}{\pi_2(t) \xrightarrow[\mathrm{v}]{\mathrm{ev}} v}$$

$$\frac{r \xrightarrow[\mathrm{v}]{\mathrm{ev}} \mathrm{tt} \qquad s \xrightarrow[\mathrm{v}]{\mathrm{ev}} u}{r\ ?\ s : t \xrightarrow[\mathrm{v}]{\mathrm{ev}} u} \qquad \frac{r \xrightarrow[\mathrm{v}]{\mathrm{ev}} \mathrm{ff} \qquad t \xrightarrow[\mathrm{v}]{\mathrm{ev}} v}{r\ ?\ s : t \xrightarrow[\mathrm{v}]{\mathrm{ev}} v}$$

$$\frac{t[u/x] \xrightarrow[\mathrm{v}]{\mathrm{ev}} v}{(\lambda x\, t)\ u \xrightarrow[\mathrm{v}]{\mathrm{ap}} v} \qquad \frac{t[u/x, \mathrm{LFP}(\varphi = \lambda x\, t)/\varphi] \xrightarrow[\mathrm{v}]{\mathrm{ev}} v}{\mathrm{LFP}(\varphi = \lambda x\, t)\ u \xrightarrow[\mathrm{v}]{\mathrm{ap}} v}$$

$$\frac{}{f\ \langle c_{a_1}, c_{a_2}\rangle \xrightarrow[\mathrm{v}]{\mathrm{ap}} c_b} \text{ if } f^{\mathfrak{A}}(a_1, a_2) \simeq b$$

FV(t). A *context* $C[*^\tau]$ is a term that contains occurrences of a special constant $*^\tau$ which is considered as a *hole*. If t is a term of type τ, then $C[t]$ denotes the result of replacing all occurrences of $*^\tau$ in $C[*^\tau]$ by t. During this process free variables of t can become bound.

4.1 Strict evaluation of programs

Given a first-order structure $\mathfrak{A}$ we define a partial function $\mathrm{eval}^{\mathrm{v}}_{\mathfrak{A}}$ which evaluates a term t to a value. If the evaluation of t does not terminate then $\mathrm{eval}^{\mathrm{v}}_{\mathfrak{A}}(t)$ is undefined. We assume that c_a is a new constant of type ι for every element $a \in A_\iota$. If the set A_ι is generated from constants using constructors, as A_{nat} and A_{list} in the example above, then we identify say the constant c_2 with the term $succ\ (succ\ 0)$ and $c_{[0,1]}$ with $cons\ \langle 0, cons\ \langle succ\ 0, nil\rangle\rangle$.

The objects returned by $\mathrm{eval}^{\mathrm{v}}_{\mathfrak{A}}$ are called *values* (or *canonical forms*). Values are denoted by u, v, w and are nothing other than terms of the following form:

$$u, v, w ::= c_a \mid f \mid \lambda x\, t \mid \langle u, v\rangle \mid \mathrm{LFP}(\varphi = \lambda x\, t).$$

We define the function $\mathrm{eval}^{\mathrm{v}}_{\mathfrak{A}}$ via two relations $t \xrightarrow[\mathrm{v}]{\mathrm{ev}} v$ and $u\ v \xrightarrow[\mathrm{v}]{\mathrm{ap}} w$. The relation $t \xrightarrow[\mathrm{v}]{\mathrm{ev}} v$ means that the term t is evaluated to the value v and $u\ v \xrightarrow[\mathrm{v}]{\mathrm{ap}} w$ means that the value u applied to the argument v returns the value w. The rules for $t \xrightarrow[\mathrm{v}]{\mathrm{ev}} v$ and $u\ v \xrightarrow[\mathrm{v}]{\mathrm{ap}} w$ are listed in Table 1.

The function $\mathrm{eval}^{\mathrm{v}}_{\mathfrak{A}}$ is defined on the argument t iff there exists a value v such that $t \xrightarrow[\mathrm{v}]{\mathrm{ev}} v$ is derivable in the call-by-value evaluation calculus. If this is the case then we set $\mathrm{eval}^{\mathrm{v}}_{\mathfrak{A}}(t)$ to the unique value v with this property.

4.2 Non-strict evaluation of programs

In the non-strict case a partial function $\mathrm{eval}^{\mathrm{n}}_{\mathfrak{A}}$ is defined in a similar way. Both the calculus and the set of values differ from the strict case. Since it is always clear from the context whether we are in the strict or non-strict case, we use the letters u, v, w

https://doi.org/10.1017/S0956796898002974 Published online by Cambridge University Press

Table 2. *Call-by-name evaluation*

$$\frac{}{v \xrightarrow[\mathrm{n}]{\mathrm{ev}} v} \text{ if } v \text{ is a value} \qquad \frac{s \xrightarrow[\mathrm{n}]{\mathrm{ev}} u \qquad u\ t \xrightarrow[\mathrm{n}]{\mathrm{ap}} v}{s\ t \xrightarrow[\mathrm{n}]{\mathrm{ev}} v}$$

$$\frac{t \xrightarrow[\mathrm{n}]{\mathrm{ev}} \langle r, s\rangle \qquad r \xrightarrow[\mathrm{n}]{\mathrm{ev}} u}{\pi_1(t) \xrightarrow[\mathrm{n}]{\mathrm{ev}} u} \qquad \frac{t \xrightarrow[\mathrm{n}]{\mathrm{ev}} \langle r, s\rangle \qquad s \xrightarrow[\mathrm{n}]{\mathrm{ev}} v}{\pi_2(t) \xrightarrow[\mathrm{n}]{\mathrm{ev}} v}$$

$$\frac{r \xrightarrow[\mathrm{n}]{\mathrm{ev}} \mathrm{tt} \qquad s \xrightarrow[\mathrm{n}]{\mathrm{ev}} u}{r\ ?\ s : t \xrightarrow[\mathrm{n}]{\mathrm{ev}} u} \qquad \frac{r \xrightarrow[\mathrm{n}]{\mathrm{ev}} \mathrm{ff} \qquad t \xrightarrow[\mathrm{n}]{\mathrm{ev}} v}{r\ ?\ s : t \xrightarrow[\mathrm{n}]{\mathrm{ev}} v}$$

$$\frac{t[s/x] \xrightarrow[\mathrm{n}]{\mathrm{ev}} v}{(\lambda x\, t)\ s \xrightarrow[\mathrm{n}]{\mathrm{ap}} v} \qquad \frac{t[s/x, \mathrm{LFP}\,(\varphi = \lambda x\, t)/\varphi] \xrightarrow[\mathrm{n}]{\mathrm{ev}} v}{\mathrm{LFP}\,(\varphi = \lambda x\, t)\ s \xrightarrow[\mathrm{n}]{\mathrm{ap}} v}$$

$$\frac{t \xrightarrow[\mathrm{n}]{\mathrm{ev}} \langle s_1, s_2\rangle \qquad s_1 \xrightarrow[\mathrm{n}]{\mathrm{ev}} c_{a_1} \qquad s_2 \xrightarrow[\mathrm{n}]{\mathrm{ev}} c_{a_2}}{f\ t \xrightarrow[\mathrm{n}]{\mathrm{ap}} c_b} \text{ if } f^{\mathfrak{A}}(a_1, a_2) \simeq b$$

to denote values in both cases. Non-strict values are of the following form:

$$u, v, w ::= c_a \mid f \mid \lambda x\, t \mid \langle s, t\rangle \mid \mathrm{LFP}\,(\varphi = \lambda x\, t)$$

The difference is that a pair $\langle s, t\rangle$ is considered to be a value for arbitrary terms s and t and not only for values s and t. Pairs are lazy pairs. This means that the components of a pair are evaluated only if needed. In the same way as for call-by-value evaluation we define two relations $t \xrightarrow[\mathrm{n}]{\mathrm{ev}} v$ and $u\ t \xrightarrow[\mathrm{n}]{\mathrm{ap}} v$ in Table 2.

The function $\mathrm{eval}^{\mathrm{n}}_{\mathfrak{A}}$ is defined on the argument t iff there exists a value v such that $t \xrightarrow[\mathrm{n}]{\mathrm{ev}} v$ is derivable in the call-by-name evaluation calculus. If this is the case then we set $\mathrm{eval}^{\mathrm{n}}_{\mathfrak{A}}(t)$ to the unique value v with this property.

The main difference between strict (call-by-value) evaluation and non-strict (call-by-name) evaluation lies in the evaluation of an application $s\ t$. In the strict case both s and t are evaluated and the value of s is applied to the value of t. In the non-strict case only s is evaluated and the argument t is left as it is. Only if the value of t is really needed, say if s evaluates to a basic function of the structure $\mathfrak{A}$, then t is evaluated.

The advantage of strict evaluation is that an argument of a function is evaluated at most once. The advantage of non-strict evaluation is that an argument of a function is only evaluated if it is really needed. Consider the *map* program from above. Let t be a term that does not terminate neither in call-by-value nor in call-by-name evaluation. For example, take

$$t :\equiv \mathrm{LFP}\,(\varphi = \lambda x\,(\varphi\ x))\ 0.$$

Then in strict evaluation $\mathrm{eval}^{\mathrm{v}}_{\mathfrak{A}}((map\ t)\ nil)$ is undefined, whereas in non-strict evaluation $\mathrm{eval}^{\mathrm{n}}_{\mathfrak{A}}((map\ t)\ nil) = nil$.

Remark 4.1

If we forget about the types and LFP, then our function $\mathrm{eval}^{\mathrm{v}}_{\mathfrak{A}}$ is exactly Plotkin's function eval_V (Plotkin, 1975). Our function $\mathrm{eval}^{\mathrm{n}}_{\mathfrak{A}}$ corresponds to Plotkin's eval_N. What we call call-by-value and call-by-name evaluation calculus, however, is not

https://doi.org/10.1017/S0956796898002974 Published online by Cambridge University Press

Plotkin's λ_V–calculus and λ_N–calculus. Plotkin's calculi also include the so-called ξ-rule $M \longrightarrow N / \lambda x\, M \longrightarrow \lambda x\, N$.

Moschovakis's function *nval* (Moschovakis, 1989, p. 1246) is in a certain sense more general then $\mathrm{eval}^{\mathrm{n}}_{\mathfrak{A}}$ and also less general. It is more general, since Moschovakis' structures $\mathfrak{A}$ are functional structures with functionals of type 2. It is less general, since the *Formal Language of Recursion* contains programs of type 2 only.

5 Interpretation of programs in type structures

In this section we interpret terms in suitable type structures. Given a first-order structure $\mathfrak{A}$ we define in a canonical way for each type τ two complete partial orders

$$\mathfrak{A}^{\mathrm{v}}_\tau = (A^{\mathrm{v}}_\tau, \sqsubseteq_\tau, \bigsqcup_\tau) \quad \text{and} \quad \mathfrak{A}^{\mathrm{n}}_\tau = (A^{\mathrm{n}}_\tau, \sqsubseteq_\tau, \bigsqcup_\tau, \bot_\tau).$$

The definition of $\mathfrak{A}^{\mathrm{v}}_\tau$ and $\mathfrak{A}^{\mathrm{n}}_\tau$ is by induction on the type.

$$\begin{array}{llll} \mathfrak{A}^{\mathrm{v}}_\iota & := \mathfrak{A}_\iota, & \mathfrak{A}^{\mathrm{n}}_\iota & := (\mathfrak{A}_\iota)_\bot, \\ \mathfrak{A}^{\mathrm{v}}_{\rho\to\sigma} & := [\mathfrak{A}^{\mathrm{v}}_\rho \xrightarrow{\sim} \mathfrak{A}^{\mathrm{v}}_\sigma], & \mathfrak{A}^{\mathrm{n}}_{\rho\to\sigma} & := [\mathfrak{A}^{\mathrm{n}}_\rho \to \mathfrak{A}^{\mathrm{n}}_\sigma]_\bot, \\ \mathfrak{A}^{\mathrm{v}}_{\rho\times\sigma} & := \mathfrak{A}^{\mathrm{v}}_\rho \times \mathfrak{A}^{\mathrm{v}}_\sigma, & \mathfrak{A}^{\mathrm{n}}_{\rho\times\sigma} & := (\mathfrak{A}^{\mathrm{n}}_\rho \times \mathfrak{A}^{\mathrm{n}}_\sigma)_\bot. \end{array}$$

By $\mathfrak{A}_\iota$ we mean the discrete CPO $(A_\iota, \sqsubseteq_\iota, \bigsqcup_\iota)$ with $x \sqsubseteq_\iota y \iff x = y$.

Note that in the construction of the space $\mathfrak{A}^{\mathrm{v}}_{\rho\to\sigma}$ we take the set of all *partial* continuous function, whereas in the space $\mathfrak{A}^{\mathrm{n}}_{\rho\to\sigma}$ we take the set of all *total* continuous functions. Although the CPO $[\mathfrak{A}^{\mathrm{n}}_\rho \to \mathfrak{A}^{\mathrm{n}}_\sigma]$ already contains a least element, a new bottom element is added in the construction of $\mathfrak{A}^{\mathrm{n}}_{\rho\to\sigma}$. This is done in order to distinguish the everywhere undefined function from the undefined object of type $\rho \to \sigma$. The structures $\mathfrak{A}^{\mathrm{v}}_\tau$ were first defined in the dissertation of Platek (1966). Platek called them HC functionals. The structures $\mathfrak{A}^{\mathrm{n}}_\tau$ are used in denotational semantics (Gunter, 1992; Winskel, 1993).

5.1 Interpretation of programs as partial functions

Programs t of type τ can be interpreted with respect to variable assignments as points in the space $\mathfrak{A}^{\mathrm{v}}_\tau$ in a canonical way. An assignment α in $\mathfrak{A}$ is a function that assigns to every variable x of type τ an object $\alpha(x)$ of $\mathfrak{A}^{\mathrm{v}}_\tau$. Assignments are total functions. We define for assignments α and β,

$$\alpha \sqsubseteq \beta \ :\iff \alpha(x) \sqsubseteq \beta(x) \text{ for all variables } x.$$

The set of all assignments into $\mathfrak{A}$ is denoted by $\mathfrak{I}^{\mathrm{v}}_{\mathfrak{A}}$. The complete partial order $(\mathfrak{I}^{\mathrm{v}}_{\mathfrak{A}}, \sqsubseteq, \bigsqcup)$ is obtained in the obvious way. For an assignment α we denote by $\alpha\frac{a}{x}$ the assignment that is the same as α except for x, to which it assigns the object a.

By induction on the structure of a term t^τ one can define a set $\mathrm{def}_{\mathfrak{A}}(t) \subseteq \mathfrak{I}^{\mathrm{v}}_{\mathfrak{A}}$ and, for each assignment $\alpha \in \mathrm{def}_{\mathfrak{A}}(t)$, an element $[\![t]\!]^{\mathrm{v}}_\alpha \in \mathfrak{A}^{\mathrm{v}}_\tau$ such that the following two invariants are satisfied:

$$(5.1)\quad \alpha \in \mathrm{def}_{\mathfrak{A}}(t) \ \&\ \alpha \sqsubseteq \beta \implies \beta \in \mathrm{def}_{\mathfrak{A}}(t) \ \&\ [\![t]\!]^{\mathrm{v}}_\alpha \sqsubseteq [\![t]\!]^{\mathrm{v}}_\beta,$$

https://doi.org/10.1017/S0956796898002974 Published online by Cambridge University Press

(5.2) $A \subseteq \mathfrak{I}^{\mathrm{v}}_{\mathfrak{A}}$ directed & $\bigsqcup A \in \mathrm{def}_{\mathfrak{A}}(t) \implies$
$A \cap \mathrm{def}_{\mathfrak{A}}(t) \neq \emptyset$ & $[\![t]\!]^{\mathrm{v}}_{\bigsqcup A} \sqsubseteq \bigsqcup_{\alpha \in A \cap \mathrm{def}_{\mathfrak{A}}(t)} [\![t]\!]^{\mathrm{v}}_{\alpha}$.

The idea is that $\mathrm{def}_{\mathfrak{A}}(t)$ is the set of assignments α for which the denotation $[\![t]\!]^{\mathrm{v}}_{\alpha}$ is defined. The functions $\mathrm{def}_{\mathfrak{A}}(\cdot)$ and $[\![\cdot]\!]^{\mathrm{v}}_{\alpha}$ have the following properties:

(5.3) $[\![x]\!]^{\mathrm{v}}_{\alpha} = \alpha(x)$, $[\![c]\!]^{\mathrm{v}}_{\alpha} = c^{\mathfrak{A}}$, $[\![f]\!]^{\mathrm{v}}_{\alpha} = f^{\mathfrak{A}}$,
(5.4) $\alpha \in \mathrm{def}\,(s\ t) \iff \alpha \in \mathrm{def}\,(s) \cap \mathrm{def}\,(t)$ & $[\![t]\!]^{\mathrm{v}}_{\alpha} \in \mathrm{dom}\,([\![s]\!]^{\mathrm{v}}_{\alpha})$,
(5.5) $\alpha \in \mathrm{def}\,(s\ t) \implies [\![s\ t]\!]^{\mathrm{v}}_{\alpha} = [\![s]\!]^{\mathrm{v}}_{\alpha}([\![t]\!]^{\mathrm{v}}_{\alpha})$,
(5.6) $a \in \mathrm{dom}\,([\![\lambda x\, t]\!]^{\mathrm{v}}_{\alpha}) \iff \alpha\frac{a}{x} \in \mathrm{def}\,(t)$,
(5.7) $a \in \mathrm{dom}\,([\![\lambda x\, t]\!]^{\mathrm{v}}_{\alpha}) \implies [\![\lambda x\, t]\!]^{\mathrm{v}}_{\alpha}(a) = [\![t]\!]^{\mathrm{v}}_{\alpha\frac{a}{x}}$,
(5.8) $\alpha \in \mathrm{def}\,(r\ ?\ s : t) \iff \alpha \in \mathrm{def}\,(r)$ &
$([\![r]\!]^{\mathrm{v}}_{\alpha} = \mathrm{tt}$ & $\alpha \in \mathrm{def}\,(s)$ or $[\![r]\!]^{\mathrm{v}}_{\alpha} = \mathrm{ff}$ & $\alpha \in \mathrm{def}\,(t))$,
(5.9) $\alpha \in \mathrm{def}\,(r\ ?\ s : t)$ & $[\![r]\!]^{\mathrm{v}}_{\alpha} = \mathrm{tt} \implies [\![r\ ?\ s : t]\!]^{\mathrm{v}}_{\alpha} = [\![s]\!]^{\mathrm{v}}_{\alpha}$,
(5.10) $\alpha \in \mathrm{def}\,(r\ ?\ s : t)$ & $[\![r]\!]^{\mathrm{v}}_{\alpha} = \mathrm{ff} \implies [\![r\ ?\ s : t]\!]^{\mathrm{v}}_{\alpha} = [\![t]\!]^{\mathrm{v}}_{\alpha}$,
(5.11) $\alpha \in \mathrm{def}\,(\langle s, t\rangle) \iff \alpha \in \mathrm{def}\,(s) \cap \mathrm{def}\,(t)$,
(5.12) $\alpha \in \mathrm{def}\,(\langle s, t\rangle) \implies [\![\langle s, t\rangle]\!]^{\mathrm{v}}_{\alpha} = \langle [\![s]\!]^{\mathrm{v}}_{\alpha}, [\![t]\!]^{\mathrm{v}}_{\alpha}\rangle$,
(5.13) $\alpha \in \mathrm{def}\,(\pi_i(t)) \iff \alpha \in \mathrm{def}\,(t)$,
(5.14) $\alpha \in \mathrm{def}\,(t)$ & $[\![t]\!]^{\mathrm{v}}_{\alpha} = \langle a, b\rangle \implies [\![\pi_1(t)]\!]^{\mathrm{v}}_{\alpha} = a$ & $[\![\pi_2(t)]\!]^{\mathrm{v}}_{\alpha} = b$,
(5.15) If φ is of type $\rho \to \sigma$, then $[\![\mathrm{LFP}\,(\varphi = \lambda x\, t)]\!]^{\mathrm{v}}_{\alpha}$ is the least function f in $[\mathfrak{A}^{\mathrm{v}}_{\rho} \xrightarrow{\sim} \mathfrak{A}^{\mathrm{v}}_{\sigma}]$ such that $f = [\![\lambda x\, t]\!]^{\mathrm{v}}_{\alpha\frac{f}{\varphi}}$.

Note that $[\![\lambda x\, t]\!]^{\mathrm{v}}_{\alpha}$ and $[\![\mathrm{LFP}\,(\varphi = \lambda x\, t)]\!]^{\mathrm{v}}_{\alpha}$ are always defined, since they are functions. Both can, however, be the empty (everywhere undefined) function. Since the denotation of a term depends on the values of the assignment to its free variables only, we can write $[\![t]\!]^{\mathrm{v}}_{\mathfrak{A}}$ for the denotation of a closed term t.

Properties (5.1) and (5.2) are proved by induction on t. In the case where t^{τ} is of the form $\mathrm{LFP}\,(\varphi = \lambda x\, r)$ we have to show the following two statements:

(5.16) $\alpha \sqsubseteq \beta \implies [\![\mathrm{LFP}\,(\varphi = \lambda x\, r)]\!]^{\mathrm{v}}_{\alpha} \sqsubseteq [\![\mathrm{LFP}\,(\varphi = \lambda x\, r)]\!]^{\mathrm{v}}_{\beta}$,
(5.17) $[\![\mathrm{LFP}\,(\varphi = \lambda x\, r)]\!]^{\mathrm{v}}_{\bigsqcup A} \sqsubseteq \bigsqcup_{\alpha \in A} [\![\mathrm{LFP}\,(\varphi = \lambda x\, r)]\!]^{\mathrm{v}}_{\alpha}$ for directed sets $A \subseteq \mathfrak{I}^{\mathrm{v}}_{\mathfrak{A}}$.

Proof
Let $\Gamma_{\alpha} : \mathfrak{A}^{\mathrm{v}}_{\tau} \to \mathfrak{A}^{\mathrm{v}}_{\tau}$ be the operator defined by $\Gamma_{\alpha}(f) := [\![\lambda x\, r]\!]^{\mathrm{v}}_{\alpha\frac{f}{\varphi}}$. By the induction hypothesis applied to $\lambda x\, r$, we obtain:

(5.18) $f \sqsubseteq g \implies \Gamma_{\alpha}(f) \sqsubseteq \Gamma_{\alpha}(g)$,
(5.19) $\Gamma_{\alpha}(\bigsqcup F) = \bigsqcup\{\Gamma_{\alpha}(f) \mid f \in F\}$ for directed sets $F \subseteq \mathfrak{A}^{\mathrm{v}}_{\tau}$,
(5.20) $\alpha \sqsubseteq \beta \implies \Gamma_{\alpha}(f) \sqsubseteq \Gamma_{\beta}(f)$,
(5.21) $\Gamma_{\bigsqcup A}(f) = \bigsqcup\{\Gamma_{\alpha}(f) \mid \alpha \in A\}$ for directed sets $A \subseteq \mathfrak{I}^{\mathrm{v}}_{\mathfrak{A}}$.

Let $f^0_{\alpha} := \emptyset$, $f^{n+1}_{\alpha} := \Gamma_{\alpha}(f^n_{\alpha})$ for $n \in \mathbb{N}$ and $\ell_{\alpha} := \bigsqcup_{n \in \mathbb{N}} f^n_{\alpha}$. Note, that the empty function $\emptyset$ belongs to $\mathfrak{A}^{\mathrm{v}}_{\tau}$ and that the sequence $(f^n_{\alpha})_{n \in \mathbb{N}}$ is increasing. We have:

(5.22) $\Gamma_{\alpha}(\ell_{\alpha}) = \ell_{\alpha}$,
(5.23) $\Gamma_{\alpha}(g) \sqsubseteq g \implies \ell_{\alpha} \sqsubseteq g$, for all $g \in \mathfrak{A}^{\mathrm{v}}_{\tau}$.
(5.24) $\ell_{\alpha} = [\![\mathrm{LFP}\,(\varphi = \lambda x\, r)]\!]^{\mathrm{v}}_{\alpha}$,

https://doi.org/10.1017/S0956796898002974 Published online by Cambridge University Press

Assertion (5.22) can be seen as follows:

$$\Gamma_\alpha(\ell_\alpha) = \Gamma_\alpha(\bigsqcup_{n\in\mathbb{N}} f_\alpha^n) = \bigsqcup_{n\in\mathbb{N}} \Gamma_\alpha(f_\alpha^n) = \bigsqcup_{n\in\mathbb{N}} f_\alpha^{n+1} = \ell_\alpha.$$

For assertion (5.23) assume that $\Gamma_\alpha(g) \sqsubseteq g$. Then by induction on n it follows that $f_\alpha^n \sqsubseteq g$. Thus $\ell_\alpha = \bigsqcup_{n\in\mathbb{N}} f_\alpha^n \sqsubseteq g$. Assertion (5.24) follows from (5.15).

To show (5.16) we assume that $\alpha \sqsubseteq \beta$. Then $\Gamma_\alpha(\ell_\beta) \sqsubseteq \Gamma_\beta(\ell_\beta) = \ell_\beta$ and, by (5.23), we can conclude that $\ell_\alpha \sqsubseteq \ell_\beta$.

To prove (5.17) we assume that A is a directed set of assignments. We have to show that $\ell_{\bigsqcup A} \sqsubseteq \bigsqcup_{\alpha\in A} \ell_\alpha$. Let $\ell := \bigsqcup_{\alpha\in A} \ell_\alpha$. We have:

$$\Gamma_{\bigsqcup A}(\ell) = \bigsqcup_{\alpha\in A} \Gamma_\alpha(\ell) = \bigsqcup_{\alpha\in A}(\bigsqcup_{\beta\in A} \Gamma_\alpha(\ell_\beta)) \sqsubseteq \bigsqcup_{\xi\in A} \Gamma_\xi(\ell_\xi) = \bigsqcup_{\xi\in A} \ell_\xi = \ell.$$

Thus we have $\Gamma_{\bigsqcup A}(\ell) \sqsubseteq \ell$ and we can conclude that $\ell_{\bigsqcup A} \sqsubseteq \ell = \bigsqcup_{\alpha\in A} \ell_\alpha$. □

Since assignments are total functions, the following substitution lemma is true only under condition that s is defined.

Lemma 5.1 (*Substitution*)
Assume that $\alpha \in \mathrm{def}_{\mathfrak{A}}(s)$ and $\beta = \alpha\frac{[\![s]\!]^{\mathrm{v}}_\alpha}{x}$. Then we have:

(a) $\alpha \in \mathrm{def}_{\mathfrak{A}}(t[s/x])$ iff $\beta \in \mathrm{def}_{\mathfrak{A}}(t)$,
(b) if $\alpha \in \mathrm{def}_{\mathfrak{A}}(t[s/x])$, then $[\![t[s/x]]\!]^{\mathrm{v}}_\alpha = [\![t]\!]^{\mathrm{v}}_\beta$.

5.2 *Interpretation of programs as total functions*

For a partial function $f : A_{\iota_1} \times A_{\iota_2} \xrightarrow{\sim} A_\kappa$ let $f_\perp : \mathfrak{A}^{\mathrm{n}}_{\iota_1\times\iota_2} \to \mathfrak{A}^{\mathrm{n}}_\kappa$ be the total function defined as follows:

$$f_\perp(a) := \begin{cases} f(a_1,a_2), & \text{if } a = (a_1,a_2),\ a_1 \in A_{\iota_1},\ a_2 \in A_{\iota_2} \text{ and } (a_1,a_2) \in \mathrm{dom}(f);\\ \perp_\kappa, & \text{otherwise.}\end{cases}$$

Assignments in $\mathfrak{A}^{\mathrm{n}}_\tau$ are functions that assign to each variable of type τ an object of $\mathfrak{A}^{\mathrm{n}}_\tau$. The set of all assignments is denoted by $\mathfrak{I}^{\mathrm{n}}_{\mathfrak{A}}$ and $(\mathfrak{I}^{\mathrm{n}}_{\mathfrak{A}}, \sqsubseteq, \bigsqcup)$ is the associated complete partial order obtained in the canonical way. The interpretation of a term t with respect to an assignment α is denoted by $[\![t]\!]^{\mathrm{n}}_\alpha$. Unlike in the partial case, $[\![t]\!]^{\mathrm{n}}_\alpha$ is always defined but can take the value $\perp$. By induction on the structure of a term t^τ the value $[\![t]\!]^{\mathrm{n}}_\alpha \in \mathfrak{A}^{\mathrm{n}}_\tau$ is defined such that

(5.25) $\alpha \sqsubseteq \beta \implies [\![t]\!]^{\mathrm{n}}_\alpha \sqsubseteq [\![t]\!]^{\mathrm{n}}_\beta$,
(5.26) $A \subseteq \mathfrak{I}^{\mathrm{n}}_{\mathfrak{A}}$ & A directed $\implies [\![t]\!]^{\mathrm{n}}_{\bigsqcup A} \sqsubseteq \bigsqcup_{\alpha\in A} [\![t]\!]^{\mathrm{n}}_\alpha$.

The function $[\![\cdot]\!]^{\mathrm{n}}_\alpha$ has the following properties:

(5.27) $[\![x]\!]^{\mathrm{n}}_\alpha = \alpha(x)$, $[\![c]\!]^{\mathrm{n}}_\alpha = c^{\mathfrak{A}}$, $[\![f]\!]^{\mathrm{n}}_\alpha = (f^{\mathfrak{A}})_\perp$,
(5.28) $[\![s]\!]^{\mathrm{n}}_\alpha \neq \perp \implies [\![s\ t]\!]^{\mathrm{n}}_\alpha = [\![s]\!]^{\mathrm{n}}_\alpha([\![t]\!]^{\mathrm{n}}_\alpha)$,
(5.29) $[\![s]\!]^{\mathrm{n}}_\alpha = \perp \implies [\![s\ t]\!]^{\mathrm{n}}_\alpha = \perp$,
(5.30) $[\![\lambda x\, t]\!]^{\mathrm{n}}_\alpha(a) = [\![t]\!]^{\mathrm{n}}_{\alpha\frac{a}{x}}$,
(5.31) $[\![r]\!]^{\mathrm{n}}_\alpha = \mathrm{tt} \implies [\![r\ ?\ s : t]\!]^{\mathrm{n}}_\alpha = [\![s]\!]^{\mathrm{n}}_\alpha$,

https://doi.org/10.1017/S0956796898002974 Published online by Cambridge University Press

(5.32) $[\![r]\!]^{\mathrm{n}}_{\alpha} = \mathrm{ff} \implies [\![r\ ?\ s : t]\!]^{\mathrm{n}}_{\alpha} = [\![t]\!]^{\mathrm{n}}_{\alpha}$,
(5.33) $[\![r]\!]^{\mathrm{n}}_{\alpha} = \bot \implies [\![r\ ?\ s : t]\!]^{\mathrm{n}}_{\alpha} = \bot$,
(5.34) $[\![\langle s, t\rangle]\!]^{\mathrm{n}}_{\alpha} = \langle [\![s]\!]^{\mathrm{n}}_{\alpha}, [\![t]\!]^{\mathrm{n}}_{\alpha}\rangle$,
(5.35) $[\![t]\!]^{\mathrm{n}}_{\alpha} = \langle a_1, a_2\rangle \implies [\![\pi_1(t)]\!]^{\mathrm{n}}_{\alpha} = a_1 \ \&\ [\![\pi_2(t)]\!]^{\mathrm{n}}_{\alpha} = a_2$,
(5.36) $[\![t]\!]^{\mathrm{n}}_{\alpha} = \bot \implies [\![\pi_1(t)]\!]^{\mathrm{n}}_{\alpha} = \bot \ \&\ [\![\pi_2(t)]\!]^{\mathrm{n}}_{\alpha} = \bot$,
(5.37) If φ is of type $\rho \to \sigma$, then $[\![\mathrm{LFP}(\varphi = \lambda x\, t)]\!]^{\mathrm{n}}_{\alpha}$ is the least function f in $[\mathfrak{A}^{\mathrm{n}}_{\rho} \to \mathfrak{A}^{\mathrm{n}}_{\sigma}]$ such that $f = [\![\lambda x\, t]\!]^{\mathrm{n}}_{\alpha\frac{f}{\varphi}}$.

Unlike in the partial case, the substitution lemma is true without any restriction. The following equality holds even if $[\![s]\!]^{\mathrm{n}}_{\alpha} = \bot$.

Lemma 5.2 (*Substitution*)
$[\![t[s/x]]\!]^{\mathrm{n}}_{\alpha} = [\![t]\!]^{\mathrm{n}}_{\beta}$, where $\beta = \alpha\frac{[\![s]\!]^{\mathrm{n}}_{\alpha}}{x}$.

As a consequence, we obtain that the β-reduction rule is true in the total case, i.e. we have $[\![(\lambda x\, t)s]\!]^{\mathrm{n}}_{\alpha} = [\![t[s/x]]\!]^{\mathrm{n}}_{\alpha}$ for all assignments α. In the partial case, this equality is only true under condition that $\alpha \in \mathrm{def}(s)$.

6 Adequacy results

In this section we show that the type structures $\mathfrak{A}^{\mathrm{v}}_{\tau}$ and $\mathfrak{A}^{\mathrm{n}}_{\tau}$ are adequate for strict and non-strict evaluation. Let t^{τ} be a closed term of arbitrary type. Then we have:

(6.1) $[\![t]\!]^{\mathrm{v}}_{\mathfrak{A}}$ is defined iff $\mathrm{eval}^{\mathrm{v}}_{\mathfrak{A}}(t)$ is defined.
(6.2) $[\![t]\!]^{\mathrm{n}}_{\mathfrak{A}} \neq \bot_{\tau}$ iff $\mathrm{eval}^{\mathrm{n}}_{\mathfrak{A}}(t)$ is defined.

Let t^{ι} be a closed term of basic type and $a \in A_{\iota}$. Then we have:

(6.3) $[\![t]\!]^{\mathrm{v}}_{\mathfrak{A}} = a$ iff $\mathrm{eval}^{\mathrm{v}}_{\mathfrak{A}}(t) = c_a$.
(6.4) $[\![t]\!]^{\mathrm{n}}_{\mathfrak{A}} = a$ iff $\mathrm{eval}^{\mathrm{n}}_{\mathfrak{A}}(t) = c_a$.

These four facts are well-known. Proofs can be found, for example, in Winskel's book (Winskel, 1993). Since we use these results in sections 8 and 9 to show that the logics VPT and NPT are adequate for strict and non-strict evaluation, we sketch the proofs here briefly.

6.1 Strict evaluation and the structures $\mathfrak{A}^{\mathrm{v}}_{\tau}$

One direction is easy. If $\mathrm{eval}^{\mathrm{v}}_{\mathfrak{A}}(t) = v$, i.e. if $t \xrightarrow[\mathrm{v}]{\mathrm{ev}} v$ is derivable according to Table 1, then t and v have the same denotation in $\mathfrak{A}^{\mathrm{v}}_{\tau}$. Note, that for values v, the interpretation $[\![v]\!]^{\mathrm{v}}_{\alpha}$ is defined for arbitrary assignments α.

Lemma 6.1 (*Soundness of call-by-value evaluation*)
(a) If $t \xrightarrow[\mathrm{v}]{\mathrm{ev}} v$ then $\alpha \in \mathrm{def}_{\mathfrak{A}}(t)$ and $[\![t]\!]^{\mathrm{v}}_{\alpha} = [\![v]\!]^{\mathrm{v}}_{\alpha}$ for all assignments $\alpha \in \mathfrak{I}^{\mathrm{v}}_{\mathfrak{A}}$.
(b) If $u\ v \xrightarrow[\mathrm{v}]{\mathrm{ap}} w$ then $\alpha \in \mathrm{def}_{\mathfrak{A}}(u\ v)$ and $[\![u\ v]\!]^{\mathrm{v}}_{\alpha} = [\![w]\!]^{\mathrm{v}}_{\alpha}$ for all $\alpha \in \mathfrak{I}^{\mathrm{v}}_{\mathfrak{A}}$.

For the other direction we need relations $a \preceq_{\tau} v$ between objects $a \in \mathfrak{A}^{\mathrm{v}}_{\tau}$ and values v of type τ. The relation $a \preceq_{\tau} v$ can be read as: a is an approximation for v.

https://doi.org/10.1017/S0956796898002974 Published online by Cambridge University Press

Definition 6.2
The relation $a \preceq_\tau v$ between points $a \in \mathfrak{A}^{\mathrm{v}}_\tau$ and values v of type τ is defined by induction on τ.

(6.5) $a \preceq_\iota v \;:\Longleftrightarrow\; v = c_a$.
(6.6) $f \preceq_{\rho\to\sigma} u \;:\Longleftrightarrow\; \forall a \in \mathrm{dom}(f) \forall v^\rho (a \preceq_\rho v \implies \exists w (u\, v \xrightarrow[\mathrm{v}]{\mathrm{ap}} w \;\&\; f(a) \preceq_\sigma w))$.
(6.7) $\langle a, b\rangle \preceq_{\rho\times\sigma} \langle u, v\rangle \;:\Longleftrightarrow\; a \preceq_\rho u \;\&\; b \preceq_\sigma v$.

We omit the subscript τ in $\preceq_\tau$ if it is clear from to context. Because of the following property, the relations $\preceq_\tau$ are sometimes called *inclusive predicates* (Gunter, 1992).

Lemma 6.3
Let $F \subseteq \mathfrak{A}^{\mathrm{v}}_\tau$ be directed and let v be of type τ. If $f \preceq_\tau v$ for each $f \in F$, then $\bigsqcup F \preceq_\tau v$.

Proof
By induction on τ. □

In the following lemma Σ denotes a substitution $[v_1/x_1, \ldots, v_n/x_n]$ of values for variables. Σ can be understood as an environment.

Lemma 6.4 (*Adequacy of call-by-value evaluation*)
If $\alpha(x) \preceq x\Sigma$ for each $x \in \mathrm{FV}(t)$ and if $\alpha \in \mathrm{def}_{\mathfrak{A}}(t)$, then there exists a value v such that $t\Sigma \xrightarrow[\mathrm{v}]{\mathrm{ev}} v$ and $[\![t]\!]^{\mathrm{v}}_\alpha \preceq v$.

Proof
The proof is by induction on the term t. We consider the case, where t is of the form $\mathrm{LFP}(\varphi = \lambda x\, r)$. Assume that $\alpha(x) \preceq x\Sigma$ for each $x \in \mathrm{FV}(t)$. Let $f_0 := \emptyset$, $f_{n+1} := [\![\lambda x\, r]\!]^{\mathrm{v}}_{\alpha\frac{f_n}{\varphi}}$ and $f := \bigsqcup_{n\in\mathbb{N}} f_n$. By definition, we have $f = [\![\mathrm{LFP}(\varphi = \lambda x\, r)]\!]^{\mathrm{v}}_\alpha$. Since $t\Sigma$ is a value, we have $t\Sigma \xrightarrow[\mathrm{v}]{\mathrm{ev}} t\Sigma$ and it remains to show that $f \preceq t\Sigma$. By Lemma 6.3, it follows that it is sufficient to show that $f_n \preceq t\Sigma$ for each $n \in \mathbb{N}$.

For $n = 0$ it is certainly true, since $\mathrm{dom}(f_0) = \emptyset$. Assume that $f_n \preceq t\Sigma$. Assume that $a \in \mathrm{dom}(f_{n+1})$ and $a \preceq u$. We have to show that there exists a value v such that $t\Sigma\, u \xrightarrow[\mathrm{v}]{\mathrm{ap}} v$ and $f_{n+1}(a) \preceq v$. We can assume that the variables φ and x are not touched by Σ and do not appear in Σ. By definition, $\alpha\frac{f_n}{\varphi}\frac{a}{x} \in \mathrm{def}(r)$ and $f_{n+1}(a) = [\![r]\!]^{\mathrm{v}}_{\alpha\frac{f_n}{\varphi}\frac{a}{x}}$. We can apply the induction hypothesis to the term r and obtain a value v such that $r(\Sigma\frac{t\Sigma}{\varphi}\frac{u}{x}) \xrightarrow[\mathrm{v}]{\mathrm{ev}} v$ and $f_{n+1}(a) \preceq v$. But now, we are done, since $(r\Sigma)[t\Sigma/\varphi, u/x]$ is the same as $r(\Sigma\frac{t\Sigma}{\varphi}\frac{u}{x})$ and applying the fixed-point rule of call-by-value evaluation we thus obtain $t\Sigma\, u \xrightarrow[\mathrm{v}]{\mathrm{ap}} v$. Hence, $f_{n+1} \preceq t\Sigma$. □

Theorem 6.5
Let t be a closed term of arbitrary type. Then $[\![t]\!]^{\mathrm{v}}_{\mathfrak{A}}$ is defined iff $\mathrm{eval}^{\mathrm{v}}_{\mathfrak{A}}(t)$ is defined.

Proof
Assume that $[\![t]\!]^{\mathrm{v}}_{\mathfrak{A}}$ is defined. By the computational adequacy of call-by-value evaluation, there exists a value v such that $t \xrightarrow[\mathrm{v}]{\mathrm{ev}} v$ and $[\![t]\!]^{\mathrm{v}}_{\mathfrak{A}} \preceq v$. Hence $\mathrm{eval}^{\mathrm{v}}_{\mathfrak{A}}(t)$ is defined. For the converse direction assume that $\mathrm{eval}^{\mathrm{v}}_{\mathfrak{A}}(t)$ is defined. This means that there exists a value v such that $t \xrightarrow[\mathrm{v}]{\mathrm{ev}} v$. By the soundness of call-by-value evaluation, we obtain that $[\![t]\!]^{\mathrm{v}}_{\mathfrak{A}}$ is defined. □

https://doi.org/10.1017/S0956796898002974 Published online by Cambridge University Press

Theorem 6.6
Let t be a closed term of basic type ι and let $a \in A_\iota$. Then $[\![t]\!]^{\mathrm{v}}_{\mathfrak{A}} = a$ iff $\mathrm{eval}^{\mathrm{v}}_{\mathfrak{A}}(t) = c_a$.

Proof
Assume that $[\![t]\!]^{\mathrm{v}}_{\mathfrak{A}}$ is defined and $[\![t]\!]^{\mathrm{v}}_{\mathfrak{A}} = a$. By the computational adequacy of call-by-value evaluation, it follows that there exists a value v such that $t \xrightarrow[\mathrm{v}]{\mathrm{ev}} v$ and $a \preceq_\iota v$. By definition of the relation $\preceq_\iota$, this means that $v = c_a$. Hence $\mathrm{eval}^{\mathrm{v}}_{\mathfrak{A}}(t) = c_a$. For the converse direction assume that $t \xrightarrow[\mathrm{v}]{\mathrm{ev}} c_a$. By the soundness of call-by-value evaluation, it follows that $[\![t]\!]^{\mathrm{v}}_{\mathfrak{A}}$ is defined and $[\![t]\!]^{\mathrm{v}}_{\mathfrak{A}} = [\![c_a]\!]^{\mathrm{v}}_{\mathfrak{A}} = a$. □

6.2 *Non-strict evaluation and the structures* $\mathfrak{A}^{\mathrm{n}}_\tau$

As in the strict case, one direction is easy. If $\mathrm{eval}^{\mathrm{n}}_{\mathfrak{A}}(t) = v$, i.e. if $t \xrightarrow[\mathrm{n}]{\mathrm{ev}} v$ is derivable according to Table 2, then t and v have the same denotation in $\mathfrak{A}^{\mathrm{n}}_\tau$.

Lemma 6.7 (*Soundness of call-by-name evaluation*)
(a) If $t \xrightarrow[\mathrm{n}]{\mathrm{ev}} v$ then $[\![t]\!]^{\mathrm{n}}_\alpha = [\![v]\!]^{\mathrm{n}}_\alpha$ for all assignments $\alpha \in \mathfrak{I}^{\mathrm{n}}_{\mathfrak{A}}$.
(b) If $u\ t \xrightarrow[\mathrm{n}]{\mathrm{ap}} v$ then $[\![u\ t]\!]^{\mathrm{n}}_\alpha = [\![v]\!]^{\mathrm{n}}_\alpha$ for all assignments $\alpha \in \mathfrak{I}^{\mathrm{n}}_{\mathfrak{A}}$.

For the other direction we need relations $a \preceq_\tau v$ between elements $a \in \mathfrak{A}^{\mathrm{n}}_\tau$ and values v of type τ and relations $a \preceq^{\mathrm{w}}_\tau t$ between elements $a \in \mathfrak{A}^{\mathrm{n}}_\tau$ different from $\bot$ and terms t of type τ. The relation $a \preceq_\tau v$ is defined by induction on τ. It can be read as: a is an approximation for the value v. The relation $a \preceq^{\mathrm{w}}_\tau t$ is an abbreviation. It means: If a is different from the bottom element, then t evaluates to a value approximated by a.

Definition 6.8
(6.8) $a \preceq^{\mathrm{w}}_\tau t \ :\Longleftrightarrow\ a = \bot_\tau$ or $\exists v\,(t \xrightarrow[\mathrm{n}]{\mathrm{ev}} v\ \ \&\ \ a \preceq_\tau v)$.
(6.9) $a \preceq_\iota v \ :\Longleftrightarrow\ v = c_a$.
(6.10) $f \preceq_{\rho\to\sigma} u \ :\Longleftrightarrow\ \forall a \in \mathfrak{A}^{\mathrm{n}}_\rho\, \forall t^\rho\,(a \preceq^{\mathrm{w}}_\rho t \implies f(a) \preceq^{\mathrm{w}}_\sigma u\ t)$.
(6.11) $\langle a, b\rangle \preceq_{\rho\times\sigma} \langle s, t\rangle \ :\Longleftrightarrow\ a \preceq^{\mathrm{w}}_\rho s\ \ \&\ \ b \preceq^{\mathrm{w}}_\sigma t$.

In the following lemma Σ is a substitution $[t_1/x_1, \ldots, t_n/x_n]$ of terms for variables.

Lemma 6.9 (*Adequacy of the call-by-name evaluation*)
If $\alpha(x) \preceq^{\mathrm{w}} x\Sigma$ for all variables $x \in \mathrm{FV}(t)$, then $[\![t]\!]^{\mathrm{n}}_\alpha \preceq^{\mathrm{w}} t\Sigma$.

Proof
The proof is by induction on t. We consider only cases that are essentially different from the corresponding case in the proof of the adequacy of call-by-value evaluation. Assume that $\alpha(x) \preceq^{\mathrm{w}} x\Sigma$ for all variables $x \in \mathrm{FV}(t)$. Since we have to show that $[\![t]\!]^{\mathrm{n}}_\alpha \preceq^{\mathrm{w}} t\Sigma$, we suppose that $[\![t]\!]^{\mathrm{n}}_\alpha \neq \bot$ and show in each case that there exists a value v such that $t\Sigma \xrightarrow[\mathrm{n}]{\mathrm{ev}} v$ and $[\![t]\!]^{\mathrm{n}}_\alpha \preceq v$.

Case $t \equiv f$. Since $f \xrightarrow[\mathrm{n}]{\mathrm{ev}} f$, we have to show that $[\![f]\!]^{\mathrm{n}}_\alpha \preceq f$. Remember that $[\![f]\!]^{\mathrm{n}}_\alpha = (f^{\mathfrak{A}})_\bot$. Let $a = \langle a_1, a_2\rangle$ and $b := (f^{\mathfrak{A}})_\bot(a)$. Assume that $a \preceq^{\mathrm{w}} t$. We have to show that $b \preceq^{\mathrm{w}} f\ t$. Suppose that $b \neq \bot$. By the definition of $(f^{\mathfrak{A}})_\bot$, it follows that $a \neq \bot$ and $a_i \neq \bot$ for $i = 1, 2$. By the definition of $\preceq^{\mathrm{w}}$, there exists a value v such that $t \xrightarrow[\mathrm{n}]{\mathrm{ev}} v$ and $a \preceq v$. The value v must be of the form $\langle s_1, s_2\rangle$ and we have $a_i \preceq^{\mathrm{w}} s_i$

https://doi.org/10.1017/S0956796898002974 Published online by Cambridge University Press

for $i = 1, 2$. Since $a_i \neq \bot$, there exist v_i such that $s_i \xrightarrow[\mathrm{n}]{\mathrm{ev}} v_i$ and $a_i \preceq v_i$, i.e. $v_i = c_{a_i}$, for $i = 1, 2$. So we obtain that $f\ t \xrightarrow[\mathrm{n}]{\mathrm{ev}} c_b$ and $b \preceq c_b$. Thus, $b \preceq^{\mathrm{w}} f\ t$.

Case $t \equiv \langle r, s\rangle$. Since $t\Sigma \equiv \langle r\Sigma, s\Sigma\rangle$ and $\langle r\Sigma, s\Sigma\rangle \xrightarrow[\mathrm{n}]{\mathrm{ev}} \langle r\Sigma, s\Sigma\rangle$, it remains to show that $[\![\langle r, s\rangle]\!]^{\mathrm{n}}_\alpha \preceq \langle r\Sigma, s\Sigma\rangle$. By definition this is the same as $[\![r]\!]^{\mathrm{n}}_\alpha \preceq^{\mathrm{w}} r\Sigma$ and $[\![s]\!]^{\mathrm{n}}_\alpha \preceq^{\mathrm{w}} s\Sigma$. This follows from the induction hypothesis.

Case $t \equiv \pi_i(r)$. By the induction hypothesis, we have $[\![r]\!]^{\mathrm{n}}_\alpha \preceq^{\mathrm{w}} r$. Since by assumption $[\![t]\!]^{\mathrm{n}}_\alpha \neq \bot$, we have $[\![r]\!]^{\mathrm{n}}_\alpha \neq \bot$, too. There exist a_1 and a_2 such that $[\![r]\!]^{\mathrm{n}}_\alpha = \langle a_1, a_2\rangle$ and $[\![t]\!]^{\mathrm{n}}_\alpha = [\![\pi_i(r)]\!]^{\mathrm{n}}_\alpha = a_i$. Thus, there exists a value v such that $r\Sigma \xrightarrow[\mathrm{n}]{\mathrm{ev}} v$ and $[\![r]\!]^{\mathrm{n}}_\alpha \preceq v$. The value v is of the form $\langle s_1, s_2\rangle$ and, by definition, we have $a_i \preceq^{\mathrm{w}} s_i$. By assumption, we know that $a_i \neq \bot$ and therefore there exists a value u such that $s_i \xrightarrow[\mathrm{n}]{\mathrm{ev}} u$ and $a_i \preceq u$. Since $r\Sigma \xrightarrow[\mathrm{n}]{\mathrm{ev}} \langle s_1, s_2\rangle$ and $s_i \xrightarrow[\mathrm{n}]{\mathrm{ev}} u$, we obtain $\pi_i(r)\Sigma \xrightarrow[\mathrm{n}]{\mathrm{ev}} u$ and $a_i \preceq u$. □

The following two theorems are special cases of the adequacy of call-by-name evaluation.

Theorem 6.10
Let t be a closed term of arbitrary type. Then $[\![t]\!]^{\mathrm{n}}_{\mathfrak{A}} \neq \bot$ iff $\mathrm{eval}^{\mathrm{n}}_{\mathfrak{A}}(t)$ is defined.

Proof
Assume that $[\![t]\!]^{\mathrm{n}}_{\mathfrak{A}} \neq \bot$. By the adequacy of call-by-name evaluation, it follows that $[\![t]\!]^{\mathrm{n}}_{\mathfrak{A}} \preceq^{\mathrm{w}} t$. By the definition of $\preceq^{\mathrm{w}}$ this means that there exists a value v such that $t \xrightarrow[\mathrm{n}]{\mathrm{ev}} v$ and $[\![t]\!]^{\mathrm{n}}_{\mathfrak{A}} \preceq v$. Hence $\mathrm{eval}^{\mathrm{n}}_{\mathfrak{A}}(t)$ is defined. For the other direction assume that $\mathrm{eval}^{\mathrm{n}}_{\mathfrak{A}}(t)$ is defined. This means that there exists a value v such that $t \xrightarrow[\mathrm{n}]{\mathrm{ev}} v$. By the soundness of call-by-name evaluation, we obtain that $[\![t]\!]^{\mathrm{n}}_{\mathfrak{A}} = [\![v]\!]^{\mathrm{n}}_{\mathfrak{A}} \neq \bot$. □

Theorem 6.11
Let t be a closed term of basic type ι and $a \in A_\iota$. Then $[\![t]\!]^{\mathrm{n}}_{\mathfrak{A}} = a$ iff $\mathrm{eval}^{\mathrm{n}}_{\mathfrak{A}}(t) = c_a$.

Proof
Assume that $[\![t]\!]^{\mathrm{n}}_{\mathfrak{A}} = a$. By the adequacy of call-by-name evaluation, it follows that $a \preceq^{\mathrm{w}}_\iota t$. Since $a \neq \bot$, there exists a value v such that $t \xrightarrow[\mathrm{n}]{\mathrm{ev}} v$ and $a \preceq_\iota v$. By definition of the relation $\preceq_\iota$, this means that $v = c_a$. Hence $\mathrm{eval}^{\mathrm{n}}_{\mathfrak{A}}(t) = c_a$. For the converse direction assume that $t \xrightarrow[\mathrm{n}]{\mathrm{ev}} c_a$. By the soundness of call-by-name evaluation, it follows that $[\![t]\!]^{\mathrm{n}}_{\mathfrak{A}} = [\![c_a]\!]^{\mathrm{n}}_{\mathfrak{A}} = a$. □

As an application of the adequacy results we mention the following well-known inclusion of call-by-value into call-by-name evaluation.

(6.12) Let t^τ be a closed term of arbitrary type.
If $\mathrm{eval}^{\mathrm{v}}_{\mathfrak{A}}(t)$ is defined, then $\mathrm{eval}^{\mathrm{n}}_{\mathfrak{A}}(t)$ is defined.

(6.13) Let t^ι be closed term of basic type and $a \in A_\iota$.
If $\mathrm{eval}^{\mathrm{v}}_{\mathfrak{A}}(t) = c_a$, then $\mathrm{eval}^{\mathrm{n}}_{\mathfrak{A}}(t) = c_a$.

These two inclusions follow easily from the previous two theorems, if we observe that call-by-value evaluation is also sound with respect to the denotation $[\![\cdot]\!]^{\mathrm{n}}$, i.e. if $t \xrightarrow[\mathrm{v}]{\mathrm{ev}} v$, then $[\![t]\!]^{\mathrm{n}}_\alpha = [\![v]\!]^{\mathrm{n}}_\alpha$ for all $\alpha \in \mathfrak{I}^{\mathrm{n}}_{\mathfrak{A}}$.

https://doi.org/10.1017/S0956796898002974 Published online by Cambridge University Press

Table 3. *Call-by-Value Truth Definition*

$$[\![s = t]\!]^{\mathrm{v}}_{\alpha} \coloneqq \begin{cases} \mathit{true}, & \text{if } \alpha \in \mathrm{def}_{\mathfrak{A}}(s) \cap \mathrm{def}_{\mathfrak{A}}(t) \text{ and } [\![s]\!]^{\mathrm{v}}_{\alpha} = [\![t]\!]^{\mathrm{v}}_{\alpha}; \\ \mathit{false}, & \text{otherwise.} \end{cases}$$

$$[\![t\!\downarrow]\!]^{\mathrm{v}}_{\alpha} \coloneqq \begin{cases} \mathit{true}, & \text{if } \alpha \in \mathrm{def}_{\mathfrak{A}}(t); \\ \mathit{false}, & \text{otherwise.} \end{cases}$$

$$[\![\neg A]\!]^{\mathrm{v}}_{\alpha} \coloneqq \begin{cases} \mathit{true}, & \text{if } [\![A]\!]^{\mathrm{v}}_{\alpha} = \mathit{false}; \\ \mathit{false}, & \text{otherwise.} \end{cases}$$

$$[\![A \wedge B]\!]^{\mathrm{v}}_{\alpha} \coloneqq \begin{cases} \mathit{true}, & \text{if } [\![A]\!]^{\mathrm{v}}_{\alpha} = \mathit{true} \text{ and } [\![B]\!]^{\mathrm{v}}_{\alpha} = \mathit{true}; \\ \mathit{false}, & \text{otherwise.} \end{cases}$$

$$[\![A \vee B]\!]^{\mathrm{v}}_{\alpha} \coloneqq \begin{cases} \mathit{true}, & \text{if } [\![A]\!]^{\mathrm{v}}_{\alpha} = \mathit{true} \text{ or } [\![B]\!]^{\mathrm{v}}_{\alpha} = \mathit{true}; \\ \mathit{false}, & \text{otherwise.} \end{cases}$$

$$[\![A \to B]\!]^{\mathrm{v}}_{\alpha} \coloneqq \begin{cases} \mathit{true}, & \text{if } [\![A]\!]^{\mathrm{v}}_{\alpha} = \mathit{false} \text{ or } [\![B]\!]^{\mathrm{v}}_{\alpha} = \mathit{true}; \\ \mathit{false}, & \text{otherwise.} \end{cases}$$

$$[\![\forall x^{\tau} A]\!]^{\mathrm{v}}_{\alpha} \coloneqq \begin{cases} \mathit{true}, & \text{if } [\![A]\!]^{\mathrm{v}}_{\alpha \frac{a}{x}} = \mathit{true} \text{ for all } a \in \mathfrak{A}^{\mathrm{v}}_{\tau}; \\ \mathit{false}, & \text{otherwise.} \end{cases}$$

$$[\![\exists x^{\tau} A]\!]^{\mathrm{v}}_{\alpha} \coloneqq \begin{cases} \mathit{true}, & \text{if there is an } a \in \mathfrak{A}^{\mathrm{v}}_{\tau} \text{ with } [\![A]\!]^{\mathrm{v}}_{\alpha \frac{a}{x}} = \mathit{true}; \\ \mathit{false}, & \text{otherwise.} \end{cases}$$

7 The basic logic of partial terms

Let $\mathfrak{A}$ be a partial first-order structure. The syntax of the basic logic of partial terms for $\mathfrak{A}$, BPT($\mathfrak{A}$), is that of many-sorted first-order predicate calculus with equality, extended by a definedness predicate. The *atomic formulas* of BPT($\mathfrak{A}$) are $t\!\downarrow$ and $s^{\tau} = t^{\tau}$. The *formulas* of BPT($\mathfrak{A}$) are generated from the atomic formulas by applying the logical connectives and quantifiers and are of the form $\neg A$, $A \wedge B$, $A \vee B$, $A \to B$, $\forall x^{\tau} A$ and $\exists x^{\tau} A$. The result of substituting a term t of type τ for a variable x of the same type in A is indicated as $A[t/x]$, or $A(t)$ when A is written as $A(x)$.

The truth value $[\![A]\!]^{\mathrm{v}}_{\alpha}$ of a formula A in the structure $\mathfrak{A}$ is defined in Table 3. In the call-by-value truth definition $[\![\cdot]\!]^{\mathrm{v}}$, variables of type τ range over $\mathfrak{A}^{\mathrm{v}}_{\tau}$. The truth value $[\![A]\!]^{\mathrm{n}}_{\alpha}$ is defined in a similar way in Table 4. In the call-by-name truth definition $[\![\cdot]\!]^{\mathrm{n}}$, variables of type τ range over $\mathfrak{A}^{\mathrm{n}}_{\tau}$ and include the element $\perp_{\tau}$.

The language of BPT has equality ($=$) and definedness ($\downarrow$) as basic predicate symbols. The partial equality $\simeq$ and the predicates $\sqsubseteq_{\tau}$ are defined symbols. For each type τ formulas $s^{\tau} \simeq t^{\tau}$ and $s^{\tau} \sqsubseteq_{\tau} t^{\tau}$ are defined. The intuitive meaning of $s \simeq t$ is that (i) s is defined iff t is defined and (ii) if they are both defined, then they are equal. The meaning of $s \sqsubseteq t$ is that s is less defined than t. In the following definition, the notion $A :\equiv B$ means that A is a syntactic abbreviation for B.

Definition 7.1

(7.1) $s \simeq t :\equiv s\!\downarrow \vee t\!\downarrow \to s = t$.

(7.2) $s \sqsubseteq_{\iota} t :\equiv s\!\downarrow \to s = t$.

(7.3) $s \sqsubseteq_{\rho\to\sigma} t :\equiv s\!\downarrow \to t\!\downarrow \wedge \forall x^{\rho}(s\ x \sqsubseteq_{\sigma} t\ x)$, where $x \notin \mathrm{FV}(s) \cup \mathrm{FV}(t)$.

(7.4) $s \sqsubseteq_{\rho\times\sigma} t :\equiv s\!\downarrow \to t\!\downarrow \wedge \pi_1(s) \sqsubseteq_{\rho} \pi_1(t) \wedge \pi_2(s) \sqsubseteq_{\sigma} \pi_2(t)$.

The partial equality $\simeq$ has the property that $[\![s \simeq t]\!]^{\mathrm{v}}_{\alpha} = \mathit{true}$ iff

(7.5) $\alpha \notin \mathrm{def}_{\mathfrak{A}}(s)$ and $\alpha \notin \mathrm{def}_{\mathfrak{A}}(t)$, or

https://doi.org/10.1017/S0956796898002974 Published online by Cambridge University Press

Table 4. *Call-by-Name Truth Definition*

$$[\![s = t]\!]^{\mathrm{n}}_{\alpha} \;:=\; \begin{cases} \mathit{true}, & \text{if } [\![s]\!]^{\mathrm{n}}_{\alpha} \neq \bot \text{ and } [\![s]\!]^{\mathrm{n}}_{\alpha} = [\![t]\!]^{\mathrm{n}}_{\alpha}; \\ \mathit{false}, & \text{otherwise.} \end{cases}$$

$$[\![t\!\downarrow]\!]^{\mathrm{n}}_{\alpha} \;:=\; \begin{cases} \mathit{true}, & \text{if } [\![t]\!]^{\mathrm{n}}_{\alpha} \neq \bot; \\ \mathit{false}, & \text{otherwise.} \end{cases}$$

$$[\![\forall x^{\tau} A]\!]^{\mathrm{n}}_{\alpha} \;:=\; \begin{cases} \mathit{true}, & \text{if } [\![A]\!]^{\mathrm{n}}_{\alpha\frac{a}{x}} = \mathit{true} \text{ for all } a \in \mathfrak{A}^{\mathrm{n}}_{\tau}; \\ \mathit{false}, & \text{otherwise.} \end{cases}$$

$$[\![\exists x^{\tau} A]\!]^{\mathrm{n}}_{\alpha} \;:=\; \begin{cases} \mathit{true}, & \text{if there is an } a \in \mathfrak{A}^{\mathrm{n}}_{\tau} \text{ with } [\![A]\!]^{\mathrm{n}}_{\alpha\frac{a}{x}} = \mathit{true}; \\ \mathit{false}, & \text{otherwise.} \end{cases}$$

(7.6) $\alpha \in \mathrm{def}_{\mathfrak{A}}(s) \cap \mathrm{def}_{\mathfrak{A}}(t)$ and $[\![s]\!]^{\mathrm{v}}_{\alpha} = [\![t]\!]^{\mathrm{v}}_{\alpha}$.

The relations $\sqsubseteq_{\tau}$ are defined in such a way that $[\![s \sqsubseteq_{\tau} t]\!]^{\mathrm{v}}_{\alpha} = \mathit{true}$ iff

(7.7) $\alpha \notin \mathrm{def}_{\mathfrak{A}}(s)$, or
(7.8) $\alpha \in \mathrm{def}_{\mathfrak{A}}(s) \cap \mathrm{def}_{\mathfrak{A}}(t)$ and $[\![s]\!]^{\mathrm{v}}_{\alpha} \sqsubseteq_{\tau} [\![t]\!]^{\mathrm{v}}_{\alpha}$ in the space $\mathfrak{A}^{\mathrm{v}}_{\tau}$.

For the call-by-name truth definition we have

(7.9) $[\![s \simeq t]\!]^{\mathrm{n}}_{\alpha} = \mathit{true}$ iff $[\![s]\!]^{\mathrm{n}}_{\alpha} = [\![t]\!]^{\mathrm{n}}_{\alpha}$,
(7.10) $[\![s \sqsubseteq_{\tau} t]\!]^{\mathrm{n}}_{\alpha} = \mathit{true}$ iff $[\![s]\!]^{\mathrm{n}}_{\alpha} \sqsubseteq_{\tau} [\![t]\!]^{\mathrm{n}}_{\alpha}$ in the space $\mathfrak{A}^{\mathrm{n}}_{\tau}$.

An alternative to this treatment would be to take $\simeq$ and $\sqsubseteq_{\tau}$ as basic predicates for each type τ and the laws of Definition 7.1 as basic axioms. Moreover, we could define $\downarrow$ in terms of equality, since $[\![t\!\downarrow]\!]^{\mathrm{v}}_{\alpha} = [\![t = t]\!]^{\mathrm{v}}_{\alpha}$ and $[\![t\!\downarrow]\!]^{\mathrm{n}}_{\alpha} = [\![t = t]\!]^{\mathrm{n}}_{\alpha}$.

The Substitution Lemma 5.1 implies that, if $\alpha \in \mathrm{def}_{\mathfrak{A}}(t)$ and $\beta = \alpha\frac{[\![t]\!]^{\mathrm{v}}_{\alpha}}{x}$, then $[\![A[t/x]]\!]^{\mathrm{v}}_{\alpha} = [\![A]\!]^{\mathrm{v}}_{\beta}$. Lemma 5.2 implies that $[\![A[t/x]]\!]^{\mathrm{n}}_{\alpha} = [\![A]\!]^{\mathrm{n}}_{\beta}$, where $\beta = \alpha\frac{[\![t]\!]^{\mathrm{n}}_{\alpha}}{x}$.

7.1 Axioms and rules of the basic logic of partial terms BPT($\mathfrak{A}$)

The axioms and rules of the basic logic of partial terms are sound with respect to both truth definitions, $[\![\cdot]\!]^{\mathrm{v}}$ and $[\![\cdot]\!]^{\mathrm{n}}$. It is important to note that most of the axioms below are actually axiom schemes and r, s, t range over arbitrary terms. Several axioms of are restricted to *syntactic values*. By that we mean terms generated as follows:

$$u, v ::= x \mid c \mid f \mid \langle u, v\rangle \mid \lambda x\, t \mid \mathrm{LFP}(\varphi = \lambda x\, t).$$

Note, that variables are syntactic values. The idea to instantiate quantified variables by syntactic values only is from (Stärk, 1997).

I. Propositional axioms: All propositional tautologies.

II. Quantifier axioms: For syntactic values v of type τ:

(7.11) $\forall x^{\tau} A(x) \to A(v)$
(7.12) $A(v) \to \exists x^{\tau} A(x)$

https://doi.org/10.1017/S0956796898002974 Published online by Cambridge University Press

III. Rules of inference:

$$\frac{A \qquad A \to B}{B} \qquad \frac{A(y^\tau) \to B}{\exists x^\tau A(x) \to B}(*) \qquad \frac{B \to A(y^\tau)}{B \to \forall x^\tau A(x)}(*)$$

(*) if the variable y does not appear free in the conclusion.

IV. Definedness axioms:

(7.13) $t\downarrow \to \exists x\,(t = x)$, for $x \notin \mathrm{FV}(t)$.
(7.14) $c\downarrow$, $f\downarrow$, $\langle u, v\rangle\downarrow$, $(\lambda x\,t)\downarrow$, $\mathrm{LFP}\,(\varphi = \lambda x\,t)\downarrow$

V. Equality axioms:

(7.15) $t\downarrow \to t = t$
(7.16) $t_1 = t_2 \to t_2 = t_1$
(7.17) $t_1 = t_2 \wedge t_2 = t_3 \to t_1 = t_3$
(7.18) $t_1 = t_2 \to t_1\downarrow \wedge t_2\downarrow$

VI. Application and abstraction:

(7.19) $s_1 \simeq s_2 \wedge t_1 \simeq t_2 \to s_1\ t_1 \simeq s_2\ t_2$
(7.20) $(s\ t)\downarrow \to s\downarrow \wedge \exists y\,(t \simeq y)$, for $y \notin \mathrm{FV}(t)$.
(7.21) $(\lambda x\,t)\ v \simeq t[v/x]$, for syntactic values v.

VII. Pairs and projections:

(7.22) $s_1 \simeq s_2 \wedge t_1 \simeq t_2 \to \langle s_1, t_1\rangle \simeq \langle s_2, t_2\rangle$
(7.23) $\langle s_1, t_1\rangle = \langle s_2, t_2\rangle \to s_1 \simeq s_2 \wedge t_1 \simeq t_2$
(7.24) $t\downarrow \to t = \langle \pi_1(t), \pi_2(t)\rangle$
(7.25) $\langle s, t\rangle\downarrow \to \exists x\,(s \simeq x) \wedge \exists y\,(t \simeq y)$, for $x \notin \mathrm{FV}(s)$ and $y \notin \mathrm{FV}(t)$.
(7.26) $\pi_i(t)\downarrow \to t\downarrow$

VIII. If-then-else:

(7.27) $r = \mathrm{tt} \to (r\ ?\ s : t) \simeq s$
(7.28) $r = \mathrm{ff} \to (r\ ?\ s : t) \simeq t$
(7.29) $(r\ ?\ s : t)\downarrow \to r\downarrow$
(7.30) $r\downarrow \to r = \mathrm{tt} \vee r = \mathrm{ff}$, if r is of type *bool*.
(7.31) $\mathrm{tt} \neq \mathrm{ff}$

IX. Extensionality:

(7.32) $s\downarrow \wedge t\downarrow \wedge \forall x\,(s\ x \simeq t\ x) \to s = t$, for $x \notin \mathrm{FV}(s) \cup \mathrm{FV}(t)$.

X. Monotonicity:

(7.33) $s_1 \sqsubseteq s_2 \wedge t_1 \sqsubseteq t_2 \to \langle s_1, t_2\rangle \sqsubseteq \langle s_2, t_2\rangle$
(7.34) $s_1 \sqsubseteq s_2 \wedge t_1 \sqsubseteq t_2 \to s_1\ t_1 \sqsubseteq s_2\ t_2$

XI. Least fixed points:

(7.35) $(\lambda x\,t)[\mathrm{LFP}\,(\varphi = \lambda x\,t)/\varphi] \sqsubseteq \mathrm{LFP}\,(\varphi = \lambda x\,t)$
(7.36) $(\lambda x\,t)[v/\varphi] \sqsubseteq v \to \mathrm{LFP}\,(\varphi = \lambda x\,t) \sqsubseteq v$, for syntactic values v.

https://doi.org/10.1017/S0956796898002974 Published online by Cambridge University Press

XII. Computational induction: Let $F :\equiv \mathrm{LFP}(\varphi = \lambda x\, t)$, where φ is of type $\sigma \to \iota$ and ι is a basic type. Assume that φ does not occur free in the formula $A(x, y)$. Then the computational induction scheme is:

$$(7.37)\quad \forall\varphi\Big[\forall x\big(\varphi\, x\downarrow \to A(x, \varphi\, x)\big) \to \forall x\big(t\downarrow \to A(x, t)\big)\Big] \to \forall x\big(F\, x\downarrow \to A(x, F\, x)\big)$$

XIII. Axioms for $\mathfrak{A}$: True axioms for the structure $\mathfrak{A}$. For instance, structural induction on natural numbers or lists. The equation $f\ \langle c_{a_1}, \ldots, c_{a_n}\rangle = c_b$ must be provable, if $f^{\mathfrak{A}}(a_1, \ldots, a_n) \simeq b$.

Remark 7.2

(a) The β-axiom (7.21) can be formulated equivalently as $(\lambda x\, t)\ x \simeq t$. The more general version (7.21) can then be derived using the quantifier rules and axioms. Note, however, that the axiom $x = y \to y = x$, for example, is weaker than the corresponding axiom scheme (7.16) for arbitrary terms s and t.

(b) Instead of the extensionality axiom (7.32) we could use the following η- and ξ-axioms:

$$(7.38)\quad t\downarrow \to \lambda x\,(t\ x) = t, \quad \text{for } x \notin \mathrm{FV}(t),$$
$$(7.39)\quad \forall x\,(s \simeq t) \to \lambda x\, s = \lambda x\, t.$$

(c) The minimality principle (7.36) is sometimes called Park's induction rule (Park, 1969). We will show below, that in the closure axiom (7.35) one could use equality instead of $\sqsubseteq$ as well [see (7.58)].

(d) The scheme of computational induction (7.37) is Shankar's version of the de Bakker-Scott induction principle (Shankar, 1989). The formula

$$B(\varphi) :\equiv \forall x\,\big(\varphi\ x\downarrow \to A(x, \varphi\ x)\big)$$

is admissible in the following sense: $B(\emptyset)$ is obviously true, and if $B(f_n)$ is true for every f_n of an increasing sequence of functions then also $B(\bigsqcup_{n\in\mathbb{N}} f_n)$ is true. Manna (1974) calls the last property *chain complete*. The premise of the principle (7.37) corresponds to the induction step from $B(f_n)$ to $B(f_{n+1})$, if $(f_n)_{n\in\mathbb{N}}$ is the sequence of functions that approximate the least fixed point of the equation $\varphi = \lambda x\, t$.

(e) One could add to BPT a scheme of *comprehension for monotonic functions*. For a formula $A(x^\sigma, y^\tau)$ let $\mathrm{mon}(x^\sigma, y^\tau, A)$ be an abbreviation for the following formula which expresses that A is the graph of a monotonic function:

$$\forall x \exists y\, A(x, y) \wedge \forall x_1, x_2, y_1, y_2\big(A(x_1, y_1) \wedge A(x_2, y_2) \wedge x_1 \sqsubseteq x_2 \to y_1 \sqsubseteq y_2\big)$$

By the comprehension scheme for monotonic functions we mean all formulas of the form

$$\mathrm{mon}(x^\sigma, y^\tau, A) \to \exists \varphi^{\sigma\to\tau} \forall x, y\big(\varphi\ x \simeq y \leftrightarrow A(x, y)\big), \quad \text{for } \varphi \notin \mathrm{FV}(A).$$

In order to validate this scheme one has to use monotonic (non-continuous) functions in the constructions of $\mathfrak{A}^{\mathrm{v}}_{\sigma\to\tau}$ and $\mathfrak{A}^{\mathrm{n}}_{\sigma\to\tau}$. The comprehension scheme for monotonic functions would increase the proof-theoretic strength dramatically. We would obtain a system similar to Farmer's partial function version of Church's simple theory of types (Farmer, 1990) or, even more stronger, Kuper's Zermelo-Fraenkel theory for partial functions (Kuper, 1994).

Lemma 7.3 (*Soundness of BPT*)
If $\mathrm{BPT}(\mathfrak{A}) \vdash A$, then $[\![A]\!]^{\mathrm{v}}_{\alpha} = \mathit{true}$ for all $\alpha \in \mathfrak{I}^{\mathrm{v}}_{\mathfrak{A}}$ and $[\![A]\!]^{\mathrm{n}}_{\alpha} = \mathit{true}$ for all $\alpha \in \mathfrak{I}^{\mathrm{n}}_{\mathfrak{A}}$.

Proof
We only show that the scheme of computational induction is valid under $[\![\cdot]\!]^{\mathrm{v}}$. The rest of the proof is routine. Let $F :\equiv \mathrm{LFP}(\varphi = \lambda x\, t)$. Assume that φ does not occur free in A and that F is of type $\sigma \to \iota$, where ι is a basic type. Assume that

$$[\![\forall \varphi\, [\forall x\, (\varphi\, x\!\downarrow \to A[\varphi\, x/y]) \to \forall x\, (t\!\downarrow \to A[t/y])]]\!]^{\mathrm{v}}_{\alpha} = \mathit{true}. \tag{$*$}$$

Let $f_0 := \emptyset$, $f_{n+1} := [\![\lambda x\, t)]\!]^{\mathrm{v}}_{\alpha \frac{f_n}{\varphi}}$ and $f := \bigsqcup_{n\in\mathbb{N}} f_n$. Then $f = [\![F]\!]^{\mathrm{v}}_{\alpha}$. We show by induction on n that

$$[\![\forall x\, (\varphi\, x\!\downarrow \to A[\varphi\, x/y])]\!]^{\mathrm{v}}_{\alpha \frac{f_n}{\varphi}} = \mathit{true}. \tag{$**$}$$

For $n = 0$ this is the case, since $\mathrm{dom}(f_0) = \emptyset$. Assume now that $(**)$ is true for n. By the assumption $(*)$, we obtain that $[\![\forall x\, (t\!\downarrow \to A[t/y])]\!]^{\mathrm{v}}_{\alpha \frac{f_n}{\varphi}} = \mathit{true}$. Assume that $a \in \mathfrak{A}^{\mathrm{v}}_{\sigma}$ and $[\![\varphi\, x\!\downarrow]\!]^{\mathrm{v}}_{\alpha \frac{f_{n+1}}{\varphi} \frac{a}{x}} = \mathit{true}$. This implies that $a \in \mathrm{dom}(f_{n+1})$. Since

$$[\![\varphi\, x]\!]^{\mathrm{v}}_{\alpha \frac{f_{n+1}}{\varphi} \frac{a}{x}} = f_{n+1}(a) = [\![\lambda x\, t]\!]^{\mathrm{v}}_{\alpha \frac{f_n}{\varphi}}(a) = [\![t]\!]^{\mathrm{v}}_{\alpha \frac{f_n}{\varphi} \frac{a}{x}},$$

we obtain that

$$[\![A[\varphi\, x/y]]\!]^{\mathrm{v}}_{\alpha \frac{f_{n+1}}{\varphi} \frac{a}{x}} = [\![A]\!]^{\mathrm{v}}_{\alpha \frac{f_n}{\varphi} \frac{a}{x} \frac{f_{n+1}(a)}{y}} = [\![A[t/y]]\!]^{\mathrm{v}}_{\alpha \frac{f_n}{\varphi} \frac{a}{x}} = \mathit{true}.$$

Thus, $(**)$ holds for $n+1$.

By definition, we have that $a \in \mathrm{dom}(f)$ and $f(a) = b$ iff there exists an $n \in \mathbb{N}$ such that $a \in \mathrm{dom}(f_n)$ and $f_n(a) = b$. Here we use the fact that ι is a basic type. Hence we obtain

$$[\![\forall x\, (\varphi\, x\!\downarrow \to A[\varphi\, x/y])]\!]^{\mathrm{v}}_{\alpha \frac{f}{\varphi}} = \mathit{true}.$$

Since $f = [\![F]\!]^{\mathrm{v}}_{\alpha}$, this implies that $[\![\forall x\, (F\, x\!\downarrow \to A[F\, x/y])]\!]^{\mathrm{v}}_{\alpha} = \mathit{true}$. □

7.2 Elementary properties of the basic logic of partial terms

It is easy to see that the partial equality $\simeq$ is an equivalence relation. The following formulas are derivable in BPT:

(7.40) $t \simeq t$
(7.41) $s \simeq t \to t \simeq s$
(7.42) $t_1 \simeq t_2 \wedge t_2 \simeq t_3 \to t_1 \simeq t_3$

The axioms for pairs and projections imply the following additional laws:

(7.43) $s \simeq t \to \pi_i(s) \simeq \pi_i(t)$
(7.44) $\langle s, t\rangle\!\downarrow \to \pi_1(\langle s, t\rangle) \simeq s \wedge \pi_2(\langle s, t\rangle) \simeq t$

Using the abbreviations of Definition 7.1 one can derive in BPT the following principles for the relations $\sqsubseteq_\tau$ by induction on the type τ:

(7.45) $t \sqsubseteq t$

https://doi.org/10.1017/S0956796898002974 Published online by Cambridge University Press

(7.46) $t_1 \sqsubseteq t_2 \wedge t_2 \sqsubseteq t_3 \to t_1 \sqsubseteq t_3$
(7.47) $s_1 \simeq s_2 \wedge t_1 \simeq t_2 \wedge s_1 \sqsubseteq t_1 \to s_2 \sqsubseteq t_2$
(7.48) $s \simeq t \leftrightarrow (s \sqsubseteq t \wedge t \sqsubseteq s)$
(7.49) $s \sqsubseteq t \wedge s\downarrow \to t\downarrow$
(7.50) $\neg s\downarrow \to s \sqsubseteq t$

In axioms (7.33) and (7.34) we postulate that pairing and application are monotonic with respect to $\sqsubseteq$. The monotonicity of projections, conditionals, abstraction and least-fixed point recursion is derivable in $\mathrm{BPT}(\mathfrak{A})$:

(7.51) $s \sqsubseteq t \to \pi_i(s) \sqsubseteq \pi_i(t)$
(7.52) $r_1 \sqsubseteq r_2 \wedge s_1 \sqsubseteq s_2 \wedge t_1 \sqsubseteq t_2 \to (r_1\ ?\ s_1 : t_1) \sqsubseteq (r_2\ ?\ s_2 : t_2)$
(7.53) $\forall x\,(s \sqsubseteq t) \to \lambda x\, s \sqsubseteq \lambda x\, t$
(7.54) $\forall \varphi, x\,(s \sqsubseteq t) \to \mathrm{LFP}\,(\varphi = \lambda x\, s) \sqsubseteq \mathrm{LFP}\,(\varphi = \lambda x\, t)$

Proof
[of (7.54)] Assume that $\forall \varphi, x\,(s \sqsubseteq t)$. From (7.53) we obtain that $\forall \varphi(\lambda x\, s \sqsubseteq \lambda x\, t)$ is derivable as well. Using the abbreviation $F :\equiv \mathrm{LFP}\,(\varphi = \lambda x\, t)$ we obtain that

$$(\lambda x\, s)[F/\varphi] \sqsubseteq (\lambda x, t)[F/\varphi].$$

By the closure axiom (7.35) and the transitivity property (7.46) it follows that $(\lambda x\, s)[F/\varphi] \sqsubseteq F$. Hence we can apply the minimality axiom (7.36) and obtain that $\mathrm{LFP}\,(\varphi = \lambda x\, s) \sqsubseteq F$. □

For the following context lemma remember that a context $C[*]$ is a typed term with one ore several occurrences of a typed hole. The lemma is proved by induction on the length of the context $C[*]$.

Lemma 7.4 (*Context*)
Let $C[*]$ be a context and s and t be terms. Assume that the list $\vec{x}$ contains all the free variables of s or t that are bound by the context in $C[s]$ or $C[t]$. Then $\mathrm{BPT}(\mathfrak{A}) \vdash \forall \vec{x}\,(s \sqsubseteq t) \to C[s] \sqsubseteq C[t]$.

As a consequence of the context lemma and the anti-symmetry (7.48), we obtain the following substitution properties for terms r and formulas A:

(7.55) $s \sqsubseteq t \to r[s/x] \sqsubseteq r[t/x]$
(7.56) $s \simeq t \to r[s/x] \simeq r[t/x]$
(7.57) $s \simeq t \wedge A[s/x] \to A[t/x]$

Another consequence of the context lemma is the fixed point property:

(7.58) $(\lambda x\, t)[\mathrm{LFP}\,(\varphi = \lambda x\, t)/\varphi] = \mathrm{LFP}\,(\varphi = \lambda x\, t)$

Proof
Let $F :\equiv \mathrm{LFP}\,(\varphi = \lambda x\, t)$. The closure axiom (7.35) says that $(\lambda x\, t)[F/\varphi] \sqsubseteq F$. From the monotonicity of terms (7.55), we obtain that

$$(\lambda x\, t)[(\lambda x\, t)[F/\varphi]/\varphi] \sqsubseteq (\lambda x\, t)[F/\varphi].$$

Now we can apply the minimality principle (7.36) and obtain $F \sqsubseteq (\lambda x\, t)[F/\varphi]$. The anti-symmetry property (7.48) yields $F = (\lambda x\, t)[F/\varphi]$. □

https://doi.org/10.1017/S0956796898002974 Published online by Cambridge University Press

An interesting consequence of the substitution principle (7.57) and the definedness axiom (7.13) are the following rules for quantifiers:

(7.59) $\forall x\, A \wedge t{\downarrow} \to A[t/x], \quad A[t/x] \wedge t{\downarrow} \to \exists x\, A$

Remember that in BPT($\mathfrak{A}$) we are allowed to instantiate quantified variables by syntactic values only. Now we can instantiate them with defined terms, too. For example, we obtain the following principles:

(7.60) $s{\downarrow} \to (\lambda x\, t)\, s \simeq t[s/x]$
(7.61) $s{\downarrow} \wedge t{\downarrow} \to \langle s, t\rangle{\downarrow}$
(7.62) $t{\downarrow} \to \pi_1(\langle s, t\rangle) \simeq s, \quad s{\downarrow} \to \pi_2(\langle s, t\rangle) \simeq t$

Instead of the least-fixed point constructs LFP($\varphi = \lambda x\, t$) most authors use constants FIX_τ of type $(\tau \to \tau) \to \tau$ with the property

$$\mathrm{FIX}_\tau\ \psi \simeq \psi\ (\mathrm{FIX}_\tau\ \psi). \qquad (+)$$

It τ is a basic type then this equation cannot be solved in most cases. For functional types $\tau = \rho \to \sigma$, however, we can define FIX_τ as a program in the following way:

$$\mathrm{FIX}_\tau :\equiv \lambda\psi^{\tau\to\tau}\mathrm{LFP}\,(\varphi^\tau = \lambda x^\rho((\psi\ \varphi)\ x)).$$

Let $F :\equiv \mathrm{LFP}\,(\varphi = \lambda x((\psi\ \varphi)\ x))$. Then we can derive in BPT:

$$\begin{array}{rcll} (\mathrm{FIX}_\tau\ \psi)\ x & \simeq & F\ x & \text{[Axioms (7.19) and (7.21)]} \\ & \simeq & \lambda x\,((\psi\ F)\ x)\ x & \text{[(7.58) and Axiom (7.19)]} \\ & \simeq & (\psi\ F)\ x & \text{[Axiom (7.21)]} \\ & \simeq & (\psi\ (\mathrm{FIX}_\tau\ \psi))\ x & \text{[Axioms (7.19) and (7.21)]} \end{array}$$

Thus we have $(\mathrm{FIX}_\tau\ \psi)\ x \simeq (\psi\ (\mathrm{FIX}_\tau\ \psi))\ x$. Using the extensionality axiom (7.32) we obtain

$$\psi\ (\mathrm{FIX}_\tau\ \psi){\downarrow} \to \mathrm{FIX}_\tau\ \psi = \psi\ (\mathrm{FIX}_\tau\ \psi).$$

More interesting examples of principles, that can be derived in BPT($\mathfrak{A}$) and are therefore true under call-by-value as well as call-by-name, can be found in the appendix.

Our interest now turns to two extensions of the basic logic of partial terms. The first one is obtained by adding more strictness axioms. It is adequate for call-by-value evaluation. In the second extension, it is allowed to instantiate variables by arbitrary (possibly undefined) terms. The second extension is adequate for call-by-name evaluation.

8 The logic of partial terms for call-by-value

The logic of partial terms for call-by-value, VPT($\mathfrak{A}$), contains in addition to the axioms and rules of the basic logic of partial terms the axiom $x^\tau{\downarrow}$ which says that variables are defined for each type τ:

$$\mathrm{VPT}\,(\mathfrak{A}) := \mathrm{BPT}\,(\mathfrak{A}) + x^\tau{\downarrow}.$$

https://doi.org/10.1017/S0956796898002974 Published online by Cambridge University Press

A consequence of the definedness of variables is that $\exists x\,(t \simeq x)$ is equivalent to $t\!\downarrow$. Therefore we obtain that application and pairs are strict in VPT$(\mathfrak{A})$. The following principles are derivable in VPT$(\mathfrak{A})$:

(8.1) $s\,t\!\downarrow \to t\!\downarrow$
(8.2) $\langle s,t\rangle\!\downarrow \to s\!\downarrow \wedge\, t\!\downarrow$

Since $[\![x\!\downarrow]\!]^{\mathrm{v}}_{\alpha} = \mathit{true}$ for arbitrary assignments $\alpha \in \mathfrak{I}^{\mathrm{v}}_{\mathfrak{A}}$, the logic VPT$(\mathfrak{A})$ is sound with respect to the truth-definition $[\![\cdot]\!]^{\mathrm{v}}$ of Table 3.

Lemma 8.1 (*Soundness of VPT*)
If VPT$(\mathfrak{A}) \vdash A$ then $[\![A]\!]^{\mathrm{v}}_{\alpha} = \mathit{true}$ for all assignments $\alpha \in \mathfrak{I}^{\mathrm{v}}_{\mathfrak{A}}$.

Call-by-value evaluation can be interpreted in VPT. In fact, it can even be interpreted in BPT.

Lemma 8.2 (*Interpretation of call-by-value evaluation in VPT*)
(a) If $t \xrightarrow[\mathrm{v}]{\mathrm{ev}} v$, then VPT$(\mathfrak{A}) \vdash t = v$.
(b) If $u\,v \xrightarrow[\mathrm{v}]{\mathrm{ap}} w$, then VPT$(\mathfrak{A}) \vdash u\,v = w$.

Proof
By induction on the definition of $\xrightarrow[\mathrm{v}]{\mathrm{ev}}$ and $\xrightarrow[\mathrm{v}]{\mathrm{ap}}$. Let us consider the call-by-value rule for the application of least fixed points:

$$\frac{t[F/\varphi, u/x] \xrightarrow[\mathrm{v}]{\mathrm{ev}} v}{F\;u \xrightarrow[\mathrm{v}]{\mathrm{ap}} v}, \quad \text{where } F :\equiv \mathrm{LFP}\,(\varphi = \lambda x\,t).$$

Then we can derive in VPT:

$$\begin{array}{lcll} F\,u & \simeq & (\lambda x\,t)[F/\varphi]\,u & \text{[Fixed point property (7.58) and Axiom (7.19)]} \\ & \simeq & t[F/\varphi, u/x] & \text{[Axiom (7.21)]} \\ & = & v & \text{[Induction hypothesis]} \end{array}$$

So the equation $F\,u = v$ is provable in VPT. □

The following theorems say that VPT is adequate for call-by-value evaluation. They follow directly from what we have shown so far. The main ingredients in their proofs are the adequacy results of section 6.

Theorem 8.3 (*Adequacy of VPT with respect to termination*)
Let t be a closed term of arbitrary type. Then the strict evaluation of t terminates iff the formula $t\!\downarrow$ is provable in VPT$(\mathfrak{A})$.

Theorem 8.4 (*Adequacy of VPT with respect to strict evaluation*)
Let t be a closed term of basic type ι and $a \in A_{\iota}$. Then the strict evaluation of t terminates with result c_a iff the equation $t = c_a$ is provable in VPT$(\mathfrak{A})$.

Theorem 8.5 (*Soundness of VPT with respect to contexts and basic observations*)
Assume that the formula $s \sqsubseteq t$ is provable in VPT$(\mathfrak{A})$. Then we have for all contexts $C[*]$ of basic type such that $C[s]$ and $C[t]$ are closed: if the strict evaluation of $C[s]$ terminates with value c_a, then the strict evaluation of $C[t]$ terminates with value c_a, too.

https://doi.org/10.1017/S0956796898002974 Published online by Cambridge University Press

Proof
Assume that $s \sqsubseteq t$ is provable in VPT$(\mathfrak{A})$ and that $C[s] \xrightarrow[\mathrm{v}]{\mathrm{ev}} c_a$. By the Context Lemma 7.4, it follows that $C[s] \sqsubseteq C[t]$ is derivable, too. Since the context $C[*]$ is of basic type, the formula $C[s] \sqsubseteq C[t]$ is an abbreviation for $C[s]\!\downarrow \to C[s] = C[t]$. By Lemma 8.2, it follows that $C[s] = c_a$ is provable. Hence $C[s]\!\downarrow$ and $C[t] = c_a$ are derivable in VPT$(\mathfrak{A})$. By Lemma 8.1 it follows that $[\![C[t]]\!]^{\mathrm{v}}_{\mathfrak{A}} = a$. Theorem 6.6 yields that $C[t] \xrightarrow[\mathrm{v}]{\mathrm{ev}} c_a$. □

As an application of the last theorem and property (7.48) we obtain that, if an equation $s \simeq t$ is provable in VPT$(\mathfrak{A})$, then the two programs s and t are observationally equivalent for contexts of basic type.

9 The logic of partial terms for call-by-name

The logic or partial terms for call-by-name, NPT$(\mathfrak{A})$, is obtained from the basic logic of partial terms by adding the axioms $\exists x^{\tau} \neg x\!\downarrow$ for each type τ which says that there exist undefined objects for each type τ.

$$\mathrm{NPT}(\mathfrak{A}) \coloneqq \mathrm{BPT}(\mathfrak{A}) + \exists x^{\tau} \neg x\!\downarrow.$$

Since $[\![\exists x^{\tau} \neg x\!\downarrow]\!]^{\mathrm{n}}_{\alpha} = \mathit{true}$, for arbitrary assigenments $\alpha \in \mathfrak{I}^{\mathrm{n}}_{\mathfrak{A}}$, the logic NPT$(\mathfrak{A})$ is sound with respect to the truth-definition $[\![\cdot]\!]^{\mathrm{n}}$ of Table 4.

Lemma 9.1 (*Soundness of NPT*)
If NPT$(\mathfrak{A}) \vdash A$ then $[\![A]\!]^{\mathrm{n}}_{\alpha} = \mathit{true}$ for all assignments $\alpha \in \mathfrak{I}^{\mathrm{n}}_{\mathfrak{A}}$.

The following prinicples are derivable in NPT$(\mathfrak{A})$:

(9.1) $\exists x\,(t \simeq x)$, for $x \notin \mathrm{FV}(t)$.
(9.2) $\forall x\, A(x) \to A(t)$, $A(t) \to \exists x\, A(x)$
(9.3) $(\lambda x\, t)s \simeq t[s/x]$
(9.4) $\langle s, t\rangle\!\downarrow$
(9.5) $\pi_1(\langle s, t\rangle) \simeq s$, $\pi_2(\langle s, t\rangle) \simeq t$

These principles are used for the interpretation of call-by-name evaluation in NPT.

Lemma 9.2 (*Interpretation of call-by-name evaluation in NPT*)
(a) If $t \xrightarrow[\mathrm{n}]{\mathrm{ev}} v$ then NPT$(\mathfrak{A}) \vdash t = v$.
(b) If $u\ t \xrightarrow[\mathrm{n}]{\mathrm{ap}} v$ then NPT$(\mathfrak{A}) \vdash u\ t = v$.

Proof
By induction on the definition of $\xrightarrow[\mathrm{n}]{\mathrm{ev}}$ and $\xrightarrow[\mathrm{n}]{\mathrm{ap}}$. Let us consider the rule for the lazy application of least fixed points:

$$\frac{t[F/\varphi, s/x] \xrightarrow[\mathrm{v}]{\mathrm{ev}} v}{F\ s \xrightarrow[\mathrm{v}]{\mathrm{ap}} v}, \quad \text{where } F \coloneqq \mathrm{LFP}(\varphi = \lambda x\, t).$$

Then we can derive in NPT:

$$\begin{array}{lcll} F\ s & \simeq & (\lambda x\, t)[F/\varphi]\ s & \text{[Fixed point property (7.58) and Axiom (7.19)]} \\ & \simeq & t[F/\varphi, s/x] & \text{[Unrestricted } \beta\text{-axiom (9.3)]} \\ & = & v & \text{[Induction hypothesis]} \end{array}$$

https://doi.org/10.1017/S0956796898002974 Published online by Cambridge University Press

So the equation $F\ s = v$ is provable in NPT. □

The main theorems which relate NPT to non-strict evaluation are the same as in the strict case. They follow directly from the previous results using computational adequacy of section 6.

Theorem 9.3 (*Adequacy of NPT with respect to termination*)
Let t be a closed term of arbitrary type. Then the non-strict evaluation of t terminates iff the formula $t\downarrow$ is provable in NPT($\mathfrak{A}$).

Theorem 9.4 (*Adequacy of NPT with respect to strict evaluation*)
Let t be a closed term of basic type ι and $a \in A_\iota$. Then the non-strict evaluation of t terminates with result c_a iff the equation $t = c_a$ is provable in NPT($\mathfrak{A}$).

Theorem 9.5 (*Soundness of NPT with respect to contexts and basic observations*)
Assume that the formula $s \sqsubseteq t$ is provable in NPT($\mathfrak{A}$). Then we have for all contexts $C[*]$ of basic type such that $C[s]$ and $C[t]$ are closed: if the non-strict evaluation of $C[s]$ terminates with value c_a, then the non-strict evaluation of $C[t]$ terminates with value c_a, too.

A comparison of VPT and NPT

The main difference between NPT and VPT is that in NPT quantifiers range over possibly undefined objects, whereas in VPT they range over defined objects only. So NPT is no longer a *logic of definedness* (Feferman, 1995) and the question is, whether this could be changed. We do not think so. How could the axiom of extensionality (7.32) for call-by-name be formulated without letting quantifiers range over the object 'undefined'? Under call-by-name two functions are equal, only if they agree on undefined arguments, too. For example, the functions

$$\lambda x^{bool}(x\ ?\ \mathbb{tt} : \mathbb{tt}) \quad \text{and} \quad \lambda x^{bool}\,\mathbb{tt}$$

agree on defined arguments, but not on the argument 'undefined'. Under call-by-name they are not considered as equal. Also in the definition of the 'less defined' relation ($\sqsubseteq$) quantifiers have to range over the object 'undefined' in the case of call-by-name (see Definition 7.1). Therefore we believe that a logic of definedness for call-by-name does not exist.

A Appendix: Simultaneous least fixed points, Moschovakis' formal language of recursion (FLR) and program transformation

Moschovakis (1989) introduces the *Formal Language of Recursion* (FLR), a formal language of terms with two semantics, a *denotational* semantics and an *intensional* semantics. He studies a *calculus of reductions and equivalence* for terms, formalizing *compilation* of terms into unique normal forms. He considers side effects and therefore the order of the evaluation of the arguments of a function is relevant.

In this appendix, we show how BPT can be used to prove the soundness of Moschovakis' reduction calculus. We refer the reader to Moschovakis (1989) for the

https://doi.org/10.1017/S0956796898002974 Published online by Cambridge University Press

exact definition of the reduction calculus. We only consider the two most important reductions here, namely the reduction of nested recursion to simultaneous recursion. This appendix can be understood as an illustration of how the logic BPT can be used to prove the correctness of program transformations both with respect to strict and non-strict evaluation.

First we have to extend the language of programs. We add new simultaneous least fixed point constructs. We extend the definition of programs by the following clause: if t_i are terms of type σ_i for $i = 1, \ldots, n$ then

$$\mathrm{SLFP}_i(\varphi_1^{\rho_1 \to \sigma_1} = \lambda x_1^{\rho_1} t_1^{\sigma_1}, \ldots, \varphi_n^{\rho_n \to \sigma_n} = \lambda x_n^{\rho_n} t_n^{\sigma_n})$$

is a term of type $\rho_i \to \sigma_i$ for $i = 1, \ldots, n$. The variables φ_i are all bound. For example, the variable φ_2 is considered bound in the term t_1.

Recursors can now be defined using simultaneous least fixed points. The following *letrec* expression is therefore not considered as a basic construct of the programming language but as a defined notion. We define

$$(\text{letrec } \varphi_1 = \lambda x_1\, t_1 \;\;\&\;\; \ldots \;\;\&\;\; \varphi_n = \lambda x_n\, t_n \text{ in } t_0)$$

to be an abbreviation for the term $t_0[s_1/\varphi_1, \ldots, s_n/\varphi_n]$, where

$$s_i :\equiv \mathrm{SLFP}_i(\varphi_1 = \lambda x_1\, t_1, \ldots, \varphi_n = \lambda x_n\, t_n) \quad \text{for } i = 1, \ldots, n.$$

Moschovakis uses the notion

$$\mathrm{rec}(x_1, \varphi_1, \ldots, x_n, \varphi_n)[t_0, t_1, \ldots, t_n]$$

for this kind of recursion. His notation has the advantage that it indicates more clearly that the variables $\varphi_1, \ldots, \varphi_n$ are bound in the terms $t_0, t_1, \ldots, t_n$.

In modern terminology, an expression $\mathrm{SLFP}(\varphi_1 = \lambda x_1\, t_1, \ldots, \varphi_n = \lambda x_n\, t_n)$ is called an *object* and the term $\mathrm{SLFP}_i(\varphi_1 = \lambda x_1\, t_1, \ldots, \varphi_n = \lambda x_n\, t_n)$ can be understood as the *invocation* of the *method* φ_i.

We extend the evaluation calculi for strict and non-strict evaluation in the obvious way. Using the abbreviation

$$s_i :\equiv \mathrm{SLFP}_i(\varphi_1 = \lambda x_1\, t_1, \ldots, \varphi_n = \lambda x_n\, t_n) \quad \text{for } i = 1, \ldots, n,$$

we add the following n rules to *call-by-value* evaluation:

$$\frac{t_i[u/x_i, s_1/\varphi_1, \ldots, s_n/\varphi_n] \xrightarrow[\mathrm{v}]{\mathrm{ev}} v}{s_i\, u \xrightarrow[\mathrm{v}]{\mathrm{ap}} v}$$

The rules for *call-by-name* evaluation are almost the same:

$$\frac{t_i[r/x_i, s_1/\varphi_1, \ldots, s_n/\varphi_n] \xrightarrow[\mathrm{n}]{\mathrm{ev}} v}{s_i\, r \xrightarrow[\mathrm{n}]{\mathrm{ap}} v}$$

In the second case, the argument r of s_i may not be a value but an arbitrary unevaluated term.

The denotations $[\![s_i]\!]^{\mathrm{v}}_{\alpha}$ of the single components of a simultaneous recursion are given by the least simultaneous fixed points of the operators Γ^i_{α} defined by

$$\Gamma^i_{\alpha}(f_1, \ldots, f_n) := [\![\lambda x_i\, t_i]\!]^{\mathrm{v}}_{\alpha \frac{f_1}{\varphi_1} \ldots \frac{f_n}{\varphi_n}} \quad \text{for } i = 1, \ldots, n.$$

https://doi.org/10.1017/S0956796898002974 Published online by Cambridge University Press

This means that $([\![s_1]\!]^{\mathrm{v}}_{\alpha},\ldots,[\![s_n]\!]^{\mathrm{v}}_{\alpha})$ is the least n-tuple of functions $(f_1,\ldots,f_n)$ such that $f_i = \Gamma^i_{\alpha}(f_1,\ldots,f_n)$ for $i = 1,\ldots,n$. The denotations $[\![s_i]\!]^{\mathrm{n}}_{\alpha}$ in the type structures $\mathfrak{A}^{\mathrm{n}}_{\tau}$ are defined in the same way.

Finally, we can formulate the axioms for SLFP in the basic logic of partial terms. We do not include computational induction, since we do not need it in this appendix. We only add the fixed point property (FIX) and the minimality property (MIN) to BPT. These principles are:

$$s_i = (\lambda x_i\, t_i)[s_1/\varphi_1,\ldots,s_n/\varphi_n] \tag{FIX}$$

$$\forall \varphi_1,\ldots,\varphi_n \left(\bigwedge_{i=1}^{n} \lambda x_i\, t_i \sqsubseteq \varphi_i \rightarrow \bigwedge_{i=1}^{n} s_i \sqsubseteq \varphi_i \right) \tag{MIN}$$

where s_i stands for the term $\mathrm{SLFP}_i(\varphi_1 = \lambda x_1\, t_1,\ldots,\varphi_n = \lambda x_n\, t_n)$ for $i = 1,\ldots,n$.

In the rest of this appendix we will show the soundness of two of Moschovakis' reduction rules with respect to BPT. The first theorem is Moschovakis' rule R4. It is the reduction of nested recursion to simultaneous recursion.

Theorem A.1
If ψ does not occur free in the program r, then we can derive in BPT the following equation:

$$(\text{letrec } \varphi = \lambda x\, r \text{ in } (\text{letrec } \psi = \lambda y\, s \text{ in } t)) \simeq (\text{letrec } \varphi = \lambda x\, r \ \ \&\ \ \psi = \lambda y\, s \text{ in } t).$$

Proof
We use the following abbreviations:

$$a_1 :\equiv \mathrm{LFP}\,(\varphi = \lambda x\, r), \quad a_2 :\equiv \mathrm{LFP}\,(\psi = (\lambda y\, s)[a_1/\varphi]),$$

$$b_i :\equiv \mathrm{SLFP}_i(\varphi = \lambda x\, r, \psi = \lambda y\, s), \quad \text{for } i = 1,2.$$

By definition, we have

$$(\text{letrec } \varphi = \lambda x\, r \text{ in } (\text{letrec } \psi = \lambda y\, s \text{ in } t)) \equiv t[a_1/\varphi, a_2/\psi].$$

We also have

$$(\text{letrec } \varphi = \lambda x\, r \ \ \&\ \ \psi = \lambda y\, s \text{ in } t) \equiv t[b_1/\varphi, b_2/\psi]$$

Thus, if we can derive $a_1 = b_1$ and $a_2 = b_2$ we are done, since BPT proves

$$a_1 = b_1 \wedge a_2 = b_2 \rightarrow t[a_1/\varphi, a_2/\psi] \simeq t[b_1/\varphi, b_2/\psi].$$

First we show that we can derive $b_1 \sqsubseteq a_1$ and $b_2 \sqsubseteq a_2$. By the closure axiom (7.35) for $\mathrm{LFP}\,(\varphi = \lambda x\, r)$, we have $(\lambda x\, r)[a_1/\varphi] \sqsubseteq a_1$. Since ψ is not free in r, the term $(\lambda x\, r)[a_1/\varphi]$ is the same as $(\lambda x\, r)[a_1/\varphi, a_2/\psi]$ and we thus obtain

$$(\lambda x\, r)[a_1/\varphi, a_2/\psi] \sqsubseteq a_1.$$

Again by the closure axiom (7.35) but this time for $\mathrm{LFP}\,(\psi = (\lambda y\, s)[a_1/\varphi])$, we obtain $(\lambda y\, s)[a_1/\varphi][a_2/\psi] \sqsubseteq a_2$. Since ψ does not occur free in a_1, we have

$$(\lambda y\, s)[a_1/\varphi, a_2/\psi] \sqsubseteq a_2.$$

Now, we can apply (MIN) and obtain that $b_1 \sqsubseteq a_1$ and $b_2 \sqsubseteq a_2$.

https://doi.org/10.1017/S0956796898002974 Published online by Cambridge University Press

For the converse inequalities, we use (FIX) and obtain $(\lambda x\, r)[b_1/\varphi, b_2/\psi] \sqsubseteq b_1$. Since ψ is not free in r, we have $(\lambda x\, r)[b_1/\varphi] \sqsubseteq b_1$ and, by the minimization axiom (7.36), we obtain $a_1 \sqsubseteq b_1$, and by antisymmetry (7.48), $a_1 = b_1$.

Again by (FIX) it follows that $(\lambda y\, s)[b_1/\varphi, b_2/\psi] \sqsubseteq b_2$. By the substitution property (7.56), we can derive

$$(\lambda y\, s)[b_1/\varphi, b_2/\psi] = (\lambda y\, s)[a_1/\varphi, b_2/\psi].$$

Since ψ is not free in a_1 the term $(\lambda y\, s)[a_1/\varphi, b_2/\psi]$ is the same as $(\lambda y\, s)[a_1/\varphi][b_2/\psi]$ and, by property (7.47), we obtain $(\lambda y\, s)[a_1/\varphi][b_2/\psi] \sqsubseteq b_2$. By the minimization axiom (7.36), we obtain $a_2 \sqsubseteq b_2$, and by antisymmetry, $a_2 = b_2$. □

The second theorem corresponds to rule R5 of Moschovakis. It is called the Bekivc-Scott principle. It is the reduction of iterated to simultaneous least fixed point recursion.

Theorem A.2
If χ does not occur free in r, s or t, then we can derive in BPT the following equation:

$$\begin{array}{c}(\text{letrec } \varphi = \lambda x\, (\text{letrec } \psi = \lambda y\, r \text{ in } s) \text{ in } t)\\ \simeq \\ (\text{letrec } \varphi = \lambda x\, s[\chi\ x/\psi] \ \ \&\ \ \chi = \lambda x\, \lambda y\, r[\chi\ x/\psi] \text{ in } t)\end{array}$$

Proof
We use the following abbreviations:

$$\begin{array}{c}a_2 :\equiv \text{LFP}\,(\psi = \lambda y\, r), \quad a_1 :\equiv \text{LFP}\,(\varphi = \lambda x\, (s[a_2/\psi])),\\ b_i :\equiv \text{SLFP}_i(\varphi = \lambda x\, s[\chi\ x/\psi], \chi = \lambda x\, \lambda y\, r[\chi\ x/\psi]), \quad \text{for } i = 1, 2.\end{array}$$

Note, that x and φ may appear free in a_2. They are, however, not free in a_1. Since (letrec $\psi = \lambda y\, r$ in s) is an abbreviation for $s[a_2/\psi]$, we have by definition that

$$(\text{letrec } \varphi = \lambda x\, (\text{letrec } \psi = \lambda y\, r \text{ in } s) \text{ in } t) \equiv t[a_1/\varphi].$$

Also by definition, we have

$$(\text{letrec } \varphi = \lambda x\, s[\chi\ x/\psi] \ \ \&\ \ \chi = \lambda x\, \lambda y\, r[\chi\ x/\psi] \text{ in } t) \equiv t[b_1/\varphi].$$

Since BPT proves

$$a_1 = b_1 \to t[a_1/\varphi] \simeq t[b_1/\varphi],$$

it is sufficient to show that $a_1 = b_1$ is derivable in BPT. It turns out, that in order to do this, we have to show that $\lambda x\, a_2[a_1/\varphi] = b_2$ is derivable in BPT, too.

We first show the inequalities $b_1 \sqsubseteq a_1$ and $b_2 \sqsubseteq \lambda x\, a_2[a_1/\varphi]$. We have

$$\begin{array}{lcl}(\lambda x\, s[\chi\ x/\psi])[a_1/\varphi, \lambda x\, a_2[a_1/\varphi]/\chi] & \equiv & (\lambda x\, s[\chi\ x/\psi])[\lambda x\, a_2/\chi][a_1/\varphi]\\ & \equiv & (\lambda x\, s[(\lambda x\, a_2)\ x/\psi])[a_1/\varphi]\\ \text{[Context Lemma 7.4]} & = & (\lambda x\, s[a_2/\psi])[a_1/\varphi]\\ \text{[Axiom (7.35)]} & \sqsubseteq & a_1.\end{array}$$

https://doi.org/10.1017/S0956796898002974 Published online by Cambridge University Press

In a similar way we obtain

$$\begin{array}{lcl}(\lambda y\, r[\chi\ x/\psi])[a_1/\varphi, \lambda x\, a_2[a_1/\varphi]/\chi] & \equiv & (\lambda y\, r[\chi\ x/\psi])[\lambda x\, a_2/\chi][a_1/\varphi] \\ & \equiv & (\lambda y\, r[(\lambda x\, a_2)\ x/\psi])[a_1/\varphi] \\ \text{[Context Lemma 7.4]} & = & (\lambda y\, r[a_2/\psi])[a_1/\varphi] \\ \text{[Axiom (7.35)]} & \sqsubseteq & a_2[a_1/\varphi].\end{array}$$

Using the monotonicity property (7.53) we obtain

$$(\lambda x\, \lambda y\, r[\chi\ x/\psi])[a_1/\varphi, \lambda x\, a_2[a_1/\varphi]/\chi] \sqsubseteq \lambda x\, a_2[a_1/\varphi].$$

Hence we can apply (MIN) and obtain $b_1 \sqsubseteq a_1$ and $b_2 \sqsubseteq \lambda x\, a_2[a_1/\varphi]$.

For the converse inequalities $a_1 \sqsubseteq b_1$ and $\lambda x\, a_2[a_1/\varphi] \sqsubseteq b_2$, we use (FIX). We have

$$(\lambda x\, \lambda y\, r[\chi\ x/\psi])[b_1/\varphi, b_2/\chi] \sqsubseteq b_2$$

and, by the definition of $\sqsubseteq$, we obtain $(\lambda y\, r[b_2\ x/\psi])[b_1/\varphi] \sqsubseteq b_2\ x$ and $(b_2\ x)\!\downarrow$. By the minimality principle (7.36), it follows that $a_2[b_1/\varphi] \sqsubseteq b_2\ x$. Using this inequation, we derive

$$\begin{array}{lcl}(\lambda x\, s[a_2/\psi])[b_1/\varphi] & \equiv & (\lambda x\, s[a_2[b_1/\varphi]/\psi])[b_1/\varphi] \\ \text{[Context Lemma 7.4]} & \sqsubseteq & (\lambda x\, s[b_2\ x/\psi])[b_1/\varphi] \\ & \equiv & (\lambda x\, s[\chi\ x/\psi])[b_1/\varphi, b_2/\chi] \\ \text{[(FIX)]} & \sqsubseteq & b_1.\end{array}$$

By the minimality principle (7.36), we obtain $a_1 \sqsubseteq b_1$. For the sake of completeness we mention that we now have $a_2[a_1/\varphi] \sqsubseteq a_2[b_1/\varphi] \sqsubseteq b_2\ x$ and $\lambda x\, a_2[a_1/\varphi] \sqsubseteq b_2$. □

Concluding remarks

The main goal of this article was to explain two different operational interpretations of the same programs by different logics. We have started with a class of simply typed programs that contain least fixed-point recursion and ended up with two logics. The first logic, VPT, is adequate for call-by-value evaluation, whereas the second one, NPT, is adequate for call-by-name evaluation. This is shown using methods from denotational semantics, mainly adequacy theorems which relate the (given) operational semantics with the denotational semantics. Neither the programs nor the logics contain the constant 'undefined'.

It is possible to prove the adequacy of VPT and NPT directly without using denotational semantics. The proof, however, is tedious, since it needs term models and a direct proof of the so-called Context Lemma for call-by-value and for call-by-name evaluation.

The systems VPT and NPT are obtained both as extensions of a system which we call the basic logic of partial terms (BPT). The basic logic of partial terms is useful, since its theorems are valid under call-by-value as well as under call-by-name interpretation. It is, however, not clear what the exact semantics of BPT is. It also

https://doi.org/10.1017/S0956796898002974 Published online by Cambridge University Press

not clear how complete BPT is, i.e. whether it proves all theorems that are common to VPT and NPT.

All logics are formulated 'locally' for a fixed structure $\mathfrak{A}$ and so the question is: how complete are these logics when we interpret them over the class of all structures of a given signature? Is there a restricted completeness theorem?

To formulate the results of this article in the same context, the partial functions of the given first-order structure are strict. For the call-by-name interpretation, it would be more natural to allow non-strict functions, too. The call-by-name evaluation rules for such functions, however, are rather awkward, because if we want to apply a built-in function f to a term t then it is not clear, whether the term t has to be evaluated and if so, which components of the result (which is a lazy pair) have to be evaluated before f is called.

The programs studied in this article are pure without side effects. In the presence of side effects the standard notion of definedness splits into a myriad of notions (always evaluating to a value, being operationally equivalent to a value, being a value, etc.) The question is therefore whether this work could be carried out for programs that contain side effects, and if so, whether this helps clarify the different notions of definedness.

If we fix a set of axioms T for the structure $\mathfrak{A}$, then several proof-theoretic questions arise. What is the proof-theoretic strength of BPT(T)? What are the fragments of BPT(T) obtained by restricting the scheme for computational inductions to certain classes of formulas? In the case of natural numbers $\mathbb{N}$ and Peano Arithmetic PA it can be shown that VPT(PA) as well as NPT(PA) have the same strength as PA, since it is possible to interpret them in PA by formalizing suitable term models. This shows that VPT and NPT are first-order logics, although their terms are higher-order.

Note added in proof

I am grateful to Reinhard Kahle for the following observation: The disjunctions

$$(\forall x_1^{\tau_1} x_1\downarrow \wedge \ldots \wedge \forall x_n^{\tau_n} x_n\downarrow) \vee (\exists x_1^{\tau_1} \neg x_1\downarrow \wedge \ldots \wedge \exists x_n^{\tau_n} \neg x_n\downarrow)$$

are in general not provable in BPT although they are provable in VPT as well as in NPT. If we add the disjunctions to BPT, then we can prove all theorems common to VPT and NPT. Assume that VPT $\vdash A$ and NPT $\vdash A$. By the deduction theorem, it follows that there exist types $\tau_1, \ldots, \tau_n$ such that the formulas

$$(\forall x_1^{\tau_1} x_1\downarrow \wedge \ldots \wedge \forall x_n^{\tau_n} x_n\downarrow) \to A \quad \text{and} \quad (\exists x_1^{\tau_1} \neg x_1\downarrow \wedge \ldots \wedge \exists x_n^{\tau_n} \neg x_n\downarrow) \to A$$

are provable in BPT.

References

Beeson, M. J. (1985) *Foundations of Constructive Mathematics.* Springer-Verlag.

Clinger, W. and Rees, J. A. (1991) The revised[4] report on the algorithmic language Scheme. *Acm Lisp Pointers*, **4**(3).

https://doi.org/10.1017/S0956796898002974 Published online by Cambridge University Press

Farmer, W. M. (1990) A partial function version of Church's simple theory of types. *J. Symbolic Logic*, **55**(3), 1269–1291.

Feferman, S. (1975) A language and axioms for explicit mathematics. In: Crossley, J. N. (ed.), *Algebra and logic: Lecture Notes in Mathematics 450*, pp. 87–139. Springer-Verlag.

Feferman, S. (1992a) Logics for termination and correctness of functional programs. In: Moschovakis, Y. N. (ed.), *Logic from Computer Science*, pp. 95–127. New York: Springer-Verlag.

Feferman, S. (1992b) Logics for termination and correctness of functional programs, II. Logics of strength PRA. In: Aczel, P., Simmons, H. and Wainer, S. S. (eds.), *Proof Theory*, pp. 195–225. Cambridge University Press.

Feferman, S. (1995) Definedness. *Erkenntnis*, **43**, 295–320.

Feijs, L. M. G. and Jonkers, H. B. M. (1992) *Formal Specification and Design*. Cambridge Tracts in Theoretical Computer Science, vol. 35. Cambridge University Press.

Gordon, M. J., Milner, A. J. and Wadsworth, C. P. (1979) *Edinburgh LCF*. Lecture Notes in Computer Science, vol. 78. Springer-Verlag.

Gunter, C. A. (1992) *Semantics of Programming Languages: Structures and Techniques*. MIT Press.

Hayashi, S. and Nakano, H. (1988) *PX: A Computational Logic*. MIT Press.

Hudlak, P., Peyton Jones, S. and Wadler, P. (1992) Report on the programming language Haskell, a non-strict purely functional language. *ACM SIGPLAN Notices*, **27**(5).

Jones, C. B. and Middelburg, K. (1994) A typed logic of partial functions reconstructed classically. *Acta Informatica*, **31**(5), 399–430.

Kuper, J. (1994) *Partiality in Logic and Computation – Aspects of Undefinedness*. PhD thesis, University of Twente, Enschede, The Netherlands.

Manna, Z. (1974) *Mathematical Theory of Computation*. McGraw-Hill.

Mason, I. A. and Talcott, C. L. (1992) Inferring the equivalence of functional programs that mutate data. *Theoretical Computer Science*, **105**, 167–215.

Milner, R., Tofte, M. and Harper, R. (1990) *The Definition of Standard ML*. MIT Press.

Moschovakis, Y. N. (1989) The formal language of recursion. *J. Symbolic Logic*, **54**(4), 1216–1252.

Park, D. (1969) Fixpoint induction and proofs of program properties. In: Meltzer, B. and Michie, D. (eds.), *Machine intelligence*, , 59–78. Edinburgh University Press.

Paulson, L. C. (1987) *Logic and Computation: Interactive Proof with Cambridge LCF*. Cambridge University Press.

Platek, R. A. (1966) *Foundations of Recursion Theory*. PhD thesis, Stanford University, Stanford, CA.

Pljuvskevivcus, R. A. (1968) A sequential variant of constructive logic calculi for normal formulas not containing structural rules. In: Orevkov, V. P. (ed.), *The Calculi of Symbolic Logic, 1. Proceedings of the Steklov Institute of Mathematics*, **98**, 175–229. AMS Translations (1971).

Plotkin, G. (1975) Call-by-name, call-by-value and the λ-calculus. *Theoretical Computer Science*, **1**, 125–159.

Riecke, J. G. (1991) *The Logic and Expressibility of Simply-typed Call-by-value and Lazy Languages*. Technical Report MIT/LCS/TR-523. MIT, Cambridge, MA.

Scott, D. S. (1979) Identity and existence in intuitionistic logic. In: Fourman, M. P., Mulvey, C. J. and Scott, D. S. (eds.), *Applications of Sheaves*, pp. 660–696. Springer-Verlag.

Shankar, N. (1989) *Recursive Programming and Proving*. Technical Report, Computer Science Laboratory, SRI International, Menlo Park, CA.

Stärk, R. F. (1997) Call-by-value, call-by-name and the logic of values. In: van Dalen, D. and Bezem, M. (eds.), *Computer Science Logic CSL '96: Selected papers*, pp. 431–445. Springer-Verlag.

Troelstra, A. S. and van Dalen, D. (1988) *Constructivism in Mathematics: An introduction.* Amsterdam.

Turner, D. A. (1986) An overview of Miranda. *ACM SIGPLAN Notices*, **21**(12), 158–166.

Winskel, G. (1993) *The Formal Semantics of Programming Languages: An Introduction.* MIT Press.