On the Observable Properties of Higher Order Functions that Dynamically Create Local Names (preliminary report)

Andrew Pitts[1] Ian Stark[2]

University of Cambridge Computer Laboratory,
Pembroke Street, Cambridge CB2 3QG, England

Tel: +44 223 334629 Fax: +44 223 334678
Email: {Andrew.Pitts,Ian.Stark}@cl.cam.ac.uk

Abstract

The research reported in this paper is concerned with the problem of reasoning about properties of higher order functions involving state. It is motivated by the desire to identify what, if any, are the difficulties created purely by *locality* of state, independent of other properties such as side-effects, exceptional termination and non-termination due to recursion. We consider a simple language (equivalent to a fragment of Standard ML) of typed, higher order functions that can dynamically create fresh *names*. Names are created with local scope, can be tested for equality and can be passed around via function application, but that is all.

We demonstrate that despite the simplicity of the language and its operational semantics, the observable properties of such functions can be very subtle. Two methods are introduced for analyzing Morris-style observational equivalence between expressions in the language. The first method introduces a notion of 'applicative' equivalence incorporating a syntactic version of O'Hearn and Tennent's relationally parametric functors and a version of *representation independence* for local names. This applicative equivalence is properly contained in the relation of observational equivalence, but coincides with it for first order expressions (and is decidable there). The second method develops a general, categorical framework for computationally adequate models of the language, based on Moggi's *monadic* approach to denotational semantics. We give examples of models, one of which is fully abstract for first order expressions. No fully abstract (concrete) model of the whole language is known.

[1]Supported by UK SERC grant GR/G53279 and CEC ESPRIT project CLICS-II

[2]Supported by UK SERC studentship 91307943 and CEC SCIENCE project PL910296

1 Introduction

Programming languages combining higher order features with the manipulation of local state present severe problems for the traditional techniques of programming language semantics and logics of programs. For denotational semantics, the problems manifest themselves as a lack of abstraction in existing semantic models: some expressions that are observationally equivalent (i.e. that can be interchanged in any program without affecting its behaviour when executed) are assigned different denotations in the model. For operational semantics, the problems manifest themselves partly in the fact that simple techniques for analyzing observational equivalence in the case of purely functional languages (such as Milner's 'Context Lemma' [9], or more generally, notions of applicative bisimulation [1]) break down in the presence of state-based features. Furthermore, operationally based approaches to properties of programs are often inconveniently intensional, e.g. the familiar congruence properties of equational logic fail to hold. (See [7, Sect. 5(A)], for example.) These problems have been intensively studied for the case of local variables in block-structured, Algol-like languages and to a lesser extent for the case of languages involving the dynamic creation of mutable locations (such as ML-style *references*). See [20, 2, 8, 3, 21, 15, 16, 7, 4]. Our interest in this subject stems primarily from a desire to improve and deepen the techniques which are available for reasoning about program behaviour in the 'impure' functional language Standard ML [10].

Our motivation here is to try to identify what, if any, are the difficulties created purely by *locality* of state, independent of other properties such as side-effects, exceptional termination and non-termination due to recursion. Accordingly we consider higher order functions which can dynamically create fresh names of things, but ignore completely what kind of thing (references, exceptions, etc.) is being named. Names are created with local scope, can be tested for equality, and are passed around via function application, but that is all. Because of this limited framework, there is some hope of obtaining *definitive* results—fully abstract models and complete proof techniques. As the vehicle for this study we formulate an extension of the call-by-value, simply typed lambda calculus, called the *nu-calculus* and introduced in Sect. 2. In ML terms, it contains higher order functions over ground types `bool` and `unit ref`—the latter being the type of dynamically created references to the unique element of type `unit`. This acts as a type of 'names' because only one thing can be (and is) stored in such a reference, so that its only characteristic is its name. We have purposely excluded recursion from the nu-calculus and as a result any closed expression evaluates to an essentially unique canonical form. Indeed, the nu-calculus appears at first sight to be an extremely simple system. On closer inspection, we find that nu-calculus expressions can exhibit very subtle behaviour with respect to an appropriate notion of observational equivalence. Thus our first contribution is somewhat in the spirit of Meyer and Seiber [8]: we observe that even for this extremely simple case of local state there are observationally equivalent expressions which traditional denotational techniques will fail to identify (Example 2.8).

In Sect. 3 we introduce a notion of 'logical relation' for the nu-calculus incorporating a version of *representation independence* for local names. Our technique is a syntactic version of the relationally parametric semantics of O'Hearn and Tennent [16]. There are also interesting similarities with Plotkin and Abadi's parametricity schema for existential types [19, Theorem 7]. We use our version of logical relations to establish the termination properties of the nu-calculus (Theorem 3.3) and to provide a useful, notion of 'applicative' equivalence between nu-calculus expressions which implies observational equivalence (The-

orem 3.5). In fact the two notions of equivalence coincide for expressions of first order types (Theorem 3.8) and are decidable there, but differ for higher order types (Example 3.7).

The denotational semantics of the nu-calculus is considered in Sect. 4. Following Moggi [12], we make use of categorical *monads* to enforce a distinction between denotations of values (expressions in canonical form) and denotations of computations (arbitrary expressions). This is helpful, since it allows us to identify explicitly and simply what structure is needed in a model to give a static meaning for the key dynamic aspect of the nu-calculus, *the action of computing a new name* (see equations (10)–(12)). Our main result here (Theorem 4.2) is to identify some simple structure on a category equipped with a strong monad sufficient to guarantee that the nu-calculus can be modelled in the category in a *computationally adequate* way. Thus if two nu-calculus expressions have equal denotations in such a category, then necessarily they are observationally equivalent. An instance of this categorical structure can be obtained by adapting Moggi's [11] 'dynamic allocation' monad to O'Hearn and Tennent's category of relationally parametric functors [16]. This model is fully abstract at first order types, but not so at higher types. Other instances of the categorical structure are known and can be used to establish subtle, higher order observational equivalences (such as (3)). Whilst no concrete model is known to be fully abstract for the whole of the nu-calculus, we conjecture that a term-model construction on (a suitable extension of) the syntax of the nu-calculus yields an instance of the categorical structure which is fully abstract.

2 The nu-calculus

Syntactically, the nu-calculus is a kind of simply typed lambda calculus. The types, σ, are built up from a ground type o of *booleans* and a ground type ν of *names*, by forming *function types*, $\sigma \to \sigma'$. Expressions take the form

M	$::=$	x	variable
	$\mid$	n	name
	$\mid$	$\mathit{true} \mid \mathit{false}$	truth values
	$\mid$	$\mathit{if}\ M\ \mathit{then}\ M\ \mathit{else}\ M$	conditional
	$\mid$	$M = M$	equality of names
	$\mid$	$\nu n\ .\ M$	local name declaration
	$\mid$	$\lambda x : \sigma\ .\ M$	function abstraction
	$\mid$	MM	function application

where $x \in \mathit{Var}$, an infinite set whose elements are called *variables*, and $n \in \mathit{Nme}$, an infinite set (disjoint from Var) whose elements are called *names*. Function abstraction is a variable-binding construct (occurrences of x in M are bound in $\lambda x : \sigma\ .\ M$), whereas local name declaration is a name-binding construct (occurrences of n in M are bound in $\nu n\ .\ M$). We write $\mathit{Var}(M)$ and $\mathit{Nme}(M)$ for the finite subsets of Var and Nme consisting of the free variables and the free names in an expression M. Henceforward, we implicitly identify expressions that differ up to α-conversion of bound variables and bound names. We denote by $M[M'/x]$ (respectively $M[M'/n]$) the result of substituting an expression M' for all free occurrences of x (respectively n) in M.

Expressions will be assigned types via typing assertions of the form

$$s, \Gamma \vdash M : \sigma$$

$$\frac{}{s,\Gamma \vdash x : \Gamma(x)}\ (x \in dom(\Gamma)) \qquad \frac{}{s,\Gamma \vdash n : \nu}\ (n \in s) \qquad \frac{}{s,\Gamma \vdash b : o}\ (b = true, false)$$

$$\frac{s,\Gamma \vdash B : o \quad s,\Gamma \vdash M : \sigma \quad s,\Gamma \vdash M' : \sigma}{s,\Gamma \vdash \mathit{if}\ B\ \mathit{then}\ M\ \mathit{else}\ M' : \sigma} \qquad \frac{s,\Gamma \vdash N : \nu \quad s,\Gamma \vdash N' : \nu}{s,\Gamma \vdash (N = N') : o}$$

$$\frac{s \oplus \{n\},\Gamma \vdash M : \sigma}{s,\Gamma \vdash \nu n\ .\ M : \sigma} \qquad \frac{s,\Gamma \oplus [x : \sigma] \vdash M : \sigma'}{s,\Gamma \vdash \lambda x : \sigma\ .\ M : \sigma \to \sigma'} \qquad \frac{s,\Gamma \vdash F : \sigma \to \sigma' \quad s,\Gamma \vdash M : \sigma}{s,\Gamma \vdash FM : \sigma'}$$

Table 1: Rules for assigning types in the nu-calculus

where s is a finite subset of *Nme*, Γ is a finite function from variables to types, σ is a type, and M is an expression satisfying $Nme(M) \subseteq s$ and $Var(M) \subseteq dom(\Gamma)$ (the domain of definition of Γ). The rules generating the valid typing assertions are given in Table 1. In these rules $s \oplus \{n\}$ indicates the finite set of names obtained from s by adjoining $n \notin s$; and $\Gamma \oplus [x : \sigma]$ denotes the finite function obtained by extending Γ by mapping $x \notin dom(\Gamma)$ to σ. Clearly, if $s,\Gamma \vdash M : \sigma$ holds, then σ is uniquely determined by s, Γ and M. We write

$$\mathrm{Exp}_\sigma(s) \quad \stackrel{def}{=} \quad \{M \mid s,\emptyset \vdash M : \sigma\}$$

for the set of *closed nu-calculus expression of type σ with free names in the set s*. The subset

$$\mathrm{Can}_\sigma(s) \quad \subseteq \quad \mathrm{Exp}_\sigma(s)$$

of *canonical* nu-calculus expressions of type σ with free names in the set s consists of those closed expressions which are either names (in s), or the booleans constants *true* and *false*, or function abstractions.

We give the operational semantics of the nu-calculus in terms of an inductively defined evaluation relation which matches the computational behaviour of equivalent ML expressions. The ML equivalent of the expression $\nu n\ .\ M$ is

`let n=ref() in` M `end`

(using the ML type `unit ref` for the type of names). In other words the effect of evaluating $\nu n\ .\ M$ should be to create a fresh name n and then use it in evaluating M. Whereas in the definition of ML [10] environments are used to bind identifiers (variables) to addresses (names), here we have chosen to simplify the form of the evaluation relation by using 'extended' expressions containing names explicitly. It would be possible to simplify the syntax of the nu-calculus even further by identifying the syntactic category of names with that of variables of type ν. We choose not to do so because names and variables have different semantic properties. For example, the operational semantics we give commutes with arbitrary substitutions on variables, but only with restricted forms of substitutions on names (viz. essentially just permutations of names).

An appropriate notion of state for this simple language is just a finite subset of *Nme*, indicating the names which have been created so far. So we will use an evaluation relation

(CAN) $$\frac{}{s \vdash C \Downarrow_\sigma C}$$

(COND1) $$\frac{s \vdash B \Downarrow_o (s_1) true \quad s \oplus s_1 \vdash M \Downarrow_\sigma (s_2) C}{s \vdash \mathit{if}\ B\ \mathit{then}\ M\ \mathit{else}\ M' \Downarrow_\sigma (s_1 \oplus s_2) C}$$

(COND2) $$\frac{s \vdash B \Downarrow_o (s_1) false \quad s \oplus s_1 \vdash M' \Downarrow_\sigma (s_2) C'}{s \vdash \mathit{if}\ B\ \mathit{then}\ M\ \mathit{else}\ M' \Downarrow_\sigma (s_1 \oplus s_2) C'}$$

(EQ) $$\frac{s \vdash N \Downarrow_\nu (s_1) n \quad s \oplus s_1 \vdash N' \Downarrow_\nu (s_2) n'}{s \vdash (N = N') \Downarrow_o (s_1 \oplus s_2) \delta_{nn'}}$$

(LOCAL) $$\frac{s \oplus \{n\} \vdash M \Downarrow_\sigma (s_1) C}{s \vdash \nu n\ .\ M \Downarrow_\sigma (\{n\} \oplus s_1) C}$$

(APP) $$\frac{\begin{array}{c} s \vdash F \Downarrow_{\sigma \to \sigma'} (s_1) \lambda x : \sigma\ .\ M' \qquad s \oplus s_1 \vdash M \Downarrow_\sigma (s_2) C \\ s \oplus s_1 \oplus s_2 \vdash M'[C/x] \Downarrow_{\sigma'} (s_3) C' \end{array}}{s \vdash FM \Downarrow_{\sigma'} (s_1 \oplus s_2 \oplus s_3) C'}$$

Table 2: Rules for evaluating nu-calculus expressions

of the form

$$s \vdash M \Downarrow_\sigma (s')C \tag{1}$$

where s and s' are *disjoint* finite sets of names, $M \in \mathrm{Exp}_\sigma(s)$ and $C \in \mathrm{Can}_\sigma(s \oplus s')$. The intended meaning of (1) is: 'in state s, expression M evaluates to canonical form C, creating fresh, local names s' in the process'. The rules for generating the relation are given in Table 2. In rule (EQ) we use the notation $\delta_{nn'}$, where

$$\delta_{nn'} \stackrel{def}{=} \begin{cases} \mathit{true} & \text{if } n = n' \\ \mathit{false} & \text{if } n \neq n' \end{cases}$$

It is important to note that the rules in Table 2 refer to the collection of judgements as in (1) that are well-formed, i.e. satisfy the conditions mentioned above. For example, in rule (LOCAL) the well-formedness of the hypothesis and the conclusion entail that n is not an element of either s or s_1.

The rules follow the state convention of Standard ML [10], i.e. order of evaluation is from left to right, with state accumulating sequentially. In fact, because we are dealing with state that can be created but cannot be mutated, some of this sequentiality is spurious. For example, in rule (COND1) the second hypothesis can be strengthened by removing s_1, without affecting the collection of valid instances of evaluation; similar strengthenings can be made to the second hypotheses of (COND2), (EQ) and (APP).

It is easy to see that evaluation is deterministic up to renaming created names, in the following sense:

Lemma 2.1 *If $s \vdash M \Downarrow_\sigma (s_1)C$ and $s \vdash M \Downarrow_\sigma (s_2)C'$, then there is a bijection $R : s_1 \leftrightarrow s_2$ so that C' is α-convertible with the expression $C[n'/n \mid (n, n') \in R]$.*

The initial state s in the evaluation (1) has the structural properties of an *affine linear logic context*, in the sense that derived rules of weakening and exchange are valid, but a rule of contraction is not. For example when M is $n = n'$ and C is *false*, then $\{n, n'\} \vdash M \Downarrow_o (\emptyset)C$ is valid, but $\{n\} \vdash M[n/n'] \Downarrow_o (\emptyset)C[n/n']$ is not. (Compare the use made of affine linear logic by O'Hearn in [14].)

The evaluation relation (1) can be used to define a Morris-style contextual equivalence between nu-calculus expressions: two expressions are equivalent if they can be interchanged in any program without affecting the observable result of evaluating it. Here we will take a 'program' to be a closed expression of type o, and the possible observable results of evaluating a program to be the booleans *true* and *false*, disregarding any local names that are created in the process of evaluation. (It would not change the notion of observational equivalence given below if we also allowed programs to be of type ν and observable results to include pre-existing names.) In the following definition, as usual the 'context' $B[-]$ is an expression in which some subexpressions have been replaced by a place-holder, $-$; and then $B[M]$ denotes the result of filling the place-holder with an expression M.

Definition 2.2 (Observational equivalence) Given $M_1, M_2 \in \mathrm{Exp}_\sigma(s)$, we write

$$s \vdash M_1 \approx_\sigma M_2$$

to mean that for all $B[-]$ and all $b \in \{\mathit{true}, \mathit{false}\}$,

$$\exists s_1(s \vdash B[M_1] \Downarrow_o (s_1)b) \quad \Leftrightarrow \quad \exists s_2(s \vdash B[M_2] \Downarrow_o (s_2)b) \quad .$$

In this case we say that M_1 and M_2 are *observationally equivalent*.

The following result shows that one need only consider contexts that immediately evaluate their arguments in order to establish observational equivalence. It is the analogue of Theorem (**ciu**) in [4].

Lemma 2.3 *$s \vdash M_1 \approx_\sigma M_2$ if and only if for all $b \in \{true, false\}$ and all $\lambda x : \sigma \,.\, B \in \mathrm{Can}_{\sigma \to o}(s)$*

$$\exists s_1 (s \vdash (\lambda x : \sigma \,.\, B) M_1 \Downarrow_o (s_1) b) \quad \Leftrightarrow \quad \exists s_2 (s \vdash (\lambda x : \sigma \,.\, B) M_2 \Downarrow_o (s_2) b) \quad .$$

The following instances of observational equivalence are easily established using the lemma.

Corollary 2.4 *(i) If $M \in \mathrm{Exp}_\sigma(s)$ and $n \notin s$, then $s \vdash \nu n \,.\, M \approx_\sigma M$.*

(ii) If $M \in \mathrm{Exp}_\sigma(s \oplus \{n\} \oplus \{n'\})$, then $s \vdash \nu n \,.\, \nu n' \,.\, M \approx_\sigma \nu n' \,.\, \nu n \,.\, M$.

(iii) If $s \vdash M \Downarrow_\sigma (s') C$, then $s \vdash M \approx_\sigma \nu s' \,.\, C$. Here $\nu s' \,.\, C$ stands for $\nu n_1 \ldots \nu n_k \,.\, C$ if $s' = \{n_1, \ldots, n_k\}$ for some $k > 0$, and stands for C if $s' = \emptyset$. (By part (ii), up to observational equivalence, it does not matter which order we enumerate the elements of s' in $\nu s' \,.\, C$.)

(iv) If $s, [x : \sigma] \vdash M : \sigma'$ and $C \in \mathrm{Can}_\sigma(s)$, then $s \vdash (\lambda x : \sigma \,.\, M) C \approx_{\sigma'} M[C/x]$.

In the next section we will show that evaluation of nu-calculus expressions always terminates (Theorem 3.3). It follows from this and the above Corollary that, up to observational equivalence, the only closed expressions of type o are *true* and *false* and the only closed expression of type ν not involving any free names is

$$new \stackrel{def}{=} \nu n \,.\, n \quad .$$

However, at higher types things become rapidly more complicated. The following example gives infinitely many expressions of type $\nu \to \nu$ which are mutually observationally inequivalent.

Example 2.5 For each $p \geq 1$, consider the nu-calculus expression of type $\nu \to \nu$ which first creates $p+1$ local names $n_0, \ldots, n_p$ and then acts as the function cyclically permuting these names and mapping any other name to n_0:

$$\begin{aligned} F_p \stackrel{def}{=} \nu n_0 \ldots \nu n_p \,.\, \lambda x : \nu \,.\, & \mathit{if}\ x = n_0\ \mathit{then}\ n_1\ \mathit{else} \\ & \mathit{if}\ x = n_1\ \mathit{then}\ n_2\ \mathit{else} \\ & \quad \ldots \\ & \quad \mathit{if}\ x = n_p\ \mathit{then}\ n_0\ \mathit{else}\ n_0 \quad . \end{aligned}$$

Then $\emptyset \vdash F_p \not\approx_{\nu \to \nu} F_{p'}$ whenever $p \neq p'$, because

$$B_q \stackrel{def}{=} \lambda f : \nu \to \nu \,.\, \nu n \,.\, (f^{(q+2)}(n) = f(n))$$

has the property that for all $q \in \{1, \ldots, p\}$, $\emptyset \vdash B_q F_p \Downarrow_o (\{n_0, \ldots, n_p, n\}) true$ if and only if $q = p$. (In B_q, $f^{(q+2)}$ indicates f iterated $q+2$ times.)

Example 2.6 Here is a simple example to illustrate the fact that local name declaration and function abstraction in general do not commute up to observational equivalence. The expressions

$$M \stackrel{def}{=} \nu n \,.\, \lambda x : \nu \,.\, n \quad \text{and} \quad N \stackrel{def}{=} \lambda x : \nu \,.\, \nu n \,.\, n$$

are not observationally equivalent, because $B \stackrel{def}{=} \lambda f : \nu \to \nu \,.\, (f new = f new)$ has the property that $\emptyset \vdash BM \Downarrow_o (\{n, n_1, n_2\})true$ whereas $\emptyset \vdash BN \Downarrow_o (\{n, n_1, n_2\})false$.

Example 2.7 The rule (APP) in Table 2 embodies a form of strict, or 'call-by-value', application. Part (iv) of Corollary 2.4 shows that the appropriate restricted form of beta-conversion (Plotkin's β_v [18]) holds up to observational equivalence. Although there is no non-termination in our simple language, the general form of beta-conversion fails for the nu-calculus, because of the dynamics of name creation. For example, the beta redex $(\lambda x : \nu \,.\, x = x)new$ is not observationally equivalent to the corresponding reduct $new = new$ since

$$\begin{array}{lll} \emptyset & \vdash & (\lambda x : \nu \,.\, x = x)new \Downarrow_o (\{n_1\})true \\ \emptyset & \vdash & (new = new) \Downarrow_o (\{n_1, n_2\})false \quad . \end{array}$$

For the simple functional language PCF, Milner's context lemma [9] shows that observational equivalence may be established by testing just with *applicative contexts*—those of the form $[-]C_1C_2 \ldots C_k$. Not surprisingly, this fails in the nu-calculus. For example, the expressions F_p are in fact indistinguishable by such applicative contexts, even though they can be distinguished by more complicated contexts (like $B_q([-])$) which carry out 'anonymous' manipulation of the private names $n_0, \ldots, n_p$. It would seem that the properties of higher order functions which create and pass around private names can be quite subtle. Two contrasting examples of observational equivalence, more subtle than those in Corollary 2.4, are given below. The first one illustrates the fact that local names are always distinct from externally supplied names; the second illustrates the fact that any two local names are indiscernible by externally supplied boolean tests. (This second equivalence is quite delicate—it certainly would not hold in languages where evaluation of functions can have side-effects on mutable state.) Operational and denotational methods for proving such observational equivalences of nu-calculus expressions will be developed in the rest of this paper.

Example 2.8

$$\emptyset \;\vdash\; \nu n \,.\, \lambda x : \nu \,.\, (x = n) \;\approx_{\nu \to o}\; \lambda x : \nu \,.\, false \tag{2}$$

$$\emptyset \;\vdash\; \nu n \,.\, \nu n' \,.\, \lambda f : \nu \to o \,.\, (fn = fn') \;\approx_{(\nu \to o) \to o}\; \lambda f : \nu \to o \,.\, true \quad . \tag{3}$$

In (3), the boolean equality test $fn = fn'$ is an abbreviation for

$$if\ fn\ then\ (if\ fn'\ then\ true\ else\ false)\ else\ (if\ fn'\ then\ false\ else\ true) \quad .$$

3 Representation independence for local names

This section develops a notion of (binary) logical relation for the nu-calculus and shows how to use it to establish instances of observational equivalence between nu-calculus expressions.

Given finite subsets $s_1, s_2 \subseteq Nme$ of names, we write $R : s_1 \rightleftharpoons s_2$ to indicate that R is (the graph of) a *partial bijection* from s_1 to s_2. In other words, $R \subseteq s_1 \times s_2$ satisfies

$$m_1\ R\ m_2 \wedge n_1\ R\ n_2 \quad \Rightarrow \quad (m_1 = n_1 \Leftrightarrow m_2 = n_2) \quad . \tag{4}$$

(We use infix notation for binary relations.) Writing $s \oplus s'$ for the union of *disjoint* sets, note that $R \oplus R' : s_1 \oplus s_1' \rightleftharpoons s_2 \oplus s_2'$ when $R : s_1 \rightleftharpoons s_2$ and $R' : s_1' \rightleftharpoons s_2'$. The *identity* (partial) bijection, $I_s : s \rightleftharpoons s$, is given by:

$$n_1\ I_s\ n_2 \quad \Leftrightarrow \quad n_1 = n_2 \quad . \tag{5}$$

Definition 3.1 For each type σ we define a family of binary relations between canonical expressions

$$(R_\sigma \subseteq \mathrm{Can}_\sigma(s_1) \times \mathrm{Can}_\sigma(s_2) \mid R : s_1 \rightleftharpoons s_2)$$

by induction on the structure of σ as in (7), (8) and (9) below; clause (9) makes use of associated relations between expressions, $\overline{R}_\sigma \subseteq \mathrm{Exp}_\sigma(s_1) \times \mathrm{Exp}_\sigma(s_2)$ defined by (6).

$$\begin{aligned} M_1\ \overline{R}_\sigma\ M_2 \quad \Leftrightarrow \quad & \exists R' : s_1' \rightleftharpoons s_2', C_1 \in \mathrm{Can}_\sigma(s_1 \oplus s_1'), C_2 \in \mathrm{Can}_\sigma(s_2 \oplus s_2') \ . \\ & s_1 \vdash M_1 \Downarrow_\sigma (s_1')C_1 \wedge s_2 \vdash M_2 \Downarrow_\sigma (s_2')C_2 \wedge C_1\ (R \oplus R')_\sigma\ C_2 \end{aligned} \tag{6}$$

$$b_1\ R_o\ b_2 \quad \Leftrightarrow \quad b_1 = b_2 \tag{7}$$

$$n_1\ R_\nu\ n_2 \quad \Leftrightarrow \quad n_1\ R\ n_2 \tag{8}$$

$$\begin{aligned} & \lambda x : \sigma\ .\ M_1\ R_{\sigma \to \sigma'}\ \lambda x : \sigma\ .\ M_2 \Leftrightarrow \\ & \qquad \forall R' : s_1' \rightleftharpoons s_2', C_1 \in \mathrm{Can}_\sigma(s_1 \oplus s_1'), C_2 \in \mathrm{Can}_\sigma(s_2 \oplus s_2') \ . \\ & \qquad\quad C_1\ (R \oplus R')_\sigma\ C_2 \Rightarrow M_1[C_1/x]\ (\overline{R \oplus R'})_{\sigma'}\ M_2[C_2/x] \end{aligned} \tag{9}$$

(It is implicit in (6) and (9) that each s_i' is required to be disjoint from s_i.)

Clause (9) of the definition is a syntactic version of O'Hearn and Tennent's approach to relational parametricity in [16]. The main interest in the definition lies in clause (6) where the relation $\overline{R}_\sigma$ on expressions is defined in terms of the relation R_σ on canonical expressions. This clause embodies a form of 'representation independence' for the dynamically created local names. (Cf. Plotkin and Abadi's parametricity schema for existential types [19, Theorem 7].)

The family $(\overline{R}_\sigma \mid \sigma)$ is a form of binary 'logical relation' for nu-calculus expressions. Since we choose in (7) to take the logical relation to be the identity at the ground type o, the whole family is determined by what we take at the other ground type ν. We wish related expressions to be mapped to related expressions by any nu-calculus function, and we have to impose the restriction (4) on the relation R to ensure this property holds for the function testing equality of names. The following proposition expresses this fundamental property of our notion of logical relation. It is proved by induction on the derivation of typing assertions.

Proposition 3.2 *Suppose*

$$s, [x_1 : \sigma_1, \ldots, x_k : \sigma_k] \vdash M : \sigma \quad .$$

Then for all $R : s_1 \rightleftharpoons s_2$ *with* s_1 *and* s_2 *disjoint from* s*, and for all* $C_i \in \mathrm{Can}_{\sigma_i}(s \oplus s_1)$ *and* $C_i' \in \mathrm{Can}_{\sigma_i}(s \oplus s_2)$ *(*$i = 1, \ldots, k$*) one has*

$$\left(\bigwedge_{i=1}^{k} C_i \; (I_s \oplus R)_{\sigma_i} \; C_i' \right) \Rightarrow$$
$$M[C_1/x_1, \ldots, C_k/x_k] \; (\overline{I_s \oplus R})_\sigma \; M[C_1'/x_1, \ldots, C_k'/x_k]$$

where I_s *is the identity partial bijection, defined in (5).*

Theorem 3.3 (Termination) *For all closed expressions* M*, of type* σ *and with free names in the set* s *say, there is some set of names* s' *(disjoint from* s*) and some canonical expression* $C \in \mathrm{Can}_\sigma(s \oplus s')$ *such that* $s \vdash M \Downarrow_\sigma (s')C$*.*

Proof The $k = 0$ case of Proposition 3.2 implies that $M \; (\overline{I_s})_\sigma \; M$ for all $M \in \mathrm{Exp}_\sigma(s)$. Termination follows from this, given the definition of $\overline{R}_\sigma$ in (6). □

We now show how the fundamental property of our notion of logical relation embodied in Proposition 3.2 can be used to establish observational equivalences.

Definition 3.4 (Applicative equivalence) We say that two expressions $M_1, M_2 \in \mathrm{Exp}_\sigma(s)$ are *applicatively equivalent* if $M_1 \; (\overline{I_s})_\sigma \; M_2$.

Theorem 3.5 *Applicative equivalence implies observational equivalence.*

Proof Suppose $M_1 \; (\overline{I_s})_\sigma \; M_2$. We employ Lemma 2.3 to see that M_1 and M_2 are observationally equivalent. By (6) there is some $R : s_1 \rightleftharpoons s_2$, and C_1, C_2 with $s \vdash M_i \Downarrow_\sigma (s_i)C_i$ $(i = 1, 2)$ and $C_1 \; (I_s \oplus R)_\sigma \; C_2$. Then for any $\lambda x : \sigma \; . \; B \in \mathrm{Can}_{\sigma \to o}(s)$, applying Proposition 3.2 we get $B[C_1/x] \; (\overline{I_s \oplus R})_o \; B[C_2/x]$. Hence by (6) again, there is some $R' : s_1' \rightleftharpoons s_2'$ and b_1, b_2 with $s \oplus s_i \vdash B[C_i/x] \Downarrow_o (s_i')b_i$ $(i = 1, 2)$ and $b_1 \; (I_s \oplus R \oplus R')_o \; b_2$, i.e. with $b_1 = b_2$ (by (7)). Applying the rules in Table 2, we deduce that $s \vdash (\lambda x : \sigma \; . \; B)M_i \Downarrow_o (s_i \oplus s_i')b_i$ with $b_1 = b_2$. Thus Lemma 2.3 and the deterministic nature of the evaluation relation (Lemma 2.1) imply that $M_1 \approx_\sigma M_2$. □

Example 3.6 Theorem 3.5 provides quite a powerful method for establishing some observational equivalences, since $(\overline{I_s})_\sigma$ is much easier to deal with than $\approx_\sigma$. For example, the observational equivalence (2) can be established by this method. For with

$$C_1 \stackrel{def}{=} \lambda x : \nu \; . \; (x = n) \qquad \text{and} \qquad C_2 \stackrel{def}{=} \lambda x : \nu \; . \; \mathit{false}$$

it is not hard to see that $C_1 \; (I_\emptyset \oplus R)_{\nu \to o} \; C_2$ where $R : \{n\} \rightleftharpoons \emptyset$ is necessarily the empty partial bijection; hence $\nu n \; . \; C_1 \; (\overline{I_\emptyset})_{\nu \to o} \; C_2$, as required.

However, not every observational equivalence can be established via Theorem 3.5, as the following example shows. Thus *applicative equivalence is in general a strictly weaker relation than observational equivalence.*

Example 3.7 The pair of second order expressions in (3) are observationally equivalent (this can be established via the denotational methods sketched in Sect. 4), but they are not related by $(\overline{I_\emptyset})_{(\nu\to o)\to o}$. For the only possible partial bijection $R : \{n, n'\} \rightharpoonup \emptyset$ is $R = \emptyset$; but $\lambda f : \nu \to o \,.\, (fn = fn')$ and $\lambda f : \nu \to o \,.\, \mathit{true}$ are not related by $(I_\emptyset \oplus R)_{(\nu\to o)\to o}$, because for the canonical expressions C_1 and C_2 defined in Example 3.6, $C_1\ (I_\emptyset \oplus R)_{\nu\to o}\ C_2$, whereas it is not the case that $(fn = fn')[C_1/f]\ (\overline{I_\emptyset \oplus R})_o\ \mathit{true}[C_2/f]$.

Nevertheless, the converse of Theorem 3.5 does hold when σ is a *first order type*, i.e. of the form $\sigma_k \to \sigma_{k-1} \to \cdots \to \sigma_0$ with each σ_i either ν or o.

Theorem 3.8 *Observational equivalence coincides with applicative equivalence for expressions of first order types.*

Proof We have to show that $s \vdash M_1 \approx_\sigma M_2$ implies $M_1\ (\overline{I_s})_\sigma\ M_2$ for first order σ. Here we merely indicate the key idea of the proof. By Theorem 3.3, $s \vdash M_i \Downarrow_\sigma (s_i)C_i$ for some $C_i \in \mathrm{Can}_\sigma(s \oplus s_i)$ $(i = 1, 2)$. Define $R \subseteq s_1 \times s_2$ to consist of those pairs of names (n_1, n_2) for which there is some $s, x : \sigma \vdash N : \nu$ with $s \oplus s_i \vdash N[C_i/x] \Downarrow_\nu (s_i')n_i$ for each $i = 1, 2$. The assumption that $s \vdash M_1 \approx_\sigma M_2$ implies that R is a partial bijection. For this R (and using the fact that σ is first order) it is possible to show that $C_1\ (I_s \oplus R)_\sigma\ C_2$. Thus $M_1\ (\overline{I_s})_\sigma\ M_2$. □

Given that the observable behaviour of nu-calculus expressions of such first order types can be complicated (see Example 2.5), the theorem is non-trivial. As a corollary of the theorem we obtain the following result.

Corollary 3.9 *The relation of observational equivalence between nu-calculus expressions of first order type is decidable.*

Proof In view of the theorem, it suffices to see that the relations $\overline{R}_\sigma$ are decidable for first order σ. For this, it is sufficient to establish the decidability of the relations R_σ (for first order σ) since Theorem 3.3 ensures that we can calculate s_1' and s_2' in clause (6), and then there are only finitely R' for which a decidable property has to be checked. The decidability of R_σ can be established by induction on the structure of the first order type σ, the base cases being trivial, and the induction step following from the fact that clause (9) can be simplified as follows when $\sigma \in \{o, \nu\}$:

$$\begin{array}{rcl} C_1\ R_{o\to\sigma'}\ C_2 & \Leftrightarrow & \forall b \in \{\mathit{true}, \mathit{false}\} \,.\, C_1 b\ \overline{R}_{\sigma'}\ C_2 b \\ C_1\ R_{\nu\to\sigma'}\ C_2 & \Leftrightarrow & \forall (n_1, n_2) \in R \,.\, C_1 n_1\ \overline{R}_{\sigma'}\ C_2 n_2 \\ & & \quad \wedge \quad C_1 n\ (\overline{R \oplus I_{\{n\}}})_{\sigma'}\ C_2 n \end{array}$$

where in the last clause n is some name not in $s_1 \cup s_2$. □

4 Denotational semantics

In this section we sketch our approach to the denotational semantics of the nu-calculus and summarize the main results. We make use of Moggi's *monadic* approach to denotational semantics [12]. The nu-calculus will be modelled in cartesian closed categories $\mathcal{C}$ equipped with, amongst other things, a strong monad T. Our notation for this categorical structure and its associated internal language (the 'computational lambda calculus') will be as in

[17, Sect. 2]. In particular, nu-calculus function types and associated expressions are interpreted via Moggi's call-by-value translation of simply typed lambda calculus into computational lambda calculus: see [17, Table 5]. To model the type of booleans we assume the coproduct $1+1$ of the terminal object 1 exists in $\mathcal{C}$. For adequacy of the model (Theorem 4.2) we require $\mathcal{C}$ to have all finite limits and for the coproduct $1+1$ to be *stable* and *disjoint* (and hence also for $\mathcal{C}$ to have a *strict* initial object 0); these are standard concepts from categorical logic—see [13, Chap. 1, Sect. 4]. To model the type of names we assume $\mathcal{C}$ contains a *decidable* object N. Decidability means that there is morphism $eq : N \times N \longrightarrow 1+1$ classifying the diagonal subobject of N, i.e. $\langle id, id\rangle : N \rightarrowtail N \times N$ is the pullback along eq of the left coproduct insertion $true : 1 \longrightarrow 1+1$. The morphism eq is used to interpret the equality test on names (as in the definition of $[\![op(\mathtt{e}_1, \mathtt{e}_2)]\!]$ in [17, Table 5]). It remains to explain how local name declaration expressions $\nu n\,.\,M$ are modelled.

In general, the use of categorical monads permits abstraction away from a detailed representation of state in denotational descriptions of languages with imperative features (see Wadler [22]). For the nu-calculus, the relevant notion of state is hardly very complicated. Nevertheless we find that the type-theoretic distinction between denotations of values (expressions in canonical form) and denotations of computations (arbitrary expressions) enforced by a monad is helpful, since it allows us to identify explicitly and simply what structure is needed in $\mathcal{C}$ to give a static meaning for the key dynamic aspect of the nu-calculus, *the action of computing a new name*. We do this by requiring $T(N)$ to possess a global element $new : 1 \longrightarrow T(N)$ such that for any morphisms $e : X \times (1+1) \times N \times N \longrightarrow T(X')$, $f : X \times N \times N \longrightarrow T(X')$ and $g : X \longrightarrow T(X')$ the following equations in the internal language of $\mathcal{C}$ are satisfied:

$$[x : X, n : N] \;\vdash\; \mathit{let}\ n' \Leftarrow \mathit{new}\ \mathit{in}\ e(x, eq(n, n'), n, n') = \mathit{let}\ n' \Leftarrow \mathit{new}\ \mathit{in}\ e(x, \mathit{false}, n, n') \tag{10}$$

$$[x : X, n : N, n' : N] \;\vdash\; \mathit{let}\ n \Leftarrow \mathit{new}\ \mathit{in}\ (\mathit{let}\ n' \Leftarrow \mathit{new}\ \mathit{in}\ f(x, n, n')) = \mathit{let}\ n' \Leftarrow \mathit{new}\ \mathit{in}\ (\mathit{let}\ n \Leftarrow \mathit{new}\ \mathit{in}\ f(x, n, n')) \tag{11}$$

$$[x : X] \;\vdash\; \mathit{let}\ n \Leftarrow \mathit{new}\ \mathit{in}\ g(x) = g(x) \tag{12}$$

where in (10) $\mathit{false} : 1 \longrightarrow 1+1$ is the right coproduct insertion. These equations could be expressed equivalently, but less comprehensibly, via commutative diagrams asserting the equality of various morphisms in $\mathcal{C}$. Equation (10) expresses statically the fundamental requirement that evaluating *new* produces something new. Equations (11) and (12) correspond respectively to properties (ii) and (i) in Corollary 2.4. (They are automatically satisfied if the monad is respectively *commutative* and *affine*—see [5].)

Given the above structure in the category $\mathcal{C}$, for each nu-calculus type σ one gets an object $[\![\sigma]\!]$ of $\mathcal{C}$ by defining:

$$[\![o]\!] \stackrel{def}{=} 1+1 \qquad [\![\nu]\!] \stackrel{def}{=} N \qquad [\![\sigma \to \sigma']\!] \stackrel{def}{=} [\![\sigma]\!] \to T([\![\sigma']\!]) \quad .$$

And for each valid typing assertion $s, \Gamma \vdash M : \sigma$ one can define, by induction on the structure of M, a morphism in $\mathcal{C}$ of the form

$$[\![M]\!] : [\![s]\!] \times [\![\Gamma]\!] \longrightarrow T([\![\sigma]\!])$$

where $[\![s]\!]$ and $[\![\Gamma]\!]$ are the finite products

$$[\![s]\!] \stackrel{def}{=} \prod_{n \in s} N \qquad [\![\Gamma]\!] \stackrel{def}{=} \prod_{x \in dom(\Gamma)} [\![\Gamma(x)]\!] \quad .$$

In particular, each $M \in \mathrm{Exp}_\sigma(s)$ gives rise to a morphism $[\![M]\!] : N^{|s|} \longrightarrow T([\![\sigma]\!])$ (where $|s|$ denotes the number of elements of the finite set s). When M is a canonical expression, this morphism factors through the unit of the monad at $[\![\sigma]\!]$, $\eta : [\![\sigma]\!] \longrightarrow T([\![\sigma]\!])$.

Proposition 4.1 (Soundness) *Using finite limits in $\mathcal{C}$, for each s form the subobject $\neq(s) \rightarrowtail N^{|s|}$ in $\mathcal{C}$ corresponding to the conjunction*

$$[x_1 : N, \ldots, x_{|s|} : N] \vdash \bigwedge_{1 \leq i \neq j \leq |s|} (eq(x_i, x_j) = \mathit{false}) \quad .$$

Then for each valid evaluation $s \vdash M \Downarrow_\sigma (s')C$, the morphisms

$$[\![M]\!] \restriction_{\neq(s)} \quad \stackrel{def}{=} \quad \left(\neq(s) \rightarrowtail N^{|s|} \stackrel{[\![M]\!]}{\longrightarrow} T([\![\sigma]\!]) \right)$$
$$[\![\nu s' \,.\, C]\!] \restriction_{\neq(s)} \quad \stackrel{def}{=} \quad \left(\neq(s) \rightarrowtail N^{|s|} \stackrel{[\![\nu s'.C]\!]}{\longrightarrow} T([\![\sigma]\!]) \right)$$

are equal (i.e. $[\![M]\!]$ and $[\![\nu s' \,.\, C]\!]$ are equal when applied to distinct *$|s|$-tuples). In particular, when $s = \emptyset$ (so that M contains no free names), $\emptyset \vdash M \Downarrow_\sigma (s')C$ implies $[\![M]\!] = [\![\nu s' \,.\, C]\!]$.*

This Proposition together with the termination property Theorem 3.3 and the compositional nature of the denotational semantics yield:

Theorem 4.2 (Adequacy) *Let the category $\mathcal{C}$ be a model of the nu-calculus as described above which is* non-degenerate*, in the sense that $0 \ncong 1$ and $\eta : (1 + 1) \longrightarrow T(1 + 1)$ is a monomorphism. Then for all $M_1, M_2 \in \mathrm{Exp}_\sigma(s)$, $[\![M_1]\!] \restriction_{\neq(s)} = [\![M_2]\!] \restriction_{\neq(s)}$ implies that $s \vdash M_1 \approx_\sigma M_2$.*

Thus instances of this kind of categorical structure can be used to establish the validity of observational equivalences via denotational equalities.

Examples 4.3 We list, without giving details, various examples of models satisfying the requirements of Theorem 4.2.

(i) $\mathcal{C}$ is the category of pullback preserving functors from the category $\mathbb{I}$ of finite ordinals and injective functions to the category $\mathcal{S}et$ of all sets and functions. The monad T is one of Moggi's 'dynamic allocation' monads [11]: the value of T at a functor X is the functor $T(X)$ sending the finite ordinal n to the quotient set

$$T(X)(n) \quad \stackrel{def}{=} \quad \{(m, x) \mid m \in \mathbb{N} \wedge x \in X(n + m)\}/\!\sim$$

where $(m_1, x_1) \sim (m_2, x_2)$ if and only if there are injective functions $f_i : m_i \rightarrowtail m$ $(i = 1, 2)$ with $X(id_n + f_1)(x_1) = X(id_n + f_2)(x_2)$. The object of names N is the inclusion functor $\mathbb{I} \hookrightarrow \mathcal{S}et$. Although this model is adequate, it is far from being fully abstract: for example neither pair of observationally equivalent expressions in Example 2.8 is equated in this model.

(ii) The model in (i) can be modified to incorporate O'Hearn and Tennent's semantic notion of 'relational parametricity' [16, Sect. 6], where the relations one takes for $\mathbb{I}$ are the partial bijections used in Sect. 3. A dynamic allocation monad can be defined on parametric functors which mimicks the key definition (6) of Sect. 3. Drawing upon

Theorem 3.8, we are able to show that *the resulting model is fully abstract for first order expressions*, i.e. in this model the implication in Theorem 4.2 can be reversed when σ is first order. However, the model is not fully abstract for the whole of the nu-calculus, since in fact the expressions in (3) are not equated.

(iii) The category $\mathcal{C}$ in (i) is known to be equivalent to the category of continuous G-sets for the topological group G of permutations of $\mathbb{N}$ topologized as a subspace of Baire space $\mathbb{N}^{\mathbb{N}}$ (see [6, Lemma 1.8] for example). In this guise, one can modify the model in (i) by considering G-PERs instead of G-sets—that is, continuous G-sets given by quotienting $\mathbb{N}$ by a *partial equivalence relation*, and G-equivariant functions which are tracked by partial recursive functions. This model gives a means of establishing the observational equivalence (3), since it can be shown that no morphism to $T(1+1)$ in it distinguishes between the denotations of these two terms (even though the denotations are in fact distinct). Observational equivalence then follows by applying Lemma 2.3.

None of the above models is *fully abstract* for the whole of the nu-calculus, in the sense that the implication in Theorem 4.2 can be reversed (i.e. $[\![M_1]\!] \restriction_{\neq(s)} = [\![M_2]\!] \restriction_{\neq(s)}$ holds in the model if and only if $s \vdash M_1 \approx_\sigma M_2$). Indeed we do not know of any 'concrete' model that is fully abstract. However we conjecture that a term-model construction on (a suitable extension of) the syntax of the nu-calculus yields an instance of the categorical structure which is fully abstract.

Acknowledgements

We are grateful to Eugenio Moggi, Peter O'Hearn, Allen Stoughton and Robert Tennent for making their unpublished work available to us. We have benefited from many conversations with them on the topic of this paper.

References

[1] S. Abramsky. The Lazy Lambda Calculus. In D. Turner (ed.), *Research Topics in Functional Programming* (Addison-Wesley, 1990), pp 65–116.

[2] H.-J. Boehm. Side-effects and aliasing can have simple axiomatic descriptions, *ACM Trans. Prog. Lang. Syst.* 7(1985) 637–655.

[3] M. Felleisen and D. P. Friedman. A Syntactic Theory of Sequential State, *Theoretical Computer Science* 69(1989) 243–287.

[4] F. Honsell, I. A. Mason, S. Smith and C. Talcott. A Variable Typed Logic of Effects. In *Proc. Computer Science Logic 1992*, Lecture Notes in Computer Science (Springer-Verlag, Berlin, 1993), *to appear.*

[5] B. Jacobs. Semantics of Weakening and Contraction. Preprint, May 1992.

[6] P. T. Johnstone. Quotients of Decidable Objects, *Math. Proc. Camb. Philos. Soc.* 93(1983) 409–419.

[7] I. A. Mason and C. Talcott. References, local variables and operational reasoning. In *Proc. 7th Annual Symp. on Logic in Computer Science*, Santa Cruz, 1992 (IEEE Computer Society Press, Washington, 1992) pp 186–197.

[8] A. Meyer and K. Sieber. Towards fully abstract semantics for local variables: preliminary report. In *Conf. Record 15th Symp. on Principles of Programming Languages*, San Diego, 1988 (ACM, New York, 1988) pp 191-203.

[9] R. Milner. Fully abstract models of typed λ-calculi. *Theoretical Computer Science* 4(1977) 1–22.

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

[11] E. Moggi. An Abstract View of Programming Languages. Lecture Notes, July 1989, 46pp.

[12] E. Moggi. Notions of Computation and Monads, *Information and Computation* 93(1991) 55–92.

[13] M. Makkai and G. E. Reyes. *First Order Categorical Logic*, Lecture Notes in Math. Vol. 611 (Springer-Verlag, Berlin, 1977).

[14] P. W. O'Hearn. A Model for Syntactic Control of Interference, *Mathematical Structures in Computer Science*, to appear.

[15] P. W. O'Hearn and R. D. Tennent. Semantics of Local Variables. In M. P. Fourman, P. T. Johnstone and A. M. Pitts (eds), *Applications of Categories in Computer Science*, L.M.S. Lecture Note Series 177 (Cambridge University Press, 1992), pp 217–238.

[16] P. W. O'Hearn and R. D. Tennent. Relational Parametricity and Local Variables. In *Conf. Record 20th Symp. on Principles of Programming Languages*, Charleston, 1993 (ACM, New York, 1993) pp 171–184.

[17] A. M. Pitts. Evaluation Logic. In G. Birtwistle (ed.), *IVth Higher Order Workshop*, Banff, 1990, *Workshops in Computing* (Springer-Verlag, Berlin, 1991), pp 162–189.

[18] G. D. Plotkin. Call-by-name, call-by-value and the lambda calculus. *Theoretical computer Science* 1(1975) 125–159.

[19] G. D. Plotkin and M. Abadi. A Logic for Parametric Polymorphism. In *Proceedings of the Conference on Typed Lambda Calculus and its Applications*, Utrecht, 1993, Lecture Notes in Computer Science Vol. 664 (Springer-Verlag, Berlin, 1993) pp 361-375.

[20] J. C. Reynolds. Syntactic Control of Interference. In *Conf. Record 5th Symp. on Principles of Programming Languages*, Tucson, 1978 (ACM, New York, 1978) pp 39–46.

[21] R. D. Tennent. Semantic Analysis of Specification Logic, *Information and Computation* 85(1990) 135–162.

[22] P. Wadler. Comprehending Monads. In *Proc. 1990 ACM Conf. on Lisp and Functional Programming* (ACM, New York, 1990) pp 61–78.