1995

Note on Algol and Conservatively Extending Functional Programming

Peter W. O'Hearn
Syracuse University

J. Functional Programming **1** (1): ??–???, ?? 1995

Note on Algol and Conservatively Extending Functional Programming

Peter W. O'Hearn†
Syracuse University

Abstract

A simple Idealized Algol is considered, based on Reynolds's "essence of Algol." It is shown that observational equivalence in this language conservatively extends observational equivalence in its assignment-free functional sublanguage.

1 Introduction

In "The essence of Algol," Reynolds (1981) presents a view of Algol as a call-by-name language based on the typed λ-calculus, with "imperative" primitive types. A central feature of the design is the interaction between assignment and procedures. Side effects are wholly isolated in a primitive type **comm** of commands, and do not occur when computing a value of functional type. That is to say, side effects in procedures are latent, in the sense that an effect occurs only by evaluating a procedure call as a term of type **comm**. As a result, function types retain a genuine "functional character." For instance, *the full β and η laws* are valid equivalences in Algol-like languages. This functional aspect of Algol has been emphasized strongly by Reynolds (1981; 1988; 1992), and echoed in the works of Tennent (1989; 1991) and Felleisen and Weeks (1993).

The purpose of this short note is to give a technical result further exemplifying this functional character. Specifically, observational (or contextual) equivalence in a simple Idealized Algol conservatively extends equivalence in a simply-typed assignment-free functional sublanguage. This means that two program fragments that can be interchanged in all assignment-free programs without affecting observable behaviour can also be safely interchanged in any context in the full imperative language. Thus, not only are β, η, and so on preserved, but so are *all* equivalences from the assignment-free fragment of the language.

The proof of conservativity utilizes denotational models. The interesting twist in the proof is the use of a non-standard model for the Algol-like language. We want to work with a model of the full imperative language in which semantic equality conservatively extends equality in a standard domain-theoretic model of functional

† Research supported by NSF grant CCR-92110829.

languages. It turns out that standard models of Algol-like languages are not suitable because they contain what Reynolds calls "snapback" operations, which cause backtracking of state changes that require copying of the entire state (cf. (O'Hearn and Tennent, 1995; O'Hearn and Reddy, 1995) for discussion). These operations violate the intuitive property of irreversibility of state changes, and Section 3 shows an example of where snapback invalidates an equivalence true in the assignment-free sublanguage. Thus, conservativity fails for the standard models. The main step in the proof is the formulation of a non-standard model for which a semantic conservativity result does hold.

The result we seek concerns not only semantic equality, but observational equivalence; that is, equivalence in all program contexts. It can be (and is often) the case that semantic equality and observational equivalence for a model and language do not match. In order to extend our result to observational equivalence we need to work with a *fully abstract* model of the assignment-free sublanguage, a model in which semantic and observational equivalence do coincide. For this we use Plotkin's (1977) fully abstract model of PPCF, a language with recursion and basic arithmetic constructs, and extended with a (determinate) parallel conditional. The proof does adapt easily to other functional sublanguages, including sequential PCF, simply by working with term models. But since this adaptation should be clear from the form of the proof it seems reasonable, for the sake of simplicity, to show the result utilizing the standard continuous-function model of parallel PCF. A fully abstract model is not required for the full Algol-like language.

I consider the result given here to be part of folklore. Amongst those with a detailed knowledge of "The essence of Algol," the result is I suspect either already known, or would become known soon after the question was considered. But it is a piece of folklore that deserves to be explicitly noted, especially in light of the growing interest in integrating functional and imperative programming, e.g., (Swarup *et al.* , 1991; Wadler, 1990b; Wadler, 1990a; Peyton-Jones and Wadler, 1993; Guzmàn and Hudak, 1990; Launchbury and Peyton Jones, 1995). Conservative extension results of the kind considered here have been a specific concern in (Odersky *et al.* , 1993; Odersky, 1994; Riecke, 1993; Riecke and Viswanathan, 1995).

2 Idealized Algol

Idealized Algol extends simply-typed functional programming with primitive types for imperative features. We take the language PCF, a typed λ-calculus with recursion and basic arithmetic constructs, as our representative pure functional language. The language IA (for Idealized Algol) extends PCF with two additional primitive types, the type **comm** of commands and the type **var** of storage variables. Altogether, the types of IA are

$$t ::= \mathbf{nat} \mid \mathbf{bool} \mid \mathbf{var} \mid \mathbf{comm} \mid t \to t \ .$$

For simplicity, we only consider storage variables that hold natural-number values; variables for the booleans could easily be added. Though we will not do so here, in the presence of product types we could take **comm** as the *only* additional type,

beyond those of PCF, by defining **var** as syntactic sugar for $(\mathbf{nat} \to \mathbf{comm}) \times \mathbf{nat}$ (Reynolds, 1981).

Many of the essential properties of IA can be immediately brought to light by considering a semantics for the types. In the following, each type t determines an ω-complete partial order $\mathcal{S}[\![t]\!]$ with a least element.

$$\begin{array}{lcl} \mathcal{S}[\![\mathbf{comm}]\!] & = & S \Rightarrow S_\perp \\ \mathcal{S}[\![\mathbf{nat}]\!] & = & S \Rightarrow N_\perp \\ \mathcal{S}[\![\mathbf{bool}]\!] & = & S \Rightarrow T_\perp \\ \mathcal{S}[\![\mathbf{var}]\!] & = & S \Rightarrow L_\perp \\ \mathcal{S}[\![t' \to t]\!] & = & \mathcal{S}[\![t']\!] \Rightarrow \mathcal{S}[\![t]\!] \end{array}$$

Here, $\Rightarrow$ is is the continuous function space, $T = \{tt, ff\}$ is a two-point set (of truth values), L is a countably infinite set (of locations), N is the set of natural numbers, and S is a suitable set of states.

The striking point to notice is that the interpretation of the function type is *exactly* as in a domain-theoretic semantics of a purely-functional language. In comparison, in most imperative languages such as Pascal, ML, or Scheme, the collection of states would be used to interpred functions themselves. Furthermore – and this is related to the interpretation of the function type – side-effects are wholly concentrated in the type **comm**, since no other primitive types have the state in an output position. The **nat** and **bool** types are state-dependent, but in a read-only way. These aspects of the language are an example of what Strachey (1972) termed *structural* properties, on display from the semantics of types alone, prior to considering primitive operations or terms at all, let alone operational semantics.

IA is an applied λ-calculus with certain constants. An infinite set of variables $x^t : t$, for each type t, is assumed, together with formation rules for λ-abstraction and application:

$$\frac{M : s \to t \quad N : s}{M\ N : t} \qquad \frac{M : s}{\lambda x^t.M : t \to s}$$

The constants come in two groups. One group consists essentially of the operations of PPCF, i.e., PCF together with a parallel conditional.

$$\begin{array}{rcl} \mathbf{succ}, \mathbf{pred} & : & \mathbf{nat} \to \mathbf{nat} \\ \mathbf{if}_b & : & \mathbf{bool} \to b \to b \to b \\ \mathbf{pif}_\delta & : & \mathbf{bool} \to \delta \to \delta \to \delta \\ \mathbf{0} & : & \mathbf{nat} \\ \mathbf{0?} & : & \mathbf{nat} \to \mathbf{bool} \\ \mathbf{tt}, \mathbf{ff} & : & \mathbf{bool} \\ \mathbf{Y}_t & : & (t \to t) \to t \end{array}$$

In the rule for $\mathbf{if}_b$, the sequential conditional, b ranges over all primitive types including **var** and **comm**. In the rule for $\mathbf{pif}_\delta$, the parallel conditional, δ ranges over only **nat** and **bool**. In the rule for $\mathbf{Y}_t$, the recursion combinator, t ranges over all types of IA.

The constants for the imperative fragment of IA are as follows.

$$\begin{array}{rcl} := & : & \mathbf{var} \to \mathbf{nat} \to \mathbf{comm} \\ \mathbf{deref} & : & \mathbf{var} \to \mathbf{nat} \\ \mathbf{skip} & : & \mathbf{comm} \\ ; & : & \mathbf{comm} \to \mathbf{comm} \to \mathbf{comm} \\ \mathbf{new} & : & \mathbf{nat} \to (\mathbf{var} \to \mathbf{comm}) \to \mathbf{comm} \end{array}$$

$\mathbf{new}\, v\, P$ creates a local storage variable ℓ, initializes its contents to v, executes $P(\ell)$, and de-allocates ℓ on completion. With this explanation the binding of an identifier denoting a local variable is accomplished using λ, as in $\mathbf{new}\, v\, (\lambda x.C)$.

PPCF is a sublanguage of IA. The PPCF types are

$$\rho ::= \mathbf{nat} \mid \mathbf{bool} \mid \rho \to \rho.$$

PPCF terms are build from variables x^ρ, abstraction, application, and the constants just given (with the restriction that in $\mathbf{if}_b$ b is **nat** or **bool**). We will denote the standard continuous-function model of PPCF by $\mathcal{P}[\![\cdot]\!]$. The interpretation of types is as usual:

$$\begin{array}{lcl} \mathcal{P}[\![\mathbf{nat}]\!] & = & N_\perp \\ \mathcal{P}[\![\mathbf{bool}]\!] & = & T_\perp \\ \mathcal{P}[\![\rho' \to \rho]\!] & = & \mathcal{P}[\![\rho']\!] \Rightarrow \mathcal{P}[\![\rho]\!] \end{array}$$

A $\mathcal{P}[\![\cdot]\!]$-environment u is a type-respecting map that assigns a value $u(x^\rho) \in \mathcal{P}[\![\rho]\!]$ to each variable x^ρ, and the meaning of a PPCF term is a (continuous) map from environments into values so that $\mathcal{P}[\![M]\!]u \in \mathcal{P}[\![\rho]\!]$ when $M : \rho$. All of the constants have their usual interpretations, with $\mathbf{pif}_\delta$ being the parallel conditional. We often suppress mention of environments when speaking of $\mathcal{P}[\![\mathbf{c}]\!]$, for $\mathbf{c}$ one of the given constants. We refer to (Plotkin, 1977; Gunter, 1992) for detailed definitions.

Returning to IA, to complete the semantics of types we have to define the set S of states. There are a number of ways to do this, one of the simplest of which is to set

$$S = L \Rightarrow (N + \{\text{unused}\})$$

The unused portion is used to define the local variable declarator **new**. For this to work, we must assume that there is a partial function $\text{new} : S \rightharpoonup L$ that selects a new unused location if there is one, and is undefined if all locations are in use; see the textbook (Tennent, 1991) for a more detailed discussion.

An $\mathcal{S}[\![\cdot]\!]$-environment u is a function associating an element $u(x^t) \in \mathcal{S}[\![t]\!]$ to each variable x^t. The following semantic equations define a continuous function $\mathcal{S}[\![M]\!] : E \Rightarrow \mathcal{S}[\![t]\!]$ for $M : t$, where E is the (componentwise ordered) domain of

environments.

$$\begin{array}{rcl@{\qquad}rcl}
\mathcal{S}[\![x^t]\!]\,u & = & u(x^t) & \mathcal{S}[\![\mathbf{Y}_t]\!]f & = & \bigsqcup_{i\in N} f^i(\bot)\\
\mathcal{S}[\![M(N)]\!]\,u & = & \mathcal{S}[\![M]\!]u(\mathcal{S}[\![N]\!]u) & \mathcal{S}[\![\mathbf{0}]\!]\,s & = & 0\\
\mathcal{S}[\![\lambda x^t.M]\!]\,u\,a & = & \mathcal{S}[\![M]\!]\,(u \mid x^t \mapsto a) & \mathcal{S}[\![\mathbf{0?}]\!]a\,s & = & \mathcal{P}[\![\mathbf{0?}]\!](a(s))\\
\mathcal{S}[\![\mathbf{pred}]\!]\,a\,s & = & \mathcal{P}[\![\mathbf{pred}]\!]\,(a(s)) & \mathcal{S}[\![\mathbf{tt}]\!]\,s & = & \mathit{tt}\\
\mathcal{S}[\![\mathbf{succ}]\!]\,a\,s & = & \mathcal{P}[\![\mathbf{succ}]\!]\,(a(s)) & \mathcal{S}[\![\mathbf{ff}]\!]\,s & = & \mathit{ff}
\end{array}$$

$$\mathcal{S}[\![\mathbf{skip}]\!]s = s$$

$$\mathcal{S}[\![;]\!]a\,b\,s = \begin{cases} b(s') & \text{if } a(s) = s' \neq \bot \\ \bot & \text{if } a(s) = \bot \end{cases}$$

$$\mathcal{S}[\![\mathbf{deref}]\!]a\,s = \begin{cases} s(\ell) & \text{if } a(s) = \ell \neq \bot \\ \bot & \text{if } a(s) = \bot \end{cases}$$

$$\mathcal{S}[\![:=]\!]\,a\,b\,s = \begin{cases} s(\ell \mapsto v) & \text{if } a(s) = \ell \neq \bot,\ b(s) = v \neq \bot \\ \bot & \text{if } a(s) = \bot \text{ or } b(s) = \bot \end{cases}$$

$$\mathcal{S}[\![\mathbf{if}_b]\!]a\,b\,c\,s = \begin{cases} b(s) & \text{if } a(s) = \mathit{tt} \\ c(s) & \text{if } a(s) = \mathit{ff} \\ \bot & \text{if } a(s) = \bot \end{cases}$$

$$\mathcal{S}[\![\mathbf{pif}_\delta]\!]a\,b\,c\,s = \begin{cases} b(s) & \text{if } a(s) = \mathit{tt} \\ c(s) & \text{if } a(s) = \mathit{ff} \text{ or } c(s) = b(s) \\ \bot & \text{if } a(s) = \bot \end{cases}$$

$$\mathcal{S}[\![\mathbf{new}_\delta]\!]\,e\,p\,s = \begin{cases} (s' \mid \ell \mapsto \text{unused}) & \text{if } \mathrm{new}(s) = \ell,\ e(s) = v \neq \bot, \\ & p(\lambda s.\ell)(s \mid \ell \mapsto v) = s' \\ \bot & \text{otherwise} \end{cases}$$

With the various constants, we have suppressed mention of environments.

3 Conservativity

3.1 Semantic Conservativity

The model of IA given in the previous section is standard and, even if it is imperfect, it is certainly *computationally adequate* wrt a suitable operational semantics (Meyer and Sieber, 1988). Thus, we may consider the semantics as a reference point, for defining the language. However, the model $\mathcal{S}[\![\cdot]\!]$ is not conservative over $\mathcal{P}[\![\cdot]\!]$, as the following example shows.

Consider the type $\mathbf{bool} \to \mathbf{bool}$.

$$\mathcal{S}[\![\mathbf{bool} \to \mathbf{bool}]\!] \quad = \quad (S \Rightarrow T_\bot) \Rightarrow (S \Rightarrow T_\bot)$$

The two occurrences of the set S of states allow us to (semantically) evaluate different parts of an expression at different states. An example is the function $g \in \mathcal{S}[\![\mathbf{bool} \to \mathbf{bool}]\!]$ defined by:

$$g(e)s \;=\; e(s \mid \ell \mapsto 0)$$

where $\ell \in L$ is a fixed location. Intuitively, g executes e after changing the state, by assigning 0 to ℓ, and so there are two states, s and $(s \mid \ell \mapsto 0)$, that play a role in the evaluation of the semantic expression $g(e)s$. To see this issue on the level of

equivalences, consider the terms

$$M \;=\; \mathbf{if}\, x\, (f(x))\; \Omega \qquad\qquad N \;=\; \mathbf{if}\, x\, (f(\mathbf{tt}))\; \Omega$$

where $f : \mathbf{bool} \to \mathbf{bool}$ and $x : \mathbf{bool}$ are identifiers, and $\Omega = \mathbf{Y}_{\mathbf{bool}}(\lambda x\,.\,x)$ is the divergent boolean. This is a valid equivalence in PPCF, $\mathcal{P}[\![M]\!] = \mathcal{P}[\![N]\!]$, because in the model f is applied directly to the value of x, which is a truth value. However, in the $\mathcal{S}[\![\cdot]\!]$ model f is applied to an argument of semantic type $S \Rightarrow T_\perp$, and so there is an opportunity to apply f in states where x is false. Specifically, define $e \in \mathcal{S}[\![\mathbf{bool}]\!]$ by

$$e(s) \;=\; \begin{cases} \mathit{ff} & \text{if } s(\ell) = 0, \\ \mathit{tt} & \text{otherwise} \end{cases}$$

Now, let s be a state where $s(\ell) \neq 0$. Then $g(e)s = \mathit{ff}$ while $g(\lambda s'.\mathit{tt})s = \mathit{tt}$. Therefore, if we consider an environment u where $u(f) = g$ and $u(x) = e$, we get $\mathcal{S}[\![M]\!]us = \mathit{ff}$ while $\mathcal{S}[\![N]\!]us = \mathit{tt}$. So M and N are not equal in the model $\mathcal{S}[\![\cdot]\!]$, and semantic equality in the standard model $\mathcal{S}[\![\cdot]\!]$ of IA is not conservative over equality in the model $\mathcal{P}[\![\cdot]\!]$ of PPCF.

The function g is an example of the "snapback" effect, so named because the state change is not recorded globally in the semantics. For instance, in an environment where f denotes function g, an assignment statement $x := f(1)$ will leave location ℓ unchanged (unless x denotes ℓ) because the change to ℓ during evaluation of $f(1)$ is temporary.

We now present a semantic model that overcomes this specific difficulty pertaining to conservativity. The model does not address the general problem of irreversibility of state change; see (O'Hearn and Tennent, 1995; Reddy, 1995; O'Hearn and Reddy, 1995) for discussion of this. The aim is to provide a simple (though ad hoc) work-around, that is just enough to achieve conservativity.

The main idea of the new semantics $\mathcal{C}[\![\cdot]\!]$ is to push the state as far outward as possible, by interpreting the PPCF fragment in a way that, given any state s, "compiles" to a meaning in the PPCF model $\mathcal{P}[\![\cdot]\!]$ by reading values of variables. In intuitive terms, we will maintain the following property for the PPCF fragment:

$$\mathcal{C}[\![M]\!]us \;=\; \mathcal{P}[\![M]\!]u' \quad \text{where } u'(x) \text{ is obtained by "looking up" } u(x) \text{ in state } s$$

Here is the semantics of types.

$$\begin{array}{lcll} \mathcal{C}[\![\rho]\!] & = & S \Rightarrow \mathcal{P}[\![\rho]\!] & \text{for PPCF types } \rho \\ \mathcal{C}[\![\mathbf{comm}]\!] & = & \mathcal{S}[\![\mathbf{comm}]\!] & \\ \mathcal{C}[\![\mathbf{var}]\!] & = & \mathcal{S}[\![\mathbf{var}]\!] & \\ \mathcal{C}[\![t' \to t]\!] & = & \mathcal{C}[\![t']\!] \Rightarrow \mathcal{C}[\![t]\!] & \text{provided one of } t', t \text{ not a PPCF type.} \end{array}$$

For PPCF types there is now only one occurrence of S, at the outermost level. For example, $\mathcal{C}[\![\mathbf{bool} \to \mathbf{bool}]\!] = S \Rightarrow (T_\perp \Rightarrow T_\perp)$.

The semantic equations for the PPCF constants must be altered in certain cases.

$$\begin{array}{rcll}
\mathcal{C}[\![M(N)]\!]\,us & = & \mathcal{C}[\![M]\!]us(\mathcal{C}[\![N]\!]us) & M, N \text{ of PPCF type} \\
\mathcal{C}[\![\lambda x^t.M]\!]\,u\,s\,a & = & \mathcal{C}[\![M]\!]\,(u \mid x^t \mapsto (\lambda s \in S\,.\,a)) & \lambda x.M \text{ of PPCF type} \\
\mathcal{C}[\![\mathbf{succ}]\!]\,s & = & \mathcal{P}[\![\mathbf{succ}]\!] & \\
\mathcal{C}[\![\mathbf{pred}]\!]\,s & = & \mathcal{P}[\![\mathbf{pred}]\!] & \\
\mathcal{C}[\![\mathbf{if}_\delta]\!]s & = & \mathcal{P}[\![\mathbf{if}_\delta]\!] & \\
\mathcal{C}[\![\mathbf{pif}]\!]\,s & = & \mathcal{P}[\![\mathbf{pif}]\!] & \\
\mathcal{C}[\![\mathbf{Y}_\rho]\!]\,s & = & \mathcal{P}[\![\mathbf{Y}_\rho]\!] &
\end{array}$$

For the remaining constants and cases the equations are exactly as for $\mathcal{S}[\![\cdot]\!]$.

The non-standard semantics of the PPCF fragment of IA can be easily seen to satisfy the laws of the typed λ-calculus. In fact, it is just an interpretation of the typed λ-calculus in the Kleisli category of a monad on the category of ω-complete posets and continuous functions. The functor part of this monad is $S \Rightarrow (-)$, and the resultant Kleisli category is cartesian closed.

Lemma 1 (Semantic Conservativity)

For all PPCF terms M, N, $\mathcal{P}[\![M]\!] = \mathcal{P}[\![N]\!]$ iff $\mathcal{C}[\![M]\!] = \mathcal{C}[\![N]\!]$.

Proof
For any PPCF term M and $\mathcal{C}[\![\cdot]\!]$-environment u, a routine induction shows that $\mathcal{C}[\![M]\!]u\,s \;=\; \mathcal{P}[\![M]\!]u'$, where u' is a $\mathcal{P}[\![\cdot]\!]$-environment such that $u'(x) = u(x)s$. As a consequence, for any closed PPCF term M, we clearly have $\mathcal{C}[\![M]\!] = \lambda s \in S.\mathcal{P}[\![M]\!]$, and so the result holds for closed terms. For open terms M and N the result follows by considering closures $\lambda\vec{x}.M$ and $\lambda\vec{x}.N$, which are equal iff M and N are (by virtue of λ-calculus laws). □

The reader may enjoy verifying that the terms M and N from the example at the beginning of this section are indeed equivalent in $\mathcal{C}[\![\cdot]\!]$.

3.2 Observational Conservativity

Observational equivalence will be generated by observing convergence at ground type. In the case of IA, this means a closed term of type **comm** or **var**, as well as terms of type **nat** or **bool**.

Definition 2 (Observational Equivalence)

1. *For PPCF terms M, N, $M \equiv_{PPCF} N$ iff for all ground PPCF contexts $C[\cdot]$,*
$$\mathcal{P}[\![C[M]]\!] = \bot \quad \Longleftrightarrow \quad \mathcal{P}[\![C[N]]\!] = \bot$$
2. *For IA terms M, N, $M \equiv_{IA} N$ iff for all ground IA contexts $C[\cdot]$,*
$$\mathcal{S}[\![C[M]]\!] = \bot \quad \Longleftrightarrow \quad \mathcal{S}[\![C[N]]\!] = \bot$$

There are typical implicit provisos in this definition, such as that M and N be of the same type and that $C[\cdot]$ be a context that captures all their free identifiers.

As we indicated before, we take the standard model $\mathcal{S}[\![\cdot]\!]$ as *defining* IA. The model $\mathcal{C}[\![\cdot]\!]$, though non-standard, is adequate wrt this model.

Lemma 3 (Adequacy)

For all closed IA terms M of ground type, $\mathcal{S}[\![M]\!] = \bot$ iff $\mathcal{C}[\![M]\!] = \bot$.

Proof
The proof uses a standard "logical-relation argument" (Tennent, 1991; Gunter, 1992) to connect the meanings in the two models. Given (complete and pointed) relations $R^b \subseteq \mathcal{C}[\![b]\!] \times \mathcal{S}[\![b]\!]$ on IA primitive types, we lift to higher types by the clauses:

$$\begin{array}{lll} (p,p') \in R^{\rho\to\rho'} & \iff & \forall (a,a') \in R^{\rho} .((\lambda s\,.\,(ps)(as)), p'(a')) \in R^{\rho'} \\ (p,p') \in R^{t\to t'} & \iff & \forall (a,a') \in R^{t} .(p(a), p'(a')) \in R^{t'} \end{array}$$

where one of t, t' is not a PPCF type. Then taking R^b as the equality relation, this generates a family of relations. One checks that each constant of IA is invariant under the resulting relation, using the fact that each R^t is pointed and closed under lubs of ω-chains in the case of fixed-point. One then shows that the meanings of all terms map related environments to related meanings in the usual way, and adequacy follows. □

This, together with lemma 1, yields the result.

Proposition 4 (Observational Conservativity)

For all PPCF terms M, N, $\quad M \equiv_{PPCF} N \iff M \equiv_{IA} N$

Proof
If a PPCF context $C[\cdot]$ distinguishes M and N in $\mathcal{P}[\![\cdot]\!]$, say $\mathcal{P}[\![C[M]]\!] \neq \bot$ and $\mathcal{P}[\![C[N]]\!] = \bot$, then by the semantic conservativity lemma we have $\mathcal{C}[\![C[M]]\!] \neq \bot$ and $\mathcal{C}[\![C[N]]\!] = \bot$. The $\Leftarrow$ direction then follows from the adequacy lemma.

Conversely, if $M \equiv_{\mathrm{PPCF}} N$ then $\mathcal{P}[\![M]\!] = \mathcal{P}[\![N]\!]$ by the full abstraction theorem for $\mathcal{P}[\![\cdot]\!]$. By the semantic conservativity lemma we get $\mathcal{C}[\![M]\!] = \mathcal{C}[\![N]\!]$, and then $M \equiv_{\mathrm{IA}} N$ follows from the adequacy lemma and the compositionality of $\mathcal{C}[\![\cdot]\!]$. □

The interesting part of this argument is the use of the non-standard model of IA. It shows that the presence of snapback operations is the *only* reason for the failure of conservativity in standard models of Algol. The result also illustrates, by way of equivalences, some of the undesirable properties of snapback operations, and thus weaknesses in the models of, e.g., (Oles, 1982; O'Hearn and Tennent, 1995; Sieber, 1994). Among the more advanced models of Algol-like languages, Tennent's (1990) model of specification logic is the only one in which a semantic conservativity result holds.

4 Conclusion

Reynolds's Algol, unlike Algol 60, disallows side effects in integer and boolean expressions. This leads to a clear distinction between the types of phrases (integers, booleans) that are evaluated for the value they produce, and commands, which are evaluated soley for their side effects. Analogous conservation results typically fail for languages where there is a less strict separation. For instance, in ML or Scheme

procedure invocation is inextricably bound up with state change, and equivalences such as $f(1)+f(2) \equiv f(2)+f(1)$ that (viewed at an appropriate level of abstraction) hold in the effect-free subset – what is often referred to as the "pure" subset – do not hold in contexts where f can have a side effect. In versions of Algol that allow side effects in expressions, such as (Weeks and Felleissen, 1993), conservativity is also lost, though the laws of the typed λ-calculus remain valid.

Some recent proposals for integrating imperative and functional programming also use types to isolate effects from the procedure mechanism (Peyton-Jones and Wadler, 1993; Launchbury and Peyton Jones, 1995). A type $T(a)$ is used for state transformers that change the state and also return a value of type a: the type **comm** in IA resembles $T(\mathrm{unit})$ for a type unit with a trivial value. In these languages integer and boolean expressions are completely state-independent, whereas in IA expressions are read-only or passive, in that they are state-dependent but side-effect free. The imperative λ-calculus (Swarup *et al.* , 1991) is even closer to IA, but also uses state-independent expressions. In order to maintain equational laws in a setting that does not allow for passive or read-only types excessive sequencing of dereferencing operations is required. This is one of the motivations for considering general notions of passivity (Reynolds, 1978; Wadler, 1990b; Reddy, 1994; O'Hearn *et al.* , 1995).

Although every specific equation true in the functional sublanguage remains true in IA, it is important to note that not all "global properties" of equivalence are preserved. One example is the context lemma (Milner, 1977): two closed terms M, N of functional type in PPCF are equivalent iff $M\vec{V} \equiv N\vec{V}$ for all closed vectors $\vec{V}$ of arguments. This property fails in IA already at the type $\mathbf{comm} \to \mathbf{comm}$. For instance, the procedures $\lambda c\,.\,c$ and $\lambda c\,.\,c; c$ are not observationally equivalent, but closed applicative contexts are not sufficient to distinguish them: up to equivalence, **skip** and Ω are the only closed terms of type **comm** in IA. To create a distinguishing context we must use **new**, as in

$$\mathbf{new}\ 0\ (\lambda x\,.\,([\cdot](x := x+1)); \mathbf{if}\ x = 1\ \mathbf{then\ skip\ else}\ \Omega)$$

This failure of the context lemma can perhaps be attributed to the presence of impure features in IA, though it is difficult to make this attribution precise since "impure" is ill-defined.

References

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

Guzmàn, J., and Hudak, P. 1990. Single-threaded polymorphic lambda calculus. *Pages 333–345 of: Proceedings, Fifth Annual IEEE Symposium on Logic in Computer Science.* Philadelphia, PA: IEEE Computer Society Press, Los Alamitos, California.

Launchbury, J., and Peyton Jones, S. 1995. State in Haskell. *Lisp and Symbolic Computation.* Special issue on State in Programming Languages. To appear.

Meyer, A. R., and Sieber, K. 1988. Towards fully abstract semantics for local variables: preliminary report. *Pages 191–203 of: Conf. Record 15th ACM Symp. on Principles of Programming Languages.* ACM, New York.

Milner, R. 1977. Fully abstract models of typed λ-calculi. *Theoretical Computer Science*, **4**, 1–22.

Odersky, M. 1994. A functional theory of local names. *In: Conf. Record 21st ACM Symp. on Principles of Programming Languages.* ACM, New York.

Odersky, M., Rabin, D., and Hudak, P. 1993. Call-by-name, assignment, and the λ-calculus. *In: Conf. Record 20th ACM Symp. on Principles of Programming Languages.* Charleston, South Carolina: ACM, New York.

O'Hearn, P. W., and Reddy, U. S. 1995. Objects, Interference and the Yoneda embedding. *In: Proceedings of the Eleventh Conference on the Mathematical Foundations of Programming Semantics.* To appear. Electronic Notes in Theoretical Computer Science, volume 1.

O'Hearn, P. W., and Tennent, R. D. 1995. Parametricity and Local Variables. *Journal of the ACM.* To appear. Preliminary version appeared in *Conf. Record 20th ACM Symp. on Principles of Programming Languages*, Charleston, South Carolina, pages 171–184. ACM, New York, 1993.

O'Hearn, P. W., Power, A. J., Takeyama, M., and Tennent, R. D. 1995. Syntactic control of interference, revisited. *In: Proceedings of the Eleventh Conference on the Mathematical Foundations of Programming Semantics.* To appear. Electronic Notes in Theoretical Computer Science, volume 1.

Oles, F. J. 1982. *A Category-Theoretic Approach to the Semantics of Programming Languages.* Ph.D. thesis, Syracuse University, Syracuse, N.Y.

Peyton-Jones, S., and Wadler, P. 1993. Imperative Functional Programming. *In: Conf. Record 20th ACM Symp. on Principles of Programming Languages.* Charleston, South Carolina: ACM, New York.

Plotkin, G. D. 1977. LCF considered as a programming language. *Theoretical Computer Science*, **5**, 223–255.

Reddy, U. S. 1994. Passivity and independence. *Pages 342–352 of: Proceedings, 9th Annual IEEE Symposium on Logic in Computer Science.* IEEE Computer Society Press, Los Alamitos, California.

Reddy, U. S. 1995. Global states considered unnecessary: Introduction to object-based semantics. *Lisp and Symbolic Computation.* Special issue on State in Programming Languages. To appear.

Reynolds, J. C. 1978. Syntactic control of interference. *Pages 39–46 of: Conf. Record 5th ACM Symp. on Principles of Programming Languages.* Tucson, Arizona: ACM, New York.

Reynolds, J. C. 1981. The essence of Algol. *Pages 345–372 of:* de Bakker, J. W., and van Vliet, J. C. (eds), *Algorithmic Languages.* Amsterdam: North-Holland.

Reynolds, J. C. 1988. *Preliminary design of the programming language Forsythe.* Tech. rept. CMU-CS-88-159. Computer Science, Carnegie Mellon University, Pittsburgh.

Reynolds, J. C. 1992. *Replacing complexity with generality: the programming language Forsythe.*

Riecke, J. G. 1993. Delimiting the scope of effects. *ACM Conference on Functional Programming and Computer Architecture*, 146–158.

Riecke, J. G., and Viswanathan, R. 1995. Isolating side effects in sequential languages. *In: Conf. Record 22nd ACM Symp. on Principles of Programming Languages.* ACM, New York. San Francisco.

Sieber, K. 1994. *Full abstraction for the second order subset of an Algol-like language (preliminary report).* Technischer Bericht A 01/94, Universitaet des Saarlandes, February.

Strachey, C. 1972. The varieties of programming language. *Pages 222–233 of: Proceedings of the International Computing Symposium.* Cini Foundation, Venice. Also technical monograph PRG-10, Programming Research Group, University of Oxford, Oxford.

Swarup, V., Reddy, U.S., and Ireland, E. 1991. Assignments for applicative languages. *Pages 193–214 of:* Hughes (ed), *Functional Programming Languages and Computer Architecture.* LNCS 523, Springer Verlag.

Tennent, R. D. 1989. Elementary data structures in Algol-like languages. *Science of Computer Programming*, **13**, 73–110.

Tennent, R. D. 1990. Semantical analysis of specification logic. *Information and Computation*, **85**(2), 135–162.

Tennent, R. D. 1991. *Semantics of Programming Languages*. Prentice-Hall International.

Wadler, P. 1990a. Comprehending monads. *Pages 61–78 of: Proceedings of the ACM Conference on* LISP *and Functional Programming*.

Wadler, P. 1990b. Linear types can change the world! *In:* Broy, M., and Jones, C. (eds), *Programming Concepts and Methods*. North Holland.

Weeks, S., and Felleissen, M. 1993. On the orthogonality of procedures and assignments in Algol. *In: Conf. Record 20th ACM Symp. on Principles of Programming Languages*. Charleston, South Carolina: ACM, New York.