Descriptive and Relative Completeness of Logics for Higher-Order Functions

Kohei Honda[1], Martin Berger[1], and Nobuko Yoshida[2]

[1] Department of Computer Science, Queen Mary, University of London
[2] Department of Computing, Imperial College London

Abstract. This paper establishes a strong completeness property of compositional program logics for pure and imperative higher-order functions introduced in [2, 15–18]. This property, called *descriptive completeness*, says that for each program there is an assertion fully describing the former's behaviour up to the standard observational semantics. This formula is inductively calculable from the program text alone. As a consequence we obtain the first relative completeness result for compositional logics of pure and imperative call-by-value higher-order functions in the full type hierarchy.

1 Introduction

Program logics such as Hoare logic offer a means to *describe* abstract behaviours of programs as logical assertions; to *verify* that a given program satisfies a specified property; and to *define* axiomatic semantics in the sense that the assertions assign meaning to a program with respect to its observable properties. Because of this strong match with observable semantics of programs in a simple and intuitive manner, many engineering activities ranging from static analyses to program testing increasingly use program logics as their theoretical foundation.

For describing properties of first-order imperative programs, Hoare logic uses a pair of assertions in number theory. For example, in the partial correctness judgement $\{x = i\}x := x+1\{x = i+1\}$, the pair of assertions $x = i$ and $x = i+1$ describes a property of the program $x := x+1$ by saying: *whatever the initial content of x would be, if this program terminates, then the final content of x is the increment of its initial one.* Here a *property* is a subset of programs taken modulo an observational congruence: for example, in `while` programs, we consider programs up to partial functions on store they represent. Since the collection of all properties is uncountable, no standard logical language can represent all properties of any non-trivial programming language. Then what classes of properties should a program logic represent and prove?

In this paper, we focus on a strong completeness result, *descriptive completeness*, which is about representability of behaviour *as a canonical formula*: given a program P, we can always find a unique assertion pair which represents (pinpoints) P's behaviour. For partial correctness, the best assertion pair for P describes all partial functions equal to or less defined than P. For example, the pair "$x = i$" and "$x = i+1$" are also satisfied by a diverging program. Dually for total correctness. A related concept are the *characteristic formulae* of Hennessy-Milner logics, which precisely characterise a CCS process up to bisimilarity [12, 31, 32]. We shift this notion from a process logic to a program logic, establishing descriptive completeness of Hoare logics for pure and imperative higher-order functions introduced in [2, 15, 17, 18].

In first-order Hoare logic, a program defines a partial function from states to states, so that the existence of characteristic formulae is not hard to establish. When we move to higher-order programs, a logic needs to describe how a program *transforms behaviour*. For example $\lambda x^{\mathsf{Nat}\Rightarrow(\alpha\Rightarrow\beta)}.x1$ is a function which receives a function and returns another function. The logics for higher-order functions and their imperative extensions [2, 15, 17, 18] involve direct description of such applicative behaviour. Due to complexity of the underlying semantic universe, it is not immediately obvious if a single pair of formulae can fully describe the behaviour of an arbitrary higher-order program. In the present paper we *construct* a characteristic formula of a program compositionally and algorithmically, following its syntactic structure, and inductively verify that the derived formula has the required properties. The induced algorithm is implemented as a prototype (1,250 LOC in Ocaml) [1]. The size of the resulting formula is asymptotically almost linear to the size of a program under a certain condition.

The generated characteristic assertions clarify the relationship between total and partial correctness for higher-order objects, following early observations [28, 29], but in the context of concrete assertion methods and proof systems. We use the duality between total and partial correctness [28] to derive descriptive completeness for partial correctness from its total counterpart. A total correctness property denotes an upward closed set of semantic points, representing liveness restricted to sequentiality, while a partial correctness formula stands for a downward closed set of semantic points, representing safety [22, 25, 28]. This duality not only subsumes the original Hoare logic's notions of total and partial correctness, but also offers a key insight into the nature of assertions for higher-order partial objects and their derivation. Finally, relative completeness [5] of our proof system is an immediate consequence of descriptive completeness. To our knowledge this work is the first to obtain descriptive and relative completeness for total and partial correctness in Hoare logic for (imperative) higher-order functions in full type hierarchy.

In the remainder, Section 2 establishes descriptive and relative completeness w.r.t. the logic of call-by-value PCF (PCFv) for total correctness. The same property for partial correctness is obtained via duality. Section 3 discusses the corresponding results for the imperative extension of the logic. Section 4 is devoted to comparisons with related work. Finally Section 5 discusses further topics, including practical implications of the presented results.

2 Descriptive Completeness for PCFv

Call-by-value PCF. The syntax and types of PCFv is standard [26], and briefly reviewed below (we can easily treat, but omit, other standard types such as sums and products [17]).

$$\alpha,\beta,\ldots ::= \mathsf{Bool} \mid \mathsf{Nat} \mid \alpha\Rightarrow\beta \qquad V,W,\ldots ::= x^{\alpha} \mid \mathtt{c} \mid \lambda x^{\alpha}.M \mid \mu f^{\alpha\Rightarrow\beta}.\lambda x^{\alpha}.M$$
$$M,N,\ldots ::= V \mid \mathtt{op}(\tilde{M}) \mid MN \mid \mathtt{if}\ M\ \mathtt{then}\ N_1\ \mathtt{else}\ N_2$$

We use numerals (0,1,2,..) and booleans ($\mathtt{t}$ and $\mathtt{f}$) as constants ($\mathtt{c}$ above) and standard first-order operations ($\mathtt{op}(\tilde{M})$ where $\tilde{M}$ denotes a vector). $V, V', \ldots$ denote values. The typing is standard; henceforth we only consider well-typed programs. A *basis* $(\Gamma, \Delta, \ldots)$ is a finite map from variables to types. If M has type α with its free variables typed following Γ, we write $\Gamma \vdash M : \alpha$. A program is *closed* if it has no free variables. The call-by-value evaluation relation is written $M \Downarrow V$. If M diverges, we write $M \Uparrow$. We use the standard contextual congruence $\cong$ and the precongruence $\lesssim$ [13, 26]: given M and N of the same type, we set $M \lesssim N$ iff, for each typed closing context $C[\,\cdot\,]$, $C[M] \Downarrow$ implies $C[N] \Downarrow$: $\cong$ is the symmetric closure of $\lesssim$.

We list three simple programs. First, the standard recursive factorial program is written $\mathtt{Fact} \stackrel{\text{def}}{=} \mu f^{\mathsf{Nat} \Rightarrow \mathsf{Nat}}.\lambda x^{\mathsf{Nat}}.\mathtt{if}\ x = 0\ \mathtt{then}\ 1\ \mathtt{else}\ x \times f(x-1)$. Second, in each arrow type we find $\Omega^{\alpha \Rightarrow \beta} \stackrel{\text{def}}{=} \mu f^{\alpha \Rightarrow \beta}.\lambda x^{\alpha}.fx$, which diverges whenever invoked. Third, $\omega^{\alpha} \stackrel{\text{def}}{=} \Omega^{\mathsf{Nat} \Rightarrow \alpha} 0$ gives an immediately diverging program (note $\Omega^{\alpha \Rightarrow \beta} \cong \lambda x^{\alpha}.\omega^{\beta}$).

Assertions and their Semantics. We use the following assertion language from [2, 17, 18], common to both total correctness and partial correctness.

$$e ::= \mathsf{c} \mid x^{\alpha} \mid \mathsf{op}(\tilde{e}) \qquad A ::= e_1 = e_2 \mid e_1 \bullet e_2 = e_3 \mid A \wedge B \mid A \vee B \mid A \supset B \mid \neg A \mid \forall x^{\alpha}.A \mid \exists x^{\alpha}.A$$

The left definition is for terms, that on the right for formulae. c denotes a constant, either numerals (0,1,2, ...) or booleans (t and f). Terms are typed as in PCFv. *Henceforth we only consider well-typed terms.* e^{α} indicates e has type α. Constants and first-order operations are from PCFv. We assume the standard bound name convention for formulae. If types of free variables in A follow Γ, we write $\Gamma \vdash A$. We set T as $1 = 1$ and F as its negation. $\equiv$ denotes logical equivalence. The assertion language is first-order, with a ternary predicate $e_1 \bullet e_2 = e_3$, called *evaluation formula*. Intuitively $e_1 \bullet e_2 = e_3$ means:

> *If a function denoted by e_1 is applied to an argument denoted by e_2 then it converges to a value denoted by e_3.*

Note $e_1 \bullet e_2 = e_3$ indicates termination. "=" in $e_1 \bullet e_2 = e_3$ is asymmetric and $\bullet$ is a non-commutative operation like application in an applicative structure. For example, assume f denotes a function which doubles the number n: then the assertion "$f \bullet 5 = 10$" means if we apply that function to 5, then the evaluation terminates and its result is 10.

Meaning of assertions is given by a simple term model. A *model* $(\xi, \xi', \ldots)$ is a finite map from typed variables to closed PCFv-values of the same types. Interpretation of terms is standard, denoted $\xi[\![e]\!]$. The satisfaction relation is written $\xi \models A$, and follows the standard clauses [23, Section 2.2] except that equality is interpreted by $\cong$ (i.e. $\xi \models e_1 = e_2$ iff $\xi[\![e_1]\!] \cong \xi[\![e_2]\!]$). In addition, for an evaluation formula, we define:

$$\xi \models e_1 \bullet e_2 = e_3 \quad \text{if} \quad \exists V.(\xi[\![e_1]\!]\xi[\![e_2]\!] \Downarrow V \wedge V \cong \xi[\![e_3]\!]). \tag{2.1}$$

We write $\Gamma \vdash \xi$ if $\mathsf{dom}(\Gamma) = \mathsf{dom}(\xi)$ and the typing of ξ follows Γ.

Judgements. The judgement for total correctness is written $[A]\, M :_u [B]$, prefixed with $\models$ for validity, and $\vdash$ for provability. It is the standard Hoare triple augmented with an anchor [2, 15, 17, 18]. An anchor is a fresh name denoting the result of evaluation. u may only occur in B. The judgement $[A]\, M :_u [B]$ intuitively says:

> *If a model ξ satisfies A, then $M\xi$ converges and ξ together with the result, named u, satisfy B.*

In $[A]\, M :_u [B]$, we assume for some Γ and α we have $\Gamma \vdash M : \alpha$, $\Gamma \vdash A$ and $\Gamma \cdot u{:}\alpha \vdash B$.

Provability $\vdash [A]\,M :_u [B]$ is defined by the proof rules [17] listed in Appendix A.1, which precisely follow the syntax of programs. Validity $\models [A]\,M :_u [B]$ is defined by the following clause (let Γ be the minimum basis under which M, A and B are typable).

$$\forall \xi.((\Gamma \vdash \xi \ \wedge\ \xi \models A) \supset (M\xi \Downarrow V \ \wedge\ \xi \cdot u : V \models B)). \tag{2.2}$$

The proof of soundness, $\vdash [A]\,M :_u [B]$ implies $\models [A]\,M :_u [B]$, is mechanical. Later we demonstrate the converse. Simple examples of judgements follow.

1. We have $\vdash [\mathsf{T}]\,\mathtt{Fact} :_u [\forall x^{\mathsf{Nat}}.f \bullet x = x!]$, saying $\mathtt{Fact}$ computes a factorial whenever invoked. We also have $\vdash [\mathsf{T}]\,\mathtt{Fact} :_u [\forall x^{\mathsf{Nat}}.(Even(x) \supset \exists i.(f \bullet x = i \wedge Even(i))]$ where $Even(n)$ says n is even.
2. We have $\vdash [\mathsf{F}]\,\omega :_u [\mathsf{F}]$, which is the best formulae we can get for ω. Note this judgement holds for arbitrary programs of the same type.
3. From 2 above, we derive $\vdash [\mathsf{T}]\,\lambda x.\omega :_u [\mathsf{T}]$. The judgement contains no information for values, in the sense that all values satisfy it: as it should be, since we had to start from the trivial judgement for ω. Similarly $\vdash [\mathsf{T}]\,\Omega :_u [\mathsf{T}]$ is the best we can get.

Characteristic Formulae. In the last examples of judgements, we have seen the notion of total correctness and compositional verification *demand* that an assertion pair in the present logic cannot directly describe divergence. For this reason the notion of an assertion pair representing a given program pinpoints its behaviour as the *least* element of the described property. We call such a formula a total characteristic assertion pair.

Definition 1. (TCAP) A pair (A,B) is a *total characteristic assertion pair*, or *TCAP*, *of M at u*, if the following conditions hold (in each clause we assume well-typedness).

1. (soundness) $\models [A]\,M :_u [B]$.
2. (MTC, minimal terminating condition) $M\xi \Downarrow$ if and only if $\xi \models A$.
3. (closure) Suppose $\models [E]\,N :_u [B]$ such that $E \supset A$. Then $\xi \models E$ implies $M\xi \lesssim N\xi$.

Proposition 2. *1. If (A,B) is a TCAP of M at u and if $\models [A]\,N :_u [B]$, then $M \lesssim N$.*
2. *(A,B) is a TCAP of M at u iff (soundness), (MTC) and the following condition hold:* (closure-2): *if $\xi \models A$ and $\xi \cdot u : V \models B$ then $M\xi \lesssim V$.*

Proof. For (1), assume $\models [A]\,N :_u [B]$. If $\xi \models A$ then by definition $N\xi \Downarrow V$ s.t. $\xi \cdot u : V \models B$, hence $M\xi \lesssim N\xi$ by (closure); if else, $M\xi \Uparrow$ by (MTC), that is $M\xi$ is the least element.

For (2), for the "if" direction, suppose $[E]N :_u [B]$ such that $E \supset A$. Suppose $\xi \models E$. By $E \supset A$ we have $\xi \models A$. By $[E]N :_u [B]$ we have $N\xi \Downarrow V$ and $\xi \cdot u : V \models B$. Hence $M\xi \sqsubseteq V \cong N\xi$, as required. For the "then" direction, suppose (A,B) is a TCAP of M at u. We show (closure-2) holds. Suppose $\xi \models A$ and $\xi \cdot u : V \models B$. Take $E = A \wedge \exists u.B$. Then (since V is a value) we have $[E]V :_u [B]$. By (closure) this means for each ξ' such that $\xi' \models E$ we have $M\xi' \lesssim V\xi' \stackrel{\text{def}}{=} V$. Taking ξ as ξ' we are done. □

By Proposition 2-1, a TCAP of M denotes a collection of behaviours whose minimum element is M, and in that sense characterises that behaviour uniquely. Note upwardly closing the property represented by a TCAP results in another TCAP characterising the same behaviour. We shall make use of such closure later.

Fig. 1 Derivation Rules for Total CAPs

$$[var]\,\frac{-}{\vdash^\star [\mathsf{T}]\, x :_u [u=x]} \qquad [const]\,\frac{-}{\vdash^\star [\mathsf{T}]\, \mathsf{c} :_u [u=\mathsf{c}]}$$

$$[op]\,\frac{\vdash^\star [A_i]\, M_i :_{m_i} [B_i]}{\vdash^\star [\bigwedge_i A_i]\, \mathsf{op}(M_1..M_n) :_u [\exists \tilde{m}.(u=\mathsf{op}(m_1..m_n) \wedge \bigwedge_i B_i)]}$$

$$[abs]\,\frac{\vdash^\star [A]\, M :_m [B]}{\vdash^\star [\mathsf{T}]\, \lambda x.M :_u [\forall x.(A \supset \exists m.(u \bullet x = m \ \wedge\ B))]} \qquad [rec]\,\frac{\vdash^\star [\mathsf{T}]\, \lambda x.M :_u [A]}{\vdash^\star [\mathsf{T}]\, \mu f.\lambda x.M :_u [A[u/f]]}$$

$$[app]\,\frac{\vdash^\star [A_1]\, M :_m [B_1] \quad \vdash^\star [A_2]\, N :_n [B_2]}{\vdash^\star [A_1 \wedge A_2 \wedge \forall mn.(B_1 \wedge B_2 \supset \exists z.m \bullet n = z)]\, MN :_u [\exists mn.(m \bullet n = u \wedge B_1 \wedge B_2)]}$$

$$[if]\,\frac{\vdash^\star [A]\, M :_m [B] \quad \vdash^\star [A_i]\, N_i :_u [B_i] \quad \mathsf{b}_1 = \mathsf{t},\ \mathsf{b}_2 = \mathsf{f}}{\vdash^\star [A \ \wedge\ \bigwedge_{i=1,2}(B[\mathsf{b_i}/m] \supset A_i)]\, \mathtt{if}\ M\ \mathtt{then}\ N_1\ \mathtt{else}\ N_2 :_u [\bigvee_{i=1,2}(B[\mathsf{b_i}/m] \wedge B_i)]}$$

Descriptive Completeness. In the following we show that all PCFv-terms have TCAPs. The idea is to generate pre/post conditions inductively following the syntax of PCFv-terms. Figure 1 presents the generation rules, which are illustrated below.

- All rules are close to the corresponding proof rules in Appendix A.1. [*var*], [*const*] and [*op*] are easily understood. [*abs*] is direct from the semantics of evaluation formulae. In [*app*], the premise says A_1 guarantees M_1's termination, A_2 that of M_2. Hence the conclusion's precondition ought to stipulate $A_{1,2}$, as well as termination of the application of the results (described by B_1 and B_2).
- A crucial rule is [*rec*], which represents recursion by simply renaming the recurring f to the anchor u. The rule intuitively says the program now uses itself for the environment f. Note that the size of the formula does not change by applying this rule.

Examples of derived assertions follow (which are, as we shall soon see, indeed TCAPs).

Example 3. 1. For the identity function, we get $\vdash^\star [\mathsf{T}]\, \lambda x.x :_u [\forall x.u \bullet x = x]$ (simplified using logical axioms) saying: *whatever value the program receives, it always (converges and) returns the same value.*

2. For $\lambda x.fx$, we get $\vdash^\star [\mathsf{T}]\, \lambda x.fx :_u [\forall xi.(f \bullet x = i \supset u \bullet x = i)]$ (simplification uses axioms for evaluation formulae [18]) which says: *if the application of f to x converges to some value, then the application of u to x converges to the same value.*
3. From 1, we obtain $\vdash^\star [\mathsf{T}]\, \mu f.\lambda x.x :_u [\forall x.u \bullet x = x]$ via vacuous renaming, as expected.
4. From 2, we obtain a TCAP for Ω as $\vdash^\star [\mathsf{T}]\, \Omega :_u [\mathsf{T}]$ by $\forall xi.(u \bullet x = i \supset u \bullet x = i) \equiv \mathsf{T}$. Since Ω is the least defined total behaviour, we cannot say anything better than T for this agent (note T is indeed a TCA of Ω).
5. The factorial program `Fact` is given the following assertion.

$$\vdash^\star [\mathsf{T}]\, \mathtt{Fact} :_u [u \bullet 0 = 1 \ \wedge\ \forall xi.(u \bullet x = i \supset u \bullet (x+1) = x \times i)] \qquad (2.3)$$

   Note the assertion closely follows the recursive behaviour of the program. Through mathematical induction we obtain $\vdash^\star [\mathsf{T}]\, \mathtt{Fact} :_u [\forall x.(u \bullet x = x!)]$, as expected.

The main result of this section follows.

Theorem 4. (descriptive completeness for total correctness) *Assume* $\Gamma \vdash M : \alpha$. *Then* $\vdash^{\star} [A]\, M :_u [B]$ *implies* (A,B) *is a TCAP of* M *at* u.

Proof. We establish the three conditions of TCAP of Def. 1 simultaneously by rule induction, using (closure-2) in Prop.2-2 for (closure). In this main section we only show the most non-trivial case **[*rec*]**, leaving other cases to Appendix C.1. First, (MTC) is vacuous. For (soundness), letting $\xi' = \xi \cdot f : \mu f.\lambda x.M\xi$, we obtain:

$$\xi' \cdot u : (\lambda x.M)\xi' \models A \wedge f = u \quad \Rightarrow \quad \xi \cdot u : (\mu f.\lambda x.M)\xi \models \exists f.(A \wedge f = u) \;\; (\equiv A[u/f]).$$

hence done. For (closure-2), assume $\xi \cdot u : V \models A[u/f]$, which is equivalent to $\xi \cdot u : V \cdot f : V \models A$. We show $\mu f.\lambda x.M\xi \lesssim V$ using the standard unfolding [27] of $\mu f.\lambda x.M$, given by: $W_0 \stackrel{\text{def}}{=} \Omega$ and $W_{n+1} \stackrel{\text{def}}{=} \lambda x.M[W_n/f]$ (for each $n \geq 0$), and show, by induction on n, that $W_n\xi \lesssim V$ for each n. The base case, $n = 0$, is immediate. For the inductive step let $W_n\xi \lesssim V$. Now

$$W_{n+1}\xi \;\stackrel{\text{def}}{=}\; \lambda x.M\xi[W_n\xi/f] \;\lesssim\; \lambda x.M\xi[V/f] \;\stackrel{\text{def}}{=}\; \lambda x.M(\xi \cdot f : V) \;\lesssim\; V$$

The left inequality holds because $[\cdot/x]$ is a monotonic operation (i.e. $V \lesssim W$ implies $M[V/x] \lesssim M[W/x]$), while the right inequality is direct from the induction hypothesis. Thus we have $W_n\xi \lesssim V$ for each n. Since if $\mu f.\lambda x.M\xi \not\lesssim V$ then $W_n \not\lesssim V$ for some n by syntactic continuity of $\lesssim$ (cf. [27]), we conclude $\mu f.\lambda x.M\xi \lesssim V$. □

Proposition 5. *If* $\vdash^{\star} [A]\, M :_u [B]$ *then the sum of the size of* A *and* B *is* $O(m \times 2^n)$ *where* m *is the size of* M *and* n *is the number of applications/conditionals in* M.

Proof. By mechanical rule induction, see Appendix C.3. □

Definition 6. *Let* x *be fresh in 2 and 3.*

1. *We define* $\sqsubseteq$ *inductively as follows: (1)* $x^{\alpha} \sqsubseteq y^{\alpha}$ *iff* $x = y$ *for* $\alpha \in \{\mathsf{Bool}, \mathsf{Nat}\}$*; and (2)* $x^{\alpha \Rightarrow \beta} \sqsubseteq y^{\alpha \Rightarrow \beta}$ *iff* $\forall z^{\alpha}, v^{\beta}.(x \bullet z = v \supset \exists w.(y \bullet z = w \wedge v \sqsubseteq w))$.
2. $\mathcal{U}(A,u) \stackrel{def}{=} \forall x.(A[x/u] \supset x \sqsubseteq u)$ *and* $\uparrow(A,u) \stackrel{def}{=} \exists x.(A[x/u] \wedge x \sqsubseteq u)$. *Dually we set* $\mathcal{L}(A,u) \stackrel{def}{=} \forall x.(A[x/u] \supset u \sqsubseteq x)$ *and* $\downarrow(A,u) \stackrel{def}{=} \exists x.(A[x/u] \wedge u \sqsubseteq x)$.
3. *Write* $\models M :_u \{A\}$ *when* $M\xi \Downarrow V$ *implies* $\xi \cdot u : V \models A$ *for each* ξ. *We say* A *is a* PCAP of M at u *when the following two conditions hold: (partial sound)* $\models M :_u \{A\}$*; and (partial closure) whenever* $\models N :_u \{A\}$ *we have* $N \lesssim M$.

Remark. The predicate $\sqsubseteq$ internalises the relation $\lesssim$ logically. Note that:

$$\xi \models x \sqsubseteq y \quad \equiv \quad \xi(x) \lesssim \xi(y) \tag{2.4}$$

Correspondingly, $\mathcal{U}(A,u)$ etc. are logical counterparts of the basic order-theoretic operations [6]. Indeed, from (2.4), we immediately observe:

$$\xi \cdot u : V \models \mathcal{U}(A,u) \quad \Leftrightarrow \quad \forall V_0.(\xi \cdot u : V_0 \models A \supset V_0 \lesssim V) \tag{2.5}$$

$$\xi \cdot u : V \models \uparrow(A,u) \quad \Leftrightarrow \quad \exists V_0.(\xi \cdot u : V_0 \models A \;\wedge\; V_0 \lesssim V), \tag{2.6}$$

dually for $\mathcal{L}(A,u)$ and $\downarrow(A,u)$. Finally a PCAP is the partial counterpart of a TCAP. In partial correctness we do not need a precondition since a(n obviously defined) partial correctness judgement $\{A\}M :_u \{B\}$ is equivalent to $\{\mathsf{T}\}M :_u \{A \supset B\}$, due to statelessness of PCFv.

Corollary 7. *1.* (observational completeness) $M \cong N$ *if and only if, for each A and B, we have* $\models [A]\,M :_u [B]$ *iff* $\models [A]\,N :_u [B]$.
2. (relative completeness) *We say B is* upward-closed at u *when* $\uparrow (B,u) \equiv B$. *Then* $\models [A]\,M :_u [B]$ *such that B is upward-closed at u implies* $\vdash [A]\,M :_u [B]$.
3. (derivability of partial characteristic assertion) *If* $\vdash^\star [A]\,M :_u [B]$ *then* $A \wedge \mathcal{L}(B,u)$ *is a PCAP of M at u.*

Remark. The restriction to upward-closed formulae in (2) is not unduly constraining since upward closure corresponds to total correctness [22, 28] (intuitively, upwards closed formulae never talk about non-termination). For Corollary 7 (3), note if M is a value, A becomes T, in which case the induced formula $\mathcal{L}(B,u)$ simply represents the downward-closed set of behaviours with the maximum element M, as expected. A proof system that can derive PCAPs is discussed in Appendix B.

Proof. For (1), for simplicity and without loss of generality we restrict our attention to values (noting $M \cong N$ iff $\lambda x.M \cong \lambda x.N$ for fresh x). If $V \cong W$ then by definition $\models [\mathsf{T}]\,V :_u [B]$ iff $\models [\mathsf{T}]\,W :_u [B]$ for each B. Conversely assume $\models [\mathsf{T}]\,V :_u [B]$ iff $\models [\mathsf{T}]\,W :_u [B]$ for each B. Let (T,A) be a TCAP of V at u and (T,A') be a TCAP of W at u. By assumption this means $\models [\mathsf{T}]\,V :_u [A']$ and $\models [\mathsf{T}]\,W :_u [A]$. By the definition of TCAP this means $V \cong W$, as required. For (2), relative completeness, we first show:

Claim. For each M, there is a TCAP (A,B) at u s.t. $B \equiv \uparrow (B,u)$ and $\vdash [A]\,M :_u [B]$.

(where $\vdash [A]\,M :_u [B]$ is the provability by the proof rules in Appendix A). The proof is elementary by Theorem 4, see Appendix C.2, page 23. Now suppose $\models [A]\,M :_u [B]$ such that B is upward-closed at u, i.e. $B \equiv \uparrow (B,u)$. Further let $\vdash [A_0]\,M :_u [B_0]$ be s.t. (A_0,B_0) is a TCAP and B_0 is upward-closed at u, by Claim above. We show $A \supset (A_0 \wedge (B_0 \supset B))$, then apply the consequence rule [Consequence-Kleymann] in Appendix A. First, $A \supset A_0$ is valid since A_0 satisfies (MTC). Second we show $(A \wedge B_0) \supset B$. Assume $\xi \cdot u{:}V \models A \wedge B_0$, then, for some W, $M\xi \Downarrow W$ and $\xi \cdot u{:}W \models B$. By Theorem 4, (closure) holds for B_0, so $W \lesssim V$. Since B is upper-closed, $\xi \cdot u{:}V \models B$.

For (3), suppose $\vdash^\star [A]\,M :_u [B]$. We first show the condition (partial-sound), i.e. $\models M :_u \{A \wedge \mathcal{L}(B,u)\}$. Suppose $M\xi \Downarrow$. By (MTC) we have $\xi \models A$. Note:

$$\xi \cdot u : M\xi \models \mathcal{L}(B,u) \quad \Leftrightarrow \quad \forall V_0.(\xi \cdot u : V_0 \models B \supset M\xi \lesssim V_0) \tag{2.7}$$

By Proposition 2 (1) we are done. For (partial-closure), assume $N :_u \{A \wedge \mathcal{L}(B,u)\}$. It suffices to show for each ξ we have $N\xi \lesssim M\xi$. This is trvial when $N\xi \Uparrow$. Suppose $N\xi \Downarrow$. Then $\xi \cdot u : N\xi \models A \wedge \mathcal{L}(B,u)$, that is $\xi \models A$ and $\xi \cdot u : N\xi \models \mathcal{L}(B,u)$. As in (2.7) the latter means:

$$N\xi \lesssim V \text{for each } \xi \cdot u : V \models B. \tag{2.8}$$

By (closure-2) we have $\xi \cdot u : M\xi \models B$ for each $\xi \models A$. That is $N\xi \sqsubseteq M\xi$. $\square$

3 Descriptive Completeness for Imperative PCFv

Logic for Imperative PCFv, Below we discuss how the method for deriving TCAPs studied in the previous section generalises to the imperative extension of the logic [18]. We consider the programming language (and the corresponding logic) without aliasing

[18]. To the grammar of types, we add the unit type Unit and a reference type $\mathsf{Ref}(\alpha)$ where α itself does not include a reference type (thus reference types are never carried inside other types, which corresponds to the lack of aliasing [18]). For programs, we add assignment $x := M$, dereferencing $!x$ and $()$ of Unit type. Typing is of the form $\Gamma;\Delta \vdash M : \alpha$, where Δ is for free references and Γ for free variables of non-reference types. $\cong$ (resp. $\lesssim$) is a typed congruence (resp. precongruence), relating two programs of a common basis, by convergence under all typed contexts which never extend nor abstract the common reference basis.[1] Formally we write $\Gamma;\Delta \vdash M \cong N : \alpha$ when M and N are typed congruent, though we often leave the basis implicit, writing $M \cong N$.

For the assertion language, we add $!x$ and $()$ to terms, and replace evaluation formulae for pure functions with their imperative refinement $[C]\ e_1 \bullet e_2 = x\ [C']$ (the x binds its free occurrences in C'), which says:

> *In any state satisfying C, if e_1 is applied to e_2, it converges to a value named x and the resulting state, together satisfying C'.*

Above we demand, for some α and β, e_1 has type $\alpha \Rightarrow \beta$, e_2 α, and x β. We also write $[C]e_1 \bullet e_2[C']$ for $[C]e_1 \bullet e_2 = z[C']$ with z being of unit type.

A judgement for total correctness is written $[C]\,M :_u [C']$, which formally has, but usually leaves implicit, a fixed basis (in detail: we assume a basis $\Gamma;\Delta$ such that (1) C and M can be typed under the basis, with M's type say α and (2) $\Gamma, u : \alpha;\Delta \vdash C'$). Some examples of judgements follow.

Example 8. 1. The assertion $[!x = i]\,M :_u [u = i+1]$ says that M reads the content of x and returns the successor of that content. It does not make any guarantee about what is stored in memory after execution of M.
2. Under Δ with domain $\{x,y\}$, the assertion $[!x = i \wedge !y = j]\,M :_u [u = i+1 \wedge !x = i \wedge !y = j]$ is like (1), but in addition ensures M does not modify any storage cells.
3. Let $A(f) \stackrel{\text{def}}{\equiv} \forall i.[!y = i]\ f \bullet () = z\ [z = !y = i+1]$. It characterises a procedure f that increments a reference y and returns the increment.
4. $[\mathsf{T}]\,\lambda().(!x)() :_u [\forall i.[A(!x) \wedge !y = i]\ u \bullet () = c\ [!y = c = i+1]]$ with A above, describes a procedure which, upon invocation, invokes the procedure stored in x, which, assuming $A(!x)$, increments the content of y and returns that increment.
5. Finally, just like in the pure functional case, $[\mathsf{F}]\,\omega :_u [\mathsf{F}]$ is the strongest total specification we can derive about ω.

As before, we write $\models [C]\,M :_u [C']$ for validity and $\vdash [C]\,M :_u [C']$ for provability. The latter is defined by the proof rules studied in [18], which are listed in Appendix A.2. For validity we first define models.

A program M is *semi-closed* if its all free names are references. A *model* $(\mathcal{M},\ldots)$ is a pair (ξ,σ) where ξ maps non-reference names to semi-closed values and σ is a store, mapping reference names to semi-closed values. A store σ maps x of type $\mathsf{Ref}(\alpha)$ in its domain to a semi-closed value of type α whose free names are exhausted in the domain of σ. We write $\Delta \vdash \sigma$ if the typing is correct in this sense w.r.t. Δ and $\mathsf{dom}(\Delta) = \mathsf{dom}(\sigma)$, similarly we write $\Gamma \vdash \xi$. The interpretation of terms is standard, written $[\![e]\!]\mathcal{M}$ (note interpretation of dereference $!x$ needs the store part of a model).

[1] Reference bases affect equality: for example, with f typed as $\mathsf{Unit} \Rightarrow \mathsf{Unit}$, two programs $f()$ and $f();f()$ are observationally congruent under the empty reference basis, but are not under say $x : \mathsf{Ref}(\mathsf{Nat})$. Change in non-reference basis has no such effects.

For satisfaction, the equality is modelled by $\cong$ while the logical connectives and quantifiers are interpreted classically. For evaluation formulae, we set, letting $\mathcal{M} = (\xi, \sigma_0)$, $\mathcal{M}^{\Gamma;\Delta} \models [C] e_1 \bullet e_2 \searrow x[C']$ when, for each σ such that $\Delta \vdash \sigma$ and $(\xi, \sigma) \models C$:

$$([\![e_1]\!]\mathcal{M}[\![e_2]\!]\mathcal{M},\ \sigma)\ \Downarrow\ (V, \sigma') \text{ such that } (\xi \cdot x{:}V,\ \sigma') \models C'\ )$$

Note that the precondition C above is about a hypothetical state: for example an assertion $!x = 1 \wedge [!x = 0] f \bullet () [!x = 2]$ (omitting the return value when it is of unit type) says: (1) the current content of x is 1; and (2) if in some state the content of x is 0 then invoking f terminates with the new content of x which is 1.

Finally we set the validity of judgements as follows: $\models [C] M :_u [C']$ iff, for each (ξ, σ) s.t. $(\xi, \sigma) \models C$, we have $((M\xi, \sigma) \Downarrow (V, \sigma')$ and $(\xi \cdot u : V, \sigma') \models C'$. Thus, intuitively speaking, $\models [C] M :_u [C']$ says:

> *M converges under any environment and state satisfying C, so that the resulting value and state together satisfy C'.*

Note the close connection of the judgement with evaluation formulae, which may as well be considered as internalisation of judgement.

We can now define TCAPs for imperative PCFv. Below in (3) $\sigma_1 \lesssim \sigma_2$ is taken pointwise.

Definition 9. (TCAP) A pair (C, C') is a *total characteristic assertion pair*, or *TCAP*, *of* $\Gamma; \Delta \vdash M : \alpha$ *at* u, if the following conditions hold (fix the basis of models as $\Gamma; \Delta$).

1. (soundness) $\models [C] M :_u [C']$.
2. (MTC, minimal terminating condition) $(\xi \cdot u : M\xi, \sigma) \Downarrow$ iff $(\xi, \sigma) \models C$.
3. (closure) Suppose $\models [E] N :_u [C']$, $E \supset C$ and $(\xi, \sigma) \models E$. Then $(M\xi, \sigma) \Downarrow (V, \sigma')$ implies $(N\xi, \sigma) \Downarrow (W, \sigma'')$ such that $V \lesssim W$ and $\sigma' \lesssim \sigma''$.

The generation of TCAPs thus defined will be presented later, after a short discussion on a useful syntactic tool for our technical development.

Sequential Let Form. In the TCAP generation, we use the class of "sequentially flattened" programs for a concise presentation of the generation rules. These flattened programs are generated by the following grammar.

$$\begin{array}{lll}
U & ::= & x \mid \mathtt{c} \mid \lambda x.L \mid \mu x.\lambda y.L \\
L & ::= & U \mid \mathtt{let}\ x = \mathtt{op}(U_1..U_n)\ \mathtt{in}\ L \mid \mathtt{let}\ x = UU'\ \mathtt{in}\ L \mid \mathtt{if}\ U\ \mathtt{then}\ L_1\ \mathtt{else}\ L_2 \\
 & \mid & x := U; L \mid \mathtt{let}\ x = !y\ \mathtt{in}\ L
\end{array}$$

We call terms generated from this grammar, *sequential let forms*. Sequential let forms are ranged over by $L, L', \ldots$, while values in sequential let forms are ranged over by $U, U', \ldots$. In sequential let forms, the evaluation ordering of expressions is directly visible as a sequence of lets, horizontally expanded. Thus, given a sequential let form, the evaluation order of expressions can be traced by looking at the let sequence from the left to the right except in recursion. Through the standard translation of "let" constructs into application and abstraction, each sequential let form can be considered as a program in imperative PCFv: for example, $\mathtt{let}\ x = !y\ \mathtt{in}\ \mathtt{let}\ z = x0\ \mathtt{in}\ fz$ becomes

$(\lambda x.(\lambda z.fz)(x0))(!y)$. In turn, all programs of imperative PCFv can be easily translated to their flattened forms without changing semantics by the following mapping.

$$[\![M]\!] \stackrel{\text{def}}{=} \langle\!\langle M \rangle\!\rangle_x[x]$$

where $\langle\!\langle M \rangle\!\rangle_x[N]$ is given as follows.

$$\begin{array}{rcl}
\langle\!\langle x \rangle\!\rangle_y[N] & \stackrel{\text{def}}{=} & \mathtt{let}\ x = y\ \mathtt{in}\ N \\
\langle\!\langle \mathsf{c} \rangle\!\rangle_y[N] & \stackrel{\text{def}}{=} & \mathtt{let}\ y = \mathtt{c}\ \mathtt{in}\ N \\
\langle\!\langle \lambda x.M \rangle\!\rangle_y[N] & \stackrel{\text{def}}{=} & \mathtt{let}\ y = \lambda x.[\![M]\!]\ \mathtt{in}\ N \\
\langle\!\langle M_1 M_2 \rangle\!\rangle_y[N] & \stackrel{\text{def}}{=} & \langle\!\langle M_1 \rangle\!\rangle_{m_1}[\langle\!\langle M_2 \rangle\!\rangle_{m_2}[\mathtt{let}\ y = m_1 m_2\ \mathtt{in}\ N]] \\
\langle\!\langle \mu f.\lambda x.M \rangle\!\rangle_y[N] & \stackrel{\text{def}}{=} & \mathtt{let}\ y = \mu f.\lambda x.[\![M]\!]\ \mathtt{in}\ N \\
\langle\!\langle \mathtt{if}\ M\ \mathtt{then}\ N_1\ \mathtt{else}\ N_2 \rangle\!\rangle_y[N] & \stackrel{\text{def}}{=} & \langle\!\langle M \rangle\!\rangle_m[\mathtt{if}\ m\ \mathtt{then}\ \langle\!\langle N_1 \rangle\!\rangle_y[N]\ \mathtt{else}\ \langle\!\langle N_2 \rangle\!\rangle_y[N]] \\
\langle\!\langle x := M \rangle\!\rangle_y[N] & \stackrel{\text{def}}{=} & \langle\!\langle M \rangle\!\rangle_z[x := z; \mathtt{let}\ y = ()\ \mathtt{in}\ N] \\
\langle\!\langle !x \rangle\!\rangle_y[N] & \stackrel{\text{def}}{=} & \mathtt{let}\ y = !x\ \mathtt{in}\ N
\end{array}$$

By regarding $\mathtt{let}\ x = U\ \mathtt{in}\ L$ as $\mathtt{let}\ x = (\lambda()U)()\ \mathtt{in}\ L$, the mapping $[\![M]\!]$ is always a sequential let form. Note the inductive translation does nothing but making explicit the evaluation order in subexpressions of M. The following derived proof rule is useful for understanding how sequential let forms interact with compositional proof rules.

$$[\text{let}]\frac{[C]M :_x [C_0] \quad [C_0]N :_u [C']}{[C]\mathtt{let}\ x = M\ \mathtt{in}\ N :_u [C']} \tag{3.1}$$

It is worth looking at how the "let" rule is derived through the standard translation of the let command, $\mathtt{let}\ x = M\ \mathtt{in}\ N \stackrel{\text{def}}{=} (\lambda x.M)N$.

1.	$[C_0]N :_u [C']$	(premise)
2.	$[\mathsf{T}]\lambda x.N :_n [[C_0]n \bullet x = u[C']]$	(2, abs)
3.	$[C]\lambda x.N :_n [C \wedge [C_0]n \bullet x = n[C']]$	(3, weak)
4.	$[C]M :_x [C_0]$	(premise)
5.	$[C \wedge [C_0]n \bullet x = n[C']]M :_x [C_0 \wedge [C_0]n \bullet x = n[C']]$	(4, inv)
6.	$[C](\lambda x.N)M :_n [C']$	(5, invariance)

where (app) etc. indicate the proof rules in Appendix A.2 (Lines 3 and 6 use structural rules admissible in the proof system). Note the evaluation order of subexpressions is precisely captured in the compositional reasoning. Hereafter we consider an extended syntax of imperative PCFv with "let" for which we assume the proof rule above (which does not change semantics), in which case we write provabilty $\vdash^{\mathsf{let}}$. For the proofs of the following lemma, see Appendix C.4 and C.5.

Lemma 10. *Below in (1), $\cong$ on $[\![M]\!]$ is defined regarding sequential let forms as imperative PCFv-terms.*

1. *For each $\Gamma;\Delta \vdash M : \alpha$, we have $\Gamma;\Delta \vdash M \cong [\![M]\!]$.*
2. *$\vdash^{\mathsf{let}} [C]\,[\![M]\!] :_u [C']$ implies $\vdash [C]\,M :_u [C']$.*

Fig. 2 Derivation Rules for TCAPs for Imperative PCFv.

$$[var]\frac{-}{\vdash^{\star\star} [\mathsf{T}]\, y :_u [u = y]} \qquad [const]\frac{-}{\vdash^{\star\star} [\mathsf{T}]\, \mathsf{c} :_u [u = \mathsf{c}]} \qquad [val]\frac{\vdash^{\star\star} [\mathsf{T}]\, U :_u [A] \quad \tilde{i}\text{ fresh}}{\vdash^{\star} [!\tilde{x} = \tilde{i}]\, U :_u [A \wedge !\tilde{x} = \tilde{i}]}$$

$$[op\text{-}val]\frac{\vdash^{\star\star} [\mathsf{T}]\, U_i :_{m_i} [A_i]}{\vdash^{\star} [\mathsf{T}]\, \mathsf{op}(U_1,..,U_n) :_u [\exists \tilde{m}.(u = \mathsf{op}(m_1,..,m_n) \wedge \bigwedge_i A_i)]}$$

$$[abs]\frac{\vdash^{\star} [C]\, L :_m [C'] \quad \tilde{i} = \mathsf{fv}(C,C') \setminus (\mathsf{fv}(L) \cup \{u\tilde{x}\})}{\vdash^{\star\star} [\mathsf{T}]\, \lambda y.L :_u [\forall y\tilde{i}.([C]u \bullet y = m[C'])]}$$

$$[let\text{-}app]\frac{\vdash^{\star\star} [\mathsf{T}]\, V_1 :_m [A] \quad \vdash^{\star\star} [\mathsf{T}]\, V_2 :_n [B] \quad \vdash^{\star} [C]\, L :_u [C'] \quad \tilde{i}\text{ fresh}}{\vdash^{\star} [!\tilde{x} = \tilde{i} \wedge \forall mn.((A \wedge B) \supset \{!\tilde{x} = \tilde{i}\} m \bullet n = y\{C\})]\, \mathtt{let}\ y = V_1 V_2\ \mathtt{in}\ L :_u [C']}$$

$$[if]\frac{\vdash^{\star\star} [\mathsf{T}]\, U :_m [A] \quad \vdash^{\star} [C_i]\, L_i :_u [C'_i] \quad \mathsf{b}_1 = \mathsf{t},\ \mathsf{b}_2 = \mathsf{f}}{\vdash^{\star} [\bigwedge_{i=1,2}(A[\mathsf{b_i}/m] \supset C_i)]\, \mathtt{if}\ U\ \mathtt{then}\ L_1\ \mathtt{else}\ L_2 :_u [\bigvee_{i=1,2}(A[\mathsf{b_i}/m] \wedge C'_i)]}$$

$$[assign]\frac{\vdash^{\star\star} [\mathsf{T}]\, U :_z [A] \quad \vdash^{\star} [C]\, L :_u [C']}{\vdash^{\star} [\forall z.(A \supset C[z/!y])]\, y := U; L :_u [C']} \qquad [deref]\frac{\vdash^{\star} [C]\, L :_u [C']}{\vdash^{\star} [C[!y/z]]\, \mathtt{let}\ z = !y\ \mathtt{in}\ L :_u [C']}$$

Descriptive Completeness. Figure 2 gives the generation rules for TCAPs, using sequential let forms without loss of generality (by Lemma 10-1). In all rules we fix, but leave implicit, a reference basis with domain $\tilde{x}$, which stays unchanged when going from premises to conclusions. In [*val*], which transforms the sequent for values (written $\vdash^{\star\star}$) to that for general programs (written $\vdash^{\star}$). We write $\vdash^{\star} [C]\, L^{\Delta} :_u [C']$ when $\vdash^{\star} [C]\, L :_u [C']$ is derived with an implicit basis Δ.

Theorem 11. (descriptive completeness in imperative PCFv) *If* $\Gamma;\Delta \vdash M : \alpha$ *then* $\vdash^{\star} [C]\, [\![M]\!]^{\Delta} :_u [C']$ *implies* (C,C') *is a TCAP of* $\Gamma;\Delta \vdash M : \alpha$ *at u.*

Convention 12. From now on we let $A,B,\ldots$ range over *stateless formulae*, i.e. those formulae in which derefences occur only in pre/post conditions of evaluation formulae.

Note satisfaction of a stateless formula does not depend on store. The convention is consistent with the usage of symbols in Figure 2.

Proof. By Lemma 10-1, it suffices to prove if $\vdash^{\star} [C]\, L :_u [C']$ then (C,C') gives a TCAP of L at u. To show this, we verify $\vdash_{\mathtt{tcapv}} [\mathsf{T}]\, U :_u [A]$ then (T,A) gives a value-TCAP of L at u, and that e if $\vdash^{\star} [C]\, L :_u [C']$ then (C,C') gives a TCAP of L at u, by rule induction of the generation rules in Figure 2, where we say (T,A) is a *value TCAP of V at u* if we have (1) (soundness) $\models [\mathsf{T}]\, V :_u [A]$ and (2) (closure) $\models [\mathsf{T}]\, W :_u [A]$ implies $V\xi \lesssim W\xi$ for each W. Observe the notion of value-TCAPs is identical with TCAPs for values in the pure PCFv. The inductive verification follows the one given Appendix C.1 as well as, for imperative commands, a related proof for the finite subset of sequential let forms in [18, Section 6.5] except for (val) and (rec). For (val), fix a reference basis with domain $\tilde{x}$. Assume (T,A) is a value-TCAP of V as well as $\models [!\tilde{x} = \tilde{i}]\, M :_u [A \wedge !\tilde{x} = \tilde{i}]$. From the latter we know, by easy calculation, that (1) M always converges under any model, and that (2) $(M\xi,\sigma) \Downarrow (V,\sigma')$ implies $M\xi \cong V$ and $\sigma \cong \sigma'$. By (T,A) being a value-TCAP of V we are done. For (rec) the proof is literally identical with the one given in Section 2, observing the store part of a model is irrelevant for value-TCAPs. $\square$

In the following we first present the consequences of 11 except the derivation of PCAP (the latter demands internalisation of $\lesssim$ which is more subtle in in imperative PCFv). In (2) below, we say C is *upward-closed at u* when for each (ξ,σ) covering free names of B except u, whenever $(\xi\cdot u : V,\sigma) \models C$ and $V \lesssim W$ we have $(\xi\cdot u : W,\sigma) \models C$.

Corollary 13.

1. (observational completeness in imperative PCFv)) *$M \cong N$ if and only if, for each C and C', we have $\models [C]\,M :_u [C']$ iff $\models [C]\,N :_u [C']$.*
2. (relative completeness for values in imperative PCFv) *$\models [\mathsf{T}]\,V :_u [A]$ such that A is upward-closed at u implies $\vdash [\mathsf{T}]\,V :_u [A]$.*

Remark. We believe (2) extends to general programs, with a notion of upward-closure for pre/post conditions suitably defined. Practically we can always turn a program into a value by vacuous abstraction, so that this does not lose generality.

Proof. The proof of (1) is literally the same as that of Corollary 7 (1). For (2), first note, for an upward closed A:

$$\models [\mathsf{T}]\,U :_u [A] \quad \Rightarrow \quad \vdash [\mathsf{T}]\,U :_u [A]. \tag{3.2}$$

which is immediate from Theorem 11, the definition of value-TCAP, and A's upward-closure. Now we reason:

$$\begin{array}{lcll} \models [\mathsf{T}]\,V :_u [A] & \Rightarrow & \models [\mathsf{T}]\,[\![V]\!] :_u [A] & \text{(Lem.10-1)} \\ & \Rightarrow & \vdash^{\mathsf{let}} [\mathsf{T}]\,[\![V]\!] :_u [A] & \text{(3.2)} \\ & \Rightarrow & \vdash [\mathsf{T}]\,V :_u [A] & \text{(Lem.10-2),} \end{array}$$

as required. □

For PCAP generation, we need to internalise $\lesssim$. In the imperative PCFv, we directly incorporate this notion as a predicate, written $x \sqsubseteq y$. It satisfies the following axioms, in addition to the standard axioms for partial order (we assume a singleton basis with the domain r for brevity).

1. $x^{\alpha} \sqsubseteq y^{\alpha}$ iff $x = y$ for $\alpha \in \{\mathsf{Bool},\mathsf{Nat}\}$; and
2. $x^{\alpha\Rightarrow\beta} \sqsubseteq y^{\alpha\Rightarrow\beta}$ iff $\forall z^{\alpha}, v^{\beta}, i.([!r = i]x \bullet z = v[v = j \wedge !r = h] \supset [!r = i]y \bullet z = v[j \sqsubseteq v \wedge h \sqsubseteq !r])$

Note the right-hand side of (2) cannot be used as the inductive definition, since the type of $!r$ can be higher than $\alpha \Rightarrow \beta$. The predicate $\sqsubseteq$ is interpreted as $\lesssim$. We expect the above axioms offer a complete axiomatisation of the relation. Using $\sqsubseteq$, we can define $\mathcal{L}(B,u)$ as before. We can now state the PCAP derivability under this extended logical language. The proof is an immediate order-theoretic argument.

Corollary 14. (derivability of partial characteristic assertion) *In the logic with $\sqsubseteq$, If $\vdash^{\star} [\mathsf{T}]\,V :_u [A]$ then $\mathcal{L}(A,u)$ is a PCAP of V at u.*

We conclude this section with examples.

Example 15. 1. Let us fix a basis for programs and judgements, assuming two imperative variables y and z storing natural numbers. Then we get the following TCAP for $\lambda x.x$ (up to straightforward simplification):

$$\vdash^{\star} [\mathsf{T}] \;\; \lambda x.x :_u \;\; [\forall xnm.([!y = n \wedge !z = m] \; u \bullet x = i \; [i = x \wedge !y = n \wedge !z = m])]$$

Under the assumed basis, the lack of change of the contents of y and z (i.e. n and m) signify that the program has no side effects. For $\lambda x.fx$ we get:

$$\begin{array}{l} \forall xnmn'm'.([!y = n \wedge !z = m] \; f \bullet x = i \; [i = i' \wedge !y = n' \wedge !z = m'] \\ \qquad\qquad\qquad \supset \quad [!y = n \wedge !z = m] \; u \bullet x = i \; [i = i' \wedge !y = n' \wedge !z = m']) \end{array}$$

Note how causality between the calls to f and $\lambda x.fx$, named u, is described by auxiliary variables n, m, n' and m'. The TCAP for $\mu f.\lambda x.fx$ is again T.

2. Next we look at the TCAP for an imperative version of factorial, given as:

$$\mathtt{FactImp} \stackrel{\text{def}}{=} \mathtt{while} \; !z \gneq 0 \; \mathtt{do} \; (y := !y \times !z; \; z := !z - 1)$$

The "`while`" construct is easily represented in the imperative PCFv using recursion [18], leading to the following TCAP for the thunk of `FactImp`[2]:

$$\vdash^{\star} [\mathsf{T}] \; \lambda x^{\mathsf{Unit}}.\mathtt{FactImp} :_u [B(u) \wedge I(u)]$$

where we set $B(u)$ and $I(u)$, and $E(u)$ used in $I(u)$, to be:

$$\begin{array}{rcl} B(u) & \stackrel{\text{def}}{=} & \forall n.[\,!y = n \wedge !z = 0] \; u \bullet () \; [\,!y = n \wedge !z = 0] \\ I(u) & \stackrel{\text{def}}{=} & \forall nmn'm'.[\,!y = n \wedge !z = m \gneq 0 \;\wedge\; E(u)] \; u \bullet () \; [\,!y = n' \wedge !z = m'] \\ E(u) & \stackrel{\text{def}}{=} & [\,!y = n \times m \wedge !z = m - 1] \; u \bullet () \; [!y = n' \wedge !z = m']. \end{array}$$

where $[C] \; u \bullet () \; [C']$ is an abbreviation of $[C] \; u \bullet () = z \; [z = () \wedge C']$ with z fresh. $B(u)$ describes the behaviour when the loop condition is no longer true, whereas $I(u)$ when the loop condition is satisfied. $E(u)$ specifies the assumed behaviour of u when z is decremented. By mathematical induction we obtain:

$$B(u) \wedge I(u) \quad \equiv \quad \forall nm.[!y = n \wedge !z = m] \; u \bullet () \; [!y = n \times m! \wedge !z = 0]$$

which eliminates the internal evaluation formula $E(u)$.

3. As a final example, we consider another imperative factorial, this time using a stored procedure to realise recursion.

$$\mathtt{CircFact} \quad \stackrel{\text{def}}{=} \quad w := \lambda x.\mathtt{if} \; x = 0 \; \mathtt{then} \; 1 \; \mathtt{else} \; x \times !w(x-1)$$

In [18], we have shown that a natural specification for `CircFact` is derivable in the logic for imperative PCFv. For this program, $\vdash^{\star}$ leads to the following TCAP:

$$[\mathsf{T}] \;\; \mathtt{CircFact} :_m [m = () \wedge B'(u) \wedge I'(u)]$$

[2] In the assertion, $B(u)$ and $I(u)$ can be easily combined into a single evaluation formula (in fact the TCAP is initially derived in such a form). We use these two formulae and a thunked form for clarity of presentation.

where we set, assuming w constitutes the only store for brevity:

$$
\begin{aligned}
B'(u) &\stackrel{\text{def}}{=} \forall f.\,[!w = f]\ u \bullet 0 = z\ [z = 1 \wedge !w = f] \wedge !w = u \\
I'(u) &\stackrel{\text{def}}{=} \forall f f' i. \forall x \gneqq 0.\,[!w = f \wedge E'(u)]\ u \bullet x = z\ \ [z = x \times i \wedge\ !w = f'] \\
E'(u) &\stackrel{\text{def}}{=} [!w = f]\ u \bullet (x-1) = z\ [z = i \wedge\ !w = f']
\end{aligned}
$$

This is the full specification of `CircFact`: it does not directly say the program computes a factorial since the procedure stored in w may change its behaviour depending on what w stores at the time of invocation (note w is not hidden). However through mathematical induction we can justify the following (strict) implication:

$$B'(u) \wedge I'(u) \quad \supset \quad \exists f.(\forall i.[!w = f](!w) \bullet i = z[z = i! \wedge\ !w = f]\ \wedge\ !w = f)$$

arriving at the "natural" specification of `CircFact` given in [18], which says: *after executing* `CircFact`*, w stores a procedure f which would calculate a factorial if w indeed stores that behaviour itself, and that w does store that behaviour.*

4 Related Work

Apart from their usage in verification condition generation [10], weakest preconditions and strongest postconditions [9] help in deriving relative completeness in Hoare logic. Cook's original proof [5] of relative completeness constructs the strongest postcondition for partial correctness. Clarke [4] uses the weakest liberal pre-condition. In both, the pre/post-conditions for loops use Gödel's β-function [23, Section 3.3]. Sokołowski [30] may be the first to give a completeness result for total correctness for the `while` language. De Bakker [7] extends these results to parameterless recursive procedures and concretely constructs what we call MTC (cf. Def. 1). Gorelick [11] seems the first to use most general formulae (MGFs, which correspond to our CAPs) for completeness in Hoare logic. Kleymann [20] introduces a powerful consequence rule and employs MGF for proving completeness of Hoare logic with parameterless recursive procedures. Halpern [14], Olderog [24] and others establish relative completeness of Hoare logics for sublanguages of Algol (these logics do not include assertions on higher-order behaviours, see [18, Section 8] for a survey). Von Oheimb's recent work [33] gives a mechanised proof of completeness for Hoare logic using MGFs.

Some authors also use abstraction on predicates to generate concise verification conditions in the setting of Floyd-Hoare assertion methods for first-order imperative programs. Blass and Gurevich [3], guided by a detailed study of Cook's completeness result, use an existential fixpoint logic. Leivant [21] uses second-order abstraction (abstraction on first-order predicates), inductively deriving a formula directly representing a partial function defined by a while program with recursive first-order procedure. Once this is done, characteristic assertions for both total and partial correctness for a given program are immediate. We suspect that the use of predicate abstraction in these works may make calculation of validity hard in practise, even for first-order programs.

There are two notable differences between the present work and these preceding studies. First, in the preceding works, generated assertion pairs describe first-order state transformation rather than the behaviour of higher-order programs. Philosophically, our method may be notable in that it extends completeness and related results to assertions which directly talk about (higher-order) behaviour. Second, the presented method for constructing characteristic formulae is very different from those employed so far, especially in its treatment of recursion. We need neither the β-predicate, loop annotation, predicate abstraction nor inductively defined formulae for generating TCAPs for recursion. Concretely, this clean treatment for recursion is made possible by evaluation formulae. A deeper reason however may lie in analytical, fine-grained nature of our assertion language, reflecting that of call-by-value higher-order computation. As far as our experience goes, evaluation formulae do not make calculation of validity unduly harder than in the first-order Hoare logic: for example, (often implicit) simplifications of assertions in Sections 2 and 3 only use simple syntactic axioms in [15, 18] combined with standard logical axioms including mathematical induction.

The order-theoretic nature of partial and total correctness is observed in early works by Plotkin and Stirling [28] (cf. [29]). The present work differs in that it substantiates these ideas at the level of concrete assertion methods and compositional proof rules (see for example a derivation example of $\lambda x.\omega$ in Section 2). Finally our emphasis on descriptive completeness, and the foundation of the logic itself, comes from Hennessy-Milner logic [15], where Graf, Ingólfsdóttir, Sifakis, Steffen and others [12, 31, 32] study characteristic formulae for first-order communicating processes.

5 Further Topics

The present work is an inquiry into the descriptive power of program logic for higher-order functions. Through inductive derivation of characteristic formulae, we have shown that the logic allows concise description of full behaviour of programs involving arbitrary higher-order types and recursion. Logics for more complex classes of imperative higher-order functions are studied in [2, 34]. Extensions of the presented results to these logics are important for treating such languages as ML.

Practically speaking, the presented method for TCAP generation, along with its properties, opens a new perspective for program validation based on verification condition generators (VCG) [10, 19]. In traditional VCG, we have a target specification $\{C\}P\{C'\}$ and an annotated version of the program. A VCG then automatically generates, usually through backward chaining [19], one or more entailments whose validity entails P's satisfaction of the specification. The presented TCAP generation has the potential to improve this existing scheme. Schematically, our TCAP generation suggests the following framework.

1. Assume given a program V (any program can be made into a value by vacuous abstraction) and a desired specification $[\mathsf{T}]V :_u [A]$.
2. We automatically generate the TCAP (T, A_0).
3. By Theorem 4, if we can validate $A_0 \supset A$, we know V conforms to A.

First, this framework dispenses with the need to annotate programs, which has been one of the obstacles preventing wide-spread adoption of the VCG-based validation methods. Second, at the level of specification, it allows direct treatment of higher-order idioms, opening the use of higher-order languages such as ML and Haskell for program certification (arguably these languages offer a suitable basis for this task through their well-studied semantic foundations). Third, the specification A above can contain assumptions on behaviour of V's environment (say existing libraries, represented as free variables of function types in V) on which V relies. As we discussed in [18, Section 2], this allows specifying complex interplay among the program and library functions beyond the separate treatment of assumptions on procedures in traditional methods. For these reasons, inquiries into the practical potential of TCAP generation for program validation would be worth pursuing. As an experiment towards this goal, we have developed a prototype implementation of the TCAP generation algorithm [1].

One of the foremost challenges towards practical use of TCAP generation is the development of tractable methods for logical calculation of entailment in Step 3 above, which demands, in addition to first-order Hoare logic, the treatment of logical primitives for (imperative) higher-order functions. It would be especially interesting to extend verification tools like Simplify [8] in this direction, combined with studies on axiom systems for e.g. evaluation formulae (see [2, 15, 18] for a preliminary study).

Finally, we believe that upper bound in Prop. 5 can be improved upon considerably, at least for large and practically relevant classes of programs.

References

1. A prototype implementation of an algorithm deriving characteristic fomulae. http://www.dcs.qmul.ac.uk/~martinb/capg, October 2005.
2. M. Berger, K. Honda, and N. Yoshida. A logical analysis of aliasing for higher-order imperative functions. In *ICFP'05*, pages 280–293, 2005. A full version in www.dcs.qmul.ac.uk/~kohei/logics.
3. A. Blass and Y. Gurevich. The Underlying Logic of Hoare Logic. In *Current Trends in Theoretical Computer Science*, pages 409–436. 2001.
4. E. M. Clarke. The characterization problem for Hoare logics. In *Proc. Royal Society meeting on Mathematical logic and programming languages*, pages 89–106, 1985.
5. S. A. Cook. Soundness and completeness of an axiom system for program verification. *SIAM J. Comput.*, 7(1):70–90, 1978.
6. B. A. Davey and H. A. Priestley. *Introduction to Lattices and Order*. Cambridge University Press, Cambridge, 1990.
7. J. W. de Bakker. *Mathematical Theory of Program Correctness*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1980.
8. D. Detlefs, G. Nelson, and J. B. Saxe. Simplify: a theorem prover for program checking. *J. ACM*, 52(3):365–473, 2005.
9. E. W. Dijkstra. *A Discipline of Programming*. Prentice Hall, 1976.
10. R. W. Floyd. Assigning meaning to programs. In *Symp. in Applied Mathematics*, volume 19, 1967.
11. G. Gorelick. A complete axiomatic system for proving assertions about recursive and non-recursive programs. Technical Report 75, Univ. of Toronto, 1975.
12. S. Graf and J. Sifakis. A modal characterization of observational congruence on finite terms of ccs. In *ICALP'84*, pages 222–234, London, UK, 1984. Springer-Verlag.
13. C. A. Gunter. *Semantics of Programming Languages*. MIT Press, 1995.
14. J. Y. Halpern. A good Hoare axiom system for an ALGOL-like language. In *11^{th} POPL*, pages 262–271. ACM Press, 1984.
15. K. Honda. From process logic to program logic. In *ICFP'04*, pages 163–174. ACM Press, 2004.
16. K. Honda. From process logic to program logic (full version of [15]). Available at: www.dcs.qmul.ac.uk/~kohei/logics, November 2004. Typescript, 52 pages.
17. K. Honda and N. Yoshida. A compositional logic for polymorphic higher-order functions. In *PPDP'04*, pages 191–202, 2004.

18. K. Honda, N. Yoshida, and M. Berger. An observationally complete program logic for imperative higher-order functions. In *LICS'05*, pages 270–279, 2005. A full version in www.dcs.qmul.ac.uk/˜kohei/logics.
19. J. C. King. A program verifier. In *IFIP Congress (1)*, pages 234–249, 1971.
20. T. Kleymann. Hoare logic and auxiliary variables. Technical report, University of Edinburgh, LFCS ECS-LFCS-98-399, October 1998.
21. D. Leivant. Logical and mathematical reasoning about imperative programs: preliminary report. In *Proc. POPL'85*, pages 132–140, 1985.
22. D. Leivant. Partial correctness assertions provable in dynamic logics. In *FoSSaCS*, volume 2987 of *LNCS*, pages 304–317, 2004.
23. E. Mendelson. *Introduction to Mathematical Logic*. Wadsworth Inc., 1987.
24. E.-R. Olderog. Sound and Complete Hoare-like Calculi Based on Copy Rules. *Acta Inf.*, 16:161–197, 1981.
25. S. Owicki and L. Lamport. Proving liveness properties of concurrent programs. *ACM Trans. Program. Lang. Syst.*, 4(3):455–495, 1982.
26. B. C. Pierce. *Types and Programming Languages*. MIT Press, 2002.
27. A. M. Pitts. Operationally-based theories of program equivalence. In *Semantics and Logics of Computation*, pages 241–298. Cambridge University Press, 1997.
28. G. D. Plotkin and C. Stirling. A framework for intuitionistic modal logics. In *TARK*, pages 399–406. Morgan Kaufmann, 1986.
29. M. Smyth. Power domains and predicate transformers: A topological view. In *ICALP'83*, volume 154 of *LNCS*, pages 662–675, 1983.
30. S. Sokołowski. Axioms for total correctness. *Acta Inf.*, 9:61–71, 1977.
31. B. Steffen. Characteristic formulae. In *ICALP'89*, pages 723–732. Springer-Verlag, 1989.
32. B. Steffen and A. Ingólfsdóttir. Characteristic formulae for processes with divergence. *Inf. Comput.*, 110(1):149–163, 1994.
33. D. von Oheimb. Hoare logic for mutual recursion and local variables. In *FSTTCS*, volume 1738 of *LNCS*, pages 168–180, 1999.
34. N. Yoshida, K. Honda, and M. Berger. Local state in hoare logic for imperative higher-order functions, 2006. www.dcs.qmul.ac.uk/˜kohei/logics.

A Proof Rules

A.1 Proof Rules for PCFv: Total Correctness

Below $A^{\text{-}x}$ denotes A in which x does not occur free. See [2, 17, 18] for illustration and examples. The consequence rule comes from Kleymann [20]. The recursion rule is strengthened in comparison with [2, 17, 18] (the original recursion rule is still known to be relatively complete when the logical language is extended with inductive definition, see Appendix C.6).

$$[\mathit{Var}]\ \frac{-}{[C[x/u]]\, x :_u [C]} \qquad [\mathit{Const}]\ \frac{-}{[C[\mathsf{c}/u]]\, \mathsf{c} :_u [C]}$$

$$[\mathit{Add}]\ \frac{[C]\, M :_m [C_0] \quad [C_0]\, N :_n [C'[m+n/u]]}{[C]\, M+N :_u [C']}$$

$$[\mathit{Abs}]\ \frac{[A^{\text{-}x} \wedge C]\, M :_m [C']}{[A]\, \lambda x.M :_u [\forall x.(C \supset \exists m.(u \bullet x = m \ \wedge\ C'))]}$$

$$[\mathit{App}]\ \frac{[C]\, M :_m [C_0] \quad [C_0]\, N :_n [\exists u.(m \bullet n = u \ \wedge\ C')]}{[C]\, MN :_u [C']}$$

$$[\mathit{If}]\ \frac{[C]\, M :_m [C_0] \quad [C_0[\mathsf{t}/m]]\, N_1 :_u [C'] \quad [C_0[\mathsf{f}/m]]\, N_2 :_u [C']}{[C]\, \mathtt{if}\ M\ \mathtt{then}\ N_1\ \mathtt{else}\ N_2 :_u [C']}$$

$$[\mathit{Rec}]\ \frac{[A^{\text{-}x}]\, \lambda y.M :_u [B]}{[A]\, \mu x.\lambda y.M :_u [B[u/x]]}$$

$$[\mathit{Conseq}]\ \frac{[A']\, M :_u [B'] \quad A \ \supset\ (\ A' \ \wedge\ (B' \supset B)\ )}{[A]\, M :_u [B]}$$

A.2 Proof Rules for Imperative PCFv: Total Correctness

The rules for expressions, first-order operators, recursion, and if-then-else are identical with those in Appendix A.1.

$$[\mathit{Abs}]\ \frac{[A^{\text{-}x} \wedge C]\, M :_m [C']}{[A]\, \lambda x.M :_u [\forall x.[C]\ u \bullet x = m\ [C']]} \qquad [\mathit{App}]\ \frac{[C]\, M :_m [C_0] \quad [C_0]\, N :_n [C_1 \wedge [C_1]\ m \bullet n = u\ [C']]}{[C]\, MN :_u [C']}$$

$$[\mathit{Deref}]\ \frac{-}{[C[!x/u]]\ !x :_u [C]} \qquad [\mathit{Assign}]\ \frac{[C]\, M :_m [C'[m/\,!x][()/u]]}{[C]\, x := M :_u [C']}$$

$$[\mathit{Conseq\text{-}Kleymann}]\ \frac{[C_0] M :_u [C_0'] \ \ C \supset \exists \tilde{j}.(\ C_0[\tilde{j}/\tilde{i}] \wedge (C_0'[\tilde{y}\tilde{j}/\tilde{x}\tilde{i}] \supset C'[\tilde{y}/\tilde{x}])\ )}{[C]\, M :_u [C']}$$

In $[\mathit{Conseq\text{-}Kleymann}]$, we assume a basis Γ (for non-reference) and Δ (for reference) and set $\{\tilde{x}\} = \mathsf{dom}(\Gamma, \Delta) \cup \{u\}$, $\{\tilde{i}\} = \mathsf{fv}(C, C', C_0, C_0') \setminus \{\tilde{x}\}$. In addition, we require the $\tilde{j}$ (resp. $\tilde{y}$) to be fresh and of the same length as $\tilde{i}$ (resp. $\tilde{x}$).

B PCAP Generation Rules for PCFv

B.1 Generation Rules

The generation rules for PCAP are gven below. Since $\{A\}M :_u \{B\}$ can be equivalently written $\{\mathsf{T}\}M :_u \{A \supset B\}$, we use sequents of the form $\vdash^\star M :_u A$ without loss of generality. Unlike the rules The recursion rule $[rec]$ directly uses inductive predicate, $\underline{A}(u,n)$, defined by:

$$\underline{A}(u,0) \stackrel{\text{def}}{=} \forall x.\langle u \bullet x = z\rangle \mathsf{F}, \qquad \underline{A}(u,n+1) \stackrel{\text{def}}{=} \exists f.(A(u) \wedge \underline{A}(f,n)).$$

Note that $\forall x.\langle u \bullet x = z\rangle \mathsf{F}$ denotes u's divergence at all arguments. $\underline{A}(u,n)$ is the precise dual of $\overline{A}(u,n)$: it denotes all semantic points below, or equal to, the n-th unfolding of $\mu f.\lambda x.M$.

$$[var]\frac{-}{\vdash^\star x :_u u = x} \qquad [const]\frac{-}{\vdash^\star \mathsf{c} :_u u = \mathsf{c}}$$

$$[op]\frac{\vdash^\star M_i :_{m_i} A_i}{\vdash^\star \mathsf{op}(M_1..M_n) :_u \exists \tilde{m}.(u = \mathsf{op}(m_1..m_n) \wedge \bigwedge_i A_i)}$$

$$[abs]\frac{\vdash^\star M :_m A}{\vdash^\star \lambda x.M :_u \forall x.\langle u \bullet x = m\rangle A} \qquad [app]\frac{\vdash^\star M :_m A_1 \quad \vdash^\star N :_n A_2}{\vdash^\star MN :_u \exists mn.(m \bullet n = u \wedge A_1 \wedge A_2)}$$

$$[\mathit{if}]\frac{\vdash^\star M :_m A \quad \vdash^\star N_i :_u B_i \quad \mathsf{b}_1 = \mathsf{t},\ \mathsf{b}_2 = \mathsf{f}}{\vdash^\star \mathtt{if}\ M\ \mathtt{then}\ N_1\ \mathtt{else}\ N_2 :_u \bigvee_{i=1,2}(A[\mathsf{b_i}/m] \wedge B_i)} \qquad [rec]\frac{\vdash^\star \lambda x.M :_u A}{\vdash^\star \mu f.\lambda x.M :_u \downarrow (\exists n.\underline{A}(u,n), u)}$$

B.2 Proof Rules

The derivation of PCAPs leads to completeness of proof rules for partial correctness. In this note we only present the proof rule for recursion, which is the only rule that is non-trivially different from the corresponding rules for total correctness.

$$[Rec]\frac{\{A(y)\}\lambda y.M :_u \{A(u)^{-y}\} \quad \mathsf{admissible}(A(u))}{\{\mathsf{T}\}\mu x.\lambda y.M :_u \{A(u)\}}$$

Above $\mathsf{admissible}(A(u))$ indicates the following two conditions hold for each ξ (to our knowledge, these conditions are first noted by Plotkin and Stirling in [28], in the context of CPOs). Below assume u is typed as $\alpha \Rightarrow \beta$.

1. $\xi \cdot u : \mu f.\lambda x.fx \models A(u)$.
2. For each $\lambda x.M$, define W_n by: (1) $W_0 \stackrel{\text{def}}{=} \mu f.\lambda x.fx$ and (2) $W_{n+1} = (\lambda x.M)[W_n/f]$. Then if $\xi \cdot u : W_n \models A(u)$ holds for each n, then $\xi \cdot u : \mu f.\lambda x.M \models A(u)$ also holds.

The use of these two conditions in syntactic reasoning will be discussed elsewhere.

C Remaining Proofs

C.1 Remaining Cases of the Proof of Theorem 4

We fist list the lemmas used in the proof of Theorem 4 (proofs are omitted for the standard results).

Lemma 16. *1. If $x \notin \mathsf{fv}(e)$, then $\xi[\![e]\!] = (\xi \cdot x : V)[\![e]\!]$.*
2. If $x \notin \mathsf{fv}(A)$ then: $\xi \cdot x : V \models A$ iff $\xi \models A$.

Proof. By straightforward inductions on e/A.

Lemma 17. *1. If $M_i \lesssim N_i$ for each $0 \leq i \leq m-1$, then also $\mathsf{op}(\tilde{M}) \lesssim \mathsf{op}(\tilde{N})$ for each op. Similarly $M_i \lesssim N_i$ ($i = 1,2$) implies also $M_1 M_2 \lesssim N_1 N_2$.*
2. If $M \lesssim N$ and $M \Downarrow V$ then some W exists with $N \Downarrow W$.

Proof. (1) is by congruency. (2) is by the reduction (hence evaluation) being a subset of $\cong$, hence of $\lesssim$. □

Lemma 18. *Let σ be an injective renaming. Then: $\xi \models A$ iff $\xi\sigma \models A\sigma$.*

Proof. Standard, formally by induction on A. □

Lemma 19. *Assume that $x \notin \mathsf{dom}(\xi)$. Then $M\xi[V/x] = M(\xi \cdot x : V)$.*

Proof. Immediate, recalling $M\xi$ is the result of substitution induced by ξ. □

Lemma 20. *If $M \longrightarrow N$ and $N \Downarrow V$ then $M \Downarrow V$.*

Proof. Since in PCFv, $M \longrightarrow N_{1,2}$ implies N_1 and N_2 are alpha-convertible. □

Lemma 21. *1. If $L \longrightarrow M$ and $M \lesssim N$ then also $L \lesssim N$.*
2. If $K \cong L \lesssim M \cong N$ then $K \lesssim N$.

Proof. (1) is by $\longrightarrow$ being a subset of $\lesssim$. (2) is direct from the definition of the precongruence and the congruence, i.e. $\cong = \lesssim \cap \lesssim^{-1}$. □

Lemma 22. *Assume (A,B) satisfies (closure-2) w.r.t. $\Gamma \vdash M : \alpha$ at u and ξ is a model typable under Γ. Then: $\xi \models B[\mathsf{T}/x]$ iff $\xi \not\models B[\mathsf{F}/x]$.*

Proof. Since if the assumption holds, then B under ξ uniquely defines x by (closure-2), noting $\lesssim$ becomes $\cong$ for atomic types. □

We now give the remaining cases of Theorem 4. For [*var, const*], the proof is trivial. For [*op*] we consider the case of binary addition, i.e. we show that

$$[op]\frac{\vdash^{\star} [A_1]\, M_1 :_{m_1} [B_1] \qquad \vdash^{\star} [A_2]\, M_2 :_{m_2} [B_2]}{\vdash^{\star} [A_1 \wedge A_2]\, M_1 + M_2 :_u [\exists m_1 m_2.(u = m_1 + m_2 \wedge\ B_1 \wedge B_2)]}$$

For soundness, assume $\xi \models A_1 \wedge A_2$. By (IH):

- There is V_1: $(M_1\xi \Downarrow V_1\ \wedge\ \xi \cdot m_1 : V_1 \models B_1)$.
- There is V_2: $(M_2\xi \Downarrow V_2\ \wedge\ \xi \cdot m_2 : V_2 \models B_2)$.

Let V be the unique value of appropriate type such that $V \Downarrow V_1 + V_2$. Hence: $(M_1 + M_2)\xi \Downarrow V$. Define $\xi' \stackrel{\text{def}}{=} \xi \cdot m_1 : V_1 \cdot m_2 : V_2$. Then

$$\begin{array}{lll}
\xi \cdot m_1 : V_1 \models B_1, \xi \cdot m_2 : V_2 \models B_2 & \Rightarrow & \xi' \models B_1 \wedge B_2 \\
& \Rightarrow & \xi' \models B_1 \wedge B_2 \\
& \Rightarrow & \xi' \cdot u : V \models u = m_1 + m_2 \wedge B_1 \wedge B_2 \\
& \Rightarrow & \xi \cdot u : V \models \exists m_1 m_2.(u = m_1 + m_2 \wedge B_1 \wedge B_2)
\end{array}$$

For the (MTC) we get the result straightforwardly from the (IH).

$$\begin{array}{rll} (M_1+M_2)\xi\Downarrow & \text{iff} & M_1\xi\Downarrow \wedge M_2\xi\Downarrow \\ & \text{iff} & \xi\models A_1 \text{ and } \xi\models A_2 \qquad \text{(IH)} \\ & \text{iff} & \xi\models A_1\wedge A_2 \end{array}$$

Finally, (closure-2): assuming that $\xi\models A_1\wedge A_2$ and $\xi\cdot u:V\models\exists m_1m_2.(u=m_1+m_2\wedge B_1\wedge B_2)$. Hence for some W_1,W_2:

$$\xi\cdot u:V\cdot m_1:W_1\cdot m_2:W_2\models\exists u=m_1+m_2\wedge B_1\wedge B_2.$$

By the definition of the satisfaction relation that means

$$V\cong W_1+W_2.$$

Now $u,m_2\notin\mathsf{fv}(B_1)$, hence, by Lemma 16:

$$\xi\cdot m_1:W_1\models B_1 \qquad \text{hence by (IH):} \qquad M_1\xi\lesssim W_1.$$

Similarly we obtain that $M_2\xi\lesssim W_2$, which, taken together gives

$$(M_1+M_2)\xi \quad = \quad M_1\xi+M_2\xi \quad \lesssim \quad W_1+W_2 \quad \cong \quad V,$$

using Lemma 17.

Next we treat [*app*]. For soundness, clearly:

1. $\xi\models A_1$, hence by (IH): $M_1\xi\Downarrow V_1$ and $\xi\cdot m_1:V_1\models B_1$.
2. $\xi\models A_2$, hence by (IH): $M_2\xi\Downarrow V_2$ and $\xi\cdot m_2:V_2\models B_2$.
3. $\xi\models\forall m_1m_2.(B_1\wedge B_2\wedge\exists z.m_1\bullet m_2=z)$

Using Lemma 16, we weaken (1, 2) to

$$\xi\cdot m_1:V_1\cdot m_2:V_2\models B_1\wedge B_2. \tag{C.1}$$

Hence for some W:

$$\xi\cdot m_1:V_1\cdot m_2:V_2\cdot z:W\models m_1\bullet m_2=z \tag{C.2}$$

This implies $V_1V_2\Downarrow W$, hence:

$$(M_1M_2)\xi=(M_1\xi)(M_2\xi)\longrightarrow\cdots\longrightarrow V_1V_2\Downarrow W.$$

Renaming (C.2) gives

$$\xi\cdot m_1:V_1\cdot m_2:V_2\cdot u:W\models m_1\bullet m_2=u$$

using Lemma 18. Using weakening (Lemma 16) with (C.1) we obtain

$$\xi\cdot m_1:V_1\cdot m_2:V_2\cdot u:W\models m_1\bullet m_2=u\wedge B_1\wedge B_2,$$

hence

$$\xi\cdot u:W\models\exists m_1m_2.(m_1\bullet m_2=u\wedge B_1\wedge B_2).$$

For (MTC), first for ($\Rightarrow$) assume $MN\xi \Downarrow V$, hence $M \Downarrow V_m$ and $N \Downarrow W_n$ for some values V_m, V_n. The (IH) yields

$$\xi \models A_1 \wedge A_2. \tag{C.3}$$

Choose values W_m, W_n and assume

$$\xi \cdot m : W_m \cdot W_n \models B_1 \wedge B_2.$$

Then $M\xi \lesssim W_m$ and $N\xi \lesssim W_n$ by (closure-2), using weakening. Lemma 17 now allows to conclude to

$$MN \lesssim W_m W_n.$$

This with Lemma 17 and (C.3) assures us of the existence of a value V' such that $W_m W_n \Downarrow V'$. This means that $\xi \cdot m : W_m \cdot W_n \cdot z : V' \models m \bullet n = z$, hence in fact:

$$\xi \models A_1 \wedge A_2 \wedge \forall mn.(m \bullet n = z \wedge B_1 \wedge B_2)$$

as required. Conversely ($\Leftarrow$) is immediate from soundness.

We conclude this case by showing that (closure-2) is preserved by application. Assume

1. $\xi \models A_1 \wedge A_2 \wedge \forall mn.(m \bullet n = z \wedge B_1 \wedge B_2)$.
2. $\xi \cdot u : V \cdot m : V_m \cdot n : V_n \models m \bullet n = u \wedge B_1 \wedge B_2$.

From (2) we immediately get $\xi \cdot m : V_m \models B_1$, hence $M\xi \lesssim V_m$ using the (IH) and Lemma 16. Similarly we obtain: $N\xi \lesssim V_n$. In addition clearly also $V_m V_n \Downarrow V' \cong V$. Hence

$$(MN)\xi \quad \lesssim \quad V_m V_n \quad \cong \quad V$$

by Lemma 17.

Soundness of [*abs*] is straightforward. Choose V and assume

$$\underbrace{\xi \cdot u : \lambda x.M \cdot x : V}_{\xi'} \models A.$$

Then by (IH): $M\xi' \Downarrow$, in fact, as $u \notin \mathsf{fv}(M)$ even: $M(\xi \cdot x : V) \Downarrow$. In addition: $\xi \cdot m : V \models B$. Hence all that is left to show for the soundness of this rule is $(\lambda x.M)\xi' V \Downarrow$, which is immediate because $(\lambda x.M)\xi' V \longrightarrow (M\xi \cdot u : \lambda x.M)[V/x] = M\xi' \Downarrow$ by Lemma 19 and 20.

The (MTC) for this rule is trivial. Finally (closure-2). Assume that

$$\xi \cdot u : U \models \forall x.(B \Rightarrow \exists m.(u \bullet x = m \wedge B)). \tag{C.4}$$

Choose arbitrary W of appropriate type with

$$\underbrace{\xi \cdot u : V \cdot x : W}_{\xi'} \models A.$$

Then (C.4) implies that some W', W'' exists with

$$VW \Downarrow W' \cong W'' \qquad \text{and} \qquad \xi' \cdot m : W'' \models B$$

By (IH) we know that $M(\xi \cdot x : W'') \lesssim W'' \cong VW$. But $(\lambda x.M)\xi W \longrightarrow M\xi[V/x] = M(\xi.cdotx : W)$. Now Lemma 21 delivers

$$(\lambda x.M)\xi W \lesssim VW.$$

As W was arbitrary, in fact

$$(\lambda x.M)\xi \lesssim V$$

as required.

The final rule is [*if*]. We omit the straightforward soundness proof. For the (MTC) assume wlog. that

$$M\xi \Downarrow \mathsf{T} \qquad \text{and} \qquad N_1\xi \Downarrow V.$$

By the (IH) then $\xi \models A$, hence $\xi \cdot m : \mathsf{T} \models B$. This implies

$$\xi \models B[\mathsf{T}/m]$$

Now Lemma 22 entails that also

$$\xi \not\models B[\mathsf{F}/m]$$

Hence clearly

$$\xi \models A \wedge (B[\mathsf{T}/m]) \wedge (B[\mathsf{F}/m]).$$

For (closure-2), let $\xi \models A \wedge (B[\mathsf{T}/m] \Rightarrow A_1) \wedge (B[\mathsf{F}/m] \Rightarrow A_2)$ and wlog. $\xi \cdot u : V \models B[\mathsf{T}/m]$. Now $\xi \models A \wedge B[\mathsf{T}/m]$ implies wlog. that $M \Downarrow \mathsf{T}$ and $\xi \cdot m : W \models B$. Hence $\xi \models A_1$. This together implies $N_1\xi \lesssim V$, using the (IH). Now $(\texttt{if}\ M\ \texttt{then}\ N_1\ \texttt{else}\ N_2)\xi \longrightarrow N_1\xi$ Hence by Lemma 20

$$(\texttt{if}\ M\ \texttt{then}\ N_1\ \texttt{else}\ N_2)\xi \quad \lesssim \quad V$$

as required. □

C.2 Proof of Claim for Corollary 7 (2)

By Theorem 4, we already know for each M there exists a TCAP (A,B) of M at u. We show $\vdash \{A\}M :_u \{\uparrow(B,u)\}$ which suffices. This is shown by establishing:

Proposition 23. $\vdash^\star [A]M :_u [B]$ *implies* $\vdash [A]M :_u [\uparrow(B,u)]$.

The proof of Proposition 23 is by rule induction on the generation rules in Figure 1. Noting $B \supset \uparrow(B,u)$, the verification is mechanical for all proof rules in Appendix A.1 (a verification of the corresponding result for an alternative recursion rule is given at the end).

C.3 Proof of Proposition 5

By induction on the structure of the program, for some constant c, we show the sum of the sizes of two formulae, A and B, of M, is less than $c \times m \times 2^n$, where m is the size of M and n is the number of applications/conditionals in M. As a non-trivial case, consider [*app*]:

$$[app]\frac{\vdash^\star [A_1]M_1 :_m [B_1] \qquad \vdash^\star [A_2]M_2 :_n [B_2]}{\vdash^\star [A_1 \wedge A_2 \wedge \forall mn.(B_1 \wedge B_2 \supset \exists z.m \bullet n = z)]M_1M_2 :_u [\exists mn.(m \bullet n = u \wedge B_1 \wedge B_2)]}$$

Write $|A|$ for the size of A. Then we note the size of the new TCAP is:

$$|A_1| + 2 \times |B_1| + |A_2| + 2 \times |B_2| + 12$$

Let the size of M_1 (resp. M_2) be m_1 (resp. m_2). Then by induction, for $i = 1, 2$:

$$|A_i| + |B_i| \leq c \times m_i \times 2^{n_i}$$

So that we have, for a sufficiently large n:

$$2 \times (|A_1| + |B_1| + |A_2| + |B_2|) + 12 \leq 2 \times c \times \sum_{i=1,2} (m_i \times 2^{n_1+n_2} + 12 \leq c \times (m_1 + m_2) \times 2^{n_1+n_2+1}$$

hence as required. Other cases are simpler.

C.4 Proof of Lemma 10 (1)

To show for each $\Gamma;\Delta \vdash M : \alpha$ we have $\Gamma;\Delta \vdash M \cong [\![M]\!]$, it suffices to prove:

$$\langle\!\langle M \rangle\!\rangle_x[N] \cong \mathtt{let}\ x = M\ \mathtt{in}\ N. \tag{C.5}$$

This suffices since $\mathtt{let}\ x = M\ \mathtt{in}\ x \cong M$ by β_v-equality. We use the structural induction on M. The base case is obvious. Here we only show the case of application.

$$\begin{aligned}
\langle\!\langle M_1 M_2 \rangle\!\rangle_x[N] &\stackrel{\text{def}}{=} \langle\!\langle M_1 \rangle\!\rangle_{m_1}[\langle\!\langle M_2 \rangle\!\rangle_{m_2}[\mathtt{let}\ y = m_1 m_2\ \mathtt{in}\ N]] \\
&\cong \mathtt{let}\ m_1 = M_1\ \mathtt{in}\ \mathtt{let}\ m_2 = M_2\ \mathtt{in}\ \mathtt{let}\ y = m_1 m_2\ \mathtt{in}\ N \\
&\cong \mathtt{let}\ y = M_1 M_2\ \mathtt{in}\ N
\end{aligned}$$

Other cases are similar.

C.5 Proof of Lemma 10 (2)

Thoughout the following proof, we consider, since it suffices while giving simpler arguments, the proof system in Appendix A.1 replacing its consequence rule with the following standard (weaker) consequence rule:

$$[\mathit{Consequence}]\, \frac{C \supset C_0 \quad [C_0]M :_u [C_0'] \quad C_0' \supset C'}{[C]\,M :_u [C']}$$

We still write $\vdash$ for the provability in this weaker system. We also consider a proof system which extends this system with the let rule (reproduced below).

$$[\text{let}]\, \frac{[C]M :_x [C_0] \quad [C_0]N :_u [C']}{[C]\mathtt{let}\ x = M\ \mathtt{in}\ N :_u [C']}$$

The provability in this extended system (still with the weaker consequence rule) is written $\vdash^{\mathsf{let}}$. Our purpose is to show:

$$\vdash^{\mathsf{let}} [C]\,[\![M]\!] :_u [C'] \text{ implies} \vdash [C]\,M :_u [C'].$$

As a preparation, we prove a couple of lemmas. The first one is about the let rule.

Lemma 24. (let-rule lemma)

1. $\vdash^{\mathsf{let}} [C]\mathtt{let}\, x = M \,\mathtt{in}\, N :_u [C']$ *implies there exists* C_0 *such that* $\vdash^{\mathsf{let}} [C_0]N :_u [C']$.
2. *Suppose* M *does not contain let's. Then* $\vdash^{\mathsf{let}} [C]\mathtt{let}\ x = M\ \mathtt{in}\ x :_u [C']$ *implies* $\vdash [C]M :_u [C']$.

Proof. For (1), since $\vdash^{\mathsf{let}}$ is compositional, we can safely set the rule is derived from [Let] followed by [Consequence]. Assume the result of applying [Let] is $\vdash^{\mathsf{let}} [G]\mathtt{let}\, x = M \,\mathtt{in}\, N :_u [G']$ so that $C \supset G$ and $G' \supset C'$. Then there is G_0 such that $\vdash^{\mathsf{let}} [G]M :_x [G_0]$ and $\vdash^{\mathsf{let}} [G_0]N :_u [G']$. By applying [Consequence] we are done. For (2), if $\vdash^{\mathsf{let}} [C]\mathtt{let}\ x = M\ \mathtt{in}\ x :_u [C']$ then $\vdash^{\mathsf{let}} [C]M :_x [C_0]$ and $\vdash^{\mathsf{let}} [C_0]x :_u [C']$. Since no let occurs in either M nor x, we know $\vdash [C]M :_x [C_0]$ and $\vdash [C_0]x :_u [C']$. Since the strongest postcondition of x with the precondition C_0 is $C_0[u/x]$, we know $C_0[u/x] \supset C'$. By [Consequence], we have $\vdash [C]M :_x [C'[x/u]]$, that is $\vdash [C]M :_u [C']$, as required. $\square$

Using the lemma above, we next prove:

Lemma 25. $\vdash^{\mathsf{let}} [C]\langle\!\langle M\rangle\!\rangle_x[N] :_u [C']$ *implies* $\vdash^{\mathsf{let}} [C]\mathtt{let}\, x = M \,\mathtt{in}\, N :_u [C']$.

Proof. By induction on M. We show two cases, constant and application. Other cases are similar. For constant:

$$\vdash^{\mathsf{let}} [C]\langle\!\langle c\rangle\!\rangle_x[N] :_u [C'] \Leftrightarrow \vdash^{\mathsf{let}} [C]\mathtt{let}\, x = c \,\mathtt{in}\, N :_u [C']$$

hence done. For application:

$$\begin{array}{lll}
\vdash^{\mathsf{let}} [C]\langle\!\langle M_1M_2\rangle\!\rangle_x[N] :_u [C'] & & \\
\quad\Leftrightarrow\ \vdash^{\mathsf{let}} [C]\langle\!\langle M_1\rangle\!\rangle_{m_1}[\langle\!\langle M_2\rangle\!\rangle_{m_2}[\mathtt{let}\, x = m_1m_2 \,\mathtt{in}\, N]] :_u [C'] & & \text{(by def.)} \\
\quad\Rightarrow\ \vdash^{\mathsf{let}} [C]\mathtt{let}\, m_1 = M_1 \,\mathtt{in}\, \mathtt{let}\, m_2 = M_2 \,\mathtt{in}\, \mathtt{let}\, x = m_1m_2 \,\mathtt{in}\, N :_u [C'] & & \text{(IH)} \\
\quad\Rightarrow\ \vdash^{\mathsf{let}} [C]\mathtt{let}\, x = M_1M_2 \,\mathtt{in}\, N :_u [C'] & & \text{(Lem.24-1)}
\end{array}$$

In the final step, we consecutively used Lemma 24-1, noting there are C, C_0 and C_1 for intermediate steps, that is:

(I-1) $\vdash [C]M_1 :_{m_1} [C_0]$.
(I-2) $\vdash [C_0]M_2 :_{m_2} [C_1]$.
(I-3) $\vdash [C_1]m_1m_2 :_x [C_2]$.

as well as $\vdash [C_2]N :_u [C']$. To infer (I-3), we may safely assume the rule is derived by [app] (it is possible (I-3) is the result of applying [Consequence]: if so we have the same provable judgements except for changes in the corresponding assertions, with precisely the same subsequent reasoning). So we need to have:

(I-3a) $\vdash [C_1]m_1 :_{z_1} [C_a]$.
(I-3b) $\vdash [C_a]m_2 :_{z_2} [C_b \wedge [C_b]z_1 \bullet z_2 = u[C_2]]$.

Since the corresponding strongest judgements are:

(I-3a') $\vdash [C_1]m_1 :_{z_1} [C_1[z_1/m_1]]$.
(I-3b') $\vdash [C_1[z_1/m_1]]m_2 :_{z_2} [C_1[z_1z_2/m_1m_2]]$

Hence we need to have

$$C_1 \supset [C_1]m_1 \bullet m_2 = u[C_2]. \tag{C.6}$$

Combining (I-2) and (C.6) we obtain:

$$\vdash [C_0]\, M_2 :_{m_2} [C_1 \wedge [C_1]m_1 \bullet m_2 = u[C_2]] \tag{C.7}$$

By (I-1), (C.7) and $\vdash [C_2]\, N :_u [C']$, we obtain $\vdash [C]\, \mathtt{let}\ x = M_1M_2\ \mathtt{in}\ N :_u [C']$, as required. □

We can now conclude the proof of Lemma 10 (2). Assume below M does not contain *let*'s.

$$\begin{array}{rlll} \vdash^{\mathsf{let}} [C]\, [\![M]\!] :_u [C'] & \Leftrightarrow & \vdash^{\mathsf{let}} [C]\, \langle\!\langle M \rangle\!\rangle_x [x] :_u [C'] & \text{(def.)} \\ & \Rightarrow & \vdash^{\mathsf{let}} [C]\, \mathtt{let}\, x = M\ \mathtt{in}\, x :_u [C'] & \text{(lem.25)} \\ & \Rightarrow & \vdash [C]\, M :_u [C'] & \text{(lem.24-2)} \end{array}$$

hence as required.

C.6 Completeness of Alternative Recursion Rule

Assuming the logical language allows extension with inductively defined predicates, we show the following proof rule for recursion is relatively complete (i.e. the proof system in Appendix A.1 is relatively complete after replacing the recursion rule there with the following one):

$$[Rec]\frac{[A^{\text{-}xi} \wedge \forall j \precneqq i.B(j)[x/u]]\, \lambda y.M :_u [B(i)^{\text{-}x}]}{[A]\, \mu x.\lambda y.M :_u [\forall i.B(i)]}$$

We work with the logic for PCFv, though the proof extends to the imperative PCFv.

To show completeness of this rule, we consider an alternative TCAP generation rule which corresponds to [Rec] above.

$$[rec\text{-}fix]\frac{\vdash^{\star} [\mathsf{T}]\, \lambda x.M :_u [A]}{\vdash^{\star} [\mathsf{T}]\, \mu f.\lambda x.M :_u [\forall n.\overline{A}(u,n)]}$$

where $\overline{A}(u,n)$ is given by the following induction:

$$\overline{A}(u,0) \stackrel{\text{def}}{=} \mathsf{T}, \qquad \overline{A}(u,n+1) \stackrel{\text{def}}{=} \exists f.(A \wedge \overline{A}(f,n)).$$

We first show:

Proposition 26. *If (T,A) satisfies (soundness), (MTC) and (closure) w.r.t. $\lambda x.M$, then $(\mathsf{T},\overline{A}(u,n))$ does so w.r.t. W_n for each n. Further $\models [\mathsf{T}]\, W_n :_u [\overline{A}(u,m)]$ for each $m \leq n$.*

Proof. We argue by induction on n. The base case is immediate. For induction, assume (soundness) and (closure) hold for $(\mathsf{T},\overline{A}(u,n))$ w.r.t. W_n. We show the same holds for $(\mathsf{T},\overline{A}(u,n+1))$ w.r.t. W_{n+1}.

(soundness) For each ξ on $\mathsf{fv}(\lambda x.M)$, and for each $m \leq n$:

$$\begin{array}{lll} \text{(assumption, IH)} & \Rightarrow & \xi \cdot f : W_m\xi \cdot u : (\lambda x.M)(\xi \cdot f : W_m\xi) \models A,\ \ \xi \cdot f : W_m\xi \models \overline{A}(f,m) \\ & \Rightarrow & \xi \cdot u : (\lambda x.M)(\xi \cdot f : W_m\xi) \models \exists f.(A\ \wedge\ \overline{A}(f,m)) \end{array}$$

(closure-2) We show $W_{n+1}\xi$ is the least among those satisfying $\overline{A}(u,n+1)$ under ξ.

$$\begin{array}{lll}
\xi\cdot u\!:\!V \models \overline{A}(u,n+1) & \Rightarrow & \xi\cdot u\!:\!V\cdot f\!:\!W \models A\wedge\overline{A}(f,n) \quad \text{for some } W \\
 & \Rightarrow & W_n\xi \lesssim W \qquad\qquad\qquad\qquad\qquad\qquad (*) \\
 & \Rightarrow & (\lambda x.M)(\xi\cdot f\!:\!W_n\xi) \lesssim (\lambda x.M)(\xi\cdot f\!:\!W) \lesssim V \quad (**).
\end{array}$$

Above $(*)$ is by $\xi, f\!:\!W \models \overline{A}(f,n)$ and by (closure) of W_n w.r.t. $\overline{A}(f,n)$. For $(**)$, the first inequation is by (closure) of $\lambda x.M$ w.r.t. A, the second is by $(*)$. $\square$

Corollary 27. *If* $(\top,A)$ *satisfies (soundness), (MTC) and (closure-2) w.r.t.* $\lambda x.M$*, then* $(\top,\forall n.\overline{A}(u,n))$ *does so w.r.t.* $\mu f.\lambda x.M$*.*

Proof. We use the standard n-th unfolding W_n of recursion [27], which approximates $\mu f.\lambda x.M$ as n tends to infinity.

$$W_0 \stackrel{\text{def}}{=} \mu f.\lambda x.(fx), \qquad W_{n+1} \stackrel{\text{def}}{=} \lambda x.M[W_n/f]$$

(soundness) is direct from Claim A, while (MTC) is vacuous. For (closure-2), using the syntactic notion of continuity [27]:

$$\begin{array}{llll}
\xi, u\!:\!W \models \forall n.\overline{A}(u,n) & \Rightarrow & \forall n.(\xi,u\!:\!W \models \overline{A}(u,n)) & \text{(def)} \\
 & \Rightarrow & \forall n.(W_n\xi \lesssim W) & \text{(IH (closure))} \\
 & \Rightarrow & \mu f.\lambda x.M\xi \lesssim W & \text{(continuity).}
\end{array}$$

as required. $\square$

Lemma 28. [rec-fix] *is admissible w.r.t.the proof system in Appendix A.1.*

Proof. We show [*rec-fix*] is derivable by combining [*Rec*] (which is the proof rule for recursion in Appendix A.1, *not* the generation rule [*rec*]) and the consequence rule. This is the same thing as:

$$\begin{array}{lll}
A(u) \supset (\overline{A}(f,n) \supset (A(u)\wedge\overline{A}(f,n))) & \Rightarrow & \overline{A}(f,n) \supset (\exists f.(A(u)\wedge\overline{A}(f,n))) \\
 & \Rightarrow & \overline{A}(f,n) \supset \overline{A}(u,n+1)
\end{array}$$

Thus, if the premise $\{\top\}M :_u \{A(u)\}$ (of [*rec-fix*]) is provable, then we can apply [*Rec*] and [*Conseq*] to obtain the conclusion of [*rec-fix*]. $\square$

By Corollary 27 and Lemma 28 we are done.