Logics for Imperative Higher-Order Functions with Aliasing and Local State: Thee Completeness Results

Martin Berger, Kohei Honda, and Nobuko Yoshida

Abstract. We establish three completeness properties (descriptive, relative and observational completeness) for the program logic for aliasing and higher-order functions, first studied in [1]. We then extend the results to the logic with local state in [6].

1 Introduction

In this note we establish three completeness properties of the logic for higher-order functions with aliasing [1] and its refinement for local state [6]. These three properties are existence of characteristic (or most general) formulae for each typed program (descriptive completeness), coincidence of validity-induced equivalence with the standard contextual equivalence (observational completeness) and derivability of all valid judgements using the proof rules (relative completeness).

The key idea used for the completeness results is simplification of the original evaluation formulae, which has the form $[C]x \bullet y = z[C']$ saying that in any hypothetical state which satisfies C, if we apply x to y, then the return value (named z) and the resulting state satisfy C'. Instead we now use a somewhat lopsided formula of the form:

$$x \bullet y = z\ [C]$$

which says that in the *current* state, if we apply x to y, then the return value (named z) and the resulting state together satisfy C. Since we can simulate arbitrary state change from the current state by this one-sided evaluation formulae, we can recover the original formulae. Further the capability to be able to represent the application in the current state allows us to treat those cases which have been formerly hard to treat.

In the rest of the note, Section 2 establishes three completeness properties for the logic with aliasing. Section 3 prove the same results for the logic with local state.

2 Completeness of Logic for Aliasing

2.1 Programming Language

The programming language we shall use is call-by-value PCF with unit, sums and products, augmented with imperative variables. Assuming an infinite set of *variables* ($x, y, z, \ldots$, also called *names*), the syntax of programs is standard [5] and given by the following grammar.

$$\begin{array}{ll} \text{(values)} & V, W ::= \mathtt{c} \mid x \mid \lambda x^{\alpha}.M \mid \mu f^{\alpha \Rightarrow \beta}.\lambda y^{\alpha}.M \\ \text{(program)} & M, N ::= V \mid MN \mid M := N \mid\ !M \mid \mathtt{op}(\tilde{M}) \mid \mathtt{if}\ M\ \mathtt{then}\ M_1\ \mathtt{else}\ M_2 \end{array}$$

Abstraction and recursion are annotated by types. Constants ($\mathtt{c}, \mathtt{c}', \ldots$) include unit $()$, natural numbers $\mathtt{n}$, booleans $\mathtt{b}$ (either true $\mathtt{t}$ or false $\mathtt{f}$) and locations ($l, l', \ldots$). $\mathtt{op}(\tilde{M})$ (where $\tilde{M}$ is a vector of programs) is a standard n-ary first-order operation such as $+$, $-$, $\times$, $=$ (equality of two numbers or that of reference names), $\neg$ (negation), $\wedge$ and $\vee$. $!M$ dereferences M while $M := N$ first evaluates M and obtains a location (say l), evaluates N and obtains a value (say V), and assigns V to l. All these constructs are standard, cf. [2, 5]. The notions of binding and α-convertibility are also conventional; $\mathsf{fv}(M)$ and $\mathsf{fl}(M)$ denotes the sets of free variables and locations in M, respectively. We use standard abbreviations such as $\lambda().M$, $M;N$ and $\mathtt{let}\ x = M\ \mathtt{in}\ N$.

Let $\mathrm{X}, \mathrm{Y}, \ldots$ range over an infinite set of type variables. Types are ranged over by $\alpha, \beta, \ldots$ and are given by the following grammar.

$$\alpha, \beta ::= \mathsf{Unit} \mid \mathsf{Bool} \mid \mathsf{Nat} \mid \alpha \Rightarrow \beta \mid \mathsf{Ref}(\alpha) \mid \mathrm{X} \mid \mu \mathrm{X}.\alpha$$

We call types of the form $\mathsf{Ref}(\alpha)$ *reference types*. All others are *value types*. A type is *closed* if it does not contain free occurrences of type variables. We write $\mathsf{ftv}(\alpha)$ to mean the set of α's free type variables.

A *basis* is a finite map from names and locations to closed types. $\Gamma, \Gamma' \ldots$ range over bases and $\mathsf{dom}(\Gamma)$ denotes the domain of Γ, while $\mathsf{cod}(\Gamma)$ denotes the range of Γ. We let $\Delta, \ldots$ range over bases whose codomains are reference types and write $\Gamma; \Delta$ for a basis where Γ maps names to value types, always assuming $\mathsf{dom}(\Gamma) \cap \mathsf{dom}(\Delta) = \emptyset$. We take the equi-isomorphic approach for recursive types, and will identify types and programs up to equi-isomorphism. The typing rules are standard [5] using sequents $\Gamma \vdash M : \alpha$, which say that M has type α under basis Γ. We often write $M^{\Gamma;\alpha}$ for $\Gamma \vdash M : \alpha$.

The dynamics of the language is given by standard call-by-value reductions using a store [2, 5], where a *store* ($\sigma, \sigma', \ldots$) is a finite map from locations to closed values. We write $\mathsf{dom}(\sigma)$ for the domain of σ. A *configuration* is a pair of a closed program and a store. Then *reduction* is a binary relation over configurations, written $(M, \sigma) \longrightarrow (M', \sigma')$, generated by the rules (Appendix A in [1]).

The call-by-value evaluation is written $(M, \sigma) \Downarrow (V, \sigma')$. If (M, σ) diverges, we write $(M, \sigma) \Uparrow$. We use the standard contextual congruence $\cong$ and the precongruence $\lesssim$.

Definition 1. (observational congruence) Assume that $\Gamma; \Delta \vdash M_{1,2} : \alpha$. We write $\Gamma; \Delta \vdash (M_1, \sigma_1) \lesssim (M_2, \sigma_2)$ if, for each closing typed context $C[\cdot]$ such that $\Delta \vdash C[M_i] : \mathsf{Unit}$ for $i = 1, 2$: if $(C[M_1],\ \sigma_1) \Downarrow$, then $(C[M_2],\ \sigma_2) \Downarrow$. We often omit σ when it is empty. We set $\cong = \lesssim \cup \lesssim^{-1}$.

As we have done [3, 4], we use the `let`-*form* for deriving TCAPs. The grammar of the `let`-form which we use for TCAP-derivations is:

$$\begin{array}{ll} \text{(values)} & U ::= \mathtt{c} \mid x \mid \lambda x^{\alpha}.L \mid \mu f^{\alpha \Rightarrow \beta}.\lambda y^{\alpha}.L \\ \text{(program)} & L ::= U \mid \mathtt{let}\ x = f y\ \mathtt{in}\ L \mid \mathtt{let}\ x = \mathtt{op}(y_1, .., y_n)\ \mathtt{in}\ L \\ & \qquad \mid \mathtt{let}\ x = !y\ \mathtt{in}\ L \mid x := y; L \mid \mathtt{if}\ x\ \mathtt{then}\ L_1\ \mathtt{else}\ L_2 \mid \mathtt{let}\ x = U\ \mathtt{in}\ L \end{array}$$

It is straightforward to define a translation $[\![\cdot]\!]$ which converts all programs of imperative PCFv to the let forms so that $[\![M]\!] \cong M$ and $\vdash [C]\,[\![M]\!] :_u [C']$ iff $\vdash [C]\,M :_u [C']$. Thus it suffices to consider deriving CAPs for `let`-programs.

2.2 Logic

The assertion language in [1] has the following expressions and formulae.

$$e ::= x^{\alpha} \mid \mathsf{c} \mid \mathsf{op}(\tilde{e}) \mid !e$$
$$C ::= e = e' \mid \neg C \mid C \star C' \mid Qx^{\alpha}.C \mid Q\mathrm{X}.C \mid e \bullet e' = x[C] \mid [!x]^{\circ} C$$

Here $\star \in \{\wedge, \vee, \supset\}$, and $Q \in \{\forall, \exists\}$. $[!x]^{\circ}C$ is the same content quantification as we used in [1]. $\langle x \rangle^{\circ} C$ is its dual.

We write $e_1 \bullet e_2[C]$ for $e \bullet e' = x[C]$ with x fresh; and $e_1 \bullet e_2 \Downarrow$ for $e \bullet e'[\mathsf{T}]$. We set T as $1 = 1$ and F as its negation. A sequent $[C]M :_u [C']$ has a fixed basis, usually left implicit. A judgement for total correctness is written $\models [C]M :_u [C']$ (for validity) and $\vdash [C]M :_u [C']$ (for provability).

$e_1 \bullet e_2 = x[C]$ means that, in the present state, if we apply a procedure e_1 to its argument e_2, then the return value (named x) and the resulting state satisfy C. It is a simplification of the original evaluation formula in [1, 4], but at the same time more expressive. Define:

$$\Box C \quad \stackrel{\text{def}}{\equiv} \quad \forall f^{\mathsf{Unit} \Rightarrow \mathsf{Unit}}.(f \bullet () \Downarrow \supset f \bullet ()[C]$$

Since such f can have an arbitrary effect on the store, $\Box C$ says that C holds now and in any possible future. Then the original evaluation formulae $[C]x \bullet y = z[C']$ can be defined as $\Box(C \supset x \bullet y = z[C'])$.

We can also define $f \bullet x = z[C]@\emptyset$, which indicates the evaluation has no effect as in [1], as follows:

$$f \bullet x = z[C]@\emptyset \quad \stackrel{\text{def}}{\equiv} \quad f \bullet x = z[C] \wedge \forall g.(g \bullet () \Downarrow \supset f \bullet x = z[g \bullet () \Downarrow])$$

Similarly we can define:

$$f \bullet x = z[C]@\tilde{w} \quad \stackrel{\text{def}}{\equiv} \quad f \bullet x = z[C] \wedge \forall g.(g \bullet () \Downarrow \supset f \bullet x = z[\langle !\tilde{w} \rangle^{\circ} g \bullet () \Downarrow])$$

which says that the write effect is restricted to $\tilde{w}$.

2.3 Model

In this section we present the model for the language with aliasing but without local store.

Definition 2. (models) A term is *closed* if it has not free variables. A *model of type* $\Theta = \Gamma; \Delta$, ranged over by $\mathcal{M}, \mathcal{M}', ...$, with $\mathsf{fv}(\Delta) \cup = \mathsf{ftv}(\Delta) = \emptyset$, is a tuple (ξ, σ) where:

- ξ, called *environment*, is a finite map from (1) $\mathsf{dom}(\Theta)$ to closed values such that, for each $x \in \mathsf{dom}(\Gamma)$, $\xi(x)$ is typed as $\Theta(x)$ under Δ, i.e. $\Delta \vdash \xi(x) : \Theta(x)$; and (2) from type variables to closed types.
- σ, called *store*, is a finite map from labels to closed values such that for each $l \in \mathsf{dom}(\sigma)$, if $\Delta(l)$ has type $\mathsf{Ref}(\alpha)$, then $\sigma(l)$ has type α under Δ, i.e. $\Delta \vdash \sigma(l) : \alpha$.

We continue with the semantics of formulae, omitting trivial cases and the semantics of expressions, except for noting that $[\![e]\!]_{\mathcal{M}}$ is the interpretation of e under $\mathcal{M}$. The following notation is useful later. Let $\mathcal{M} = (\xi, \sigma)$.

- Given $u \notin \mathsf{fv}(\mathcal{M})$, we write $\mathcal{M} \cdot u : V$, or often $(\xi \cdot u : V, \sigma)$, for a model that extends $\mathcal{M}$ by one entry with the value V.
- We sometimes write $M_{\mathcal{M}}$ for $M\xi$, assuming typability.
- If $l \in \mathsf{dom}(\sigma)$, $\mathcal{M} \cdot [l \mapsto V]$ is the model obtained from $\mathcal{M}$ by updating the store at l with V. Similarly, and assuming appropriate typing, $\mathcal{M}[x \mapsto V]$ means $\mathcal{M}[l \mapsto V]$, where the reference x is mapped to location l by $\mathcal{M}$.
- If u is fresh we write $\mathcal{M}[u : V]$ (resp. $\mathcal{M}[u : e]$) when adding the entry $V\xi$ (resp. $[\![e]\!]_{\mathcal{M}}$) to $\mathcal{M}$. We write $\mathcal{M}[u : M] \Downarrow \mathcal{M}'$ to state that $(M\xi, \sigma) \Downarrow (V, \sigma')$ and $(\xi \cdot u : V, \sigma') = \mathcal{M}'$; likewise, for $\mathcal{M}[u : e] \Downarrow \mathcal{M}'$. Relatedly, $\mathcal{M}[x \mapsto M] \Downarrow \mathcal{M}'$ stands for $(M\xi, \sigma) \Downarrow (V, \sigma')$ with $\mathcal{M}' = (\xi, \sigma'[\xi(x) \mapsto V])$. We also write $\mathcal{M}[x : V/x]$ for the model that is exactly like $\mathcal{M}$, except that x now denotes V.
- We generate $\mathcal{M} \rightsquigarrow \mathcal{M}'$ inductively by: (1) $\mathcal{M} \rightsquigarrow \mathcal{M}$; and (2) if $\mathcal{M} \rightsquigarrow \mathcal{M}_0$ and $\mathcal{M}_0[u : N] \Downarrow \mathcal{M}'$ then $\mathcal{M} \rightsquigarrow \mathcal{M}'$.
- We write $\mathcal{M}_1 \lesssim \mathcal{M}_2$, where $\mathcal{M}_i = (\tilde{u} : \tilde{V}_i, \sigma_i)$, if for all M of type Unit with $\mathsf{fv}(M) \subseteq \tilde{x}$: $(\mathtt{let}\ \tilde{x} = \tilde{V}_1\ \mathtt{in}\ M, \sigma_1) \Downarrow$ implies $(\mathtt{let}\ \tilde{x} = \tilde{V}_2\ \mathtt{in}\ M, \sigma_2) \Downarrow$.
- Symbols $i, j, \ldots$ range over auxiliary names. A, B denote *stateless* formulae: C is stateless when $\Box C \equiv C$.

Now we can list the key non-trivial clauses for the semantics of formulae. The remaining definitions can be found in [1]. The semantics for the formulae needed for local state are defined in the next section.

- $\mathcal{M} \models \Box C$ if $\forall \mathcal{M}'.(\mathcal{M} \rightsquigarrow \mathcal{M}' \supset \mathcal{M}' \models C)$.
- $\mathcal{M} \models e \bullet e' = x[C]$ if $\exists \mathcal{M}'.(\mathcal{M}[x : ee'] \Downarrow \mathcal{M}' \ \wedge\ \mathcal{M}' \models C)$.
- $\mathcal{M} \models [!x]C \equiv \mathcal{M} \models [!x]^{\circ}C$ if $[\![x]\!]_{\mathcal{M}} = l$ and $\forall V\colon \mathcal{M}[l \mapsto V] \models C$.

These definitions are adapted from [1, 6]. Note that the definition of satisfaction for $\Box C$ can be simplified to $(\xi, \sigma) \models \Box C$ if $\forall \sigma'.(\xi, \sigma') \models C$. The more formulation above generalises smoothly to local references.

2.4 Key Definitions

We define three notions for formulae that form the backbone of the completeness proofs.

Definition 3 (weakest precondition, WPC). We say C is a *weakest precondition (or WPC) of C' with respect M at u* written $\models^{wpc} [C]\, M :_u [C']$, if $\forall \mathcal{M}.(\mathcal{M} \models C$ iff $\exists \mathcal{M}'.(\mathcal{M}[u : M] \Downarrow \mathcal{M}' \wedge \mathcal{M}' \models C')$.

The idea behind WPCs is that they, as the name suggests, state (up to logical equivalence, the weakest formula, that guarantees a given postcondition to hold of a program. We will later show that the inference system in Figure 1 does produce WPCs.

Definition 4 (total characteristic assertion pair, 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, assuming $\mathcal{M} = (\xi, \sigma)$.

Fig. 1 Derivation Rules for TCAPs for Imperative PCFv.

$$[var]\frac{-}{\vdash^{\star\star}[\mathsf{T}]\,y :_u [u=y]} \qquad [const]\frac{-}{\vdash^{\star\star}[\mathsf{T}]\,\mathsf{c} :_u [u=\mathsf{c}]} \qquad [val]\frac{\vdash^{\star\star}[\mathsf{T}]\,V :_u [A] \quad g \text{ fresh}}{\vdash^{\star}[\forall u.(A \supset g \bullet u \Downarrow)]\,V :_u [g \bullet u \Downarrow]}$$

$$[abs]\frac{\vdash^{\star}[C]\,M :_m [C'] \quad \tilde{i} = \mathsf{fv}(C,C') \setminus (\mathsf{fv}(M) \cup \{m\})}{\vdash^{\star\star}[\mathsf{T}]\,\lambda y.M :_u [\Box \forall y \tilde{i}.(C \supset u \bullet y = m[C'])]} \qquad [rec]\frac{\vdash^{\star\star}[\mathsf{T}]\,\lambda x.M :_u [A]}{\vdash^{\star\star}[\mathsf{T}]\,\mu f.\lambda x.M :_u [A[u/f]]}$$

$$[op\text{-}val]\frac{\vdash^{\star}[C]\,M :_u [C']}{\vdash^{\star}[C[\mathsf{op}(m_1,..,m_n)/x]]\,\mathtt{let}\ x = \mathsf{op}(m_1,..,m_n)\ \mathtt{in}\ M :_u [C']}$$

$$[if]\frac{\vdash^{\star}[C_i]N_i :_u [C'] \quad \mathsf{b}_1 = \mathsf{t},\ \mathsf{b}_2 = \mathsf{f}}{\vdash^{\star}[\bigwedge_{i=1,2}(x = \mathsf{b}_i \supset C_i)]\,\mathtt{if}\ x\ \mathtt{then}\ N_1\ \mathtt{else}\ N_2 :_u [C']}$$

$$[app]\frac{\vdash^{\star}[C]\,M :_u [C']}{\vdash^{\star}[f \bullet y = x[C]]\,\mathtt{let}\ x = f y\ \mathtt{in}\ M :_u [C']} \qquad [let\text{-}val]\frac{\vdash^{\star\star}[\mathsf{T}]\,V :_x [A] \quad \vdash^{\star}[C]\,M :_u [C']}{\vdash^{\star}[\forall x.(A \supset C)]\,\mathtt{let}\ x = V\ \mathtt{in}\ M :_u [C']}$$

$$[assign]\frac{\vdash^{\star}[C]\,M :_u [C']}{\vdash^{\star}[C\{y/!x\}]\,x := y; M :_u [C']} \qquad [deref]\frac{\vdash^{\star}[C]\,M :_u [C']}{\vdash^{\star}[C[!y/z]]\,\mathtt{let}\ z = !y\ \mathtt{in}\ M :_u [C']}$$

1. (soundness) $\models [C]\,M :_u [C']$.
2. (MTC, minimal terminating condition) $(M\xi,\sigma) \Downarrow$ iff $\mathcal{M} \models \exists \tilde{i}.C$ with $\tilde{i}$ covering exactly the auxiliary variables in C.
3. (closure) Suppose $\mathcal{M} \models [C]\,N :_u [C']$ then: $\mathcal{M}[u : M] \Downarrow \mathcal{M}'$ implies $\mathcal{M}[u : N] \Downarrow \mathcal{M}''$ such that $\mathcal{M}' \lesssim \mathcal{M}''$).

TCAPs are pairs of formulae, that are up-closed. That means that whenever a program P satisfies a TCAP, any other program that is exactly like P, except that it diverges less often, also satisfies this TCAP. Since $\lesssim$ is well-founded, TCAPs (C,C') have canonical representatives: the least defined program P such that $\models [C]P :_u [C']$.

Definition 5 (value total characteristic assertion pair, VTCAP). A pair (T,A) is a *value total characteristic assertion pair of* $\Gamma;\Delta \vdash V : \alpha$ *at* u, if following conditions hold, assuming $\mathcal{M} = (\xi,\sigma)$.

1. (soundness) $\models [\mathsf{T}]\,V :_u [A]$.
2. (closure) $\mathcal{M} \models [\mathsf{T}]\,W :_u [A]$ implies $V_{\mathcal{M}} \lesssim W_{\mathcal{M}}$.

Figure 1 gives the generation rules for TCAPs. In all rules, we fix, but leave implicit, a reference basis with domain $\tilde{x}$.

We illustrate the derivation rules. [*val*] transforms a judgement for values (written turnstile $\vdash^{\star\star}$) to that for general programs (written turnstile $\vdash^{\star}$). We cannot infer $\vdash^{\star}[\mathsf{T}]\,V :_u [A]$ from $\vdash^{\star\star}[\mathsf{T}]\,V :_u [A]$, because in general (T,A) is not strong enough an assertion pair to guarantee that V is the unique least defined inhabitant if we also consider non-values. [*abs*] is as in [3], except for the aforementioned decomposition using $\Box$ and the one-sided evaluation formula. [*abs, rec, if, assign, if, deref*] are essentially like in [3]. The rule for application [*app*] makes full use of the one-sided evaluation

formula. [3], uses $\tilde{x}=\tilde{i}\wedge[\tilde{x}=\tilde{i}]\ f\bullet x=y\ [C']\wedge ...$, where $\tilde{x}$ ranges over all references, because the rule needed to ensure that the function is evaluated in the current state. With one-sided evaluation formulae, this is no longer required, leading to a constant-length specification (other than that of the `let`-body).

2.5 Completeness

Lemma 6. *1. Let $\Theta\vdash M:\alpha$, $x\notin\mathsf{dom}(\Theta)$ and y fresh, then: $\mathcal{M}[u:M]\Downarrow\mathcal{M}'$ iff $\mathcal{M}[y/x][u:M]\Downarrow\mathcal{M}'[y/x]$. Furthermore: $\mathcal{M}[u:M]\Downarrow\mathcal{M}_0$ iff $\mathcal{M}[x:V/x][u:M]\Downarrow\mathcal{M}_0[x:V/x]$.*
2. Let $\Theta\vdash M:\alpha$, $x\notin\mathsf{dom}(\Theta)$.
3. Let $\Theta\vdash C$ and $x\notin\mathsf{dom}(\Theta)$. Then: $\mathcal{M}[x:V]\models C$ iff $\mathcal{M}'\models C$.
4. Let $\mathcal{M}=(\xi,\sigma)$. Then: $(\xi,\sigma)\models C[e/x]$ iff $(\xi\cdot x[\![e]\!]_{\mathcal{M}},\sigma)\models C$ and $(\xi,\sigma)\models C\{\!\{e/x\}\!\}$ iff $(\xi,\sigma[\xi(x)\mapsto[\![e]\!]_{\mathcal{M}})\models C$.
5. Let σ be an injective renaming, then: $\mathcal{M}\models C$ iff $\mathcal{M}\sigma\models C\sigma$.
6. If $V\lesssim W$ then also $MV\lesssim MW$ and $M[V/x]\lesssim M[W/x]$.
7. The congruence $\lesssim$ is syntactically continuous, i.e. if $V_0\stackrel{def}{=}\Omega, V_{n+1}\stackrel{def}{=}\lambda x.M[V_n/f]$ and $\lambda x.V_n\lesssim W$ for all n, then $\mu f.\lambda x.M\lesssim W$. Here $\Omega\stackrel{def}{=}\mu f.\lambda x.fx$.

Proof. Straightforward, see [1].

Proposition 7 (WPC as TCAP).

1. *If $\models^{wpc}[C]M:_u[g\bullet u\Downarrow]$ with g auxiliary, then $(C,g\bullet u\Downarrow)$ is a TCAP of M w.r.t. u.*
2. *If (T,A) is a VTCAP of V w.r.t. u, then $\models^{wpc}[\forall u.(A\supset g\bullet u\Downarrow)]V:_u[g\bullet u\Downarrow]$.*

Proof. For (1), soundness is by definition. For (MTC), we reason as follows, assuming V_c to be a constant function of appropriate type that is hereditarily convergent for any input. Note g may or may not occur in $\mathcal{M}$.

$$\begin{array}{lll}
\mathcal{M}[u:M]\Downarrow & \Rightarrow & \mathcal{M}[g'/g][u:M]\Downarrow \qquad (g' \text{ fresh, Lemma 6.1})\\
& \Rightarrow & \mathcal{M}[g'/g][g:V_c][u:M]\Downarrow\mathcal{M}'\models g\bullet u\Downarrow \qquad (\text{Lemma 6.3})\\
& \Rightarrow & \mathcal{M}[g'/g][g:V_c]\models C \qquad (\text{WPC})\\
& \Rightarrow & \mathcal{M}[g'/g]\models\exists g.C\\
& \Rightarrow & \mathcal{M}[g'/g][g/g']\models(\exists g.C)[g/g'] \qquad (\text{Lemma 6.5})\\
& \Rightarrow & \mathcal{M}\models\exists g.C\\
& \Rightarrow & \mathcal{M}\models\exists\tilde{i}.C
\end{array}$$

For (Closure), assume $\mathcal{M}[u:M]\Downarrow\mathcal{M}'$. As shown in the argument above, we can set $g\notin\mathsf{fv}(\mathcal{M})$ without loss of generality. By soundness, $\mathcal{M}[u:N]\Downarrow\mathcal{M}''$. Assume $\mathcal{M}'\not\lesssim\mathcal{M}''$, then for some closed V of type $\alpha\Rightarrow\mathsf{Unit}$ with α the type of u:

$$\mathcal{M}''[g:V][w:gu]\Downarrow \qquad \text{but} \qquad \mathcal{M}'[g:V][w:gu]\not\Downarrow.$$

By Lemma 6.1, we obtain:

$$\mathcal{M}[g:V][u:N]\Downarrow\mathcal{M}'\not\models g\bullet u\Downarrow$$

which contradicts the assumption.

For (2) if $(\xi \cdot u : V\xi, \sigma) \models g \bullet u \Downarrow$ and at the same time $(\xi \cdot u : W\xi, \sigma) \models A$, then by (closure): $V_{\mathcal{M}} \lesssim W_{\mathcal{M}}$, hence clearly $(\xi \cdot u : W\xi, \sigma) \models g \bullet u \Downarrow$. The reverse direction is trivial. □

Lemma 8. *1. (i)* $\vdash^{\star\star} [\mathsf{T}]V :_u [A]$ *implies A is downward-closed w.r.t. free variables of V except when V is a variable. (ii)* $\vdash^{\star} [C]M :_u [C']$ *implies C is upper-closed w.r.t. free variables of M.*

2. $\vdash^{\star} [C]M :_u [C']$ *implies* $\models^{wpc} [C]M :_u [C']$.

3. $\vdash^{\star\star} [\mathsf{T}]V :_u [A]$ *implies* (T, A) *is a VTCAP of V w.r.t. u.*

Proof. For (1), we show (i) and (ii) simultaneously by rule induction. The reasoning is mechanical (note that, when we apply [*val*] for $V \stackrel{\text{def}}{=} y$, then $\forall u.(A \supset g \bullet u \Downarrow)$ becomes $g \bullet y \Downarrow$ hence it is indeed upper-closed at y, so that the special case for [*var*] does not affect the argument). For (2) and (3), we again use simultaneous induction on the derivation of $\vdash^{\star} [C]M :_u [C']$ and $\vdash^{\star\star} [\mathsf{T}]V :_u [C']$. We have essentially three cases for (2): [*if*], [*val*] and all those of the form $\mathtt{let}\, x = M\, \mathtt{in}\, N$, noting that $x := y; M$ is $\mathtt{let}\, z = (x := y)\, \mathtt{in}\, M$.

We start with programs $\mathtt{let}\, x = M\, \mathtt{in}\, N$ by showing that

$$(\models^{wpc} [C]M :_x [C_0] \wedge \models^{wpc} [C_0]N :_u [C']) \;\Rightarrow\; \models^{wpc} [C]\mathtt{let}\, x = M\, \mathtt{in}\, N :_u [C'].$$

This is immediate by the (IH) because

$$(\mathcal{M}[x : M] \Downarrow \mathcal{M}'', \mathcal{M}''[u : N] \Downarrow \mathcal{M}') \quad \text{iff} \quad \mathcal{M}[u : \mathtt{let}\, x = M\, \mathtt{in}\, N] \Downarrow \mathcal{M}'$$

as is easy to verify. But $\models^{wpc} [C_0]N :_u [C'])$ by (IH), so all we have to verify is $\models^{wpc} [C]M :_x [C_0]$ for $M \in \{!y, x := y, fy, V\}$.

Let $M \stackrel{\text{def}}{=} (\xi, \sigma)$. For $M = !y$ we have

$$\mathcal{M} \models C[!y/z] \quad \text{iff} \quad (\xi \cdot z : [\![!y]\!]_{\mathcal{M}}, \sigma) \models C$$

by Lemma 6.4.

If $M = fy$ then by definition of $f \bullet y = x[C]$:

$$\mathcal{M} \models f \bullet y = x[C] \quad \text{iff} \quad ([\![f]\!]_{\mathcal{M}}[\![y]\!]_{\mathcal{M}}, \sigma) \Downarrow (V, \sigma'), (\xi \cdot x : V, \sigma) \models C.$$

This immediately implies the required equivalence.

For [let-val], assume

$$\mathcal{M}[u : \mathtt{let}\, x = V\, \mathtt{in}\, M] \Downarrow \mathcal{M}' \models C'$$

that is

$$\mathcal{M}[x : V][u : M] \Downarrow \mathcal{M}' \models C'.$$

By (IH) we have $\mathcal{M}[x : V] \models C$. By A being VTCAP, we know $\mathcal{M}[x : W] \models A$ implies $V_{\mathcal{M}} \prec W_{\mathcal{M}}$. Since C is upperclosed at x by (1), we know $\mathcal{M}[x : W] \models A$ implies $\mathcal{M}[x : W] \models C$, that is $\mathcal{M} \models \forall x.(A \supset C)$. The other direction is immediate.

For assignment, clearly

$$\mathcal{M} \models C\{\!|y/!x|\!\} \quad \text{iff} \quad (\xi, \sigma[[\![x]\!]_{\mathcal{M}} \mapsto [\![y]\!]_{\mathcal{M}}]) \models C$$

by Lemma 6.4.

For [*if*] assume w.l.o.g. that $\xi(x) = \mathsf{T}$. Then

$$\begin{array}{lll} \mathcal{M} \models \bigwedge_i (x = \mathsf{b}_i \supset C_i) & \text{iff} & \mathcal{M} \models C_i \\ & \text{iff} & \mathcal{M}[u : N_1] \Downarrow \mathcal{M}' \models C' \\ & \text{iff} & \mathcal{M}[u : \mathtt{if}\ x\ \mathtt{then}\ N_1\ \mathtt{else}\ N_2] \Downarrow \mathcal{M}' \models C' \end{array}$$

Here the second equivalence is by (IH).

For $\vdash^{\star\star}$, [*val*] is immediate by the (IH) and Prop. 7.2. Furthermore, [*var*] and [*const*] are trivial. Similarly, [*abs*] is immediate from the (IH) and the definitions using closure. For [*rec*] let $(\xi \cdot u : W\xi, \sigma) \models [\mathsf{T}]\, W :_u [A[u/f]]$. Then

$$\mathcal{M} \models A \qquad \text{where } \mathcal{M} \stackrel{\text{def}}{=} (\xi \cdot u, f : W\xi, \sigma) \tag{2.1}$$

by the (IH) and Lemma 6.4. We construct $V_0 \stackrel{\text{def}}{=} \Omega$ and $V_{n+1} \stackrel{\text{def}}{=} \lambda x.M[V_n/f]$ and show by induction on n that $(V_n)_{\mathcal{M}} \lesssim W_{\mathcal{M}}$.

$$\begin{array}{lll} (V_{n+1})_{\mathcal{M}} & = & (\lambda x.M[V_n/f])_{\mathcal{M}} \\ & \lesssim & (\lambda x.M[W/f])_{\mathcal{M}} \qquad \text{(inner (IH), Lemma 6.6)} \\ & \lesssim & W_{\mathcal{M}} \qquad (2.1). \end{array}$$

By syntactic continuity of $\lesssim$ (cf. Lemma 6.7) $\mu f.\lambda x.M \lesssim W$. □

Noting that all postconditions inductively generated by $\vdash^{\star}$ are of the form $g \bullet u \Downarrow$, Proposition 7 and Lemma 8 yield:

Theorem 9 (descriptive completeness). $\vdash^{\star} [C]\, M :_u [C']$ *implies* (C, C') *is a TCAP of* M *w.r.t.* u.

This theorem has two powerful consequences. The first of which, given next, is the perfect match between operational and logical semantics.

Corollary 10 (observational completeness). $M \cong N$ *if and only if, for each* C *and* C', *we have* $\models [C]\, M :_u [C']$ *iff* $\models [C]\, N :_u [C']$.

We next show relative completeness, which means that relative provability (by the proof rules in Appendix A) coincided with validity, i.e. $\vdash^{\star} [C]\, M :_m [C']$ implies $\vdash [C]\, M :_m [C']$ where $\vdash$ is the provability in the logic for alias. For this purpose we relate provability in the standard programs to TCAP derivability of the let forms. The map from the standard programs to the let forms is given as follows.

$$\begin{array}{rcll} [\![\mathsf{c}]\!] & \stackrel{\text{def}}{=} & \mathsf{c} & \\ [\![x]\!] & \stackrel{\text{def}}{=} & x & \\ [\![\lambda x.M]\!] & \stackrel{\text{def}}{=} & \lambda x.[\![M]\!] & \\ [\![\mu f.\lambda x.M]\!] & \stackrel{\text{def}}{=} & \mu f.\lambda x.[\![M]\!] & \\ [\![M]\!] & \stackrel{\text{def}}{=} & \langle\!\langle M \rangle\!\rangle_x[x] & \text{(other cases)} \end{array}$$

In the last line we set:

$$\begin{array}{rcl}
\langle\!\langle M_1 + M_2 \rangle\!\rangle_y[L] & \stackrel{\text{def}}{=} & \langle\!\langle M_1 \rangle\!\rangle_{y_1}[\langle\!\langle M_2 \rangle\!\rangle_{y_2}[\texttt{let } y = y_1 + y_1 \texttt{ in } L]] \\
\langle\!\langle M_1 M_2 \rangle\!\rangle_y[L] & \stackrel{\text{def}}{=} & \langle\!\langle M_1 \rangle\!\rangle_{m_1}[\langle\!\langle M_2 \rangle\!\rangle_{m_2}[\texttt{let } y = m_1 m_2 \texttt{ in } L]] \\
\langle\!\langle \texttt{if } M \texttt{ then } N_1 \texttt{ else } N_2 \rangle\!\rangle_y[L] & \stackrel{\text{def}}{=} & \langle\!\langle M \rangle\!\rangle_m[\texttt{if } m \texttt{ then } \langle\!\langle N_1 \rangle\!\rangle_y[L] \texttt{ else } \langle\!\langle N_2 \rangle\!\rangle_y[L]] \\
\langle\!\langle M := N \rangle\!\rangle_y[L] & \stackrel{\text{def}}{=} & \langle\!\langle M \rangle\!\rangle_m[\langle\!\langle N \rangle\!\rangle_n[m := n; L]] \\
\langle\!\langle !M \rangle\!\rangle_y[L] & \stackrel{\text{def}}{=} & \langle\!\langle M \rangle\!\rangle_m[\texttt{let } y = !m \texttt{ in } L]
\end{array}$$

Note that, by definition, $\langle\!\langle M \rangle\!\rangle_y[L]$ is a concatenation of two let sequences, the expansion of M and L.

We have already seen the relationship between the TCAP rules and WPC. In fact, on the basis of $\vdash^\star$ and $\vdash^{\star\star}$, we can further construct the rules for deriving WPC of let forms by a slight change of the TCAP rules. This derivation relation is written $\vdash^{\mathsf{wpc}} [C]\, L :_u [C']$ and given by the same rules as Figure 1 except:

1. We take off [*var, const, abs*] from the rules.
2. We replace [*val*] with, assuming C is well-typed as a postcondition of V w.r.t. u:

$$[\textit{val-wpc}]\ \frac{\vdash^{\star\star} [\mathsf{T}]\, V :_u [A]}{\vdash^{\mathsf{wpc}} [\forall u.(A \supset C)]\, V :_u [C]}$$

3. For the remaining rules, we replace $\vdash^\star$ with $\vdash^{\mathsf{wpc}}$.

We now observe:

Lemma 11. *1. If C_0 is WPC of C' w.r.t. M at u and $\models [C]M :_u [C']$, then $C \supset C_0$.*
2. If $\vdash^{\mathsf{wpc}} [C]\, L :_u [C']$ and C' is upper-closed at u, then (i) C is upper-closed w.r.t. free variables of L, and (ii) C is WPC of C' w.r.t. L at u.
3. $\vdash^{\mathsf{wpc}} [C]\, [\![M]\!] :_u [C']$ implies $\vdash [C]\, M :_u [C']$.

(1) is the standard property of weakest pre-condition. The proofs of these results are identical with those for local state, and are left to the next section.

Theorem 12 (relative completeness). *For each well typed M, $\models [C]\, M :_u [C']$ such that C' is upward-closed at u implies $\vdash [C]\, M :_u [C']$.*

Proof. Let $\vdash^{\mathsf{wpc}} [C_0]\, [\![M]\!] :_u [C']$ with C' upper-closed. By Lemma 11 (2), this implies C_0 is a WPC of C' w.r.t. $[\![M]\!]$ at u. Further by Lemma 11 (3) we have $\vdash [C_0]M :_u [C']$. Since $[\![M]\!] \cong M$, C_0 is also a WPC of C' w.r.t. M at u. Now assume $\models [C]M :_u [C']$. By Lemma 11 (1) we have $C \supset C_0$, hence done. □

3 Local State

3.1 Language and Logic

We now extend the completeness results to a language with local state. The programming and logical languages are extended as follows:

$$\begin{array}{rcl}
M & ::= & \ldots \mid \texttt{ref}(M) \\
C & ::= & \ldots \mid \nu x.C \mid \overline{\nu} x.C \mid e \hookrightarrow e' \mid e \# e'
\end{array}$$

The syntax adds the reference generation $\mathtt{ref}(M)$, which behaves as: first M of type α is evaluated and becomes a value V; then a *fresh* reference of type $\mathsf{Ref}(\alpha)$ with initial content V is generated. This behaviour is formalised by the following reduction rule:

$$(\mathtt{ref}(V),\ \sigma) \longrightarrow (\nu\, l)(l,\ \sigma \uplus [l \mapsto V]) \qquad (l \text{ fresh})$$

Above σ is a store, a finite map from locations to closed values, denoting the initial state; whereas $\sigma \uplus [l \mapsto V]$ is the result of disjointly adding a pair (l,V) to σ. The resulting configuration uses a binder. Its general form is $(\nu\,\tilde{l})(M,\sigma)$ where $\tilde{l}$ is a vector of distinct locations occurring in σ (the order is irrelevant). We write (M,σ) for $(\nu\,\varepsilon)(M,\sigma)$. We use $\cong$ (resp. $\lesssim$) for the observational congruence (resp. precongruence).

In the logical language, the hiding quantifiers, $\nu x.C$ (read: "for some hidden reference x, C holds") and $\overline{\nu} x.C$ (read: "for each hidden reference x, C holds"), which are mutually dual, are new and quantify only variables of reference types (x's type is $\alpha = \mathsf{Ref}(\beta)$). $e_1 \hookrightarrow e_2$ (with e_2 of a reference type) is *reachability predicate*, which says that: *We can reach the reference named by e_2 from a datum denoted by e_1. . $y\#x$* is the negation of $x \hookrightarrow y$, which says: *One can never reach a reference y starting from a datum denoted by x.*

3.2 Model

We extend a model by introducing ν-binder: a model of type $(\Gamma;\Delta)$ is a structure $(\nu\,\tilde{l})(\xi,\sigma)$ with (ξ,σ) being an open model of type $\Gamma;\Delta\cdot\Delta'$ with $\mathsf{dom}(\Delta') = \{\tilde{l}\}$. $(\nu\,\tilde{l})$ act as binders. We also use the same notations as Section 2.3, extending to $(\nu\,\tilde{l})(\xi,\sigma)$ from (ξ,σ). In particular,

- We write $\mathcal{M}_1 \lesssim \mathcal{M}_2$, where $\mathcal{M}_i = (\nu\,\tilde{l_i})(\tilde{u} : \tilde{V}_i, \sigma_i)$, if for all M of type Unit with $\mathsf{fv}(M) \subseteq \tilde{x}$: $(\nu\,\tilde{l_i})(\mathtt{let}\ \tilde{x} = \tilde{V_1}\ \mathtt{in}\ M, \sigma_1) \Downarrow$ implies $(\nu\,\tilde{l_i})(\mathtt{let}\ \tilde{x} = \tilde{V_2}\ \mathtt{in}\ M, \sigma_2) \Downarrow$.
- Let $\mathcal{M} = (\nu\,\tilde{l})(\xi,\sigma)$. We write $M_{\mathcal{M}}$ for $(\nu\,\tilde{l})(M\xi,\sigma)$, assuming typability.

Then the definitions of TCP and VTCP stay as the same by replacing (ξ,σ) by $(\nu\,\tilde{l})(\xi,\sigma)$. We set:

$$\mathcal{M} \models e_1 \hookrightarrow e_2 \quad \text{ if } [\![e_2]\!]_{\xi,\sigma} \in \mathsf{lc}(\mathsf{fl}([\![e_1]\!]_{\xi,\sigma}),\sigma) \text{ for each } (\nu\,\tilde{l})(\xi,\sigma) \approx \mathcal{M}$$

where $\mathsf{lc}(S,\sigma)$ (with S a set of locations) is the minimum set S' of locations such that: (1) $S \subset S'$ and (2) If $l \in S'$ then $\mathsf{fl}(\sigma(l)) \subset S'$. The clause says that the set of hereditarily reachable names from e_1 includes e_2 up to $\approx$.

The universal hiding-quantifier has the following semantics.

$$\mathcal{M} \models \overline{\nu} x.C \quad \text{ if } \quad \forall \mathcal{M}'.((\nu\, l)\mathcal{M}' \approx \mathcal{M} \supset \mathcal{M}'[x : l] \models C)$$

where l is fresh, i.e. $l \notin \mathsf{fl}(\mathcal{M})$ where $\mathsf{fl}(\mathcal{M})$ denotes free labels in $\mathcal{M}$. Dually, with l fresh again:

$$\mathcal{M} \models \nu x.C \quad \text{ if } \quad \exists \mathcal{M}'.((\nu\, l)\mathcal{M}' \approx \mathcal{M} \wedge \mathcal{M}'[x : l] \models C)$$

which says that x denotes a hidden reference, say l, and the result of taking it off from $\mathcal{M}$ satisfies C. We also refine $\mathcal{M} \models [!x]^{\circ} C$ if $\forall \mathcal{M}'.(\mathcal{M}[x \mapsto N] \Downarrow \mathcal{M}' \wedge \forall V.(\mathcal{M}[x \mapsto V] \approx \mathcal{M}'[x \mapsto V]) \supset \mathcal{M}' \models C)$. [6] lists full definitions.

3.3 Completeness

Descriptive and Observational Completeness. We fist introduce the following predicate, with f, X and i fresh:

$$\mathsf{new}(x)(C) \stackrel{\text{def}}{=} \exists f.\forall \mathrm{X}, \forall i^{\mathrm{X}}.f \bullet () = x[x\#i \wedge C]@\emptyset$$

which means: *there exists a function f such that, when we invoke it with $()$ from the current state, return a fresh x and leaves the state such that it satisfies C.* Above we write $x \bullet y = z[C]@\emptyset$ for $\forall g.(g \bullet () \Downarrow \supset x \bullet y = z[C \wedge g \bullet () \Downarrow])$. We add to the previous rule:

$$[\mathit{letref}]\frac{\vdash^{\star} [C]M :_u [C']}{\vdash^{\star} [\mathsf{new}(x)(C \wedge !x = y)]\,\mathtt{let}\ x = \mathtt{ref}(y)\ \mathtt{in}\ M :_u [C']}$$

The rule reads:

> *If C is a WPC of C' w.r.t. M and u, then " the initial state is such that creating fresh x in that state leads to C and the content of x is y" is a WPC of C' w.r.t.* $\mathsf{new}(x)(C \wedge !x = y)$ *and C'.*

The same statements as Lemmas 6 and 8 and Proposition 7 hold incorporating $[\mathit{letref}]$. The arguments are precisely the same argument as before, reading $\mathcal{M}$ to be a model with hiding, except for adding the following argument for $[\mathit{letref}]$ to the proof of Lemma 8.

Lemma 13 (WPC for Ref). *Let C' be thin and upper-closed w.r.t. u. Then* $\models^{wpc} [C]M :_u [C']$ *implies* $\models^{wpc} [\mathsf{new}(x)(C \wedge !x = y)]\,\mathtt{let}\ x = \mathtt{ref}(y)\ \mathtt{in}\ M :_u [C']$.

Proof. Let $\mathcal{M}[u : \mathtt{let}\ x = \mathtt{ref}(y)\ \mathtt{in}\ M] \Downarrow \mathcal{M}'$ (which always converges). For a fresh l, we have $\mathcal{M}' = (\nu l)(\mathcal{M}[x : l][l \mapsto y])$ (to be precise, $\mathcal{M}'$ is the result of evaluating the r.h.s.). By $\mathcal{M}' \models C$ and by setting f to be $\lambda().\mathtt{ref}(y)$ as the witness for the existential, we are done.

For the other direction, suppose $\mathcal{M} \models \mathsf{new}(x)(C \wedge !x = y)$. Thus there is some function W such that, up to $\approx$ and for fresh l and f:

$$\mathcal{M}[f : W][x : f()] \Downarrow \mathcal{M}'' = (\nu l)(\mathcal{M}[f : W][x : l][l \mapsto y]) \models C'.$$

Let $\mathcal{M}' \stackrel{\text{def}}{=} \mathcal{M}''/f$. Since C' is thin, we have $\mathcal{M}' \models C'$. Thus we have $\mathcal{M}[x : \mathtt{ref}(y)] \Downarrow \mathcal{M}' \models C'$, as required. □

By precisely the same arguments as before, and noting $g \bullet u \Downarrow$ is obviously thin, we obtain:

Theorem 14 (descriptive completeness). *In the derivation system incorporating* [letref], *if $\vdash^{\star} [C]M :_u [C']$ then (C, C') is a TCAP of M w.r.t. u.*

The following is a direct corollary [3] of Theorem 14.

Corollary 15 (observational completeness). *$M \cong N$ iff, for each C and C' and fresh u, we have $\models [C]M :_u [C']$ iff $\models [C]N :_u [C']$.*

Preparation for Relative Completeness: thinness. The proof rules in Figure 2 in Appendix A restrict some of the postconditions to be *thin*, in the following sense (this restriction of the proof rules does not affect the relative completeness result, in the sense that we prove the relative completeness for all total correctness formulae which may not necessarily be thin).

Definition 16. (1) We say C is *thin* iff for each $\mathcal{M}$ and for each $y \in \mathsf{fv}(\mathcal{M}) \backslash \mathsf{fv}(C)$, $\mathcal{M} \models C$ implies $\mathcal{M}/y \models C$. (2) We say C is *weak* iff for each $\mathcal{M}$ and for each $y \in \mathsf{fv}(\mathcal{M}) \backslash \mathsf{fv}(C)$, $\mathcal{M}/y \models C$ implies $\mathcal{M} \models C$.

We can check if C is thin then, with $x \notin \mathsf{fv}(C)$, we have $\nu x.C \supset C$ (we believe this formula characterises thinness), dually for weak formulae. We also note:

Proposition 17. *1. $e = e'$ and $e \hookrightarrow e'$ are both thin and weak. C is thin iff $\neg C$ is weak.*
2. If C, C' are thin, then $C \wedge C'$, $C \vee C'$, $\forall x^{\alpha}.C$ for all α, $\exists x^{\alpha}.C$ with $\alpha \in \{\mathsf{Unit}, \mathsf{Bool}, \mathsf{Nat}\}$, $\exists \mathrm{X}.C$, $\forall \mathrm{X}.C$, $\nu x.C$, $\Box C$ $\boxdot C$, and $[!x]^{\circ} C$ are thin. Dually for weak formulae.
3. If C is thin (resp. weak) then so is $e \bullet e' = x[C]$. If C is weak then $e \bullet e' = x[C]$ is thin (resp. weak).

Proof. (1) is immediate. For (2), suppose C and C' are thin, $x \notin \mathsf{fv}(C, C')$ and $\mathcal{M} \models C \wedge C'$. Then $\mathcal{M} \models C$ hence $\mathcal{M}/x \models C$, similarly for C', hence $\mathcal{M}/x \models C \wedge C'$. Similarly for other cases. For (3), let C be thin and x be fresh. Suppose $\mathcal{M} \models f \bullet y = z[C]$, i.e. $\mathcal{M}[z : fy] \Downarrow \mathcal{M}' \models C$. Then we have $\mathcal{M}/x[z : fy] \Downarrow \mathcal{M}'/x$. Since C is thin, we have $\mathcal{M}'/x \models C$, as required. Symmetrically let C be weak and x be fresh. Suppose $\mathcal{M}/x \models f \bullet y = z[C]$, i.e. $\mathcal{M}/x[z : fy] \Downarrow \mathcal{M}' \models C$. Note the second half means $\mathcal{M}[z : fy] \Downarrow \mathcal{M}''$ such that $\mathcal{M}' = \mathcal{M}''/x$. Since C is weak, we have $\mathcal{M}'' \models C$, as required. □

The above results give us an easy way to establish thin-ness and its dual syntactically. In particular we obtain:

Proposition 18. *$g \bullet u \Downarrow$ is both thin and weak.*

In the proof rules, we assume the thin-ness in the postcondition of the conclusion in: [*App*], [*Assign*] and [*Deref*] (as well as those derived from these rules). We note:

Remark 19. *In all reasoning examples in [6], the derivation of provability conforms to the condition on thin-ness as stipulated above.*

Relative Completeness. The proof of relative completeness uses the map $[\![M]\!]$ as in Section 2 , which is augmented by:

$$\langle\!\langle \mathtt{ref}(M) \rangle\!\rangle_x[L] \quad \stackrel{\text{def}}{=} \quad \langle\!\langle M \rangle\!\rangle_y[\mathtt{let}\ x = \mathtt{ref}(y)\ \mathtt{in}\ L]$$

Precisely following Section 2 we define $\vdash^{\mathsf{wpc}}$, except we add:

$$[\mathit{letref}] \frac{\vdash^{\mathsf{wpc}} [C] M :_u [C']}{\vdash^{\mathsf{wpc}} [\mathsf{new}(x)(C \wedge !x = y)]\, \mathtt{let}\ x = \mathtt{ref}(y)\ \mathtt{in}\ M :_u [C']}$$

Other rules remain precisely the same as before. Note that, by Lemma 13, this rule always produces a WPC as far as the premise gives such.

We can now show:

Lemma 20. *1. Assume C_0 is a WPC of C' w.r.t. M at u. Then:* (i) *if $\models [C]M :_u [C']$, then $C \supset C_0$.* (ii) *if C' is thin, then C is also thin.*
2. *If $\vdash^{\mathsf{wpc}} [C]\,L :_u [C']$ and C' is upper-closed at u, then (i) C is upper-closed w.r.t. free variables of L, and (ii) C is WPC of C' w.r.t. L at u.*
3. *If $\vdash^{\mathsf{wpc}} [C]\,\langle\!\langle M \rangle\!\rangle_x [L] :_u [C']$ then $\vdash^{\mathsf{wpc}} [C]\,[\![M]\!] :_x [C_0]$ and $\vdash^{\mathsf{wpc}} [C_0]\,L :_u [C']$.*
4. *If $\vdash^{\mathsf{wpc}} [C]\,[\![M]\!] :_u [C']$ and if C' is thin, then $\vdash [C]\,M :_u [C']$.*
5. *$\vdash^{\star} [C]\,[\![M]\!] :_u [C']$ implies $\vdash [C]\,M :_u [C']$. Also $\vdash^{\star\star} [\mathsf{T}]\,[\![V]\!] :_u [A]$ implies $\vdash [\mathsf{T}]\,V :_u [A]$.*

Proof. (1-i) is immediate [for each $\mathcal{M}$, if $\mathcal{M} \models C$ then $\mathcal{M}[u : M] \Downarrow \mathcal{M}' \models C'$ hence by the definition of WPC we have $\mathcal{M} \models C_0$]. (1-ii) is also direct from the definition. (2)-(i) is easy rule induction by checking the variance and noting $\vdash^{\star\star} [\mathsf{T}]\,V :_u [A]$ implies A is downward-closed w.r.t. free variables in V except when V itself is a variable. For (2)-(ii), all rules easily derive a WPC from a WPC (for ref, we use Lemma 13). The only interesting case is [*val*], which is reasoned as follows (we only show the non-trivial direction). We use the fact that, with $x \notin \mathsf{fv}(C)$, we have $\mathcal{M}[x : V] \models C$ iff $\mathcal{M} \models C$.

$$
\begin{array}{lcl}
\mathcal{M}[u : \mathtt{let}\ x = V\ \mathtt{in}\ M] \Downarrow \models C' & \equiv & \mathcal{M}[x : V][u : M] \Downarrow \models C' \\
& \equiv & \mathcal{M}[x : V] \models C \\
& \Rightarrow & \forall W.(V_{\mathcal{M}} \prec W_{\mathcal{M}} \Rightarrow \mathcal{M}[x : W] \models C) \\
& \Rightarrow & \forall W.(\mathcal{M}[x : W] \models A \ \Rightarrow\ \mathcal{M}[x : W] \models C) \\
& \Rightarrow & \mathcal{M} \models \forall x.(A \supset C)
\end{array}
$$

Note the second line to the last uses the upper-closure of C w.r.t. x. (3) is immediate by the construction of $\vdash^{\mathsf{wpc}}$. (4) is by induction and is given in Appendix B. Finally (5) follows from (4), through (1-ii) and noting $g \bullet u \Downarrow$ is thin. □

Below we prove the relative completeness for values. This result is not restrictive in practice since any program can be transformed to values by (vacuous) abstraction.

Theorem 21 (relative completeness, values). *If $\models [\mathsf{T}]\,V :_u [A]$ such that A is upward-closed at u, then $\vdash [\mathsf{T}]\,V :_u [A]$.*

Proof. Let $\models [\mathsf{T}]\,V :_u [A]$. Calculate A_0 by $\vdash^{\star} [\mathsf{T}]\,V :_u [A_0]$. By Lemma 20, we have $\vdash [\mathsf{T}]\,V :_u [A_0]$. By definition $A_0 \supset A$, hence $\vdash [\mathsf{T}]\,V :_u [A]$, as required. □

For extending relative completeness to general programs, we cannot use the same arguments as in the proof of Theorem 12 because the provability of WPC in Lemma 20 (4) uses thin-ness of the postcondition. We can however use the strengthened consequence rule, [*Cons-Eval*] in Appendix A to derive $[C]\,M :_u [C']$ from $\vdash [C_0]\,M :_u [C'_0]$ where $\vdash^{\star} [C_0]\,M :_u [C'_0]$ as far as C' is upper-closed at u.

Theorem 22 (relative completeness, general programs). *If $\models [C]\,M :_u [C']$ such that C is upward-closed at u, then $\vdash [C]\,M :_u [C']$ in the proof system with [*Cons-Eval*].*

Proof. Suppose $\vdash^{\star} \models [C_0]\,M :_u [C'_0]$. By Lemma 20, we know $\vdash \models [C_0]\,M :_u [C'_0]$. By (C_0, C'_0) being a TCAP, we know, with $\tilde{i}$ auxiliary, $\mathcal{M} \models \forall \tilde{i}.([C_0]m \bullet () = u[C'_0])$ implies the projection of $\mathcal{M}$ to m is more defined than $\lambda().M$ under $\mathcal{M}/m$. Hence we have $\mathcal{M} \models \forall \tilde{i}.([C]m \bullet () = u[C'])$. (note C' is upper-closed w.r.t. u). We can now use [*Cons-Eval*] to obtain $\vdash^{\star} \models [C]\,M :_u [C']$, as required. □

References

1. M. Berger, K. Honda, and N. Yoshida. A logical analysis of aliasing for higher-order imperative functions. In *ICFP'05*, pages 280–293, 2005. Full version is available at: www.dcs.qmul.ac.uk/˜kohei/logics.
2. C. A. Gunter. *Semantics of Programming Languages*. MIT Press, 1995.
3. K. Honda, M. Berger, and N. Yoshida. Descriptive and relative completeness for logics for higher-order functions. In *ICALP'06*, volume 4052 of *LNCS*, pages 360–371, 2006.
4. K. Honda, N. Yoshida, and M. Berger. An observationally complete program logic for imperative higher-order functions. In *Proc. LICS'05*, pages 270–279, 2005. Full version is is available at: www.dcs.qmul.ac.uk/˜kohei/logics.
5. B. C. Pierce. *Types and Programming Languages*. MIT Press, 2002.
6. N. Yoshida, K. Honda, and M. Berger. Logical reasoning for functions with local state. http://www.doc.ic.ac.uk/˜yoshida/local.

A Proof Rules

Figure 2 presents all compositional proof rules for the logic for aliasing (treated in Section 2 and its extension with local state (treated in Section 3). We assume that judgements are well-typed in the sense that, in $[C]M :_u [C']$ with $\Gamma;\Delta \vdash M : \alpha, \Gamma, \Delta, \Theta \vdash C$ and $u : \alpha, \Gamma, \Delta, \Theta \vdash C'$ for some Θ s.t. $\mathsf{dom}(\Theta) \cap (\mathsf{dom}(\Gamma, \Delta) \cup \{u\}) = \emptyset$.

In the rules, $C^{\neg \tilde{x}}$ indicates $\mathsf{fv}(C) \cap \{\tilde{x}\} = \emptyset$. Symbols $i, j, \ldots$ range over auxiliary names. In [*Abs, Rec*], A, B denote *stateless* formulae: C is stateless when $\Box C \equiv C$. [*Assign*] uses *logical substitution* which is built with content quantification to represent substitution of content of a possibly aliased reference [1].

$$C\{\!|e_2/!e_1|\!\} \stackrel{\text{def}}{=} \forall m.(m = e_2 \supset [!e_1]^{\circ}(!e_1 = m \supset C)).$$

with m fresh.

When we are in the logic with local state, we assume C' is thin in [*App*], [*Deref*] and [*Assign*].

B Proofs for Lemma 20 (4,5)

We prove this with the second statement of (5) ((A) below) simultaneously.

(A) $\vdash^{\star\star} [\mathsf{T}]\,[\![V]\!] :_u [A]$ implies $\vdash [\mathsf{T}]V :_u [A]$.
(B) If $\vdash^{\mathsf{wpc}} [C]\,[\![M]\!] :_u [C']$ and if C' is thin, then $\vdash [C]M :_u [C']$.

Case x**:** Obvious by letting $C = u = y$.

Case c**:** Obvious by letting $C = u = \mathsf{c}$.

Case $\lambda y.[\![N]\!]$**:** Suppose $\vdash^{\star\star} [\mathsf{T}]\lambda y.[\![N]\!] :_u [\Box \forall \tilde{y}\tilde{i}.(C \supset u \bullet y = m[C'])]$ is derived from $\vdash^{\mathsf{wpc}} [C]\,[\![N]\!] :_m [C']$ with $\tilde{i} = \mathsf{fv}(C, C') \setminus (\mathsf{fv}([\![N]\!]) \cup \{m\})$. Since C' is thin, by induction by (B), we know $\vdash [C]N :_m [C']$.

Case $\mu f.\lambda x.N$**:** The same as the above case.

(B)

Case M_1M_2**:** Let $\vdash^{\mathsf{wpc}} [C]\,[\![M_1M_2]\!] :_u [C']$. Then by definition, $[\![M_1M_2]\!] \stackrel{\text{def}}{=} \langle\!\langle M_1M_2 \rangle\!\rangle_x[x] \stackrel{\text{def}}{=} \langle\!\langle M_1 \rangle\!\rangle_{m_1}[\langle\!\langle M_2 \rangle\!\rangle_{m_2}[\mathtt{let}\ x = m_1m_2\ \mathtt{in}\ x]]$, and by Lemma 20 (3), we know:

Fig. 2 Proof Rules for Local PCFv.

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

$$[Add]\ \frac{[C]\, M_1 :_{m_1} [C_0] \quad [C_0]\, M_2 :_{m_2} [C'[m_1 + m_2/u]]}{[C]\, M_1 + M_2 :_u [C']}$$

$$[If]\ \frac{[C]\, M :_b [C_0] \quad [C_0[\mathsf{t}/b]]\, M_1 :_u [C'] \quad [C_0[\mathsf{f}/b]]\, M_2 :_u [C']}{[C]\, \mathtt{if}\ M\ \mathtt{then}\ M_1\ \mathtt{else}\ M_2 :_u [C']}$$

$$[Abs]\ \frac{[A^{\text{-}x\tilde{X}\tilde{i}} \wedge C]\, M :_m [C'] \quad \{\tilde{i}\tilde{X}\} \subset \mathsf{fv}(C, C') \setminus (\mathsf{fv}(M) \cup \{m\})}{[A]\, \lambda x.M :_u [\Box \forall x \tilde{X} \tilde{i}.(C \supset u \bullet x = m[C'])]} \qquad [Rec]\ \frac{[C^{\text{-}f}]\, \lambda x.M :_u [C']}{[C]\, \mu f.\lambda x.M :_u [C'[u/f]]}$$

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

$$[Deref]\ \frac{[C]\, M :_m [C'[!m/u]]}{[C]\, !M :_u [C']} \qquad [Assign]\ \frac{[C]\, M :_m [C_0] \quad [C_0]\, N :_n [C'\{\!|n/\, !m|\!\}]}{[C]\, M := N :_u [C']}$$

$$[Ref]\ \frac{[C]\, M :_m [C']}{[C]\, \mathtt{ref}(M) :_u [\forall x.(C'[!u/m] \ \wedge\ u \# i^{\mathrm{X}} \wedge u = x)]}$$

$$[Cons\text{-}Eval]\ \frac{[C_0]\, M :_m [C_0'] \quad \forall \tilde{i}.[C_0]\ x \bullet () = m\ [C_0'] \supset \forall \tilde{i}.[C]\ x \bullet () = m\ [C'] \ \ x \text{ fresh, } \tilde{i} \text{ auxiliary}}{[C]\, M :_m [C']}$$

- $\vdash^{\mathsf{wpc}} [C]\, [\![M_1]\!] :_{m_1} [C_1]$, and $\vdash^{\mathsf{wpc}} [C_1]\, [\![M_2]\!] :_{m_2} [C_2]$
- $\vdash^{\mathsf{wpc}} [C_2]\, \mathtt{let}\ x = m_1 m_2\ \mathtt{in}\ x :_u [C']$ with $C_2 = m_1 \bullet m_2 = u[C']$.

Then by induction, we know $\vdash [C]\, M_1 :_{m_1} [C_1]$ and $\vdash [C_1]\, M_2 :_{m_2} [m_1 \bullet m_2 = u[C']]$. Now applying [*App*] in Figure 2, we conclude this case.

Case $M_1 := M_2$**:** The same as the above case by replacing by $\mathtt{let}\ x = m_1 := m_2\ \mathtt{in}\ x$ and $C_2 = C'\{\!|m_2/\, !m_1|\!\}$.

Case $M_1 + M_2$**:** The same as the above case by replacing by $\mathtt{let}\ x = m_1 + m_2\ \mathtt{in}\ x$ and $C_2 = C'[m_1 + m_2/u]$.

Case $!N$**:** Let $\vdash^{\mathsf{wpc}} [C]\, [\![!N]\!] :_u [C']$. Then by definition, $[\![!N]\!] \stackrel{\text{def}}{=} \langle\!\langle !N \rangle\!\rangle_x[x] \stackrel{\text{def}}{=} \langle\!\langle N \rangle\!\rangle_n[\mathtt{let}\ x = !n\ \mathtt{in}\ x]$, and by Lemma 20 (3), we know:

- $\vdash^{\mathsf{wpc}} [C]\, [\![N]\!] :_n [C_1]$
- $\vdash^{\mathsf{wpc}} [C_1]\, \mathtt{let}\ x = !n\ \mathtt{in}\ x :_u [C']$ with $C_1 = C'[!n/u]$.

Then by induction, we know $\vdash [C]\, N :_n [C'[!n/u]]$. By [*Deref*] in Figure 2, we have $\vdash [C]\, !N :_u [C']$, as required.

Case $\mathtt{if}\ M_0\ \mathtt{then}\ N_1\ \mathtt{else}\ N_2$**:** Let $\vdash^{\mathsf{wpc}} [C]\, [\![\mathtt{if}\ M_0\ \mathtt{then}\ N_1\ \mathtt{else}\ N_2]\!] :_u [C']$. Then by definition, $\langle\!\langle \mathtt{if}\ M_0\ \mathtt{then}\ N_1\ \mathtt{else}\ N_2 \rangle\!\rangle_x[x] \stackrel{\text{def}}{=} \langle\!\langle M_0 \rangle\!\rangle_m[\mathtt{if}\ m\ \mathtt{then}\ \langle\!\langle N_1 \rangle\!\rangle_x[x]\ \mathtt{else}\ \langle\!\langle N_2 \rangle\!\rangle_x[x]]$, and by Lemma 20 (3), we know:

- $\vdash^{\mathsf{wpc}} [C]\, [\![M_1]\!] :_x [C_0]$,

– $\vdash^{\mathsf{wpc}} [C_1]\,[\![N_1]\!] :_u [C']$, and $\vdash^{\mathsf{wpc}} [C_2]\,[\![M_2]\!] :_u [C']$ with $C_0 = \bigwedge_{i=1,2}(x = \mathsf{b_i} \supset C_i)$ and $\mathsf{b}_1 = \mathsf{t}, \mathsf{b}_2 = \mathsf{f}$.

We note that $C_0[\mathsf{t}/x] \equiv ((\mathsf{t} = \mathsf{t}) \supset C_1) \ \wedge \ ((\mathsf{t} = \mathsf{f}) \supset C_2) \equiv C_1$. Hence by induction, we know $\vdash [C]\,M_1 :_x [C_0]$, $\vdash [C_0[\mathsf{t}/x]]\,[\![N_1]\!] :_u [C']$, and $\vdash [C_0[\mathsf{f}/x]]\,[\![N_2]\!] :_u [C']$. By [*If*] in Figure 2, we conclude the case.

Case $\mathtt{ref}(N)$**:** Let $\vdash^{\mathsf{wpc}} [C]\,[\![\mathtt{ref}(N)]\!] :_u [C']$. Then by definition, $[\![\mathtt{ref}(N)]\!] \stackrel{\text{def}}{=} \langle\!\langle \mathtt{ref}(N) \rangle\!\rangle_x[x] \stackrel{\text{def}}{=} \langle\!\langle N \rangle\!\rangle_n[\mathtt{let}\ x = \mathtt{ref}(n)\ \mathtt{in}\ x]$, and by Lemma 20 (3), we know:

– $\vdash^{\mathsf{wpc}} [C]\,[\![N]\!] :_n [C_1]$
– $\vdash^{\mathsf{wpc}} [C_1]\,\mathtt{let}\ x = \mathtt{ref}(n)\ \mathtt{in}\ x :_u [C']$ with $C_1 = \mathsf{new}(x)(C'[x/u] \wedge !x = n)$.

Then by induction, we know $\vdash [C]\,N :_n [C_1]$. Then by applying [*Ref*], we have

$$\vdash [C]\,\mathtt{ref}(N) :_u [\forall x.(\mathsf{new}(x)(C'[x/u][!u/n] \wedge !x = !u) \ \wedge \ u\#i \ \wedge \ u = x)]$$

Note that $C' = g \bullet u \Downarrow$, hence $C'[x/u][!u/n] = g \bullet x \Downarrow$. Note u is fresh $u\#i$ and $\mathsf{new}(x)(g \bullet x \Downarrow \ \wedge !x = !u)$ means that fresh x is identical with u, and by [*Cons-Eval*], we derive:

$$\vdash [C]\,\mathtt{ref}(N) :_u [g \bullet u \Downarrow]$$

as desired.