B. Appendix: Models

B.1 Observational Congruence

Programs Write \((v^2)(M, \sigma) \Downarrow (v^2)(V, \sigma')\) for \((v^2)(M, \sigma) \rightarrow^* (v^2)(V, \sigma')\); and \((v^2)(M, \sigma) \Downarrow (v^2)(V, \sigma')\) for some \((v^2)(V, \sigma')\). Let \(\Gamma; \Delta \vdash M_{i,2} : \alpha\). Then we write \(\Gamma; \Delta \vdash (v^2)(M_{i,1}, \sigma_1) \equiv (v^2)(M_{2,2}, \sigma_2)\) if, for each typed closing context \(C.\) of type Unit which is typeable under \(\Delta\) and in which no labels from \(I_{1,2}\) occur, we have:

\[
(v^2)(C[M_{1,1}], \sigma_1) \Downarrow \text{iff } (v^2)(C[M_{2,2}], \sigma_2) \Downarrow
\]

We often write \((v^2)(M_i, \sigma_i) \equiv (v^2)(M_i', \sigma_i)\), leaving type information implicit.

Models Given models \(M[\Gamma; \Delta] = (v^2)(\{y_i : V_{i,1}, \ldots, \} : V_{i,m})\), for \(i = 1, 2\), we set \(\Gamma; \Delta \vdash M_i \equiv M_{i,2}\) iff

\[
(v^2)(\{V_{i,1}, \ldots, V_{i,m}\}, \sigma_i) \equiv (v^2)(\{V_{2,1}, \ldots, V_{2,m}\}, \sigma_2)
\]

B.2 Semantics

Let \(\Gamma; \Delta \vdash e : \alpha; \Gamma; \Delta \vdash M\) and \(M = (v^2)(\xi, \sigma)\). Then the interpretation of \(e\) under \((\xi, \sigma)\), denoted \(\llbracket e \rrbracket_{\xi, \sigma}\), is inductively given by:

1. \(\llbracket x \rrbracket_{\xi, \sigma} = \xi(x)\)
2. \(\llbracket \xi \rrbracket_{\xi, \sigma} = \xi\)
3. \(\llbracket e_1 \rrbracket_{\xi, \sigma} = n\)
4. \(\llbracket b \rrbracket_{\xi, \sigma} = b\)
5. \(\llbracket \text{op}(e) \rrbracket_{\xi, \sigma} = \text{op}(\llbracket e \rrbracket_{\xi, \sigma})\)
6. \(\llbracket \pi(e) \rrbracket_{\xi, \sigma} = \pi(\llbracket e \rrbracket_{\xi, \sigma})\)
7. \(\llbracket \text{inj}(e) \rrbracket_{\xi, \sigma} = \text{inj}(\llbracket e \rrbracket_{\xi, \sigma})\)

Then we define semantics of the assertions (the notations are illustrated below); All omitted cases are by de Morgan duality. Let \(u, u'\) be fresh.

- \(\Vdash e_1 = e_2\) if \(\Vdash \alpha = \alpha\).
- \(\Vdash C_1 \land C_2\) if \(\Vdash C_1\) and \(\Vdash C_2\).
- \(\Vdash \neg C\) if not \(\Vdash C\).
- \(\Vdash \forall x.e\) if \(\forall e(x) \Vdash C\).
- \(\Vdash \exists x.e\) if \(\exists e(x) \Vdash C\).

Above we use the following notations (assuming well-typedness):

\[
M[x := a] \Downarrow M' [x := a]
\]

\[
\llbracket M rbracket_{\xi, \sigma} = \text{fl}(\llbracket M \rrbracket_{\xi, \sigma})
\]

iff it satisfies the clause of the evaluation formula above and the following, letting \(M_0 \equiv (v^2)(\xi, \sigma_0)\) and \(M' \equiv (v^2)(\xi, \sigma')\),

\[
\llbracket \text{inj}(\llbracket e \rrbracket_{\xi, \sigma} | I \rightarrow V) \rrbracket \equiv (v^2)(\llbracket \xi, \sigma_0[l] | I \rightarrow V)\]

\(i \in \{I\}\) where \(l \in \{I\}\). If \(\llbracket (\xi, l, \sigma_0) \rrbracket = E\) this says:

The value stored at each location \(z\) satisfying \(\neg E(z)\) in \(M_0\), is exactly preserved when the application at \(M_0\) results in \(M'\), taking \(M'\) up to \(\approx\).

For formal details, see [2, C.2].

C. Derivations for Examples in Section 5

This appendix lists the derivations omitted in Section 5.

C.1 Derivation for [LetRef]

We can derive [LetRef] as follows.

1. \(\{C\} M_{\text{in}} \{C_0\}\) (premise)
2. \(\{C_0[x/m] \land x \# \tilde{e}\} N_{\text{in}} \{C'\}\) with \(x \notin \text{fn}(\tilde{e})\) (premise)
3. \(\{C\} \text{ref}(\tilde{M}) \vdash \{\#x.C_0[x/m]\}\) (1,Ref)
4. \(\{C\} \text{ref}(\tilde{M}) \vdash \{\#x.(C_0[x/m] \land x \# \tilde{e})\}\) (Subs n-times)
5. \(\{C\} \text{ref}(\tilde{M}) \vdash \{\forall y.((C_0[x/m] \land x \# \tilde{e}) \land y)\} \{C' \land y = x\}\) (Conseq)
6. \(\{C_0[x/m] \land x \# \tilde{e} \land y = x\} N_{\text{in}} \{C' \land y = x\}\) (2, Invariance)
7. \(\{C\} \text{let } x = \text{ref}(\tilde{M}) \in N_{\text{in}} \{\forall x.\tilde{C}'\}\) (5, LetOpen)
8. \(\{C\} \text{let } x = \text{ref}(\tilde{M}) \in N_{\text{in}} \{\forall x.\tilde{C}'\}\) (Conseq)

Lines 5 and 8 use the standard logical law (discussed below). Lines 4 and 7 use the following derived/admissible proof rules:

\[
\text{[Subst]}\{C\} M_{\text{in}} \{C'\} \quad u \notin \text{fn}(\tilde{e})
\]
\[
\{C\} M_{\text{in}} \{C'\} \quad \{C' \land y = x\}\]

\[
\text{[LetOpen]}\{C\} \text{let } x = M \in N_{\text{in}} \{\forall x.\tilde{C}'\}\]

[LetOpen] opens the “scope” of \(\tilde{y}\) to \(N\). The crucial step is Line 5, which turns freshness “\#” into locality “\&” through the standard law of equality and existential, \(C \equiv \exists y. (C \land y = x)\) with \(y\) fresh.

C.2 Derivation for IncUnShared

For illustration, we contrast the inference of IncShared with:

\[
\text{IncUnShared} \equiv a := \text{Inc}; b := \text{Inc}; c_1 := \text{Inc}(a); c_2 := \text{Inc}(b); (c_1 \land c_2)
\]

This program assigns to \(a\) and \(b\) two separate instances of \(\text{Inc}\). This lack of sharing between \(a\) and \(b\) in IncUnShared is captured by the following derivation:

\[
\{T\} \text{Inc}_{\text{in}} \{\forall x.\text{inc}(\! a \times u, 0\})
\]
\[
\{T\} a := \text{Inc} \{\forall x.\text{inc}(\! a \times u, 0\})
\]
\[
\{C_0[\! \langle a \times u, 0\rangle \} b := \text{Inc} \{\forall y.\text{inc}(\! b \times y, 0\})
\]
\[
\{C_0[\! \langle a \times u, 0\rangle \} c_1 := \text{Inc}(\! a\{\! \langle 0 \rangle \} \land c_1 = 1\}
\]
\[
\{C_0[\! \langle a \times u, 0\rangle \} c_2 := \text{Inc}(\! b\{\! \langle 1 \rangle \} \land c_2 = 1\}
\]
\[
\{c_1 = 1 \land c_2 = 1\} \{c_1 \land c_2 \land u = 2\}
\]
\[
\{T\} \text{IncUnShared}_{\text{in}} \{\forall x.u = 2\}
\]
\[
\{T\} \text{IncUnShared}_{\text{in}} \{u = 2\}
\]

Above \(\text{inc}^n(\!a \times m\}) = \text{inc}(\! a \times n, x) \land \text{inc}(\! b \times y, m) \land u \neq y\). Note \(x \neq y\) is guaranteed by [LetRef]. This is in contrast to the derivation for IncShared, where, in Line 3, \(x\) is automatically shared after “\(b := !a\)” which leads to scope extrusion.
The derivation is similar to safeFact.
1. \{T\} \lambda n. t ; m (T) @ 0

2. \{T\} mutualParity; \![y] ; m \{\exists gh.IsOddEven(gh, gu, xy, n)\} @ xy

3. \{T\} mutualParity; \![y] ; m \{\exists gh.(Even_a \land Even_b)\} @ xy

4. \{\![xy \# i]\} mutualParity; \![y] ; m \{\exists gh.(xy \# i) \land Even_a \land Even_b\} @ xy

5. \{T\} safeEven ; v \{v \# xy \exists gh.(Even_a \land Even_b)\} @ 0

6. \{T\} m;i = u \{\!v \# xy \exists gh.(Even_a \land Even_b)\}

\cup \{T\} m;i = u \{ValEven(u)\} \quad \text{(by (AIH))}

7. \{T\} safeEven ; u \{ValEven(u)\} @ 0

C.4 Derivation for Meyer-Seiber

For the derivation of (5.6) we use (ε is the empty string): I = lnw(f(\text{Even}(x)), x, e, ε). G_0 = \{\text{Even}(x) \land x \# g \} \ast f(\text{Even}(x))\}, and G_1 = \{T\} g \ast f(\text{T} \text{. The derivation follows. Below} M_1, 2 \text{ is the}
body of the first/second lets, respectively.

\[ \{ \text{Even}(x) \land G_0 \} \ g f \ \{ \text{Even}(x) \} \]

(App)

\[ \{ \text{Even}(x) \land I \land G_1 \} \ g f \ \{ \text{Even}(x) \} \]

(1, Conseq)

\[ \{ E \land \lambda [x]C \land \lambda _x \# g \} \ g f \ \{ C' \} \ @ \ ^\uparrow w x \]

(App)

\[ \{ E \land [x]C \land \lambda _x \# g \} \ g f \ \{ \text{Even}(x) \land C' \} \ @ \ ^\uparrow w x \]

(2, 3, Conj)

\[ \{ \text{Even}(x) \land C' \} \ if \ even \ (x) \ then \ () \ else \ \Omega \} \ \{ C' \} \ @ \ ^\uparrow \Theta \]

(IF)

\[ \{ E \land [x]C \land \lambda _x \# g \} M_2 \{ C' \} \ @ \ ^\uparrow w x \]

(4, 5, Seq)

\[ \{ \text{Even}(x) \} \Lambda \{ x \} := \{ x + 2 \ ; \ ^\uparrow I \} \ @ \ ^\uparrow \Theta \]

(Abs etc.)

\[ \{ E \land [x]C \land \text{Even}(x) \land \lambda _x \# g \} M_1 \{ C' \} \ @ \ ^\uparrow w x \]

(7, 6, LetRef)

\[ \{ E \land C \} \ 0 : n \ { E \land C \land \text{Even}(m) } \ @ \ ^\uparrow \Theta \]

(Const)

\[ \{ E \land C \} \ \text{MeyerSieber} \ \{ C' \} \ @ \ ^\uparrow w \]

(9, LetRef)

Line 2 uses the axiom in Proposition 9. Line 4 uses the standard structural rule. Line 10 cancels \([x]C\) from \([x]C\) which is possible since \(m\) does not occur in \(C\).

C.5 Derivation for Object

We need the following generalisation: The procedure \(u\) in \((\text{AH})\) is of a function type \(\alpha \rightarrow \beta\); when values of other types such as \(\alpha \times \beta\) or \(\alpha + \beta\) are returned, we can make use of a generalisation. For simplicity we restrict our attention to the case when types do not contain recursive or reference types.

\[
\begin{align*}
\text{inv}(u^\alpha & \times \beta, C_0, \tilde{r}, \tilde{w}) = \\
\text{inv}(u^\alpha + \beta, C_0, \tilde{r}, \tilde{w}) &= \land (u = \text{inj}_2(y)) \lor \text{inv}(y, C_0, \tilde{r}, \tilde{w}))
\end{align*}
\]

Using this extension, we can generalise \((\text{AH})\) so that the cancelling of \(C_0\) is possible for all components of \(u\). For example, if \(u\) is a pair of functions, those two functions need to satisfy the same condition as in \((\text{AH})\). This is what we shall use for \(\text{cellGen}\). We call the resulting generalised axiom \((\text{AH}_G)\).

Let \(\text{cell}\) be the internal \(\lambda\)-abstraction of \(\text{cellGen}\). First, it is easy to obtain:

\[
\{ \text{T} \} \ \text{cell} \ {}_\gamma \ \{ I_0 \land G_1 \land G_2 \land E' \}
\]

where, with \(I_0 = \{ x_0 = _! x_1 \} \land E' = \{ x_0 = z \} .

\[
\begin{align*}
G_1 &= \{ I_0 \} \pi_1 (o) () = v \{ v = _! x_1 \} \ @ \ ^\uparrow \Theta \\
G_2 &= \forall w. \{ I_0 \} \pi_1 (o) \bullet w \{ _! x_0 = w \land I_0 \} @ _! x_0
\end{align*}
\]

which will become, after taking off the invariant \(I_0\):

\[
\begin{align*}
G'_1 &= \{ \text{T} \} \pi_1 (o) () = v \{ v = _! x_1 \} \ @ \ ^\uparrow \Theta \\
G'_2 &= \forall w. \{ \text{T} \} \pi_1 (o) \bullet w \{ _! x_0 = w \} @ _! x_0
\end{align*}
\]

Note \(I_0\) is stateless except \(x_0\). In \(G_1\), notice the empty write set means \(x_1\) does not change from the pre to the postcondition. We now present the inference. We set \(\text{cell}' \ \text{def} \ \text{let} \ y = \text{ref}(0) \) in \(\text{cell}\) below.

\[
\{ \text{T} \} \ \text{cell} {}_\gamma \ \{ I_0 \land G_1 \land G_2 \land E' \}
\]

\[
\{ \text{T} \} \ \text{cell}' {}_\gamma \ \{ I_0 \land G_1 \land G_2 \land E' \}
\]

(LetRef)

\[
\{ \text{T} \} \ \text{let} \ x_1 = z \ \text{in} \ \text{cell}' {}_\gamma \ \{ \forall x_1. (I_0 \land G_1 \land G_2) \land E' \}
\]

(LetRef)

\[
\{ \text{T} \} \ \text{let} \ x_1 = z \ \text{in} \ \text{cell}' {}_\gamma \ \{ G'_1 \land G'_2 \land E' \}
\]

(\text{AH}_G, \text{ConsEval})

\[
\{ \text{T} \} \ \text{let} \ x_{0,1} = z \ \text{in} \ \text{cell}' {}_\gamma \ \{ \forall x_1. (G'_1 \land G'_2 \land E') \}
\]

(LetRef)

\[
\{ \text{T} \} \ \text{cellGen} {}_\gamma \ \{ \text{CellGen}(u) \}
\]

(Abs)