

Environments and Completeness

April 18, 1989

1 Terminology

Definition 1.1 An *environment* E_i is a pair $\langle D_i, U_i \rangle$, where D_i is a set $\{d_{i1}, \dots, d_{iin}\}$ of *defined names* and U_i is a set $\{u_{i1}, \dots, u_{iim}\}$ of *free names* (or *uses*).

Consider a set of environments $\{E_i\}$. Let $M : \text{Ints} \times \text{Ints} \rightarrow \text{Ints} \times \text{Ints} \cup \{\perp\}$ give a desired binding of every free function name u_{ij} to a defined name d_{hk} . In other words, $M(i, j) = \langle h, k \rangle$ indicates that the free name u_{ij} should receive the definition d_{hk} . If $M(i, j) = \perp$, then u_{ij} should remain free.

Fix a set of environments. The question is: for every M , is there a program which makes the bindings specified by M ? To make this question precise, we must say what it means for a program to “make a binding”.

Let a *binding* be a partial function of type $\text{Names} \rightarrow \text{Ints} \times \text{Ints} \cup \{\perp\}$. A binding is associated with every free name; what binding is associated with a free name is determined by the programming language. We assume that a program consists of environments put together with operations that create bindings. The operations determine what definitions are accessible to the free names.

Definition 1.2 A free function name u_{ij} *sees* d_{hk} if u_{ij} is associated with an binding B where $B(u_{ij}) = \langle h, k \rangle$.

Definition 1.3 A programming language is *complete* for environments if for every set of environments $\{E_i\}$ and possible M , there is a program containing one occurrence of each of the E_i such that

$$M(i, j) = \langle h, k \rangle \Leftrightarrow u_{ij} \text{ sees } d_{hk}$$

In what follows, we examine the completeness of sets of operations on bindings.

2 A Complete System

The section defines a few operations on bindings. These operations have the functionality $(\text{Binding} \rightarrow \text{Binding})^* \rightarrow (\text{Binding} \rightarrow \text{Binding})$; that is, an operation takes a list of functions from bindings to bindings and returns a function from bindings to bindings. A program is understood to be a function of type $\text{Binding} \rightarrow \text{Binding}$ evaluated in the everywhere undefined binding $\lambda b. \perp$.

Let b_1 and b_2 be bindings. The notation $b_1[s \leftarrow i]$ stands for the function b_1 updated at s to return i . The auxiliary functions b_1/b_2 (b_1 modified by b_2) and $b_1|(\dots, n_i, \dots)$ (b_1 restricted to a set of names) are defined below.

$$\begin{aligned} b_1/b_2 &= b_1[s \leftarrow i] \quad \forall s \text{ s.t. } b_2(s) = i \wedge i \neq \perp \\ b_1|(\dots, n_i, \dots) &= \lambda n. \perp [n_i \leftarrow b_1(n_i)] \end{aligned}$$

Every environment E_i defines a function from bindings to bindings $\lambda b. \perp [d_{hk} \leftarrow \langle h, k \rangle]$. Every free name in E_i sees b . We now define the operations **union**, **uses**, **hide**, **rec**, and **alias**.

Definition 2.1 (union)

$$A \text{ union } B = \lambda b. A(b) \sqcup B(b)$$

If $A(b) \sqcup B(b)$ is undefined (i.e., A and B define the same name), then the result is undefined.

Definition 2.2 (uses)

$$A \text{ uses } B = \lambda b. B(b)/A(B(b))$$

Definition 2.3 (hide)

$$\text{hide}(n_1, \dots, n_k)A = \lambda b. A(b)[n_i \leftarrow \perp]$$

Definition 2.4 (rec) Let $A_{\text{rec}} = (A(\lambda b. \perp))|(\dots, n_i, \dots)$.

$$\text{rec}(\dots, n_i, \dots)A = \lambda b. A(b/A_{\text{rec}})$$

Definition 2.5 (alias)

$$\text{alias}(\dots, \langle n_i, m_i \rangle, \dots)A = \lambda b. A(b)[m_i \leftarrow A(b)(n_i)]$$

unless $A(b)(n_i) = \perp$, in which case the result is undefined.

In the following, curly braces $\{\dots\}$ are used to show precedence, and empty braces $\{\}$ denote the empty environment.

Theorem 2.6 The system consisting of **union**, **uses**, **hide**, **rec**, and **alias** is complete.

Proof: Let v_{ij} be a fresh (unused) name. For each E_i , let A_i be

$$\{\text{hide}(d_{ik})\{\text{alias}(\langle d_{ik}, v_{ik} \rangle)E_i\}\} \text{ uses } \{\text{alias}(\langle v_{M(i,j)}, u_{ij} \rangle)\{\}\}$$

The program

$$\text{rec}(v_{ij})\{\dots \text{union } A_i \text{ union } \dots\}$$

satisfies the theorem. $\square$

3 Uses is not primitive

The operator **uses** can be implemented by **alias**, **rec**, and **union**. Let w_i , z_i , and x_i be fresh names. Let A and B be environments, with defined names d_{ai} and d_{bi} and free names u_{ai} and u_{bi} respectively. Then

$$\begin{aligned} A \text{ uses } B &= \lambda b. \text{hide}(w_i)\{\text{alias}(\langle w_i, d_{ai} \rangle) \\ &\quad \{\text{hide}(x_i)\{\text{alias}(\langle x_i, d_{bi} \rangle)\{\text{rec}(x_i)\{A'(b) \text{ union } B'(b)\}\}\}\}\} \end{aligned}$$

where

$$A' = \lambda b. \text{rec}(u_{ai})\{\{\text{hide}(d_{ai})\{\text{alias}(\langle d_{ai}, w_i \rangle)A(b)\}\} \text{ union } \{\text{hide}(u_{ai})\{\text{alias}(\langle u_{ai}, x_j \rangle)\{\}\}\}\}$$

where $d_{bj} = u_{ai}$, and where

$$B' = \lambda b. \text{rec}(u_{bi})\{\{\text{hide}(d_{bi})\{\text{alias}(\langle d_{bi}, x_i \rangle)B(b)\}\} \text{ union } \{\text{hide}(z_i)\{\text{alias}(\langle z_i, u_{bi} \rangle)\{\}\}\}\}$$

4 Necessity

The following lemmas show that each of **rec**, **union**, **alias**, and **hide** are necessary.

Lemma 4.1 **rec** is necessary.

Proof: Let $E = \langle \{f\}, \{f\} \rangle$. Let $M(f) = f$. No combination of **union**, **alias**, and **hide** can implement this binding. With **rec**:

$$\text{rec}(f)E$$

$\square$

Lemma 4.2 **union** is necessary.

Proof: Let $E_1 = \langle \{f\}, \{\} \rangle$ and let $E_2 = \langle \{g\}, \{\} \rangle$. Let $M = \lambda x. \perp$. No combination of **rec**, **alias**, or **hide** can implement a program with this (null) binding, because it is impossible to introduce more than one environment using these operators. With **union**:

$$E_1 \text{ union } E_2$$

□

Lemma 4.3 **alias** is necessary.

Proof: Let $E_1 = \langle \{f\}, \{\} \rangle$ and let $E_2 = \langle \{\}, \{g\} \rangle$. Let $M(g_2) = f_1$. No combination of **rec**, **hide**, and **union** can implement this binding, because without **alias** a name can only be bound to another occurrence of the same name. With **alias**:

$$\text{rec}(g) \{ \{ \text{alias}(\langle f, g \rangle) E_1 \} \text{ union } E_2 \}$$

□

Lemma 4.4 **hide** is necessary.

Proof: Let $E_1 = \langle \{f\}, \{\} \rangle$ and let $E_2 = \langle \{f\}, \{\} \rangle$. Let $M = \lambda x. \perp$. No combination of **rec**, **union**, and **alias** can build a program with these two environments, because **alias** merely adds aliases for a name. Thus, a **union** of any renaming of these two environments is undefined. With **hide**:

$$E_1 \text{ union } \{ \text{hide}(f) E_2 \}$$

or

$$E_2 \text{ union } \{ \text{hide}(f) E_1 \}$$

□

As a byproduct of these proofs, we see that the **include** facility cannot realize the effect of any of the necessary constructs.