Semantics of the typed -calculus with substitution in a cartesian closed category

J.W. Gray

UNITÉ DE RECHERCHE
INRIA-ROCQUENCOURT

Institut National
de Recherche
en Informatique
et en Automatique

Domaine de Voluceau
Rocquencourt
B.P. 105
78153 Le Chesnay Cedex
France
Tél.:(1) 39 63 55 11

Rapports de Recherche

N° 1261

Programme 1
Programmation, Calcul Symbolique
et Intelligence Artificielle

SEMANTICS OF THE TYPED λ-CALCULUS WITH SUBSTITUTION IN A CARTESIAN CLOSED CATEGORY

John W. GRAY

Juillet 1990

RR-1261

SEMANTICS OF THE TYPED λ-CALCULUS WITH SUBSTITUTION IN A CARTESIAN CLOSED CATEGORY.

by John W. Gray
UIUC and INRIA

Abstract: Two versions of semantics of the typed λ-calculus with substitution are given. The first uses a global environments object which is the domain of all interpreted terms, while the second uses individual environments objects for each term. The algebraic properties of the environments object(s) play a central role in showing that the interpretation function is invariant under the operational semantics given by substitution.

SEMANTIQUE DU λ-CALCUL TYPÉ AVEC SUBSTITUTION DANS UNE CATEGORIE CARTESIENNE FERMÉE .

John W. Gray
UIUC et INRIA

Resumé: On donne deux versions de sémantique du λ-calcul typé avec substitution. La premiere se sert d'un objet d'environments globale qui et le domain de touts termes interpretés. Dans la deuxieme version il y a des objets d'environments individuels pour chaque terme. Les propriétés algebriques des objets d'environments joue un role principal dans la démonstration de l'invariance de la function d'interpretation sous la semantiques operationelle donné par substitution.

1 INTRODUCTION

By the typed λ-calculus with substitution, we mean the ordinary typed lambda calculus with explicit substitution as in [Abadi et al 90] and [Hardin and Lévy 90] except that we treat the usual lambda calculus with explicit variables (Cf. Hindley and Seldin, 86] whereas their work is all done with de Bruijn notation. The usual rule for β-reduction which says that $(\lambda x \,.\, f)\ g \Rightarrow [\ g\ /\ x\ ]\ f$ treats the substitution of g for x in f as a one step process, even though this substitution is defined by recursive rules depending on the structure of f. It is assumed that these steps are carried out in some meta-realm, independent of the reduction steps involved in β-reducing a term to normal form. But, it is a familiar feature of computer implementations of the λ-calculus that this optimistic assumption leads to many difficulties. (See [Gray 91a and 91b].) For instance, [Hardin and Lévy 90] is the first place where is it shown that if substitutions are explicitely included in the rewrite rules, then they result in a confluent system. (Note that this is specifically for de Bruijn syntax.) We are not concerned with such syntactic results here, but rather with describing a semantics with values in a cartesian closed category for this situation. This aspect of semantics is totally ignored in the standard treatment of the λ-calculus in [Lambek and Scott 86] to which we refer the interested reader. Here we study the *environment* model of the typed lambda calculus in a cartesian closed category. It turns out that there is an *algebra of environments* whose properties are exactly what is needed to interpret substitutions. There are two kinds of environment models, those based on *global environments* and those based on *local envirnoments*. Both kinds are treated here. The collection of rewrite rules for the typed λ-calculus with substitution determines an *operational semantics* for it, symbolized by $f \Rightarrow g$. The *denotational semantics* in a cartesian closed category **C** is indicated by $[\![f]\!]_{\mathbf{C}}$, which assigns objects in **C** to types and morphisms in **C** to terms. The main result is that

if $f \Rightarrow g$, then $[\![f]\!]_{\mathbf{C}} = [\![g]\!]_{\mathbf{C}}$;

i.e., if terms are operationally equivalent, then their denotations in any cartesian closed category are equal.

2 THE SYNTAX AND OPERATIONAL SEMANTICS OF THE λ-CALCULUS WITH SUBSTITUTION.

2.1 Syntax.

Substitution is a basic ingredient on the lambda calculus, and its analysis in this context is one of the main contributions of the λ-calculus to general mathematical understanding. It is the main ingredient of the *operational semantics* of the lambda calculus.

2.1.1 Definition. A *typed λ-calculus* L consists of types and terms of each type.

i) The set Type of types is given recursively as follows:
   a) There is a finite or countable set B of basic types
   b) If σ and τ are types then $[\sigma \to \tau]$ is a type.

   A grammar for types is given by: $\tau ::= b \mid [\tau \to \tau]$, where $b \in B$.

ii) For each type τ, there are
   a) a countable set, Var^{τ}, of variables of type τ,
   b) a finite or countable set, $Const^{\tau}$, of constants of type τ, and

c) the set, $Atom^\tau = Var^\tau + Const^\tau$, of atoms of type τ.

iii) Write $f : \tau$ for "f is a term of type τ". The set $Terms^\tau$ of terms of type τ is described recursively as follows:

a) $Terms^\tau \supseteq Atom^\tau$

b) If $f : [\sigma \to \tau]$ and $g : \sigma$, then $(f\ g) : \tau$;

c) If $g : \tau$ and $x \in Var^\sigma$, then $(\lambda x : \sigma\ .\ g) : [\sigma \to \tau]$;

iv) If $Const^\tau = \emptyset$ for all types τ, then L is called a *pure typed λ calculus.*

A grammar for terms is given by: $f ::= x \mid c \mid (f\ f) \mid (\lambda x\ .\ f)$, where x is a variable and c is a constant. Only terms satisfying the above type restrictions are admitted in this grammar. (Actually, there should be separate, interlinked grammars for each type.)

Note that if $B = \emptyset$, then there are no types at all, and hence no terms. This is an acceptable case, giving the empty λ-calculus.

An important role is played by the free variable function FV which takes terms to finite sets of variables. The following notation will be used in order to describe it succinctly.

$Var = \cup_{\tau \in Type} Var^\tau$, $Const = \cup_{\tau \in Type} Const^\tau$

$P_f(Var)$ = the set of finite subsets of Var.

$Terms = \cup_{\tau \in Type} Terms^\tau$.

type : Terms $\to$ Type is the function taking terms of type τ to τ.

Thus, $f : \sigma$ if and only if $type(f) = \sigma$.

2.1.2 Definition. $FV : Terms \to P_f(Var)$ is the function defined by structural recursion as follows:

i) If $x \in Var$, then $FV(x) = \{x\}$.

ii) If $c \in Const$, then $FV(c) = \emptyset$.

iii) $FV((f\ g)) = FV(f) \cup FV(g)$.

iv) $FV((\lambda x\ .\ g)) = FV(g) - \{x\}$.

A term f is called *closed* or a *program* if $FV(f) = \emptyset$.

2.2 Operational semantics.

2.2.1 Definition. (Operational semantics) The operational semantics consists of some of the following rewrite rules. We use "$\Rightarrow$" here to mean the left hand side rewrites to the right hand side in a single step. The transitive closure of $\Rightarrow$ is written as $\Rightarrow^*$.

i) (α - conversion) $(\lambda x\ .\ f) \Rightarrow (\lambda y\ .[y / x]\ f)$ providing $y \notin FV(f)$.

ii) (β - conversion) $(\lambda x\ .\ f)\ g \Rightarrow [g / x]\ f$. (See below.)

iii) (η - conversion) $(\lambda x\ .\ f)\ x \Rightarrow f$ (or the reverse rule)

iv) (rewrite schemes)

a) $\dfrac{h \Rightarrow k}{(h\ g) \Rightarrow (k\ g)}$ b) $\dfrac{h \Rightarrow k}{(f\ h) \Rightarrow (f\ k)}$ c) $\dfrac{f \Rightarrow g}{(\lambda x\ .\ f) \Rightarrow (\lambda x\ .\ g)}$

v) (δ - conversion) Special rewrite rules for constants.

2.2.2 Definition. Let $x \in$ Var and $f, g \in$ Terms with type(x) = type(f). Then the operation $[f/x]\,g$ of *substituting* f *for* x *in* g is defined recursively by the rules:

i) $[f/x]\,c \Rightarrow c$ if c is a constant

ii) $[f/x]\,x \Rightarrow f$

iii) $[f/x]\,y \Rightarrow y$ if $y \neq x$ and y is a variable

iv) $[f/x]\,(h\,k) \Rightarrow (([f/x]\,h)\,([f/x]\,k))$

v) $[f/x]\,(\lambda x\,.\,g) \Rightarrow (\lambda x\,.\,g)$ (i.e., don't substitute for bound variables.)

vi) $[f/x]\,(\lambda y\,.\,g) \Rightarrow (\lambda y\,.\,[f/x]\,g)$ if $x \neq y$ and $y \notin FV(f)$

vii) $[f/x]\,(\lambda y\,.\,g) \Rightarrow (\lambda z\,.\,[f/x]\,[z/y]\,g)$ if $x \neq y$ and $y \in FV(f)$,
where $z \neq x$, $z \neq y$, type(z) = type(y), and $z \notin FV[f] \cup FV[g]$.
(This prevents the capture of free variables.)

2.2.3 Definition. The *pure λ-calculus with substitution* (based on a set B of basic types) L is the λ– calculus in which

i) The set of basic types is B.

ii) There are no constant terms.

iii) The rewrite rules are given by α-conversion, β-conversion, and the three rules in 2.2.1 iv).

iv) Substitution is given the rules in 2.2.2.

3. DENOTATIONAL SEMANTICS FOR THE TYPED λ-CALCULUS WITH SUBSTITUTION: THE GLOBAL ENVIRONMENT MODEL.

Let **C** be a cartesian closed category with countable products. A global environment model, denotational semantics for L in **C**, consists of an interpretation of types in L as objects in **C** and terms as morphisms in **C** whose domains are all equal to an *environment object* in **C** and whose codomains depend on their types. To express things in a simple form, we need some notation describing what happens to products when their sets of indices are changed.

3.1 Definition.

i) Let I and J be index sets and let $h : J \to I$ be a function. If $\{X_i : i \in I\}$ is a family of objects in a category **C** with products, then $pr_h : \prod_{i \in I} X_i \to \prod_{j \in J} X_{h(j)}$ denotes the morphism, called a *generalized projection*, that is uniquely determined by the requirement that $pr_j \circ pr_h = pr_{h(j)}$; i.e., $pr_h = \langle pr_{h(j)} \mid j \in J \rangle$, where pr_j is the j-th projection morphism. Notice that if an element $i \in I$ is replaced by its name, $name(i) : 1 \to I$, then $pr_i = pr_{name(i)}$.

ii) In particular, if h is the inclusion of a subset J into I, then
$$pr_J = pr_h : \prod_{i \in I} X_i \to \prod_{i \in J} X_i.$$

iii) If h is the inclusion of the complement of a single element i_0 of I, then
$$pr_{i^\wedge} = pr_{I - \{i_0\}} : \prod_{i \in I} X_i \to \prod_{i \in I - \{i_0\}} X_i.$$

3.2 Definition. Let L be a pure λ-calculus with substitution based on B.

i) A *type interpretation* of L in **C** consists of a function D : Type → obj(**C**) such that

a) If $b \in B$, then $D(b) \in$ obj(**C**) is some user chosen object in **C**.

b) $D([\sigma \to \tau] = [D(\sigma) \to D(\tau)]_C$.

We frequently write D_σ for $D(\sigma)$.

ii) The object of *global environments* is the product $Env = \prod_{x \in Var} D_{type(x)}$.

iii) If $f \in$ Terms, then the object of *local environments for* f is the product
$Env(f) = \prod_{x \in FV(f)} D_{type(x)}$.

In order to describe the interpretation of terms in **C**, we need an operation of updating an environment. The usual expression, $\rho[\, d / x \,]$, for an updated environment depends on an environment ρ, a variable x, and an element d in $D_{type(x)}$, and produces a new environment, so it can be regarded as a function from $Env \times D_{type(x)}$ to Env. Since Env is defined as a product in **C**, morphisms into it are determined by their projections onto each factor.

3.3 Definition. For each $x \in$ Var, $update_x : Env \times D_{type(x)} \to Env$ is the unique morphism such that for every $w \in$ Var, $pr_w \circ update_x = pr_w \circ pr_1$ if $w \neq x$, and $pr_x \circ update_x = pr_2$; i.e., $update_x = \langle u_w \mid w \in Var \rangle$ where $u_w = pr_w \circ pr_1$ if $w \neq x$ and $u_x = pr_2$. Here, pr_1 and pr_2 refer to the projections from the product $Env \times D_{type(x)}$ to its two factors.

3.4 Definition.
For each type τ, the *term interpretation* function $[\![-]\!]_\tau : Terms^\tau \to \mathbf{C}(Env, D_\tau)$ is defined recursively by the clauses:

i) $[\![x : \tau]\!]_\tau = pr_x : Env \to D_\tau$.

ii) If $f : [\sigma \to \tau]$ and $g : \sigma$, then
$[\![(f\ g)]\!]_\tau = app_{\sigma,\tau} \circ \langle [\![f]\!]_{[\sigma \to \tau]}, [\![g]\!]_\sigma \rangle : Env \to D_\tau$

iii) If $f : \tau$, then
$[\![\lambda x : \sigma \,.\, f]\!]_{[\sigma \to \tau]} = curry([\![f]\!]_\tau \circ update_x) : Env \to [D_\sigma \to D_\tau]$.

Part ii) makes sense since $[\![f]\!]_{[\sigma \to \tau]} : Env \to [D_\sigma \to D_\tau]$ and $[\![g]\!]_\sigma : Env \to D_\sigma$ so $\langle [\![f]\!]_{[\sigma \to \tau]}, [\![g]\!]_\sigma \rangle : Env \to [D_\sigma \to D_\tau] \times D_\sigma$. To make sense of part iii), we have to clarify what morphism is being curried. But if $\sigma = type(x)$, then $update_x : Env \times D_\sigma \to Env$, so $[\![f]\!]_\tau \circ update_x : Env \times D_\sigma \to D_\tau$.

If, instead of $update_x$, we use the curried version, $update_x^\# : Env \to [D_{type(x)} \to Env]$, then there is an equally nice formula for the semantics of an abstraction:

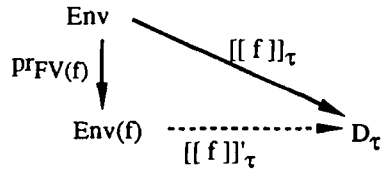
$[\![\lambda x : \sigma \,.\, f]\!]_{[\sigma \to \tau]} = [id_{D_\sigma} \to [\![f]\!]_\tau] \circ update_x^\# : Env \to [D_\sigma \to D_\tau]$.

However, the operations described in 3.7 for the environments algebra no longer have a reasonable and easily recognizable form, so we don't use this formulation here.

Our main goal is to show that the intended meaning of the λ-calculus embodied in the rewrite rules that constitute the operational semantics of the language agrees with denotational semantics given by the interpretation function. Thus, we want to show that if $f \Rightarrow^* g$ then $[\![f]\!] = [\![g]\!]$. It is, of course, sufficient to prove this for a single step $f \Rightarrow g$. Some preliminary results are required before this can be carried out.

3.5 Proposition. If $f : \tau$, then $[[f]]_\tau : Env \to D_\tau$ factors through Env(f). (I.e., $[[f]]_\tau$ depends only on the free variables of f.)

Proof. It must be shown that $[[f]]_\tau = [[f]]'_\tau \circ pr_{FV(f)}$ for some suitable morphism $[[f]]'_\tau : Env(f) \to D_\tau$; i.e., that there is a commutative diagram

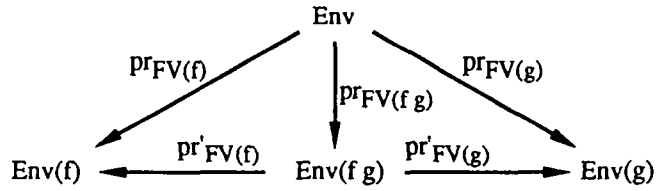


This is proven by structural induction on the form of f.

i) $[[x : \tau]]_\tau = pr_x = id \circ pr_{FV(x:\tau)}$

ii) $[[(f\ g)]]_\tau = app_{\sigma,\tau} \circ \langle [[f]]_{[\sigma \to \tau]}, [[g]]_\sigma \rangle$

But, $FV((f\ g)) = FV(f) \cup FV(g)$ so there are commutative diagrams of projections:



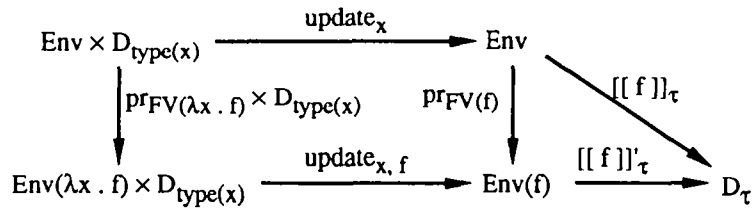
where, for instance, $pr'_{FV(f)}$ corresponds to the inclusion $FV(f) \to FV(f) \cup FV(g)$. Thus, omitting types,

$$
\begin{aligned}
[[(f\ g)]] &= app \circ \langle [[f]], [[g]] \rangle \\
&= app \circ \langle [[f]]' \circ pr_{FV(f)}, [[g]]' \circ pr_{FV(g)} \rangle \\
&= app \circ \langle [[f]]' \circ pr'_{FV(f)} \circ pr_{FV(f) \cup FV(g)}, \\
&\qquad [[g]]' \circ pr'_{FV(g)} \circ pr_{FV(f) \cup FV(g)} \rangle \\
&= app \circ \langle [[f]]' \circ pr'_{FV(f)}, [[g]]' \circ pr'_{FV(g)} \rangle \circ pr_{FV(f) \cup FV(g)} \\
&= app \circ \langle [[f]]' \circ pr'_{FV(f)}, [[g]]' \circ pr'_{FV(g)} \rangle \circ pr_{FV(f\ g)}
\end{aligned}
$$

iii) $[[\lambda x : \sigma . f]]_{[\sigma \to \tau]} = curry([[f]]_\tau \circ update_x)$

$$
\begin{aligned}
&= curry([[f]]'_\tau \circ pr_{FV(f)} \circ update_x) \\
&= curry([[f]]'_\tau \circ update_{x,\, f} \circ (pr_{FV(\lambda x . f)} \times D_{type(x)})\ ) \\
&= curry([[f]]'_\tau \circ update_{x,\, f}) \circ pr_{FV(\lambda x . f)}.
\end{aligned}
$$

This follows from the diagram



in which the triangle commutes by the definition of $[[f]]'_\tau$. As for the rectangle, it is easy to see that the compositions of both paths with $pr_w : Env(f) \to D_{type(w)}$ is the same for all $w \in FV(f)$.

Next, the definition of an interpretation has to be extended to substitutions.

3.6 Definition. Let $f : \tau$ and $x : \tau$. Then $[[f / x]] : Env \to Env$ is the unique morphism such that

$$pr_w \circ [[f / x]] = pr_w \text{ if } w \neq x, \text{ and } pr_x \circ [[f / x]] = [[f]]_\tau;$$

i.e., $[[f / x]] = \langle u_w \mid w \in Var \rangle$ where $u_w = pr_w$ if $w \neq x$ and $u_x = [[f]]_\tau$.

There are many relations between the morphisms $[[f]]$, $[[f / x]]$, and $update_x$. Some of them are listed in the following proposition which provides the crucial technical tools for the main result of this section.

3.7 Proposition. (The global environments algebra)

i) $[[f / x]] = update_x \circ \langle id_{Env}, [[f]] \rangle$

ii) If V is a subset of Var and $y \notin V$, then

$$pr_V \circ update_y = pr_V \circ pr_1 : Env \times D_{type(y)} \to \prod_{x \in V} D_{type(x)}.$$

Here, pr_1 is the projection from $Env \times D_{type(x)}$ to Env.

iii) If $y \notin FV(f)$, then

$$[[f]] \circ update_y = [[f]] \circ pr_1 : Env \times D_{type(y)} \to D_{type(f)}.$$

iv) If $x \neq y$, then

$$update_x \circ (update_y \times D_{type(x)}) = update_y \circ (update_x \times D_{type(y)}) \circ ac.$$

Here $ac : Env \times D_{type(y)} \times D_{type(x)} \to Env \times D_{type(x)} \times D_{type(y)}$ is the isomorphism rearranging the last two factors.

v) $update_x \circ ([[f / x]] \times D_{type(x)}) = update_x$

vi) If $y \notin FV(f)$, then

$$update_y \circ ([[f / x]] \times D_{type(y)}) = [[f / x]] \circ update_y.$$

vii) If $x \neq y$, $x \notin FV(g)$ and $y \notin FV(f)$, then

$$[[f / x]] \circ [[g / y]] = [[g / y]] \circ [[f / x]].$$

viii) If $z \neq y$, then

$$pr_{z^\wedge} \circ [[z / y]] \circ update_z = pr_{z^\wedge} \circ update_y$$
$$: Env \times D_{type(y)} \to \prod_{x \in Var - \{z\}} D_{type(x)}.$$

Proof.

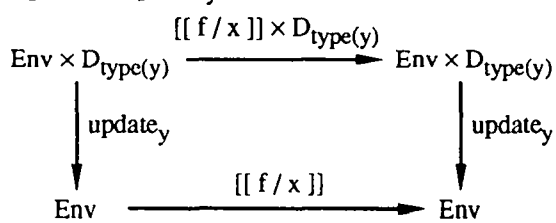
i) and ii). Check that the compositions of both sides of the equation with pr_w for all $w \in Var$ (respectively, V) are the same.

iii) $[[f]] \circ update_y = [[f]]' \circ pr_{FV(f)} \circ update_y$
$= [[f]]' \circ pr_{FV(f)} \circ pr_1 = [[f]] \circ pr_1,$

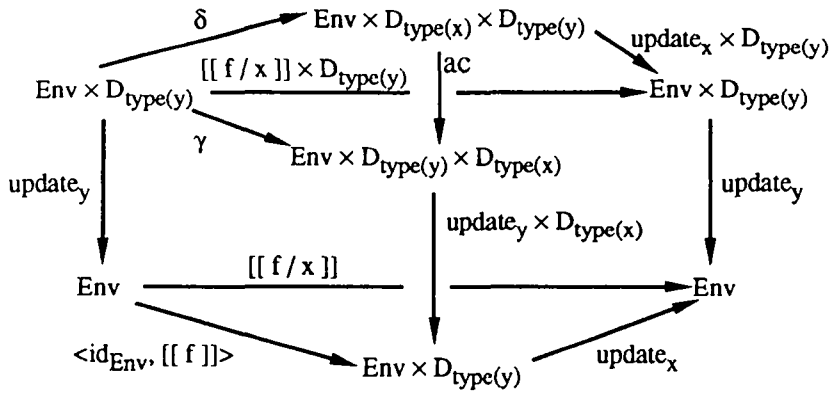
by part ii), provided $y \notin FV(f)$.

iv) and v) Check these by composing with pr_w for all $w \in Var$.

vi) The equation $update_y \circ ([[f / x]] \times id) = [[f / x]] \circ update_y$ says that the diagram



commutes. Expanding $[[f / x]]$ using part i), gives the diagram



where $\delta = \langle id_{Env}, [[f]] \rangle \times D_{type(y)}$ and $\gamma = \langle id_{Env \times D_{type(y)}}, [[f]] \circ pr_1 \rangle$. But in this 3-dimensional diagram, we recognize the right hand side as the commutative pentagon from part iv). The top and bottom triangles commute by part i), and, on the left hand side, the triangle commutes for trivial reasons because it is just rearranging components. The rectangle there commutes by part iii). Hence the main rectangle commutes.

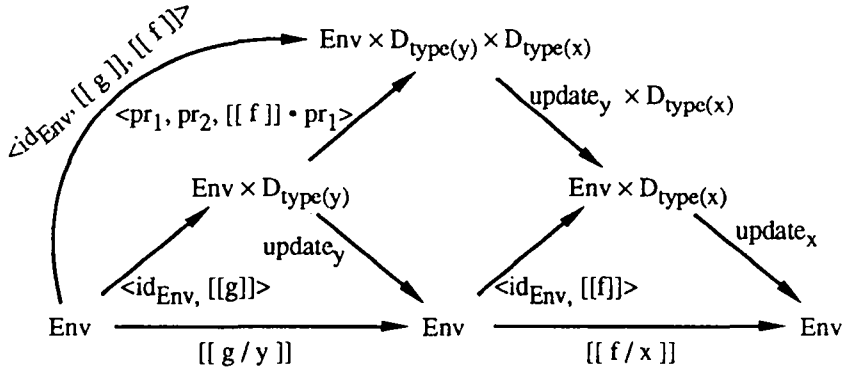
vii) To verify the equation

$$[[f / x]] \circ [[g / y]] = [[g / y]] \circ [[f / x]]$$

when $x \neq y$, $x \notin FV(g)$ and $y \notin FV(f)$, consider the diagram below in which the two triangles commute by part i) and the central diamond commutes by part iii) since $y \notin FV(f)$. The curvilinear part commutes trivially. Hence

$$\begin{aligned} &[[f / x]] \circ [[g / y]] \\ &\quad = update_x \circ (update_y \times D_{type(x)}) \circ \langle id_{Env}, [[g]], [[f]] \rangle \\ &\quad = update_y \circ (update_x \times D_{type(y)}) \circ ac \circ \langle id_{Env}, [[g]], [[f]] \rangle \\ &\quad = update_y \circ (update_x \times D_{type(y)}) \circ \langle id_{Env}, [[f]], [[g]] \rangle \\ &\quad = [[g / y]] \circ [[f / x]] \end{aligned}$$

the second equation following from part iv), the third by the definition of ac, and the fourth by symmetry.



viii) Check this by composing with pr_w for all $w \in Var$.

3.8 Definition..

i) There is a left action on the set C(X, Env) of generalized elements from X to Env by the set

$$\text{SameType} = \{(x, f) \mid x \in \text{Var}, f \in \text{Terms, and type}(f) = \text{type}(x)\}$$

(cf. 2.2.2 and 3.6) using the rule $[[\, f / x \,]](\rho) = [[\, f / x \,]] \circ \rho : X \to \text{Env}$ where $\rho : X \to \text{Env}$ is a generalized element of Env and $(x, f) \in$ SameType.

ii) Similarly, define a right action on the set $\mathbf{C}(X, \text{Env})$ by the set

$$\text{GeneralizedElements} = \cup\{\mathbf{C}(X, D_{\text{type}(x)}) \mid x \in \text{Var}\}.$$

using the rule $\rho[\, d / x \,] = \text{update}_x \circ \langle \rho, d \rangle : X \to \text{Env}$.

where $(d : X \to D_{\text{type}(x)}) \in$ GeneralizedElements.

Using the notation

$$[[\, f \,]](\rho) = [[\, f \,]] \circ \rho : X \to D_{\text{type}(f)},.$$

the Environments algebra can be expressed in terms of these constructions.

3.9 Proposition.

i) (The left-right exchange law.) $[[\, f / x \,]](\rho) = \rho[\, [[\, f \,]](\rho) / x \,]$

ii) —

iii) If $y \notin FV(f)$, then $[[\, f \,]](\rho[\, d / y \,]) = [[\, f \,]](\rho)$.

iv) (The right commutative law for $y \neq x$.)

$$(\rho[\, d / y \,])[\, d' / x \,] = (\rho[\, d' / x \,])[\, d / y \,]$$

v) $([[\, f / x \,]]\rho)[\, d / x \,] = \rho[\, d / x \,]$

vi) (The left-right commutative law.) If $y \notin FV(f)$ and $y \neq x$, then

$$[[\, f / x \,]](\rho[\, d / y \,]) = ([[\, f / x \,]](\rho))[\, d / y \,]$$

vii) (The left commutative law for $y \neq x$, $x \notin FV(g)$ and $y \notin FV(f)$)

$$[[\, f / x \,]]([[\, g / y \,]]\rho) = [[\, g / y \,]]([[\, f / x \,]]\rho).$$

Proof. These follow directly from 3.7.

This form of the Environment algebra seems as though it ought to be useful. However, we have been unable to use it in proving the substitution lemma that follows, so it does not actually play any role in the rest of the paper. In the substitution lemma we will need the notion of the length of an expression since one step in its inductive proof requires induction on the length of the expression..

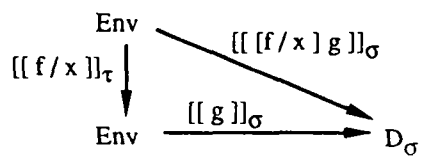
3.10 Definition. The length of an expression is defined recursively by the clauses:

i) $\text{length}(x : \sigma) = 1$ if $x \in$ Var.

ii) $\text{length}(f\ g) = \text{length}(f) + \text{length}(g)$

iii) $\text{length}(\lambda x : \sigma\ .\ f) = \text{length}(f) + 1$.

3.11 Proposition. (The substitution lemma) Let $f : \tau$, $x : \tau$ and $g : \sigma$. Then $[[\, [f / x]\ g \,]]_\sigma = [[\, g \,]]_\sigma \circ [[\, f / x \,]]_\tau$; i.e., the diagram



commutes. (Thus, substitution is interpreted by composition with a suitable morphism.)

Proof. The proof is by structural induction on the form of g together with induction on the length of g. (Cf. [Brooks 89].) We omit type information wherever possible.

i) The cases g = x and g = y follow directly from 3.4 and 3.6.

ii) $[f / x] (h\ k) = (([f / x] h) ([f / x] k))$ by 2.2.2 iv), so

$$\begin{aligned}
&[[[f / x] (h\ k)]] \\
&\quad = [[([f / x] h) ([f / x] k)]] \\
&\quad = \mathrm{app} \circ \langle [[[f / x] h]], [[[f / x] k]] \rangle \\
&\quad = \mathrm{app} \circ \langle [[h]] \circ [[f / x]], [[k]] \circ [[f / x]] \rangle
\end{aligned}$$

(by structural induction (or, equally well, by induction on the length of the expression)

$$\begin{aligned}
&\quad = \mathrm{app} \circ \langle [[h]], [[k]] \rangle \circ [[f / x]] \\
&\quad = [[(h\ k)]] \circ [[f / x]].
\end{aligned}$$

iii) If, instead of just g, there is a lambda expression, $\lambda y . g$, then we have to show that

$$\begin{aligned}
[[[f / x] (\lambda y . g)]] &= [[\lambda y . g]] \circ [[f / x]] \\
&= \mathrm{curry}([[g]] \circ \mathrm{update}_y) \circ [[f / x]],
\end{aligned}$$

using the definition of $[[\lambda y . g]]$ from 3.4. There are three cases to be considered:

a) If x = y, then $[f / x] (\lambda x . g) = (\lambda x . g)$, so

$$\begin{aligned}
&[[[f / x] (\lambda x . g)]] = [[(\lambda x . g)]] \\
&\quad = \mathrm{curry}([[g]] \circ \mathrm{update}_x) \\
&\quad = \mathrm{curry}([[g]] \circ \mathrm{update}_x \circ ([[f / x]] \times D_{\mathrm{type}(x)})), \text{ by 3.7 vi)}, \\
&\quad = \mathrm{curry}([[g]] \circ \mathrm{update}_x) \circ [[f / x]], \\
&\quad = [[\lambda x . g]] \circ [[f / x]].
\end{aligned}$$

b) If $x \neq y$ and $y \notin \mathrm{FV}(f)$, then $[f / x] (\lambda y . g) = \lambda y . [f / x] g$, so

$$\begin{aligned}
&[[[f / x] (\lambda y . g)]] = [[\lambda y . [f / x] g]] \\
&\quad = \mathrm{curry}([[[f / x] g]] \circ \mathrm{update}_y) \\
&\quad = \mathrm{curry}([[g]] \circ [[f / x]] \circ \mathrm{update}_y), \text{ by structural induction}, \\
&\quad = \mathrm{curry}([[g]] \circ \mathrm{update}_y \circ ([[f / x]] \times D_{\mathrm{type}(y)})),
\end{aligned}$$

(by 3.7 vii) since $y \notin \mathrm{FV}(f)$),

$$\begin{aligned}
&\quad = \mathrm{curry}([[g]] \circ \mathrm{update}_y) \circ [[f / x]], \\
&\quad = [[\lambda y . g]] \circ [[f / x]].
\end{aligned}$$

c) If $x \neq y$ and $y \in \mathrm{FV}(f)$, then

$$[f / x] (\lambda y . g) = \lambda z . [f / x] [z / y] g$$

where $\mathrm{type}(z) = \mathrm{type}(y)$, $z \neq x$, $z \neq y$, and $z \notin \mathrm{FV}(f) \cup \mathrm{FV}(g)$, so

$$\begin{aligned}
&[[[f / x] (\lambda y . g)]] = [[\lambda z . [f / x] [z / y] g]] \\
&\quad = \mathrm{curry}([[[f / x] [z / y] g]] \circ \mathrm{update}_z) \\
&\quad = \mathrm{curry}([[[z / y] g]] \circ [[f / x]] \circ \mathrm{update}_z),
\end{aligned}$$

(by induction on the length of expressions, since $[z / y] g$ is shorter than $\lambda y . g$),

$$\begin{aligned}
&\quad = \mathrm{curry}([[[z / y] g]] \circ \mathrm{update}_z \circ ([[f / x]] \times D_{\mathrm{type}(z)})),
\end{aligned}$$

(by 3.7 vii) since $z \notin \mathrm{FV}(f)$),

$$\begin{aligned}
&\quad = \mathrm{curry}([[[z / y] g]] \circ \mathrm{update}_z) \circ [[f / x]], \\
&\quad = \mathrm{curry}([[g]] \circ [[[z / y]]] \circ \mathrm{update}_z) \circ [[f / x]]
\end{aligned}$$

(by structural induction),

$= \text{curry}([\![g]\!] \circ \text{update}_y) \circ [\![f / x]\!]$, by 3.7 viii) ,

(since $[\![g]\!] = [\![g]\!]' \circ \text{pr}_{FV(g)}$ and $z \notin FV(g)$),

$= [[\lambda y . g]] \circ [\![f / x]\!]$.

This completes the proof for all cases.

3.12 Theorem. For terms in the pure typed lambda calculus,

if $f \Rightarrow g$, then $[\![f]\!] = [\![g]\!]$.

Proof. i) We treat the case of β–reductions in detail, so we have to show that the reduction $((\lambda x.g)\ f) \Rightarrow [\ f / x\ ]\ g$, implies that $[\![(\lambda x.g)\ f]\!] = [\![[\ f / x\ ]\ g]\!]$. By Definition 3.4 ii),

$$[\![((\lambda x.g)\ f)]\!]_\tau = \text{app}_{\sigma,\tau} \circ \langle [\![(\lambda x.g)]\!]_{[\sigma \to \tau]}, [\![f]\!]_\sigma \rangle$$

where type(x) = type(f) = σ and type(g) = τ. But, dropping type subscripts,

$[\![(\lambda x.g)]\!] = \text{curry}([\![g]\!] \circ \text{update}_x)$ (by Definition 3.4 iii)),

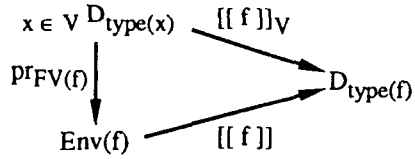
and $\langle [\![(\lambda x.g)]\!], [\![f]\!] \rangle = ([\![(\lambda x.g)]\!] \times \text{id}) \circ \langle \text{id}, [\![f]\!] \rangle$, so

$$\begin{aligned}
[\![((\lambda x.g)\ f)]\!] &= \text{app} \circ \langle [\![(\lambda x.g)]\!], [\![f]\!] \rangle \\
&= \text{app} \circ ([\![(\lambda x.g)]\!] \times D_\sigma) \circ \langle \text{id}_{Env}, [\![f]\!] \rangle \\
&= \text{app} \circ (\text{curry}([\![g]\!] \circ \text{update}_x) \times D_\sigma) \circ \langle \text{id}_{Env}, [\![f]\!] \rangle \\
&= [\![g]\!] \circ \text{update}_x \circ \langle \text{id}_{Env}, [\![f]\!] \rangle. \\
&= [\![g]\!] \circ [\![f / x]\!] \text{ by 3.7 i)} \\
&= [\![[\ f / x\ ]\ g]\!] \text{ by the Substitution Lemma.}
\end{aligned}$$

ii) The other kinds of reductions that have to be considered are α-reductions and the three rewrite schemes in 2.2.1 iv). The rewrite schemes are simple exercises in structural induction and α-reduction follows from 3.7 viii).

4 THE LOCAL ENVIRONMENTS MODEL.

4.1 Discussion. Because of 3.5, it is tempting to try to define a tighter semantics in which the separate environment for each term, Env(f) as defined in 3.2, are used and the semantics is defined as a morphism $[[f]] : Env(f) \to D_{type(f)}$. It is straight forward to set this up. The preceding development then can be carried out until one arrives at the cases in the substitution lemma for lambda expressions. The case $[\ f / x\ ]\ (\lambda x.g) = (\lambda x.g)$ corresponds to β-reduction of a term of the form $(\lambda x.\lambda x.g)\ f$ to $(\lambda x.g)$. Here, it is obvious that β-reduction does not preserve free variables since the left hand side includes the free variables of f while the right hand side doesn't mention f at all. This means that the analogue of 3.12 no longer holds. However, it fails in a "trivial" way in that the two sides of $[\![f]\!] = [\![g]\!]$ differ only by a projection onto a product of a smaller number of factors. One solution is to label all reductions in the operational semantics by sets of variables; e.g., $f \Rightarrow_X g$ where X is a set of variables containing at least FV(f) and FV(g). The theorem then says that if $f \Rightarrow_X g$, then, considered as morphisms from $\prod_{x \in X} D_{type(x)}$ to $D_{type(f)}$, it does follow that $[\![f]\!] = [\![g]\!]$. This is essentially the solution adopted in [Lambek and Scott, 85] and more or less the solution in [Reynolds 74]. Another solution is to define $[\![f]\!]_V$ for any set V of variables containing FV(f) as the illustrated composition:



Then the theorem says that $f \Rightarrow g$ implies $[[f]]_V = [[g]]_V$ for any V containing $FV(f) \cup FV(g)$. The details are sufficiently different, particularly in the aspects involving substitution, that we treat them in detail here.

4.2 Definition. For each $x \in$ Var and $f \in$ Terms, $update_{x, f}$ is defined by two cases:

i) If $x \in FV(f)$, then $update_{x, f} : Env(\lambda x . f) \times D_{type(x)} \to Env(f)$ is the unique morphism such that for every $w \in FV(f)$,

$pr_w \circ update_{x, f} = pr_w \circ pr_1$, if $w \neq x$

$pr_x \circ update_{x, f} = pr_2$.

ii) If $x \notin FV(f)$, then $FV(\lambda x . f) = FV(f)$, so $Env(\lambda x . f) = Env(f)$ and

$update_{x, f} = pr_1 : Env(\lambda x . f) \times 1 \to Env(f)$

4.3 Definition. For each $x \in$ Var, and $f, g \in$ Terms with $type(x) = type(f)$, $[[f / x]]_g$ is defined by two cases:

i) If $x \in FV(g)$, then $[[f / x]]_g : Env([f / x] g) \to Env(g)$ is the unique morphism such that for every $w \in FV(g)$,

$pr_w \circ [[f / x]]_g = pr_w$ if $w \neq x$,

$pr_x \circ [[f / x]]_g = [[f]]$.

ii) If $x \notin FV(g)$, then $FV([f / x] g) = FV(g)$ so $Env([f / x]g) = Env(g)$, and

$[[f / x]]_g = id_{Env(g)} : Env([f / x] g) \to Env(g)$.

For the last definition to make sense, we need the following lemma.

4.4 Lemma. $FV([f / x] g) =$ if $x \in FV(g)$ then $FV(f) \cup (FV(g) - \{x\})$ else $FV(g)$

Proof. The proof is by cases on the form of g and by induction on the length of g.

4.5 Definition. Let L be a pure λ-calculus with substitution based on B.

i) A *type interpretation* of L in **C**, as in 3.2, consists of a function $D : Type \to obj(\mathbf{C})$ such that

a) If $b \in B$, then $D(b) \in obj(\mathbf{C})$ is some user chosen object in **C**.

b) $D([\sigma \to \tau] = [D(\sigma) \to D(\tau)]_{\mathbf{C}}$.

As before, we write D_σ for $D(\sigma)$.

ii) The *(local environments) term interpretation* function $[[-]] : Terms \to mor(\mathbf{C})$ assigns to each term f in L a morphism $[[f]] : Env(f) \to D_{type(f)}$. These morphisms are defined recursively by the clauses:

a) If $x \in$ Var, then $[[x]] = id_{D_{type(x)}} : D_{type(x)} \to D_{type(x)}$.

b) If $f : [\sigma \to \tau]$ and $g : \sigma$, then

$[\![(f\ g)]\!] = app_{\sigma,\tau} \circ \langle [\![f]\!] \circ pr_{FV(f)}, [\![g]\!] \circ pr_{FV(g)} \rangle : Env(\ (f\ g)\ ) \to D_\tau$

c) Let $x \in$ Var and $f \in$ Terms. If $x \in FV(f)$, then

$[\![\lambda x\ .\ f]\!] = curry([\![f]\!] \circ update_{x,f}) : Env\ (\ \lambda x\ .\ f\ ) \to [D_{type(x)} \to D_{type(f)}]$.

else

$[\![\lambda x\ .\ f]\!] = curry([\![f]\!] \circ update_{x,f}) : Env\ (\ \lambda x\ .\ f\ ) \to [1 \to D_{type(f)}]$.

Part b) makes sense since $[\![f]\!] : Env(f) \to [D_\sigma \to D_\tau]$ and $[\![g]\!] : Env(g) \to D_\sigma$. Now $Env(\ (f\ g)\ ) = \prod_{w \in FV(f) \cup FV(g)} D_{type(w)}$ so there are generalized projections

$pr_{FV(f)} : Env(\ (f\ g)\ ) \to Env(f)$ and $pr_{FV(g)} : Env(\ (f\ g)\ ) \to Env(g)$

Thus

$\langle [\![f]\!] \circ pr_{FV(f)}, [\![g]\!] \circ pr_{FV(g)} \rangle : Env(\ (f\ g)\ ) \to [D_\sigma \to D_\tau] \times D_\sigma$

while $app_{\sigma,\tau} : [D_\sigma \to D_\tau] \times D_\sigma \to D_\tau$.

To make sense of part c), we have to clarify what morphism is being curried. If $x \in FV(f)$, then $update_{x,f} : Env(\ \lambda x\ .\ f) \times D_{type(x)} \to Env(f)$, so

$[\![f]\!] \circ update_{x,f} : Env(\ \lambda x\ .\ f) \times D_{type(x)} \to D_{type(f)}$.

Hence,

$curry([\![f]\!] \circ update_{x,f}) : Env(\ \lambda x\ .\ f) \to [D_{type(x)} \to D_{type(f)}]$,

as desired. If $x \notin FV(f)$, then $update_{x,f} = pr_1 : Env(\ \lambda x\ .\ f) \times 1 \to Env(f)$, so as before,

$curry([\![f]\!] \circ update_{x,f}) : Env(\ \lambda x\ .\ f) \to [1 \to D_{type(f)}]$,

4.6 Proposition. (The local environments algebra).

i) If $x \in FV(g)$, then $[\![f / x]\!]_g = update_{x,g} \circ \langle pr_{FV(\lambda x\ .\ g)}, [\![f]\!] \circ pr_{FV(f)} \rangle$
else $[\![f / x]\!]_g = update_{x,g} \circ \langle pr_{FV(\lambda x\ .\ g)}, ! \rangle$.

ii) $pr_{FV(h)} \circ [\![f / x]\!]_{(hk)} = [\![f / x]\!]_h \circ pr_{FV([\ f / x\ ]\ h)}$
$pr_{FV(k)} \circ [\![f / x]\!]_{(hk)} = [\![f / x]\!]_k \circ pr_{FV([\ f / x\ ]\ k)}$

iii) $update_{x,g} = update_{x,g} \circ ([\![f / x]\!]_{(\lambda x\ .\ g)} \times D_{type(x)})$

iv) If $y \notin FV(f)$, then

$[\![f / x]\!]_g \circ update_{y,[f/x]g}) = update_{y,g} \circ ([\![f / x]\!]_{(\lambda y\ .\ g)} \times D_{type(y)})$

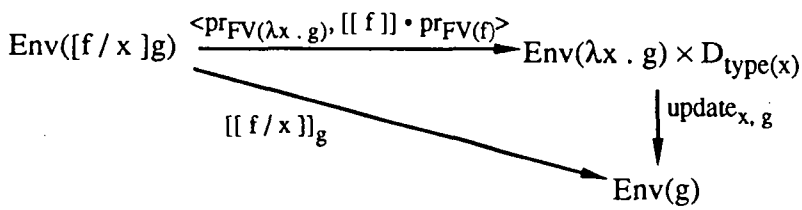
v) $[\![f / x]\!]_{[\ z / y\ ]g} \circ update_{z,[\ f / x\ ][\ z / y\ ]g}) =$

$update_{z,[\ z / y\ ]g} \circ ([\![f / x]\!]_{(\lambda y\ .\ g)} \times D_{type(z)}))$,

vi) $[\![[\ z / y\ ]]\!]_g \circ update_{z,[\ z / y\ ]g} = update_{y,g}$

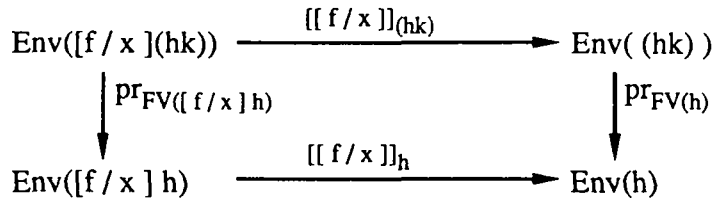
Proof.

i) This says that the diagram



commutes, which follows by considering the projections onto the factors of Env(g).

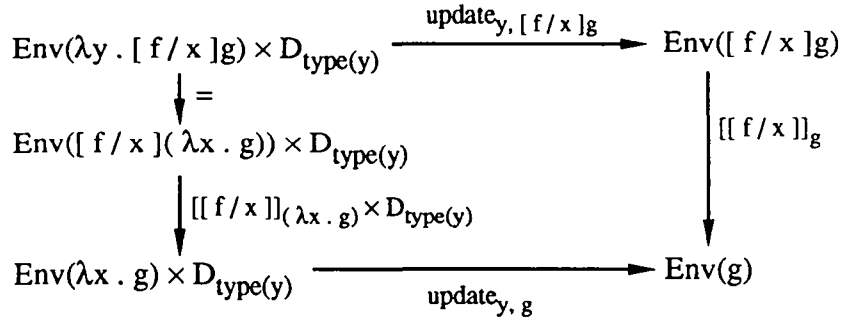
ii) The first equation here says that the diagram



commutes, which follows by considering the projections onto the factors of Env(h). The second equation is proved similarly.

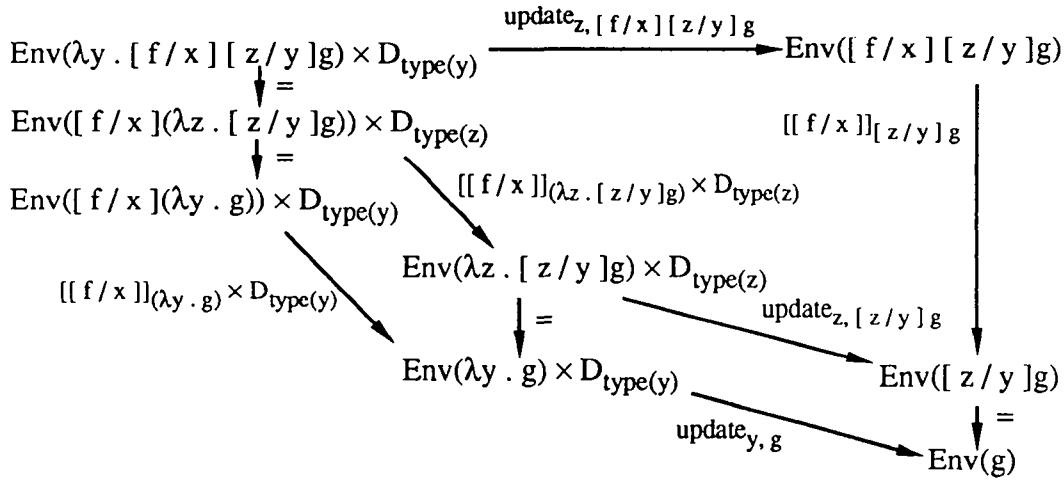
iii) Since $x \notin FV(\lambda x \,.\, g)$, $[[\, f / x \,]]_{(\lambda x \,.\, g)} = id_{Env(\lambda x \,.\, b)}$, so this is immediate.

iv) This says that the diagram



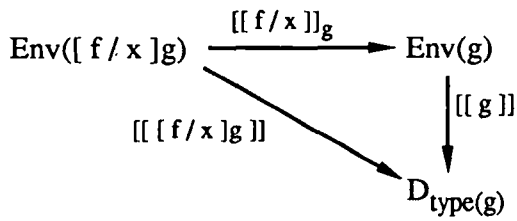
commutes, which follows by considering the projections onto the factors of Env(g). The equality in the upper left hand corner follows from $FV(\lambda y \,.\, [\, f / x \,]g) = FV([\, f / x \,](\lambda y \,.\, g))$ which is a simple computation using 4.4.

v) This says that the diagram



commutes, which follows by considering the projections onto the factors of Env(g). The many equalities follow from similar equalities for sets of free variables, which are exercises using 4.4.

4.7 Proposition. (The substitution lemma). Let $x \in$ Var and f, g $\in$ Terms with type(x) = type(f).Then $[[\, [f / x]\, g \,]] = [[\, g \,]] \circ [[\, f / x \,]]_g$; i.e., the diagram



commutes.

Proof. The proof is by structural induction on the form of g together with induction on the length of g. We omit type information wherever possible.

i) The left hand side is [f / x] x = f by 2.2.2 ii), so [[[f / x] x]] = [[f]], while the right hand side is, $[[x]] \circ [[f / x]]_x = id_{D_{type(x)}} \circ [[f / x]]_x = [[f]]$ by definitions 4.5 ii) and 4.3.i), since type(x) = type(f) and

$$[[f / x]]_x : Env([f / x] x) \to Env(x)$$
$$: Env(f) \to D_{type(f)}.$$

ii) [f / x] y = y, if y ≠ x, so $[[[f / x] y]] = [[y]] = id_{D_{type(y)}}$. On the other hand,

$$[[y]] \circ [[f / x]]_y = id_{D_{type(x)}} \circ id_{Env(y)} = id_{D_{type(y)}},$$

again by definition 4.5 ii), since Env(y) = $D_{type(y)}$.

iii) [f / x] (h k) = (([f / x] h) ([f / x] k)) by 2.2.2 iv), so

[[[f / x] (h k)]]

= [[([f / x] h) ([f / x] k)]]

= app ∘ <[[[f / x] h]] ∘ $pr_{FV([f / x] h)}$, [[[f / x] k]] ∘ $pr_{FV([f / x] k)}$>

= app ∘ < [[h]] ∘ [[f / x]]h ∘ $pr_{FV([f / x] h)}$,

[[k]] ∘ [[f / x]]k ∘ $pr_{FV([f / x] k)}$ >

(by structural induction (or, equally well, by induction on the length of the expression)

= app ∘ <[[h]] ∘ $pr_{FV(h)}$ ∘ [[f / x]](hk), [[k]] ∘ $pr_{FV(k)}$ ∘ [[f / x]](hk) >

(by 4.6.ii))

= app ∘ <[[h]] ∘ $pr_{FV(h)}$, [[k]] ∘ $pr_{FV(k)}$ > ∘ [[f / x]](hk)

= [[(h k)]] ∘ [[f / x]](hk).

iv) If g is a lambda expression, λy . g, then we have to show that

[[[f / x] (λy . g)]] = [[λy . g]] ∘ [[f / x]](λy . g)

= curry([[g]] ∘ $update_{y, g}$) ∘ [[f / x]](λy . g),

using the definition of [[λy . g]] from 4.5 ii). There are three cases to be considered:

a) If x = y, then [f / x] (λx . g) = (λx . g), so

[[[f / x] (λx . g)]] = [[(λx . g)]]

= curry([[g]] ∘ $update_{x, g}$)

= curry([[g]] ∘ $update_{x, g}$ ∘ ([[f / x]](λx . g) × $D_{type(x)}$)), by 4.6 iii),

= curry([[g]] ∘ $update_{x, g}$) ∘ [[f / x]](λx . g),

= [[λx . g]] ∘ [[f / x]](λx . g).

b) If x ≠ y and y ∉ FV(f), then [f / x] (λy . g) = λy . [f / x] g, so

[[[f / x] (λy . g)]] = [[λy . [f / x] g]]

= curry([[[f / x] g]] ∘ $update_{y, [f / x]g}$)

= curry([[g]] ∘ [[f / x]]g ∘ $update_{y, [f / x]g}$), by structural induction,

$= \mathrm{curry}([\![g]\!] \circ \mathrm{update}_{y,\,g} \circ ([\![f / x]\!](\lambda y\,.\,g) \times D_{\mathrm{type}(y)}))$,

(by 4.6.iv) since $y \notin FV(f)$),

$= \mathrm{curry}([\![g]\!] \circ \mathrm{update}_{y,\,g}) \circ [\![f / x]\!](\lambda y\,.\,g)$,

$= [[\lambda y\,.\,g]] \circ [\![f / x]\!](\lambda y\,.\,g)$.

c) If $x \neq y$ and $y \in FV(f)$, then

$[f / x]\,(\lambda y\,.\,g) = \lambda z\,.\,[f / x]\,[z / y]\,g$

where $\mathrm{type}(z) = \mathrm{type}(y)$, $z \neq x$, $z \neq y$, and $z \notin FV(f) \cup FV(g)$, so

$[\![[f / x]\,(\lambda y\,.\,g)]\!] = [\![\lambda z\,.\,[f / x]\,[z / y]\,g]\!]$

$= \mathrm{curry}([\![[f / x]\,[z / y]\,g]\!] \circ \mathrm{update}_{z,\,[f / x]\,[z / y]g})$

$= \mathrm{curry}([\![[z / y]\,g]\!] \circ [\![f / x]\!]_{[z / y]g} \circ \mathrm{update}_{z,\,[f / x]\,[z / y]g})$,

(by induction on the length of expressions, since $[z / y]\,g$ is shorter than $\lambda y\,.\,g$),

$= \mathrm{curry}([\![[z / y]\,g]\!] \circ \mathrm{update}_{z,\,[z / y]g} \circ ([\![f / x]\!](\lambda y\,.\,g) \times D_{\mathrm{type}(z)}))$,

(by 4.6 v) since $z \notin FV(f)$),

$= \mathrm{curry}([\![[z / y]\,g]\!] \circ \mathrm{update}_{z,\,[z / y]g}) \circ [\![f / x]\!](\lambda y\,.\,g)$,

$= \mathrm{curry}([\![g]\!] \circ [\![[z / y]]\!]_g \circ \mathrm{update}_{z,\,[z / y]g}) \circ [\![f / x]\!](\lambda y\,.\,g)$

(by structural induction),

$= \mathrm{curry}([\![g]\!] \circ \mathrm{update}_{y,\,g}) \circ [\![f / x]\!](\lambda y\,.\,g)$, by 4.6 vi) ,

(since $z \notin FV(g)$),

$= [[\lambda y\,.\,g]] \circ [\![f / x]\!](\lambda y\,.\,g)$.

This completes the proof for all cases.

4.8 Definition. Let $f \in$ Terms and let V be a subset of Var containing FV(f). Then we introduce the notation $[\![f]\!]_V = [\![f]\!] \circ \mathrm{pr}_V$.

4.9 Theorem. If $f \Rightarrow g$, then for any $V \supseteq FV(f) \cup FV(g)$, $[\![f]\!]_V = [\![g]\!]_V$.

Proof. As before, we just treat the case of β–reductions, so we have to show that the reduction $((\lambda x.g)\ f) \Rightarrow [f / x]\ g$, implies that $[\![(\lambda x.g)\ f]\!]_V = [\![[f / x]\ g]\!]_V$. By Definition 4.5 ii),

$[\![((\lambda x.g)\ f)]\!] = \mathrm{app}_{\sigma,\,\tau} \circ \langle [\![\lambda x\,.\,g]\!] \circ \mathrm{pr}_{FV(\lambda x\,.\,g)}\,,\ [\![f]\!] \circ \mathrm{pr}_{FV(f)} \rangle$

where $\mathrm{type}(x) = \mathrm{type}(f) = \sigma$ and $\mathrm{type}(g) = \tau$. Suppose first that $x \in FV(g)$. Then, dropping type subscripts,

$[\![((\lambda x.g)\ f)]\!] = \mathrm{app} \circ \langle [\![\lambda x\,.\,g]\!] \circ \mathrm{pr}_{FV(\lambda x\,.\,g)}\,,\ [\![f]\!] \circ \mathrm{pr}_{FV(f)} \rangle$

$= \mathrm{app} \circ ([\![\lambda x\,.\,g]\!] \times [\![f]\!]) \circ \langle \mathrm{pr}_{FV(\lambda x\,.\,g)}\,,\ \mathrm{pr}_{FV(f)} \rangle$

$= \mathrm{app} \circ ([\![(\lambda x.g)]\!] \times \mathrm{id}_{D_\sigma}) \circ (\mathrm{id}_{Env} \times [\![f]\!]) \circ \langle \mathrm{pr}_{FV(\lambda x\,.\,g)}\,,\ \mathrm{pr}_{FV(f)} \rangle$

$= \mathrm{app} \circ (\mathrm{curry}([\![g]\!] \circ \mathrm{update}_{x,\,g}) \times \mathrm{id}_{D_\sigma})$

$\circ (\mathrm{id}_{Env} \times [\![f]\!]) \circ \langle \mathrm{pr}_{FV(\lambda x\,.\,g)}\,,\ \mathrm{pr}_{FV(f)} \rangle$

$= [\![g]\!] \circ \mathrm{update}_{x,\,g} \circ \langle \mathrm{pr}_{FV(\lambda x\,.\,g)},\ [\![f]\!] \circ \mathrm{pr}_{FV(f)} \rangle$

$= [\![g]\!] \circ [\![f / x]\!]_g$, by 4.6.i),

$= [\![[f / x]\ g]\!]$ by the Substitution Lemma.

If $x \notin FV(g)$, then the last three lines are replaced by:

$= [[\ g\ ]] \circ update_{x,\ g} \circ \langle pr_{FV(\lambda x\ .\ g)},\ [[\ f\ ]] \circ pr_{FV(f)} \rangle$
$= [[\ g\ ]] \circ pr_1 \circ \langle pr_{FV(\lambda x\ .\ g)},\ [[\ f\ ]] \circ pr_{FV(f)} \rangle$
$= [[\ g\ ]] \circ id_{Env(\lambda x\ .\ g)}$
$= [[\ g\ ]] \circ [[\ f\ /\ x\ ]]_g$, by 4.3.ii)

This completes the proof in all cases.

5 REFERENCES.

[Abadi, et al 90] M. Abadi, L. Cardelli, P.-L. Curien, J.-J. Lévy, Explicit Substitutions, Rapports de Recherche No 1176, INRIA, Le Chesnay, Mars 1990

[Curien 90] P.-L. Curien, Substitution up to Isomorphism, Rapport de Recherche du LIENS - 90 - 9, Ecole Normale Supérieure, Paris, February 1990.

[Gray 91a] J. W. Gray, Sketches for typed λ-calculi with substitution, to appear.

[Gray 91b] J. W. Gray, Extended PCF, to appear.

[Hardin,Lévy 90] T. Hardin and J.-J. Lévy, A confluent calculus of substitutions, Preprint Juin 1990.

[Hindley,Seldin 86] J. R. Hindley and J. P. Seldin, *Introduction to Combinators and λ-calculus*, Cambridge University Press, 1986.

Imprimé en France
par
l'Institut National de Recherche en Informatique et en Automatique

ISSN 0249-6399