Parametric Computation

Ian A. Mason
Computer Science @ U.N.E
iam@turing.une.edu.au
http://maths.une.edu.au/~iam/

August 9, 1996

Contents

1 Introduction

Contexts, expressions with holes in them, play an increasingly important role in the lambda calculus, but are never truly elevated to first class status. Some recent examples of their use are:

1. The description of reduction strategies (e.g. lazy, left-first depth first) via evaluation and reduction contexts [6].
2. The operational semantics of control, imperative and concurrent features [5, 11, 3].
3. The definition of operational equivalence [14].
4. The characterization of operational equivalence via context lemmas [13].
5. As modalities akin to Hoare triples in logics for functional programming languages with effects [9].
6. As a powerful technique for establishing operational principles for a very general class of programming languages [18].

The purpose of this *short* note is to explain how this can be rectified. One can define contexts in such a way as to enable them to be manipulated, in particular α-converted and β-reduced. In particular we generalize the set of lambda expressions to include holes, and on these generalized entities define β-reduction (i.e. substitution) and filling. In this paper our holes will be black: $\bullet$. Given an expression e that may contain holes, we let $e[\![e_0]\!]$ be the result of filling each occurrence of a hole by the expression e_0. This operation is different from substitution in that the trapping of free

variables can occur. We also define two notions of substitution which (for want of better terms) we shall call *weak* and *strong* substitution. $e[x := e_0]_{\mathtt{w}}$ is the result of *weakly* substituting e_0 for the free occurrences of the variable x in e, again taking care not to trap free variables of e_0. $e[x := e_0]_{\mathtt{s}}$ is the result of *strongly* substituting e_0 for the free occurrences of the variable x in e, again taking care not to trap free variables of e_0. The two notions of substitution differ only in what takes place at holes.

- We use weak substitution to define a notion of α-congruence, $\stackrel{\alpha}{\equiv}$, for contexts that ensures that filling and both forms of substitution preserve α-congruence.
- Substitution and filling are defined in such a way to ensure that filling commutes with strong substitution.
- β-reduction is then defined via the appropriate substitution[1] and it is shown that β-reduction and hole filling commute.

That these two aims can be achieved is interesting in itself. But this is not the end of the story. To achieve these two aims one must do two things. Firstly, one must generalize the notion of a context. Secondly, one must modify the way one handles trappings at holes. We generalize the notion of a hole, so that (occurrences of) holes are decorated with substitutions. This is to enable one to substitute into expressions containing holes. So rather than make the trappings at holes implicit in the surrounding expression, we make them explicit at the hole itself. In this paper we will let σ denote substitutions (finite maps from variable to expressions). The generalized holes are holes decorated by (generalized) substitutions. These syntactic entities are written thus:

$$\bullet^{\sigma}$$

Since this world is more complex than the world without holes we shall take care to define such notions as the *free*, *bound*, *trapped* and *captured* variables of an expression. For example, consider the following skeleton of a term e:

$$\lambda z. \ldots \lambda x. \ldots \bullet^{\{z:=\ldots x \ldots z \ldots x \ldots\}} \ldots$$

for the sake of argument let us assume that neither x nor z occur elsewhere in e. In this expression all occurrences of x are bound, as is the rightmost occurrence of z. The occurrence of z in the domain of the substitution (second from the right) is *neither* free nor bound; rather it is what we shall call a *trapped* variable. The occurrences of x and z in the range of the substitution are called *captured* variables. Note that the terms *trapped* and *captured* have different meanings. α-congruence is defined in such a way that:

$$\lambda z. \ldots \lambda x. \ldots \bullet^{\{z:=\ldots x \ldots z \ldots x \ldots\}} \ldots \qquad \stackrel{\alpha}{\equiv} \qquad \lambda w. \ldots \lambda y. \ldots \bullet^{\{z:=\ldots y \ldots w \ldots y \ldots\}} \ldots$$

and filling is defined so that filling either with z yields (upto α-congruence)

$$\lambda w. \ldots \lambda y. \ldots\ldots y \ldots w \ldots y \ldots$$

In this paper we shall treat contexts simply as special types of expressions. Namely those which contain these decorated holes. Our aim in this paper is to define α-congruence, weak and strong substitution and hole filling so that the above properties hold. The approach adopted in this paper derives from the more abstract de Bruijn theory presented in [16, 17]. However, to be honest, the underlying aim of this work is enable a clean elegant encoding of various operational semantics and programming logics into the current generation of logical frameworks. In our previous work [7] we have described an encoding of the syntax and proof theory of a modern programming logic [10] into a generic theorem prover. Encoding the syntax and proof theory of the logic was a relatively painless procedure.[2] Since the semantics of the underlying programming language ($\lambda_{\mathtt{mk}}$-calculus) is operational, and the semantics of the logic is defined strictly in terms of syntactic entities, it seems not unreasonable to expect an implementation to be capable of encoding the semantics as well as the syntax and proof theory. This would allow for both proof theoretical and semantic reasoning to be carried out at the same time in the same context [19]. It will also allow the system to semantically verify its own proof system, an attractive idea. It would also allow for the dynamic enrichment of the proof theory by introducing new, semantically verified, principles. Thus the logic implemented would be truly

[1] We leave the reader to guess which one at this point.

[2] Especially when compared with the contortions required for logics of the Hoare and Dynamic ilk [12, 1].

extensible or dynamic. The only obstacle to successfully encoding the semantics is the problem of encoding lambda calculus style contexts and hole filling (i.e the corresponding notion of substitution with variable binding capture). This is the the aim of this paper.

We use the usual notation for set membership and function application. $[X \xrightarrow{\omega} Y]$ is the set of functions f whose domain, $\mathrm{Dom}(f)$, is a finite subset of X and whose range, $\mathrm{Rng}(f)$, is contained in Y. $\mathbf{P}_\omega(X)$ is the collection of all finite subsets of X. For any function f, $f\{y := y'\}$ is the function f' such that $\mathrm{Dom}(f') = \mathrm{Dom}(f) \cup \{y\}$, $f'(y) = y'$, and $f'(z) = f(z)$ for $z \neq y, z \in \mathrm{Dom}(f)$. $\mathbb{N} = \{0, 1, 2, \ldots\}$ is the natural numbers and $i, j, n, n_0, \ldots$ range over $\mathbb{N}$. We use $\underline{\lambda}$ as a *meta-lambda* to define functions; it is to be distinguished from any object lambdas that may occur.

2 The Syntax and Structure of Terms

2.1 The Syntactic Domains

We begin with a set, $\mathbb{A}$, of atoms, together with a countably infinite set of variables, $\mathbb{X}$, well-ordered by $\leq$. We let $a, a_0, \ldots$ range over $\mathbb{A}$ and $x, x_0, \ldots, y, \ldots$ range over $\mathbb{X}$. From the given sets we define λ-abstractions, $\mathbb{L}$, expressions, $\mathbb{E}$, and substitutions, $\mathbb{S}$, by induction as the least sets satisfying the following equations:

$$\mathbb{E} = \bullet^{\mathbb{S}} \cup \mathbb{A} \cup \mathbb{X} \cup \mathbb{L} \cup \mathtt{app}(\mathbb{E}, \mathbb{E}) \qquad \mathbb{L} = \lambda\mathbb{X}.\mathbb{E} \qquad \mathbb{S} = \mathbb{X} \xrightarrow{\omega} \mathbb{E}$$

A substitution, σ, is a finite map from variables to expressions. We write $\{x_0 := e_0, \ldots, x_n := e_n\}$ for the substitution σ with domain $\{x_0, \ldots, x_n\}$ such that $\sigma(x_i) = e_i$ for $i < n$. Note that holes can appear anywhere an expression can. In particular holes can appear inside elements of the range of a substitution.

These syntactic domains are constructed in the usual way: defined inductively by a sequence of monotonically increasing sets $\mathbb{E}_i, \mathbb{L}_i$ and $\mathbb{S}_i$ for $i \in \mathbb{N}$ such that

$\mathbb{E} = \bigcup_{i\in\mathbb{N}} \mathbb{E}_i$	where	$\mathbb{E}_0 = \mathbb{A} \cup \mathbb{X}$	and	$\mathbb{E}_{i+1} = \{\bullet^{\mathbb{S}_i}\} \cup \mathbb{L}_i \cup \mathtt{app}(\mathbb{E}_i, \mathbb{E}_i)$	$e, e_0, \ldots$ range over $\mathbb{E}$
$\mathbb{L} = \bigcup_{i\in\mathbb{N}} \mathbb{L}_i$	where	$\mathbb{L}_0 = \emptyset$	and	$\mathbb{L}_{i+1} = \lambda\mathbb{X}.\mathbb{E}_i$	$\lambda x.e, \lambda x_0.e_0, \ldots$ range over $\mathbb{L}$
$\mathbb{S} = \bigcup_{i\in\mathbb{N}} \mathbb{S}_i.$	where	$\mathbb{S}_0 = \emptyset$	and	$\mathbb{S}_{i+1} = \mathbb{X} \xrightarrow{\omega} \mathbb{E}_i$	$\sigma, \sigma_0, \ldots$ range over $\mathbb{S}$

These sets give rise to the standard notion of rank of an expression, $\rho(e)$, (and substitution, $\rho(\sigma)$,) being the least $j \in \mathbb{N}$ for which $e \in \mathbb{E}_j$ ($\sigma \in \mathbb{S}_j$ respectively).

Definition ($\rho(e)$ $\rho(\sigma)$):

$$\rho(e) = \begin{cases} \rho(\sigma) + 1 & \text{if } e = \bullet^\sigma \\ 0 & \text{if } e \in \mathbb{A} \cup \mathbb{X} \\ 1 + \rho(e_0) & \text{if } e = \lambda z.e_0 \\ 1 + \mathbf{max}(\rho(e_0), \rho(e_1)) & \text{if } e = \mathtt{app}(e_0, e_1) \end{cases}$$

$$\rho(\sigma) = 1 + \mathbf{max}_{x\in\mathrm{Dom}(\sigma)} \rho(\sigma(x))$$

As usual λ is a binding operator. Since this world is more complex than the world without holes we shall take care to define such notions as the *free*, *bound*, *trapped* and *captured* variables of an expression.

Definition ($\mathrm{FV}(e)$ $\mathrm{FV}(\sigma)$):

$$\mathrm{FV}(e) = \begin{cases} \mathrm{FV}(\sigma) & \text{if } e = \bullet^\sigma \\ \{e\} \cap \mathbb{X} & \text{if } e \in \mathbb{X} \cup \mathbb{A} \\ \mathrm{FV}(e_0) - \{z\} & \text{if } e = \lambda z.e_0 \\ \mathrm{FV}(e_0) \cup \mathrm{FV}(e_1) & \text{if } e = \mathtt{app}(e_0, e_1) \end{cases}$$

$$\mathrm{FV}(\sigma) = \bigcup_{x\in\mathrm{Dom}(\sigma)} \mathrm{FV}(\sigma(x))$$

Note that for any $\sigma \in \mathbb{S}$ and any $e \in \mathbb{E}$ we have that $\mathrm{FV}(\sigma)$ and $\mathrm{FV}(e)$ are defined in terms of syntactic entities of lesser rank, and so in particular this definition is well-founded. Also note that variables in the domain of a substitution decorating a hole in an expression are *not* considered free in that expression. Similarly we define the set of all variables occurring in either an expression or a substitution.

Definition ($\mathrm{V}(e)$ $\mathrm{V}(\sigma)$):

$$\mathrm{V}(e) = \begin{cases} \mathrm{V}(\sigma) & \text{if } e = \bullet^{\sigma} \\ \{e\} \cap \mathbb{X} & \text{if } e \in \mathbb{X} \cup \mathbb{A} \\ \mathrm{V}(e_0) \cup \{z\} & \text{if } e = \lambda z.e_0 \\ \mathrm{V}(e_0) \cup \mathrm{V}(e_1) & \text{if } e = \mathtt{app}(e_0, e_1) \end{cases}$$

$$\mathrm{V}(\sigma) = \bigcup_{x \in \mathrm{Dom}(\sigma)} \mathrm{V}(\sigma(x)) \cup \mathrm{Dom}(\sigma)$$

We use this definition to define a function from finite sets of syntactic entities to variables called $\mathbf{fresh}$:

$$\mathbf{fresh} : \mathbf{P}_{\omega}(\mathbb{E} \cup \mathbb{S}) \mapsto \mathbb{X} \quad \text{where} \quad \mathbf{fresh}(S) = \text{the } \leq\text{-least } x \in \mathbb{X} \text{ such that } x \notin \textstyle\bigcup_{s \in S} \mathrm{V}(s)$$

The *trapped* variables of either an expression or a substitution are those that appear anywhere in the domain of a hole decorated by a substitution, or in the case of a substitution the domain of the substitution itself:

Definition ($\mathrm{TV}(e)$ $\mathrm{TV}(\sigma)$):

$$\mathrm{TV}(e) = \begin{cases} \mathrm{TV}(\sigma) & \text{if } e = \bullet^{\sigma} \\ \emptyset & \text{if } e \in \mathbb{A} \cup \mathbb{X} \\ \mathrm{TV}(e_0) & \text{if } e = \lambda z.e_0 \\ \mathrm{TV}(e_0) \cup \mathrm{TV}(e_1) & \text{if } e = \mathtt{app}(e_0, e_1) \end{cases}$$

$$\mathrm{TV}(\sigma) = \bigcup_{x \in \mathrm{Dom}(\sigma)} \mathrm{TV}(\sigma(x)) \cup \mathrm{Dom}(\sigma)$$

The definition of a *captured* variable (or one that possibly can be captured) is more complex, but fortunately we can safely keep it informal: an occurrence of a variable is *captured* if it occurs in the range of a substitution that decorates a hole which in turn occurs in the scope of a λ binding that variable.

2.2 Weak Substitution and α-Equivalence

We will let $e[\sigma]_{\mathtt{w}}$ be the result of simultaneous *weak substitution* of free occurrences of the variables $x \in \mathrm{Dom}(\sigma)$ in e by $\sigma(x)$, taking care not to capture variables. It defined as follows:

Definition ($e[\sigma]_{\mathtt{w}}$ $\sigma_1[\sigma_2]_{\mathtt{w}}$):

$$e[\sigma]_{\mathtt{w}} = \begin{cases} \bullet^{\sigma_1[\sigma]_{\mathtt{w}}} & \text{if } e = \bullet^{\sigma_1} \\ e & \text{if } e \in \mathbb{A} \cup (\mathbb{X} - \mathrm{Dom}(\sigma)) \\ \sigma(e) & \text{if } e \in \mathrm{Dom}(\sigma) \\ \lambda\nu.(e_0[z := \nu]_{\mathtt{w}}[\sigma]_{\mathtt{w}}) & \text{if } e = \lambda z.e_0, \text{ and } \nu = \mathbf{fresh}(\{e, \sigma\}) \\ \mathtt{app}(e_0[\sigma]_{\mathtt{w}}, e_1[\sigma]_{\mathtt{w}}) & \text{if } e = \mathtt{app}(e_0, e_1) \end{cases}$$

$$\sigma_1[\sigma]_{\mathtt{w}} = \underline{\lambda} x \in \mathrm{Dom}(\sigma_1).(\sigma_1(x)[\sigma]_{\mathtt{w}})$$

In the definition we have made use of the function $\mathbf{fresh}$, thus ensuring $e[\sigma]_{\mathtt{w}}$ is indeed a function, rather than a relation. Elsewhere in this paper the term *fresh* simply means *any* variable that does not occur in the expressions or substitutions involved (thus dropping the requirement that it be the $\leq$-least such element of $\mathbb{X}$). It is important to note that $\mathrm{Dom}(\sigma_1[\sigma]_{\mathtt{w}}) = \mathrm{Dom}(\sigma_1)$, this has the consequence that weak substitution does not compose in the time honored fashion:

Example (1): $e[\sigma_0]_{\mathtt{w}}[\sigma_1]_{\mathtt{w}} \neq e[\sigma_0[\sigma_1]_{\mathtt{w}}]_{\mathtt{w}}$

Proof of Example (1): Simply take

$$e = z \qquad \sigma_0 = \{x := 3\} \qquad \sigma_1 = \{z := 4\}$$

Then

$$e[\sigma_0]_{\mathtt{w}}[\sigma_1]_{\mathtt{w}} = z[\{x := 3\}]_{\mathtt{w}}[\{z := 4\}]_{\mathtt{w}} = z[\{z := 4\}]_{\mathtt{w}} = 4$$
$$e[\sigma_0[\sigma_1]_{\mathtt{w}}]_{\mathtt{w}} = z[\{x := 3\}[\{z := 4\}]_{\mathtt{w}}]_{\mathtt{w}} = z[\{x := 3[\{z := 4\}]_{\mathtt{w}}\}]_{\mathtt{w}} = z[\{x := 3\}]_{\mathtt{w}} = z$$

□

When $\sigma = \{x := e\}$ we often write $e_0[x := e]_{\mathtt{w}}$ rather than $e_0[\{x := e\}]_{\mathtt{w}}$, when there is no cause for confusion. Often we will use weak substitution to replace one variable by another, in this case the reader should note that the operation corresponds to a simple renaming (leaving the domains of substitutions) unchanged.

We now define α-congruence on expressions. It agrees with the usual definition on expressions not containing holes and is a simple extension of the usual definition given via weak substitution [4, 15].

Definition ($\overset{\alpha}{\equiv}$): The relation $\overset{\alpha}{\equiv}$ is defined (inductively) by the following rules:

(0) $$\frac{}{a \overset{\alpha}{\equiv} a} \qquad a \in \mathbb{A}$$

(1) $$\frac{}{x \overset{\alpha}{\equiv} x} \qquad x \in \mathbb{X}$$

(2) $$\frac{e_0 \overset{\alpha}{\equiv} e_1 \quad e_0' \overset{\alpha}{\equiv} e_1'}{\mathtt{app}(e_0, e_0') \overset{\alpha}{\equiv} \mathtt{app}(e_1, e_1')}$$

(3) $$\frac{\sigma_0(x) \overset{\alpha}{\equiv} \sigma_1(x) \qquad \text{for each } x \in \mathrm{Dom}(\sigma_i)}{\sigma_0 \overset{\alpha}{\equiv} \sigma_1} \qquad \text{provided } \mathrm{Dom}(\sigma_0) = \mathrm{Dom}(\sigma_1)$$

(4) $$\frac{\sigma_0 \overset{\alpha}{\equiv} \sigma_1}{\bullet^{\sigma_0} \overset{\alpha}{\equiv} \bullet^{\sigma_1}}$$

(5) $$\frac{e_0[x_0 := \nu]_{\mathtt{w}} \overset{\alpha}{\equiv} e_1[x_1 := \nu]_{\mathtt{w}}}{\lambda x_0.e_0 \overset{\alpha}{\equiv} \lambda x_1.e_1} \qquad \text{for } \nu \text{ fresh}$$

Notice the simplicity of this proof system: each syntactic type has exactly one rule that will establish conclusions of that form, consequently by a simple application of the *inversion* principle we may conclude that if $e_0 \overset{\alpha}{\equiv} e_1$ is provable, then, modulo new variables introduced using rule (**5**), such a proof is unique. Another simple consequence of this definition is that $\overset{\alpha}{\equiv}$ is an equivalence relation, actually it is a congruence but this requires some proof (c.f. lemma ($\overset{\alpha}{\equiv}$**.6**)). A somewhat more involved property is that weak substitution is functional modulo α-equivalence. This we will establish anon. Note that at holes the domain of the decorating substitution remains unchanged, for example,

Example (2): $\lambda z.\, \bullet^{\{z:=z\}} \overset{\alpha}{\equiv} \lambda \nu.\bullet^{\{z:=\nu\}}$.

Proof of Example (2): Choose ν_0 fresh. Then

(1) $\nu_0 \overset{\alpha}{\equiv} \nu_0$ by rule ($\overset{\alpha}{\equiv}$**.1**)

(2) $\{z := \nu_0\} \overset{\alpha}{\equiv} \{z := \nu_0\}$ from (**1**) by rule ($\overset{\alpha}{\equiv}$**.3**)

(3) $\bullet^{\{z:=\nu_0\}} \stackrel{\alpha}{\equiv} \bullet^{\{z:=\nu_0\}}$ from (**2**) by rule ($\stackrel{\alpha}{\equiv}$**.4**)

(4) $\bullet^{\{z:=z\}}\,[z := \nu_0]_{\mathtt{w}} \stackrel{\alpha}{\equiv} \bullet^{\{z:=\nu\}}\,[\nu := \nu_0]_{\mathtt{w}}$ from (**3**) by definition of $.[.]_{\mathtt{w}}$

(5) $\lambda z.\bullet^{\{z:=z\}} \stackrel{\alpha}{\equiv} \lambda\nu.\bullet^{\{z:=\nu\}}$ from (**4**) by rule ($\stackrel{\alpha}{\equiv}$**.5**)

□

Two trivial properties of $\stackrel{\alpha}{\equiv}$ and weak substitution are: the renaming of a variable in an expression results in an expression of the same rank; and that α-equivalent expressions must be of the same rank. These are proved by extremely simple inductions on the ranks of the expressions involved and are omitted from this paper.

Lemma ($\stackrel{\alpha}{\equiv}$.0):

(0) $\rho(e_0) = \rho(e_0[x := y]_{\mathtt{w}})$

(1) $e_0 \stackrel{\alpha}{\equiv} e_1 \;\Rightarrow\; \rho(e_0) = \rho(e_1)$

Perhaps the most basic and important property of the two definitions is that weak substitution preserves $\stackrel{\alpha}{\equiv}$ equivalence classes:

$(e_0 \stackrel{\alpha}{\equiv} e_1 \;\wedge\; \sigma_0 \stackrel{\alpha}{\equiv} \sigma_1) \;\Rightarrow\; e_0[\sigma_0]_{\mathtt{w}} \stackrel{\alpha}{\equiv} e_1[\sigma_1]_{\mathtt{w}}$

To prove this fact actually requires us to first simultaneously establish several simpler properties:

Proposition ($.[.]_{\mathtt{w}}$.1):

(a) $(e_0' \stackrel{\alpha}{\equiv} e_1' \;\wedge\; e_0 \stackrel{\alpha}{\equiv} e_1) \;\Rightarrow\; e_0'[x := e_0]_{\mathtt{w}} \stackrel{\alpha}{\equiv} e_1'[x := e_1]_{\mathtt{w}}$

(b) $(\nu_0 \notin \mathrm{FV}(e) \;\wedge\; \nu_0 \neq x \neq \nu_1) \;\Rightarrow\; e_0[\nu_0 := \nu_1]_{\mathtt{w}}[x := e]_{\mathtt{w}} \stackrel{\alpha}{\equiv} e_0[x := e]_{\mathtt{w}}[\nu_0 := \nu_1]_{\mathtt{w}}$

(c) $e_0[x := \nu]_{\mathtt{w}}[\nu := y]_{\mathtt{w}} \stackrel{\alpha}{\equiv} e_0[x := y]_{\mathtt{w}}$ for $\nu \notin \mathrm{FV}(e_0)$

(d) $\lambda x.e_0 \stackrel{\alpha}{\equiv} \lambda\nu.e_0[x := \nu]_{\mathtt{w}}$ ν fresh

And then prove a moderately more elaborate version of the desired fact, namely:

Proposition ($\stackrel{\alpha}{\equiv}$.1):

(a) $(e_0 \stackrel{\alpha}{\equiv} e_1 \;\wedge\; \sigma_0 \stackrel{\alpha}{\equiv} \sigma_1) \;\Rightarrow\; e_0[\sigma_0]_{\mathtt{w}} \stackrel{\alpha}{\equiv} e_1[\sigma_1]_{\mathtt{w}}$

(b) $(\nu_0 \notin \mathrm{V}(\sigma) \;\wedge\; \nu_1 \notin \mathrm{Dom}(\sigma)) \;\Rightarrow\; e_0[\nu_0 := \nu_1]_{\mathtt{w}}[\sigma]_{\mathtt{w}} \stackrel{\alpha}{\equiv} e_0[\sigma]_{\mathtt{w}}[\nu_0 := \nu_1]_{\mathtt{w}}$

(c) $(\lambda\nu.e_0)[\sigma]_{\mathtt{w}} \stackrel{\alpha}{\equiv} \lambda\nu.(e_0[\sigma]_{\mathtt{w}})$ if $\nu \notin \mathrm{V}(\sigma)$

Now that the most basic principles have been established, we can be somewhat more freewheeling. For example we can show that α-equivalence is a congruence simply by establishing the following useful derived rule.

Lemma ($\stackrel{\alpha}{\equiv}$.6): The following rule is derivable:

$$(\stackrel{\alpha}{\equiv}.6)\quad \frac{e_0 \stackrel{\alpha}{\equiv} e_1}{\lambda x.e_0 \stackrel{\alpha}{\equiv} \lambda x.e_1}$$

Proof of Lemma ($\stackrel{\alpha}{\equiv}$.6): Assume that $e_0 \stackrel{\alpha}{\equiv} e_1$. Then by part (**a**) of ($\stackrel{\alpha}{\equiv}$**.1**) we have that $e_0[x := \nu]_{\mathtt{w}} \stackrel{\alpha}{\equiv} e_1[x := \nu]_{\mathtt{w}}$. Thus by the rule ($\stackrel{\alpha}{\equiv}$**.4**) we may conclude $\lambda x.e_0 \stackrel{\alpha}{\equiv} \lambda x.e_1$ □

We can also strengthen part (**b**) of proposition ($.[.]_{\mathtt{w}}$**.1**) to the following more useful principle.

Proposition ($.[.]_{\mathtt{w}}$.2): If $(\mathrm{Dom}(\sigma_1) - \mathrm{Dom}(\sigma_0)) \cap \mathrm{FV}(e) = \emptyset$, then

$e[\sigma_0]_{\mathtt{w}}[\sigma_1]_{\mathtt{w}} \stackrel{\alpha}{\equiv} e[\sigma_0[\sigma_1]_{\mathtt{w}}]_{\mathtt{w}}$

2.3 Trapping & Filling

To define the process of hole filling requires a stronger notion of substitution. However from a technical (and perhaps conceptual) point of view it is simpler if we first introduce a restricted notion, that of *trapping*. This process is quite intuitive, given a finite set of variables X and a syntactic entity Φ (either an expression or a substitution), we define $\mathbf{Trap}(\Phi, X)$ to be the syntactic entity obtained by enlarging the domains of any substitution occurring in Φ (by mapping $x \in X$ to x itself, if x is not already in the domain of the substitution) so that they include X. Formally:

Definition ($\mathbf{Trap}(e, X)$ $\mathbf{Trap}(\sigma, X)$):

$$\mathbf{Trap}(e, X) = \begin{cases} \bullet^{\mathbf{Trap}(\sigma, X)} & \text{if } e = \bullet^{\sigma} \\ e & \text{if } e \in \mathbb{A} \cup \mathbb{X} \\ \lambda\nu.\mathbf{Trap}(e_0[z := \nu]_{\mathtt{w}}, X) & \text{if } e = \lambda z.e_0, \text{ and } \nu \text{ is fresh} \\ \mathtt{app}(\mathbf{Trap}(e_0, X), \mathbf{Trap}(e_1, X)) & \text{if } e = \mathtt{app}(e_0, e_1) \end{cases}$$

$$\mathbf{Trap}(\sigma, X) = \underline{\lambda} y \in \mathrm{Dom}(\sigma) \cup X. \begin{cases} \mathbf{Trap}(\sigma(y), X) & \text{if } y \in \mathrm{Dom}(\sigma) \\ y & \text{if } y \in X - \mathrm{Dom}(\sigma) \end{cases}$$

Observe that if an expression e contains no holes, then the process of trapping leaves the expression unchanged modulo α-congruence. The first fact we need to establish is that the process of trapping is functional with respect α-congruence.

Proposition ($\overset{\alpha}{\equiv}$.2):

$$e_0 \overset{\alpha}{\equiv} e_1 \;\Rightarrow\; \mathbf{Trap}(e_0, X) \overset{\alpha}{\equiv} \mathbf{Trap}(e_1, X)$$

The second fact establishes that trapping interacts simply with weak substitution and λ-abstraction.

Proposition ($\mathbf{Trap}(.,.)$.1): If $\nu \notin X$, then

(a) $\mathbf{Trap}(e, X)[\nu := y]_{\mathtt{w}} \overset{\alpha}{\equiv} \mathbf{Trap}(e[\nu := y]_{\mathtt{w}}, X)$

(b) $\mathbf{Trap}(\lambda\nu.e, X) \overset{\alpha}{\equiv} \lambda\nu.\mathbf{Trap}(e, X)$

We are now in a position to define hole filling:

Definition ($e[\![e_0]\!]$ $\sigma[\![e_0]\!]$):

$$e[\![e_0]\!] = \begin{cases} \mathbf{Trap}(e_0, \mathrm{Dom}(\sigma))[\sigma[\![e_0]\!]]_{\mathtt{w}} & \text{if } e = \bullet^{\sigma} \\ e & \text{if } e \in \mathbb{A} \cup \mathbb{X} \\ \lambda\nu.(e_1[z := \nu]_{\mathtt{w}}[\![e_0]\!]) & \text{if } e = \lambda z.e_1, \text{ and } \nu \text{ is fresh} \\ \mathtt{app}(e_1[\![e_0]\!], e_2[\![e_0]\!]) & \text{if } e = \mathtt{app}(e_1, e_2) \end{cases}$$

$$\sigma[\![e_0]\!] = \underline{\lambda} x \in \mathrm{Dom}(\sigma).(\sigma(x)[\![e_0]\!])$$

Before we delve in any further depth into this definition note that filling is defined so that capturing, if it is to occur, must be done by the substitution decorating the hole, rather than by the surrounding expression. Examples of this, as well as proof that weak substitution does not commute with filling are provided in the following example.

Example (4):

(a) $\neg(e[\![e_0]\!][\sigma[\![e_0]\!]]_{\mathtt{w}} \overset{\alpha}{\equiv} e[\sigma]_{\mathtt{w}}[\![e_0]\!])$

(b) $(\lambda x.\bullet)[\![x]\!] \overset{\alpha}{\equiv} \lambda y.x$

(c) $(\lambda\nu.\bullet^{\{x:=\nu\}})[\![x]\!] \overset{\alpha}{\equiv} \lambda x.x$

Proof of Example (4): Simply take

$$e = \bullet \qquad e_0 = x \qquad \sigma = \{x := 3\}$$

Then

$$e[\![e_0]\!][\sigma[\![e_0]\!]]_{\mathtt{w}} = \bullet[\![x]\!][\{x := 3\}[\![x]\!]]_{\mathtt{w}} = x[x := 3]_{\mathtt{w}} = 3$$
$$e[\sigma]_{\mathtt{w}}[\![e_0]\!] = \bullet[\{x := 3\}]_{\mathtt{w}}[\![x]\!] = \bullet[\![x]\!] = x$$

$\square_4$

There is (hopefully) only one aspect of the definition that requires explanation. Namely, why is it that:

$$\bullet^{\sigma}[\![e_0]\!] = \mathbf{Trap}(e_0, \mathrm{Dom}(\sigma))[\sigma[\![e_0]\!]]_{\mathtt{w}}$$

To anticipate matters slightly we should point out that *strong substitution* will be defined in such a way as to ensure that

$$\bullet^{\sigma}[\![e_0]\!] = \mathbf{Trap}(e_0, \mathrm{Dom}(\sigma))[\sigma[\![e_0]\!]]_{\mathtt{w}} = e_0[\sigma[\![e_0]\!]]_{\mathtt{s}}.$$

With this in mind, consider the case when $e_0 = \bullet^{\sigma_0}$. To keep things simple we shall assume that neither σ nor σ_0 contain any holes. The idea is that $\bullet^{\sigma}$ represents the situation where the substitutions described by σ have taken place at an initially simple hole $\bullet$. If we had started with $\bullet^{\sigma_0}$ rather than the simple hole, we would be in a situation best described informally by $(\bullet^{\sigma_0})^{\sigma}$. By analysis we see that the result of the informal process is $\bullet^{\sigma_1}$ where

$$\sigma_1 = \underline{\lambda} y \in \mathrm{Dom}(\sigma_0) \cup \mathrm{Dom}(\sigma). \begin{cases} \sigma_0(y)[\sigma]_{\mathtt{w}} & \text{if } y \in \mathrm{Dom}(\sigma_0) \\ \sigma(y) & \text{if } y \in \mathrm{Dom}(\sigma) - \mathrm{Dom}(\sigma_0) \end{cases}$$

and it is a simple computation to show that this is α-congruent to the right hand side of the equation under consideration. When there are holes present in either σ or σ_0 the process is slightly more complex. The underlying idea, however, remains the same. For example, consider the following instructive example.

Example (5):

$$\bullet^{\{x:=\bullet^{\{z:=y\}}\}}[\![\bullet^{\{w:=\bullet^{\{v:=u\}}\}}]\!] = \bullet^{\{x:=\bullet^{\{z:=y,w:=\bullet^{\{z:=y,v:=u\}}\}},w:=\bullet^{\{x:=\bullet^{\{z:=y,w:=\bullet^{\{z:=y,v:=u\}}\},v:=u\}}\}}$$

Proof of Example (5): Let us first work informally with the left hand side of this equation.

$$\begin{aligned}
&\bullet^{\{x:=\bullet^{\{z:=y\}}\}}[\![\bullet^{\{w:=\bullet^{\{v:=u\}}\}}]\!] \\
&\quad = (\bullet^{\{w:=\bullet^{\{v:=u\}}\}})^{\{x:=(\bullet^{\{w:=\bullet^{\{v:=u\}}\}})^{\{z:=y\}}\}} \\
&\quad = (\bullet^{\{w:=\bullet^{\{v:=u\}}\}})^{\{x:=\bullet^{\{z:=y,w:=\bullet^{\{z:=y,v:=u\}}\}}\}} \\
&\quad = \bullet^{\{x:=\bullet^{\{z:=y,w:=\bullet^{\{z:=y,v:=u\}}\}},w:=\bullet^{\{x:=\bullet^{\{z:=y,w:=\bullet^{\{z:=y,v:=u\}}\},v:=u\}}\}}
\end{aligned}$$

Now we work with the actual definition.

$$\begin{aligned}
&\bullet^{\{x:=\bullet^{\{z:=y\}}\}}[\![\bullet^{\{w:=\bullet^{\{v:=u\}}\}}]\!] \\
&\quad = \mathbf{Trap}(\bullet^{\{w:=\bullet^{\{v:=u\}}\}}, \{x\})[x := \bullet^{\{z:=y\}}[\![\bullet^{\{w:=\bullet^{\{v:=u\}}\}}]\!]]_{\mathtt{w}} \\
&\quad = \bullet^{\{x:=x,w:=\bullet^{\{x:=x,v:=u\}}\}}[x := \mathbf{Trap}(\bullet^{\{w:=\bullet^{\{v:=u\}}\}}, \{z\})[z := y]_{\mathtt{w}}]_{\mathtt{w}} \\
&\quad = \bullet^{\{x:=x,w:=\bullet^{\{x:=x,v:=u\}}\}}[x := \bullet^{\{z:=z,w:=\bullet^{\{z:=z,v:=u\}}\}}[z := y]_{\mathtt{w}}]_{\mathtt{w}} \\
&\quad = \bullet^{\{x:=x,w:=\bullet^{\{x:=x,v:=u\}}\}}[x := \bullet^{\{z:=y,w:=\bullet^{\{z:=y,v:=u\}}\}}]_{\mathtt{w}} \\
&\quad = \bullet^{\{x:=\bullet^{\{z:=y,w:=\bullet^{\{z:=y,v:=u\}}\}},w:=\bullet^{\{x:=\bullet^{\{z:=y,w:=\bullet^{\{z:=y,v:=u\}}\},v:=u\}}\}}
\end{aligned}$$

□

Having convinvced ourselves that the definition of filling is correct, we proceed to establish some of its more basic properties. The first being its functionality with repect to α-congruence.

Proposition ($\stackrel{\alpha}{\equiv}$.3):

$$(e_0 \stackrel{\alpha}{\equiv} e_1 \ \wedge \ e_0' \stackrel{\alpha}{\equiv} e_1') \ \Rightarrow \ e_0[\![e_0']\!] \stackrel{\alpha}{\equiv} e_1[\![e_1']\!]$$

Filling also interacts nicely with renaming and λ-abstraction under suitable hygene conditions. These properties are used in the proofs of theorems (**2**) and (**3**).

Lemma (.[![.]!].1):

(a) $\nu \notin \mathrm{FV}(e_0) \ \Rightarrow \ e[\nu := y]_{\mathtt{w}}[\![e_0]\!] \stackrel{\alpha}{\equiv} e[\![e_0]\!][\nu := y]_{\mathtt{w}}$

(b) $(\lambda\nu.e)[\![e_0]\!] \stackrel{\alpha}{\equiv} \lambda\nu.(e[\![e_0]\!]) \qquad$ if $\nu \notin \mathrm{FV}(e_0)$

2.4 Strong Substitution

We complete the picture by defining strong substitution.

Definition ($e[\sigma]_{\mathtt{s}} \quad \sigma_1[\sigma_2]_{\mathtt{s}}$):

$$e[\sigma]_{\mathtt{s}} = \begin{cases} \bullet^{\sigma_1[\sigma]_{\mathtt{s}}} & \text{if } e = \bullet^{\sigma_1} \\ e & \text{if } e \in \mathbb{A} \cup (\mathbb{X} - \mathrm{Dom}(\sigma)) \\ \sigma(e) & \text{if } e \in \mathrm{Dom}(\sigma) \\ \lambda\nu.(e_0[z := \nu]_{\mathtt{w}}[\sigma]_{\mathtt{s}}) & \text{if } e = \lambda z.e_0, \text{ and } \nu \text{ is fresh} \\ \mathtt{app}(e_0[\sigma]_{\mathtt{s}}, e_1[\sigma]_{\mathtt{s}}) & \text{if } e = \mathtt{app}(e_0, e_1) \end{cases}$$

$$\sigma_1[\sigma]_{\mathtt{s}} = \underline{\lambda} x \in \mathrm{Dom}(\sigma_1) \cup \mathrm{Dom}(\sigma).\begin{cases} \sigma_1(x)[\sigma]_{\mathtt{s}} & \text{if } x \in \mathrm{Dom}(\sigma_1), \\ \sigma(x) & \text{if } x \in \mathrm{Dom}(\sigma) - \mathrm{Dom}(\sigma_1) \end{cases}$$

A comparison between this definition and that of weak substitution reveals that the solitary difference between the two definitions lies in what takes place at holes. In this respect it is important to note that $\mathrm{Dom}(\sigma_1[\sigma]_{\mathtt{s}}) = \mathrm{Dom}(\sigma) \cup \mathrm{Dom}(\sigma_1)$, in constrast with weak substitution. This has the following consequence:

Example (6): $\neg(e[x := \nu_0]_{\mathtt{s}}[\nu_0 := \nu_1]_{\mathtt{s}} \stackrel{\alpha}{\equiv} e[x := \nu_1]_{\mathtt{s}})$ for ν_i fresh.

Proof of Example (6): Simply take $e = \bullet$. Then

$$e[x := \nu_0]_{\mathtt{s}}[\nu_0 := \nu_1]_{\mathtt{s}} = \bullet[x := \nu_0]_{\mathtt{s}}[\nu_0 := \nu_1]_{\mathtt{s}} = \bullet^{\{x:=\nu_0\}}[\nu_0 := \nu_1]_{\mathtt{s}} = \bullet^{\{x:=\nu_1, \nu_0:=\nu_1\}}$$
$$e[x := \nu_1]_{\mathtt{s}} = \bullet[x := \nu_1]_{\mathtt{s}} = \bullet^{\{x:=\nu_1\}}$$

and clearly $\neg(\{x := \nu_0, \nu_0 := \nu_1\} \stackrel{\alpha}{\equiv} \{x := \nu_1\})$ since they do not have the same domains. □

An equivalent definition of strong substitution can be given in terms of trapping and weak substitution, as we mentioned in the previous section.

Lemma (.[.]$_{\mathtt{s}}$.1):

$$e[\sigma]_{\mathtt{s}} \stackrel{\alpha}{\equiv} \mathbf{Trap}(e, \mathrm{Dom}(\sigma))[\sigma]_{\mathtt{w}}$$

Thus an immediate consequence of this is that $\bullet^{\sigma}[\![e_0]\!] = e_0[\sigma[\![e_0]\!]]_{\mathtt{s}}$. It also follows that strong substitution is functional modulo α-congruence.

Corollary ($\stackrel{\alpha}{\equiv}$.4):

$$(e_0 \stackrel{\alpha}{\equiv} e_1 \wedge \sigma_0 \stackrel{\alpha}{\equiv} \sigma_1) \Rightarrow e_0[\sigma_0]_{\mathtt{s}} \stackrel{\alpha}{\equiv} e_1[\sigma_1]_{\mathtt{s}}$$

Proof of Corollary ($\stackrel{\alpha}{\equiv}$.4): Assume the hypothesis. Then

$e_0[\sigma_0]_{\mathtt{s}} \stackrel{\alpha}{\equiv} \mathbf{Trap}(e_0, \mathrm{Dom}(\sigma_0))[\sigma_0]_{\mathtt{w}}$ by the previous lemma

$\stackrel{\alpha}{\equiv} \mathbf{Trap}(e_0, \mathrm{Dom}(\sigma_0))[\sigma_1]_{\mathtt{w}}$ by proposition ($\stackrel{\alpha}{\equiv}$.1) since $\sigma_0 \stackrel{\alpha}{\equiv} \sigma_1$

$= \mathbf{Trap}(e_0, \mathrm{Dom}(\sigma_1))[\sigma_1]_{\mathtt{w}}$ since $\sigma_0 \stackrel{\alpha}{\equiv} \sigma_1$ implies $\mathrm{Dom}(\sigma_0) = \mathrm{Dom}(\sigma_1)$

$\stackrel{\alpha}{\equiv} \mathbf{Trap}(e_1, \mathrm{Dom}(\sigma_1))[\sigma_1]_{\mathtt{w}}$ by propositions ($\stackrel{\alpha}{\equiv}$.1) and ($\stackrel{\alpha}{\equiv}$.2) since $e_0 \stackrel{\alpha}{\equiv} e_1$

$\stackrel{\alpha}{\equiv} e_1[\sigma_1]_{\mathtt{s}}$ again by the previous lemma

□

Example (7): Suppose that σ_0 contains no holes. Then

$$\bullet^{\sigma_0}[\![\bullet^{\sigma_1}]\!] \stackrel{\alpha}{\equiv} \bullet^{\sigma_1[\sigma_0]_{\mathtt{s}}}$$

Proof of Example (7):

$\bullet^{\sigma_0}[\![\bullet^{\sigma_1}]\!] = \mathbf{Trap}(\bullet^{\sigma_1}, \mathrm{Dom}(\sigma_0))[\sigma_0[\![\bullet^{\sigma_1}]\!]]_{\mathtt{w}}$ by definition

$= \mathbf{Trap}(\bullet^{\sigma_1}, \mathrm{Dom}(\sigma_0))[\sigma_0]_{\mathtt{w}}$ since σ_0 contains no holes

$= \bullet^{\sigma_1}[\sigma_0]_{\mathtt{s}}$ by (.[.]$_{\mathtt{s}}$.1)

$= \bullet^{\sigma_1[\sigma_0]_{\mathtt{s}}}$ by definition

□

Lemma (.[.]$_{\mathtt{s}}$.2):

(a) $(\nu_0 \notin (\mathrm{FV}(\sigma) \cup \mathrm{Dom}(\sigma)) \wedge \nu_1 \notin \mathrm{Dom}(\sigma)) \Rightarrow e[\nu_0 := \nu_1]_{\mathtt{w}}[\sigma]_{\mathtt{s}} \stackrel{\alpha}{\equiv} e[\sigma]_{\mathtt{s}}[\nu_0 := \nu_1]_{\mathtt{w}}$

(b) $(\lambda\nu.e)[\sigma]_{\mathtt{s}} \stackrel{\alpha}{\equiv} \lambda\nu.(e[\sigma]_{\mathtt{s}})$ if $\nu \notin (\mathrm{FV}(\sigma) \cup \mathrm{Dom}(\sigma))$

In contrast to weak substitution, strong substitution satisfies the usual composition principle.

Proposition (.[.]$_{\mathtt{s}}$.3):

$$e[\sigma_0]_{\mathtt{s}}[\sigma_1]_{\mathtt{s}} \stackrel{\alpha}{\equiv} e[\sigma_0[\sigma_1]_{\mathtt{s}}]_{\mathtt{s}}$$

Proposition (.[.]$_{\mathtt{s}}$.4):

$$\mathbf{Trap}(e, X) \stackrel{\alpha}{\equiv} e[\underline{\lambda}x \in X.x]_{\mathtt{s}}$$

We may collect all the main results of the previous sections into a single theorem:

Theorem (1):

(1) $(e_0 \stackrel{\alpha}{\equiv} e_1 \wedge \sigma_0 \stackrel{\alpha}{\equiv} \sigma_1) \Rightarrow e_0[\sigma_0]_{\mathtt{w}} \stackrel{\alpha}{\equiv} e_1[\sigma_1]_{\mathtt{w}}$

(2) $e_0 \stackrel{\alpha}{\equiv} e_1 \Rightarrow \mathbf{Trap}(e_0, X) \stackrel{\alpha}{\equiv} \mathbf{Trap}(e_1, X)$

(3) $(e_0 \stackrel{\alpha}{\equiv} e_1 \wedge e_0' \stackrel{\alpha}{\equiv} e_1') \Rightarrow e_0[\![e_0']\!] \stackrel{\alpha}{\equiv} e_1[\![e_1']\!]$

(4) $(e_0 \stackrel{\alpha}{\equiv} e_1 \wedge \sigma_0 \stackrel{\alpha}{\equiv} \sigma_1) \Rightarrow e_0[\sigma_0]_{\mathtt{s}} \stackrel{\alpha}{\equiv} e_1[\sigma_1]_{\mathtt{s}}$

The main result of this section is that, unlike weak substitution, strong substitution commutes with filling.

Theorem (2):

$$e[\sigma]_{\mathtt{s}}[\![e']\!] \stackrel{\alpha}{\equiv} e[\![e']\!][\sigma[\![e']\!]]_{\mathtt{s}}$$

Corollary (2):

$$\mathbf{Trap}(e[\![e_0]\!], \mathrm{Dom}(\sigma))[\sigma[\![e_0]\!]]_{\mathtt{w}} \stackrel{\alpha}{\equiv} \mathbf{Trap}(e, \mathrm{Dom}(\sigma))[\sigma]_{\mathtt{w}}[\![e_0]\!]$$

3 Notions of Reduction

3.1 Classical β-Reduction

The first problem that must be faced is: *In this more general setting is β-reduction defined via strong or weak substitution?* This turns out to be quite easy to answer, although the answer may be surprising. Suppose that β is defined via strong substitution:

(β.s) $\quad \mathtt{app}(\lambda x.e_0, e_1) \stackrel{\beta}{\longmapsto} e_0[x := e_1]_{\mathtt{s}}$

Then by theorem **(2)** β reduction commutes with filling:

$$\begin{array}{ccc} \mathtt{app}(\lambda x.e, e_1)[\![e_0]\!] & \stackrel{[\![]\!]}{\longrightarrow} & \mathtt{app}((\lambda x.e)[\![e_0]\!], e_1[\![e_0]\!]) \\ \downarrow_{\beta.\mathrm{s}} & & \downarrow_{\beta.\mathrm{s}} \\ (e[x := e_1]_{\mathtt{s}})[\![e_0]\!] & \stackrel{[\![]\!]}{\longrightarrow} & (e[\![e_0]\!])[x := e_1[\![e_0]\!]]_{\mathtt{s}} \end{array}$$

However it is not functional modulo α–conversion: Suppose for sake of argument that β-reduction (so defined) does preserve equivalence. Observe by proposition (.[.]$_{\mathtt{w}}$**.1.d**) that for ν fresh

$$\lambda x.e_0 \stackrel{\alpha}{\equiv} \lambda \nu.e_0[x := \nu]_{\mathtt{w}}$$

consequently β reducing the application of either to e_1 yields

$$e_0[x := e_1]_{\mathtt{s}} \stackrel{\alpha}{\equiv} e_0[x := \nu]_{\mathtt{w}}[\nu := e_1]_{\mathtt{s}}$$

by considering the case when e_0 is the undecorated hole $\bullet$ we are forced into the unpleasant situation of concluding that

$$\bullet^{\{x:=e_1\}} \stackrel{\alpha}{\equiv} \bullet^{\{\nu:=e_1\}}.$$

On the other hand, suppose that β is defined via weak substitution:

(β.w) $\quad \mathtt{app}(\lambda x.e, e_1) \stackrel{\beta}{\longmapsto} e[x := e_1]_{\mathtt{w}}$

Then by virtue of proposition ($\stackrel{\alpha}{\equiv}$ **.1**) it is functional modulo α–conversion, but it is not at all not obvious that this form β reduction commutes with filling. Indeed, the fact that weak substitution does *not* commute with filling suggests that the contrary is true. Happily this is an illusion:

Theorem (3):

$$\begin{array}{ccc} \mathtt{app}(\lambda x.e, e_1)[\![e_0]\!] & \stackrel{[\![]\!]}{\longrightarrow} & \mathtt{app}((\lambda x.e)[\![e_0]\!], e_1[\![e_0]\!]) \\ \downarrow_{\beta.\mathrm{w}} & & \downarrow_{\beta.\mathrm{w}} \\ (e[x := e_1]_{\mathtt{w}})[\![e_0]\!] & \stackrel{[\![]\!]}{\longrightarrow} & (e[\![e_0]\!])[x := e_1[\![e_0]\!]]_{\mathtt{w}} \end{array}$$

This theorem is established by proving the following *weak* form of commutation between weak substitution and filling:

Lemma (3): If ν is fresh, then

$$(e[x := e_1]_{\mathtt{w}})[\![e_0]\!] \stackrel{\alpha}{\equiv} (e[x := \nu]_{\mathtt{w}})[\![e_0]\!][\nu := e_1[\![e_0]\!]]_{\mathtt{w}}$$

As a result of these observations we make the following definition.

Definition ($\stackrel{\beta}{\longmapsto}$):

(β) $\quad \mathtt{app}(\lambda x.e_0, e_1) \stackrel{\beta}{\longmapsto} e_0[x := e_1]_{\mathtt{w}}$

We define unrestricted multiple step reduction $\stackrel{*}{\longmapsto}$ be the smallest reflexive transitive relation generated by:

$(\beta\kappa)$ $\quad e[\![\mathtt{app}(\lambda x.e_0, e_1)]\!] \stackrel{*}{\longmapsto} e[\![e_0[x := e_1]_{\mathtt{w}}]\!]$

Many (but not all, e.g. the λ-I calculus) variations of the λ-calculus can be obtained by restricting, in various ways, the $\beta\kappa$ reductions and perhaps adding new forms of reductions. We give several examples to illustrate this idea.

3.2 Call-by-Value λ-calculus

To describe the call-by-value λ-calculus we must first define the set of value expressions, $\mathbb{V}$, or more simply the set of values: $\mathbb{V} = \mathbb{A} \cup \mathbb{X} \cup \mathbb{L}$. The call-by-value β-reduction is defined by:

(β_v) $\quad \mathtt{app}(\lambda x.e_0, e_1) \stackrel{\beta}{\longmapsto} e_0[x := e_1]_{\mathtt{w}}$ $\qquad$ provided $e_1 \in \mathbb{V}$

We define unrestricted multiple step reduction $\stackrel{*}{\longmapsto}$ be the smallest reflexive transitive relation generated by:

$(\beta_v\kappa)$ $\quad e[\![\mathtt{app}(\lambda x.e_0, e_1)]\!] \stackrel{*}{\longmapsto} e[\![e_0[x := e_1]_{\mathtt{w}}]\!]$ $\qquad$ provided $e_1 \in \mathbb{V}$

3.3 Left-first call-by-Value λ-calculus

To describe the left first call-by-value λ-calculus (ISWIM) we first define reduction contexts (a.k.a evaluation contexts) and use them to restrict the order of evaluation. Reduction contexts identify the subexpression of an expression that is to be evaluated next, they correspond to the standard reduction strategy (left-first, call-by-value) of [15] and were first introduced in [6]. The set of reduction contexts, $\mathbb{R}$, is the subset of $\mathbb{E}$ defined by

$$\mathbb{R} = \{\bullet\} + \mathtt{app}(\mathbb{R}, \mathbb{E}) + \mathtt{app}(\mathbb{V}, \mathbb{R})$$

We define left-first multiple step reduction to be the smallest reflexive transitive relation generated by:

$(\beta_v\kappa.\mathrm{lf})$ $\quad e[\![\mathtt{app}(\lambda x.e_0, e_1)]\!] \stackrel{*}{\longmapsto} e[\![e_0[x := e_1]_{\mathtt{w}}]\!]$ $\qquad$ provided $e_1 \in \mathbb{V}$ and $e \in \mathbb{R}$

3.4 $\lambda_{\mathtt{cc}}$-calculus

Following [8] we present the $\lambda_{\mathtt{cc}}$-calculus as the left-first call-by-value λ-calculus enriched with two control primitives $\mathtt{A}$ and $\mathtt{C}$ in $\mathbb{A}$. The reduction relation is the least transitive, reflexive releation generated by ($\beta_v\kappa$**.lf**) extended by the two delta rules $\delta.\mathtt{A}$ and $\delta.\mathtt{C}$

$(\beta_v\kappa.\mathrm{lf})$ $\quad e[\![\mathtt{app}(\lambda x.e_0, e_1)]\!] \stackrel{*}{\longmapsto} e[\![e_0[x := e_1]_{\mathtt{w}}]\!]$ $\qquad$ provided $e_1 \in \mathbb{V}$ and $e \in \mathbb{R}$

$(\delta.\mathtt{A})$ $\quad e[\![\mathtt{app}(\mathtt{A}, e_0)]\!] \stackrel{\delta}{\longmapsto} e_0$ $\qquad$ provided $e \in \mathbb{R}$

$(\delta.\mathtt{C})$ $\quad e[\![\mathtt{app}(\mathtt{C}, e_0)]\!] \stackrel{\delta}{\longmapsto} \mathtt{app}(e_0, \lambda z.\mathtt{app}(\mathtt{A}, e[\![z]\!]))$ $\qquad$ provided $e \in \mathbb{R}$

3.5 $\lambda_{\mathtt{M}}$-calculus

Following [9] we define the $\lambda_{\mathtt{M}}$-calculus as left-first call-by-value λ-calculus enriched with the three memory primitives $\mathtt{M}$, $\mathtt{G}$ and $\mathtt{S} \in \mathbb{A}$ (called *make, get* and *set*) together an pairing operation $\mathtt{D}$. The pairing operation separates the part of the expression that represents the memory, from the part that represents the current computation (see §2.2.1 of [9] for details). In what follows we will give describe the actual terms as well as give more readable variations using the traditional `let` abbreviation for λ-application, function application $F(X)$ to represent $\mathtt{app}(F, X)$ (similarly $F(X_1, X_2)$ for binary application $\mathtt{app}(\mathtt{app}(F, X_1), X_2)$), $\mathtt{I}$ for the identity $\lambda y.y$, infix ; for the sequencing construct, and following [18] $e_0 : e_1$ to abbreviate $\mathtt{app}(\mathtt{app}(\mathtt{D}, e_0), e_1)$. The set of memory contexts, $\mathbb{M}$, is defined inductively as the smallest subset of $\mathbb{E}$ containing $\bullet$ and closed under the following formation rules:

If $e \in \mathbb{M}$, and $x \in \mathbb{X}$ is fresh, then

$$\mathtt{app}(\lambda x.(e[\![\bullet^{\{x:=x\}}]\!]), \mathtt{app}(\mathtt{M}, \lambda y.y)) \in \mathbb{M} \qquad \text{i.e.} \qquad \mathtt{let}\{x := \mathtt{M(I)}\}(e[\![\bullet^{\{x:=x\}}]\!]) \in \mathbb{M}$$

If $e \in \mathbb{M}$, $v \in \mathbb{V}$, and $x \in \mathrm{TV}(e)$, then

$$e[\![\mathtt{app}(\lambda z.\bullet, \mathtt{app}(\mathtt{app}(\mathtt{S}, x), v))]\!] \in \mathbb{M} \qquad \text{i.e.} \qquad e[\![\mathtt{S}(x, v) ; \bullet]\!] \in \mathbb{M}$$

Note that no hygene conditions are needed concerning z. If $e \in \mathbb{M}$, then either e is a `let`-context (i.e. contains no occurrences of $\mathtt{S}$) or upto α-congruence there exists unique $e_0, e_1 \in \mathbb{E}$ and $\sigma \in \mathbb{S}$ such that $e_1 \in \mathbb{M}$ has a unique hole $\bullet^\sigma$, σ is a bijection and its range is in $\mathbb{X}$, $e \overset{\alpha}{\equiv} e_1[\![e_0 ; \bullet]\!]$ and $e_0 \overset{\alpha}{\equiv} \mathtt{S}(x, v)$ with $x \in \mathrm{Dom}(\sigma)$. Using this decomposition we can define a simple rewrite system:

Definition ($\overset{z}{\hookrightarrow}$): Assume that $e \in \mathbb{M}$ decomposes into $e_1[\![\mathtt{S}(x, v) ; \bullet]\!]$ with $\bullet^\sigma$ being the unique hole of e_1 .

$$e_1[\![\mathtt{S}(x, v) ; \bullet]\!] \overset{z}{\hookrightarrow} \begin{cases} v[\sigma^{-1}]_{\mathtt{w}} & \text{if } \sigma^{-1}(x) = z \\ e_1 & \text{otherwise} \end{cases}$$

Similarly we write $\overset{z*}{\hookrightarrow}$ for the transitive closure of this relation. We are now in a position to define the reduction rules for the $\lambda_{\mathtt{M}}$-calculus.

($\mathbb{M}\beta_v\kappa$.lf) $\mathtt{app}(\mathtt{app}(\mathtt{D}, e_2), e[\![\mathtt{app}(\lambda x.e_0, e_1)]\!]) \overset{*}{\longmapsto} \mathtt{app}(\mathtt{app}(\mathtt{D}, e_2), e[\![e_0[x := e_1]_{\mathtt{w}}]\!])$

provided $e_2 \in \mathbb{M}$, $e_1 \in \mathbb{V}$ and $e \in \mathbb{R}$

i.e. $e_2 : e[\![\mathtt{app}(\lambda x.e_0, e_1)]\!] \overset{*}{\longmapsto} e_2 : e[\![e_0[x := e_1]_{\mathtt{w}}]\!]$

($\mathbb{M}\delta$.M) $\mathtt{app}(\mathtt{app}(\mathtt{D}, e_2), e[\![\mathtt{app}(\mathtt{M}, e_1)]\!]) \overset{\delta}{\longmapsto}$

$\mathtt{app}(\mathtt{app}(\mathtt{D}, \mathtt{app}(\lambda x.(e_2[\![\mathtt{app}(\lambda z.\bullet^{\{x:=x\}}, \mathtt{app}(\mathtt{app}(\mathtt{S}, x), e_1))]\!]), \mathtt{app}(\mathtt{M}, \lambda y.y))), e[\![x]\!])$

provided $e_2 \in \mathbb{M}$, $e_1 \in \mathbb{V}$, $e \in \mathbb{R}$ and $x \in \mathbb{X}$ is fresh

i.e. $e_2 : e[\![\mathtt{app}(\mathtt{M}, e_1)]\!] \overset{\delta}{\longmapsto} \mathtt{let}\{x := \mathtt{M(I)}\}(e_2[\![\mathtt{S}(x, e_1) ; \bullet^{\{x:=x\}}]\!]) : e[\![x]\!]$

($\mathbb{M}\delta$.G) $\mathtt{app}(\mathtt{app}(\mathtt{D}, e_2), e[\![\mathtt{app}(\mathtt{G}, e_1)]\!]) \overset{\delta}{\longmapsto} \mathtt{app}(\mathtt{app}(\mathtt{D}, e_2), e[\![e_3]\!])$

provided $e_2 \in \mathbb{M}$, $e_1 \in \mathbb{X} \cap \mathrm{TV}(e_2)$, $e \in \mathbb{R}$ and $e_2 \overset{e_1*}{\hookrightarrow} e_3 \in \mathbb{V}$.

i.e. $e_2 : e[\![\mathtt{G}(e_1)]\!] \overset{\delta}{\longmapsto} e_2 : e[\![e_3]\!]$

($\mathbb{M}\delta$.S) $\mathtt{app}(\mathtt{app}(\mathtt{D}, e_2), e[\![\mathtt{app}(\lambda z.\bullet, \mathtt{app}(\mathtt{app}(\mathtt{S}, x), e_1))]\!]) \overset{\delta}{\longmapsto} \mathtt{app}(\mathtt{app}(\mathtt{D}, e_2[\![\mathtt{app}(\mathtt{app}(\mathtt{S}, x), e_1)]\!]), e[\![\lambda y.y]\!])$

provided $e_2 \in \mathbb{M}$, $e_1 \in \mathbb{V}$, $e \in \mathbb{R}$ and $x \in \mathrm{TV}(e_2)$

i.e. $e_2 : e[\![\mathtt{S}(x, e_1)]\!] \overset{\delta}{\longmapsto} e_2[\![\mathtt{S}(x, e_1) ; \bullet]\!] : e[\![\mathtt{I}]\!]$

Observe that

$$(\mathtt{let}\{x := \mathtt{M(I)}\}(e[\![\bullet^{\{x:=x\}}]\!]))[\![\mathtt{S}(x,v)\,;\bullet]\!] \stackrel{\alpha}{\equiv} \mathtt{let}\{x := \mathtt{M(I)}\}(e[\![\mathtt{S}(x,v)\,;\bullet^{\{x:=x\}}]\!])$$

and so in the δ rule ($\mathbb{M}.\delta.\mathtt{M}$) the resulting *memory component* is indeed an element of $\mathbb{M}$. Thus the reduction system so defined, contains only one complex side condition. Other than the δ rule ($\mathbb{M}.\delta.\mathtt{G}$) all reductions rely on simple recursive conditions. The side conditions on ($\mathbb{M}.\delta.\mathtt{G}$) are expressed via a simple reduction system.

4 Conclusions & Future Work

In this paper we have presented a *named variable* version of Talcott's binding structure theory [16, 17] and established several important results concerning it. We have also shown that computing with contexts, via the various forms of reduction presented in §3, is not problematic. We have also presented reduction systems for the lambda calculus enriched with control and imperative features that should be easily encoded into modern logical frameworks. Thus enabling the syntax and semantics of logics such as VTLoE to be encoded [9]. This would allow for both proof theoretical and semantic reasoning to be carried out at the same time in the same framework. It will also allow the system to semantically verify the soundness of its own proof system. It would also allow for the dynamic enrichment (via a sort of *meta-rule*) of the proof theory by introducing new, semantically verified, principles. Thus the logic implemented would be truly dynamic.

The actual choice of logical framework & the subsequent encoding is the subject of future work. It would also be of interest to compare the theory presented in this paper with the recent (de Bruijn) theories of explicit substitutions [2].

Acknowledegments: Most of this work was done while the author was at the University of Tasmania Launceston, and partially supported by ARC grant M0008202. The author would also like to thank Carolyn Talcott for providing much of the inspiration and error detection.

References

[1] A.Avron, F.Honsell, I.A. Mason, and Robert Pollack. Using Typed Lambda Calculus to Implement Formal Systems on a Machine. *Journal of Automated Reasoning*, 9:309–354, 1992.

[2] M. Abadi, L Cardeli, P.-L. Curien, and J.-J. Lévy. Explicit substitutions. In *Seventeenth Annual ACM Symposium on Principles of Programmings Languages*, pages 31–46, 1990.

[3] G. Agha, I. A. Mason, S. F. Smith, and C. L. Talcott. A foundation for actor computation, 1999. to appear in the Journal of Functional Programming.

[4] H.B. Curry and R. Feys. *Combinatory Logic*. Studies in Logic and the Foundations of Mathematics. North-Holland, 1958.

[5] M. Felleisen. *The Calculi of Lambda-v-cs Conversion: A Syntactic Theory of Control and State in Imperative Higher-Order Programming Languages*. PhD thesis, Indiana University, 1987.

[6] M. Felleisen and D.P. Friedman. Control operators, the SECD-machine, and the λ-calculus. In M. Wirsing, editor, *Formal Description of Programming Concepts III*, pages 193–217. North-Holland, 1986.

[7] J. Frost and I. A. Mason. An Operational Logic of Effects. In *Proceedings of the Australasian Theory Symposium, CATS '96*, pages 147–156, 1996.

[8] T. G. Griffin. A formulae as types notion of control. In *Seventeenth Annual ACM Symposium on Principles of Programmings Languages*, pages 47–58, 1990.

[9] F. Honsell, I. A. Mason, S. F. Smith, and C. L. Talcott. A Variable Typed Logic of Effects. *Information and Computation*, 119(1):55–90, May 1995.

[10] I. A. Mason. A First Order Logic of Effects, 1996. submitted to *Theoretical Computer Science*.

[11] I. A. Mason and C. L. Talcott. Equivalence in functional languages with effects. *Journal of Functional Programming*, 1:287–327, 1991.

[12] I.A. Mason. Hoare's Logic in the LF. Technical Report ECS-LFCS-87-32, Laboratory for foundations of computer science, University of Edinburgh, 1987.

[13] R. Milner. Fully abstract models of typed λ-calculi. *Theoretical Computer Science*, 4:1–22, 1977.

[14] J. H. Morris. *Lambda calculus models of programming languages*. PhD thesis, Massachusetts Institute of Technology, 1968.

[15] G. Plotkin. Call-by-name, call-by-value and the lambda calculus. *Theoretical Computer Science*, 1:125–159, 1975.

[16] C. L. Talcott. Binding structures. In Vladimir Lifschitz, editor, *Artificial Intelligence and Mathematical Theory of Computation*. Academic Press, 1991.

[17] C. L. Talcott. A theory of binding structures and its applications to rewriting. *Theoretical Computer Science*, 112:99–143, 1993.

[18] C. L. Talcott. Reasoning about functions with effects. In A. Gordon and A. Pitts, editors, *Higher Order Operational Techniques in Semantics*. (to appear), 1997.

[19] R. W. Weyhrauch. Prolegomena to a theory of formal reasoning. *Artificial Intelligence*, 13:133–170, 1980.