The structure of call-by-value

Carsten Führmann

Doctor of Philosophy
University of Edinburgh
2000

To my parents

Abstract

Understanding procedure calls is crucial in computer science and everyday programming. Among the most common strategies for passing procedure arguments (‘evaluation strategies’) are ‘call-by-name’, ‘call-by-need’, and ‘call-by-value’, where the latter is the most commonly used. While reasoning about procedure calls is simple for call-by-name, problems arise for call-by-need and call-by-value, because it matters how often and in which order the arguments of a procedure are evaluated.

We shall classify these problems and see that all of them occur for call-by-value, some occur for call-by-need, and none occur for call-by-name. In that sense, call-by-value is the ‘greatest common denominator’ of the three evaluation strategies.

Reasoning about call-by-value programs has been tackled by Eugenio Moggi’s ‘computational lambda-calculus’, which is based on a distinction between ‘values’ and arbitrary expressions. However, the computational lambda-calculus deals only implicitly with the evaluation order and the number of evaluations of procedure arguments. Therefore, certain program equivalences that we should be able to spot immediately require long proofs. We shall remedy this by introducing a new calculus—the ‘let-calculus’—that deals explicitly with evaluation order and the number of evaluations. For dealing with the number of evaluations, the let-calculus has mechanisms known from linear, affine, and relevant logic. For dealing with evaluation order, it has a mechanism which seems to be completely new.

We shall also introduce a new kind of denotational semantics for call-by-value programming languages. The key idea is to consider how categories with finite products are commonly used to model call-by-name languages, and remove the axioms which break for call-by-value. The resulting models we shall call ‘precartesian categories’. These relatively simple structures have remarkable mathematical properties, which will inspire the design of the let-calculus.

Precartesian categories will provide a semantics of both the let-calculus and the computational lambda-calculus. This semantics not only validates the same program equivalences as Moggi’s monad-based semantics of the computational lambda-calculus; It is also ‘direct’ by contrast to Moggi’s semantics, which implicitly performs a language transform. Our direct semantics has practical benefits:

It clarifies issues that are related with the evaluation order and the number of evaluations of procedure arguments, and it is also very easy to remember.

The thesis is rounded up by three applications of the let-calculus and precartesian categories: First, construing well-established models of partiality (i.e. categories of generalised partial functions) as precartesian categories, and specialising the let-calculus accordingly. Second, adding global state to a given computational system and construing the resulting system as a precartesian category. Third, analysing an implementation technique called 'continuation-style transform' by construing the source language of such a transform as a precartesian category.

Acknowledgements

I would like to thank my pleasant and easy-going supervisors Stuart Anderson and John Power for their patient and constant support throughout my postgraduate years in Edinburgh.

Many thanks to Hayo Thielecke, whose work on categorical models of continuations provided the entryway to my research.

Thanks to Peter O'Hearn for inviting me to the Queen Mary and Westfield College, and engaging me in many helpful discussions.

Alex Simpson and Anna Bucalo, who kindly invited me to take part in a joint article, enabled me to write the chapter on partiality in this thesis.

My friend Konstantinos Tourlas, who never got tired of reviewing drafts of this thesis, helped me with many suggestons that revealed both deep insight and common sense.

Peter Selinger, who involved me in several inspiring discussions, gave me some valuable mathematical clues.

Thanks to Alan Jeffrey and Ralph Schweimeier for pointing out to me some advanced issues (in particular, Alan's 'premonoidal-copyable' morphisms), which I may address in my future work

My friend Martin Wehr, who spurred my research by showing curiosity and enthusiasm, has been a constant source of inspiration.

Many thanks to Bob Tennent for reviewing a considerable part of this thesis and making several helpful suggestions.

Thanks also to Fabio Gaducci for comments, discussions, and references to the literature.

Thanks to Paul Taylor for his diagrams package, which I used for typesetting a dozen or so pages of this thesis.

I am particularly grateful to my friend Terry Stroup, who years ago incited my scientific curiosity and encouraged me to go to Edinburgh.

Many thanks to all my friends, and especially to my (ex-)flatmates Steve, Kostas, Conor, and Melanie, for their various contributions to my well-being during the last three years.

I dedicate this thesis to my parents, who supported me whenever necessary.

This thesis has been supported by the EU TMR Marie Curie Research Training Grant N°ERBFMBICT982906.

Declaration

I declare that this thesis was composed by myself and that the work contained therein is my own, except where explicitly stated otherwise in the text.

(*Carsten Führmann*)

Table of Contents

List of Figures

Chapter 1

Introduction

1.1 Procedure calls and equational reasoning

Almost every programming language has procedures that accept zero, one, or more arguments. Some procedures are part of the language itself, like the two-argument +, some are provided by libraries, like `printf` in C, and new procedures may be defined by the user. Understanding procedure calls is obviously crucial in computer science and everyday programming. At a first glance, reasoning about procedure calls seems simple. Deceptively so, because it matters how often and in which order the arguments of a procedure are evaluated.

1.1.1 Number of evaluations

For example, compare the procedures `p1` and `p2` in the following piece of JAVA code, where `p` is a given procedure that accepts and integer and returns an integer[1].

Example 1.1 (JAVA).

```
int p1(int x) { return q(p(x)); }
int q(int y) { return y+y; }

int p2(int x) { return p(x)+p(x); }
```

One might be misled to conclude that `p1` and `p2` behave in the same way. Now suppose that the definition of `p` is

```
int p(int x) { print("hello!"); return x; }
```

Given an argument n, both `p1` and `p2` return the value $2n$. However, `p1` prints "`hello!`" once, whereas `p2` prints "`hello!`" twice: In the body of `p1`, `p(x)` is

[1] `print` is supposed to be defined in terms of `System.out.print` in the obvious way

evaluated once (before the call of q), and in the body of p2, p(x) is evaluated twice (before the call of +). This is so because JAVA's evaluation strategy (for arguments of basic types like int and bool) is *call-by-value*—that is, before each procedure call, all arguments are evaluated[2].

If the evaluation strategy was call-by-need, then p1 and p2 would behave like in the call-by-value case, but for different reasons: p(x) would not be evaluated before it is passed to q. However, the side effect would happen when the first y is evaluated, and not for the second y. In p2, the side effect would occur twice because there is no sharing between the two copies of p(x).

If the evaluation strategy was call-by-name, p1 and p2 would behave the same, because passing p(x) to q can be seen as the formal substitution of p(x) for y.

We can write counterparts of p1 and p2 in almost every programming language—their behaviours depend only on the evaluation strategy. In Standard ML (SML), for example, we can eliminate the procedure q to get a more elegant version of Example 1.1:

Example 1.2 (SML).

```
fun p x = (print "hello!"; x);

fun p1 x = (let val y = (p x) in y+y end);
fun p2 x = (p x)+(p x);
```

Because SML is call-by-value, the behaviours of p1 and p2 differ like in the JAVA case. Minor changes would give us an example in LISP or SCHEME.

The reason for the different behaviours of p1 and p2 lies in how often p is evaluated—once or twice. We can change our examples in such a way that p is evaluated once and not at all, respectively:

Example 1.3 (JAVA).

```
int p1(int x) { return q(p(x)); }
int q(int y) { return 0; }

int p2(int x) { return 0; }
```

The SML counterpart is

Example 1.4 (SML).

```
fun p1 x = (let val y = (p x) in 0 end);
fun p2 x = 0;
```

[2]For arguments denoting class instances, JAVA's evaluation strategy is *call by reference*, but that does not concern us here.

With the previous definition of the procedure `p`, given any argument n, `p1` prints "`hello!`" once and `p2` prints "`hello!`" not at all.

If the evaluation strategy was call-by-name, neither `p1` nor `p2` would print "`hello!`". Unlike in Example 1.1, call-by-need now behaves like call-by-name rather than call-by-value.

1.1.2 Evaluation order

The failures of substitution we have seen so far have to do with the number of evaluations of a procedure argument. There is also a different aspect that can cause a failure of substitution. Consider the following SML example:

Example 1.5 (SML).

```
fun q1 x = (print "1"; x);
fun q2 x = (print "2"; x);

fun p1 x = (let val y1 = (q1 x) in
            let val y2 = (q2 x) in x*y1+y2 end end);
fun p2 x = (let val y2 = (q2 x) in
            let val y1 = (q1 x) in x*y1+y2 end end);
```

Given an argument n, both `p1` and `p2` return the value $n^2 + n$, but `p1` prints "`12`", whereas `p2` prints "`21`". The JAVA version of the example requires auxiliary procedures `h1` and `h2`:

Example 1.6 (JAVA).

```
int p1(int x) { return h1(q1(x),x); }
int h1(int y1,int x) { return q(x,y1,q2(x)); }

int p2(int x) { return h2(x,q2(x)); }
int h2(int x,int y2) { return q(x,q1(x),y2); }

int q(int x,int y1,int y2) { return x*y1+y2; }

int q1(int x) { print("1"); return x; }
int q2(int x) { print("2"); return x; }
```

Here the different behaviour of `p1` and `p2` is due to the *evaluation order* of the arguments of `q`. For `p1`, the argument `q1(x)` is evaluated before `q2(x)`, and for `p2` it is the other way round. In JAVA and SML, it so happens that `p1(x)` is equivalent to `x*q1(x)+q2(x)`—however, this is only so because these two languages choose to evaluate the arguments of `+` from left to right.

In the cases of call-by-name and call-by-need, `p1` and `p2` would behave the same.

1.1.3 Consequences

Let's summarise the three examples from the preceding sections, using a succinct notation which is an idealisation of the SML syntax: For fitting expressions M and N we may have the observational inequalities

$$(\mathit{let}\, y = M \;\mathit{in}\, y + y) \not\equiv M + M$$
$$(\mathit{let}\, y = M \;\mathit{in}\, 0) \not\equiv 0$$

and

$$\begin{aligned}&(\mathit{let}\, y_1 = M_1 \;\mathit{in}\; \mathit{let}\, y_2 = M_2 \;\mathit{in}\, x * y_1 + y_2)\\ \not\equiv&(\mathit{let}\, y_2 = M_2 \;\mathit{in}\; \mathit{let}\, y_1 = M_1 \;\mathit{in}\, x * y_1 + y_2)\end{aligned}$$

Each of the three inequalities shows that the equation

$$(\mathit{let}\, x = M \;\mathit{in}\, N) \equiv N[x := M] \qquad (\text{let}.\beta)$$

can be observationally false (as usual, $N[y := M]$ stands for the expression that results from replacing all free occurrences of y in N by M, avoiding variable capture). Accepting call-by-value as a reasonable evaluation strategy, we must forbid the unrestricted use of Equation (let.β).

Here it is important to keep in mind that in this discussion $(\mathit{let}\, x = M \;\mathit{in}\, N)$ simply means that M is passed as an actual parameter to the a procedure with body N and formal parameter x. For example, in JAVA the expression $(\mathit{let}\, y = 42 \;\mathit{in}\, x + y)$ would stand for an expression `p(x,42)`, assuming a procedure definition

```
int p(int x,int y) { return x+y; }
```

(In particular, $(\mathit{let}\, x = M \;\mathit{in}\, N)$ does *not* force that the evaluation of M if the evaluation strategy is call-by-need or call-by-name.) So the meaning of Equation (let.β) is defined for almost every programming language, even if the let-construct is not part of the actual syntax, and independently of the evaluation strategy.

That substituting arbitrary expressions for variables can be unsound has been known for a long time. In particular, for call-by-value, the β-rule

$$(\lambda x.M)N \equiv N[x := M] \qquad (\beta)$$

of the lambda-calculus has been changed into the rule (β_v) by requiring that N be a *value*—that is, an expression which is deemed to be already evaluated [Plo75]. In fact, in the presence of a lambda-operator, our expression ($let\, x = N\, in\, M$) is just another way of writing $(\lambda x.M)N$. However, we avoid the lambda-operator in this foundational discussion, because most programming languages don't have functions as first-class citizens.

1.2 Empirically correct calculi

1.2.1 The computational lambda-calculus

Eugenio Moggi introduced the *computational lambda-calculus* (also called 'λ_C-calculus'), which, according to the abstract of [Mog88]

> *provides a correct basis for proving equivalence of programs, independent from any specific computational model.*

This claim seems empirically true (we shall discuss this some more in Section 3.4). The computational lambda-calculus has sequents $(\Gamma \vdash M : A)$ where Γ is an *environment*—that is, a finite, non-repetitive list of typed variables, M is a program expression whose free variables are contained in Γ, and A is the type of M. The term formation rules are presented in Figures 1.1–1.3. We can think of the unusual type TA as $unit \rightharpoonup A$, of $[M]$ as $\lambda x : unit.M$ (where x is fresh), and of $\mu(M)$ as the function application $M()$. (We shall discuss these *monadic operators* some more in Remark 5.6.) The computational lambda-calculus has two kinds of judgements: Equations of the form $(\Gamma \vdash M \equiv N : A)$, and judgements of the form $(\Gamma \vdash M \downarrow A)$ (where $(\Gamma \vdash M : A)$ and $(\Gamma \vdash N : A)$ are derivable sequents). A judgement $(\Gamma \vdash M \downarrow A)$ states that M is a value (Moggi says 'M exists'). Values can be substituted in all judgements for any free variable x of the same type. The derivation rules of the computational lambda-calculus are presented in Figure 1.4.

1.2.2 Linear, affine, relevant, and more

Observational inequalities like

$$(let\, y = M\, in\, y + y) \not\equiv M + M$$
$$(let\, y = M\, in\, 0) \not\equiv 0$$

and observational equalities like

$$(let\, y = M\, in\, y) \equiv M$$

var	$x_1 : A_1, \ldots, x_n : A_n \vdash x_i : A_i$
let	$\dfrac{\Gamma \vdash M : A \qquad \Gamma, x : A \vdash N : B}{\Gamma \vdash \mathit{let}\, x = M\ \mathit{in}\, N : B}$
$*$	$\Gamma \vdash () : \mathit{unit}$
$(-,-)$	$\dfrac{\Gamma \vdash M : A \qquad \Gamma \vdash N : B}{\Gamma \vdash (M, N) : A * B}$
π_i	$\dfrac{\Gamma \vdash M : A_1 * A_2}{\Gamma \vdash \pi_i(M) : A_i}$
constant $f : A_1, \ldots, A_n \longrightarrow B$	$\dfrac{\Gamma \vdash M_1 : A_1 \quad \ldots \quad \Gamma \vdash M_n : A_n}{\Gamma \vdash f(M_1, \ldots, M_n) : B}$

Figure 1.1: First-order fragment of the computational lambda-calculus (with minor syntactic changes)

λ	$\dfrac{\Gamma, x : A \vdash M : B}{\Gamma \vdash \lambda x : A.M : A \rightharpoonup B}$
app	$\dfrac{\Gamma \vdash M : A \rightharpoonup B \qquad \Gamma \vdash N : A}{\Gamma \vdash MN : B}$

Figure 1.2: Higher-order fragment of the computational lambda-calculus

$[-]$	$\dfrac{\Gamma \vdash M : A}{\Gamma \vdash [M] : TA}$
μ	$\dfrac{\Gamma \vdash M : TA}{\Gamma \vdash \mu(M) : A}$

Figure 1.3: Monadic fragment of the computational lambda-calculus

$\equiv$ is a congruence

$$\Gamma \vdash x \downarrow A \qquad \Gamma \vdash () \downarrow \mathit{unit} \qquad \Gamma \vdash (x_1, x_2) \downarrow A_1 * A_2$$

$$\Gamma \vdash \pi_i(x) \downarrow A_i \qquad \Gamma \vdash [M] \downarrow TA \qquad \Gamma \vdash \lambda x : A.M \downarrow A \rightharpoonup B$$

$$\frac{\Gamma \vdash M \downarrow A \qquad \Gamma, x : A \vdash J}{\Gamma \vdash J[x := M]} \quad \text{where } J \text{ is a judgement}$$

$\Gamma \vdash (\mathit{let}\, x = M\ \mathit{in}\, x) \equiv M : A$
$\Gamma \vdash (\mathit{let}\, x_2 = (\mathit{let}\, x_1 = M_1\ \mathit{in}\, M_2)\ \mathit{in}\, M) \equiv (\mathit{let}\, x_1 = M_1\ \mathit{in}\, (\mathit{let}\, x_2 = M_2\ \mathit{in}\, M)) : A$
$\Gamma \vdash (\mathit{let}\, x_1 = x_2\ \mathit{in}\, M) \equiv M[x_1 := x_2] : A$
$\Gamma \vdash f(M_1, \ldots, M_n) \equiv (\mathit{let}\, x_1 = M_1\ \mathit{in}\ \ldots \mathit{let}\, x_n = M_n\ \mathit{in}\, f(x_1, \ldots, x_n)) : A$
$\Gamma \vdash \mu([M]) \equiv M : A$
$\Gamma \vdash [\mu(x)] \equiv x : TA$
$\Gamma \vdash () \equiv x : \mathit{unit}$
$\Gamma \vdash (M_1, M_2) \equiv (\mathit{let}\, x_1 = M_1\ \mathit{in}\ \mathit{let}\, x_2 = M_2\ \mathit{in}\, (x_1, x_2)) : A_1 * A_2$
$\Gamma \vdash \pi_1((x_1, x_2)) \equiv x_i : A_i$
$\Gamma \vdash (\pi_1(x), \pi_2(x)) \equiv x : A_1 * A_2$
$\Gamma \vdash (\lambda x_1 : A_1.M_2)(x_1) \equiv M_2 : A_2$
$\Gamma \vdash \lambda x_1 : A_1.x(x_1) \equiv x : A_1 \rightharpoonup A_2$

Figure 1.4: Derivation rules of the computational lambda-calculus

show that whether we can substitute expressions for a variable y depends on the number of free occurrences of y. The expression $y + y$ is *relevant in y*—that is, y occurs at least once, but it is not *linear* in y—that is, y does not occur exactly once. The expression 0 is *affine* in y—that is, y occurs at most once—but it too is not linear. So non-linearity obstructs substitution.

Although the computational lambda-calculus seems to prove the right equations, it has no explicit account for the number of variable occurrences.

In this thesis, we shall see that, with a certain restriction, if an expression M is *copyable* in that

$$(let\, y = M\; in\; (y, y)) \equiv (M, M) \tag{1.1}$$

then the equation

$$(let\, y = M\; in\; N) \equiv N[y := M] \qquad\qquad (\text{let}.\beta)$$

holds whenever N is relevant in y. We shall also see that, with the same restriction, Equation (let.β) holds whenever M is *discardable* in that

$$(let\, y = M\; in\; ()) \equiv () \tag{1.2}$$

and N is affine in y. The restriction comes from the fact that evaluation order also matters. For example, we may have the observational inequality

$$\begin{aligned} &(let\, y_1 = M_1\; in\; let\, y_2 = M_2\; in\; x * y_1 + y_2) \\ \not\equiv &(let\, y_2 = M_2\; in\; let\, y_1 = M_1\; in\; x * y_1 + y_2) \end{aligned}$$

although y_1 and y_2 occur linear in $x * y_1 + y_2$. We shall see that if N is linear in y, then Equation (let.β) holds if M is *central* in that for all sequents $(\Gamma' \vdash M' : A')$ such that Γ and Γ' share no variables, it holds that

$$\begin{aligned} \Gamma, \Gamma' \vdash &(let\, y = M\; in\; let\, y' = M'\; in\; (y, y')) \\ \equiv &(let\, y' = M'\; in\; let\, y = M\; in\; (y, y')) : A * A' \end{aligned} \tag{1.3}$$

In this thesis, we shall introduce a new calculus for proving program equivalences—the *let-calculus*—which is based on counting the number of occurrences of free variables and considering an extra property of variables which is related with centrality. The expressions of the let-calculus agree with the first-order fragment of the computational lambda-calculus. By contrast, the new derivation rules for program equalities will enable us to see immediately program equivalences that would take a long proof in the computational lambda-calculus. The vocabulary that comes with the let-calculus ('relevant', 'affine', 'copyable', 'discardable', 'central', and more) seems to be a real help in discussing realistic programs.

1.3 Denotational semantics of procedure calls

1.3.1 Failure of the naïve semantics

The first-order fragment of the computational lambda-calculus clearly inspires a semantics in a category with finite products. Here we shall describe this naïve semantics and show why it fails. The type- and term-formation rules dictate how to use the categorical structure: The semantics of types is

$$\begin{aligned} [\![A * B]\!] &= [\![A]\!] \times [\![B]\!] \\ [\![unit]\!] &= 1 \end{aligned}$$

where $\times$ is the cartesian product, and 1 is the terminal object. Following common practice, the semantics of a sequent $(x_1 : A_1, \ldots, x_n : A_n \vdash M : A)$ is a morphism

$$A_1 \times \cdots \times A_n \xrightarrow{f} A$$

(For convenience, we simply write A_i for $[\![A_i]\!]$.) The semantics of first-order expressions is presented in Figure 1.5. The failure of this semantics has to do with the observational falsity of the equations 1.1, 1.2, and 1.3. A routine proof shows that the left side and the right side of Equation 1.1 denote the left-bottom and the top-right path, respectively, of the following diagram (where $\delta : A \longrightarrow A \times A$ is the diagonal $\langle id_A, id_A \rangle$):

$$\begin{tikzcd} \Gamma \arrow[r, "\delta"] \arrow[d, "f"'] & \Gamma \times \Gamma \arrow[d, "f \times f"] \\ A \arrow[r, "\delta"] & A \times A \end{tikzcd} \tag{1.4}$$

In a category with finite products this diagram commutes, so the model identifies expressions that may behave differently. The left side and the right side of Equation 1.2 denote the left-bottom and the top-right path, respectively, of the following diagram

$$\begin{tikzcd} \Gamma \arrow[r, "!"] \arrow[d, "f"'] & I \arrow[d, equal] \\ A \arrow[r, "!"] & I \end{tikzcd} \tag{1.5}$$

Again, this always holds in a category with finite products, but may be false for the program behaviour. As for Equation 1.3, let $f' : \Gamma' \longrightarrow A'$ be the denotation

Rule	Syntax	Semantics
var	$x_1 : A_1, \ldots, x_n : A_n \vdash x_i : A_i$	$= A_1 \times \cdots \times A_n \xrightarrow{\pi_i} A_i$
let	$\dfrac{\Gamma \vdash M : A \qquad \Gamma, x : A \vdash N : B}{\Gamma \vdash let\, x = M\ in\ N : B}$	$= \Gamma \xrightarrow{f} A$ $= \Gamma \times A \xrightarrow{g} B$ $= \Gamma \xrightarrow{\langle id,f \rangle} \Gamma \times A \xrightarrow{g} B$
$()$	$\Gamma \vdash () : unit$	$= \Gamma \xrightarrow{!} 1$
$(-,-)$	$\dfrac{\Gamma \vdash M : A \qquad \Gamma \vdash N : B}{\Gamma \vdash (M, N) : A * B}$	$= \Gamma \xrightarrow{f} A$ $= \Gamma \xrightarrow{g} B$ $= \Gamma \xrightarrow{\langle f,g \rangle} A \times B$
π_i	$\dfrac{\Gamma \vdash M : A_1 * A_2}{\Gamma \vdash \pi_i(M) : A_i}$	$= \Gamma \xrightarrow{f} A_1 \times A_2$ $= \Gamma \xrightarrow{f} A_1 \times A_2 \xrightarrow{\pi_i} A_i$
$f : A_1, \ldots, A_n \longrightarrow B$	$\dfrac{\Gamma \vdash M_i : A}{\Gamma \vdash f(M_1, \ldots, M_n) : B}$	$= \Gamma \xrightarrow{g_i} A_i$ $= \Gamma \xrightarrow{\langle g_1,\ldots,g_n \rangle} A_1 \times \cdots \times A_n \xrightarrow{f} B$

Figure 1.5: Naive semantics of the let-language

of $(\Gamma' \vdash M' : A')$. A routine proof shows that the two sides of Equation 1.3 denote the top-right path and the left-bottom path, respectively, of the following diagram:

$$\begin{tikzcd}
\Gamma \times \Gamma' \arrow[r, "f \times \Gamma'"] \arrow[d, "\Gamma \times f'"'] & A \times \Gamma' \arrow[d, "A \times f'"] \\
\Gamma \times A' \arrow[r, "f \times A'"] & A \times A'
\end{tikzcd} \tag{1.6}$$

This diagram too commutes in every category with finite products, so again the model identifies expressions that may behave differently.

So categories with finite products are too crude for modelling some programming languages (if we assume the straightforward semantics in Figure 1.5). In Chapter 2, we shall tackle this problem by introducing a generalisation of categories with finite products such that Diagrams 1.4, 1.5, and 1.6 need no longer commute.

1.3.2 Monadic semantics

A semantics of the computational lambda-calculus that avoids the problems described in the previous section is given by Eugenio Moggi's 'computational models'. In a realistic case, such a model is based on a category with finite products. By contrast to the naïve semantics of the previous section, a sequent $(x_1 : A_1, \ldots, x_n : A_n \vdash M : B)$ denotes a morphism

$$A_1 \times \cdots \times A_n \xrightarrow{f} TB$$

where the A_i are objects of values of type A_i, and TB is the object of computations of type B. In Moggi's own words [Mog88]:

> *There are many possible choices for TB corresponding to different notions of computations, for instance in the category of sets the set of partial computations (of type B) is the lifting $B + \{\bot\}$ and the set of non-deterministic computations is the powerset $\mathcal{P}(B)$.*

Attempting to identify the general properties that the object TB of computations must have, Moggi employed structures from category theory called *monads*. A monad $T = (T, \eta, \mu)$ in a category C consists of a functor $T : C \longrightarrow C$ and two natural transformations

$$\eta : Id_C \longrightarrow T, \qquad\qquad \mu : T^2 \longrightarrow T$$

(the *unit* and *multiplication* of the monad) which make certain diagrams commute. Moggi's computational models also require a *strength*, which is a natural transformation

$$t_{A,B} : A \times TB \longrightarrow T(A \times B)$$

that satisfies certain equations. (A monad together with a strength is called a *strong monad*.) Moggi's semantics employs a category C with finite products, together with a strong monad T in C. This semantics is often given by inductively assigning to a sequent $(\Gamma \vdash M : A)$ a morphism $\Gamma \longrightarrow TA$ of C. However, we shall now use a neat alternative, which emphasises a conceptually important point. Because C is a category with finite products, we can use the first-order fragment of the computational lambda-calculus to denote morphisms of C according to the naïve semantics in Figure 1.5. We describe the 'real' semantics of the computational lambda-calculus by a language transform—the *monadic-style transform* (MS transform)—that sends each sequent $(\Gamma \vdash M : A)$ to a sequent $(\Gamma \vdash M^\sharp : TA)$ which is to be interpreted in C. It is important to see that, despite having the same syntax, the transform's source language and target language differ conceptually in the same way as the source language and the target language of a real-life compiler. The categorical semantics of the source language is given by the transform $(-)^\sharp$, followed by the categorical semantics of the target-language in C.

Because the semantics uses the strong monad T in C, the target language needs extra syntax. A nice way to add this is the *bind-construct*:

Rule	Syntax	Semantics
bind	$\Gamma \vdash M : TA$	$= \Gamma \xrightarrow{f} TA$
	$\Gamma, x : A \vdash N : TB$	$= \Gamma \times A \xrightarrow{g} TB$
	$\Gamma \vdash \mathit{bind}\, x \Leftarrow M \mathit{\ in\ } N : TB$	$= \Gamma \xrightarrow{\langle id, f\rangle} \Gamma \times TA \xrightarrow{t} T(\Gamma \times A) \xrightarrow{Tg} TTB \xrightarrow{\mu} TB$

Intuitively, if M is an expression of type TA and x is a variable of type A, then $(\mathit{bind}\, x \Leftarrow M \mathit{\ in\ } N)$ forces the evaluation of M and binds the resulting value, if any, to x. The unit η of the monad 'wraps up' a value as a computation[3].

The transform $(-)^\sharp$ for the first-order expressions of the computational lambda-calculus is presented in Figure 1.6.

[3]Originally, Moggi used the word '*let*' instead of '*bind*', so '*let*' appeared in two different term formation rules. We use '*bind*' to rule out any confusion.

$$
\begin{array}{rcl}
x^\sharp & = & \eta(x) \\
(let\, x = M\; in\; N)^\sharp & = & \big(bind\, x \Leftarrow M^\sharp\; in\; N^\sharp\big) \\
()^\sharp & = & \eta() \\
(M, N)^\sharp & = & \big(bind\, x \Leftarrow M^\sharp\; in\; bind\, y \Leftarrow N^\sharp\; in\; \eta(x, y)\big) \\
(\pi_i(M))^\sharp & = & \big(bind\, x \Leftarrow M^\sharp\; in\; \eta(\pi_i(x))\big) \\
(f(M_1, \ldots, M_n))^\sharp & = & \Big(bind\, x_1 \Leftarrow M_1^\sharp\; in\; \ldots\, bind\, x_n \Leftarrow M_n^\sharp\; in\; f(x_1, \ldots, x_n)\Big)
\end{array}
$$

Figure 1.6: Monadic-style (MS) transform

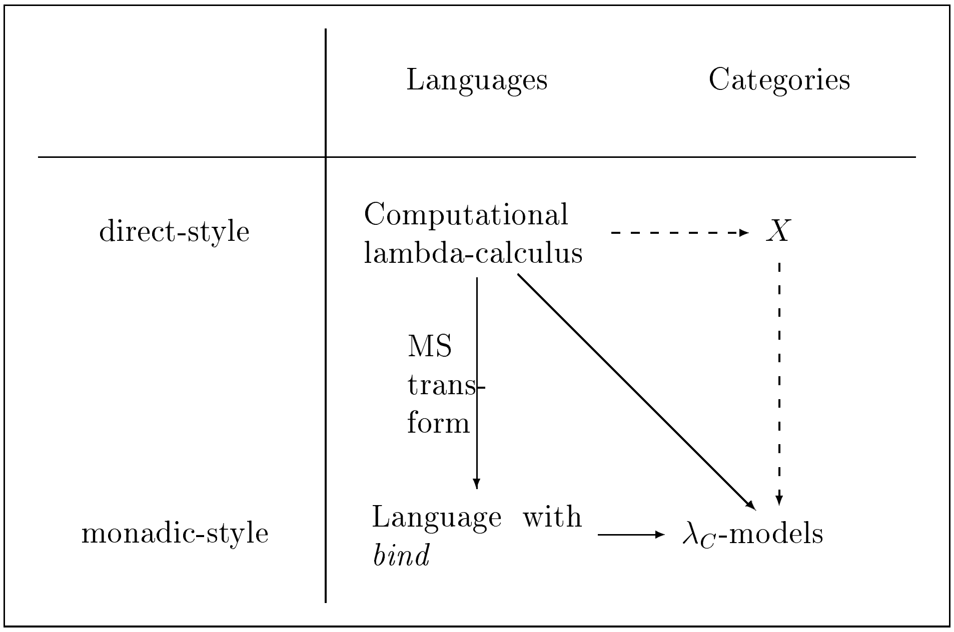


Figure 1.7: The missing direct models, marked by X

1.3.3 The missing direct semantics

The naïve semantics in Figure 1.5 assigns to a sequent $(x_1 : A_1, \ldots, x_n : A_n \vdash M : B)$ a morphism $A_1 \times \cdots \times A_n \longrightarrow B$, but it validates too many equations. By contrast, Moggi's finer-grained semantics assigns to the sequent a morphism $A_1 \times \cdots \times A_n \longrightarrow TB$, at the cost of employing a language transform (whether made explicit or not). Conceptually, Moggi's computational model provides the semantics of the target language of that transform. We do not seem to have a sufficiently fine-grained *direct* semantics of the computational lambda-calculus. The situation is depicted in Figure 1.7. The diagonal of Diagram 1.7 is the denotational semantics of the computational lambda-calculus which is usually presented. The bottom horizontal arrow is the direct semantics of the language with the bind-construct. What is missing is a class of categorical models X that provides a direct semantics (depicted by the dashed horizontal arrow) of the computational lambda-calculus—that is, a semantics that sends a sequent $(x_1 : A_1, \ldots, x_n : A_n \vdash M : B)$ to a morphism $A_1 \times \cdots \times A_n \longrightarrow B$. If we had such a semantics, there should be a semantic counterpart (depicted by the vertical dashed arrow) of the monadic-style transform.

In this thesis we shall introduce the missing class X of direct models. We shall find X by generalising categories with finite products in such a way that the problematic equations 1.4, 1.5, and 1.6 need no longer hold. We shall call the new class of models *precartesian categories*. A crucial point is that precartesian categories, while validating the same program equivalences as Moggi's semantics, are conceptually simpler and open remarkable new ways of discussing realistic computer programs. At the core of our analysis will stand notions like *copyability*, *discardability*, and *centrality*, which correspond to the program equivalences 1.1, 1.2, and 1.3.

1.4 Related work

Premonoidal categories and Freyd categories Vital for this thesis was the discovery of *premonoidal categories* by John Power and Edmund Robinson [PR97]. Premonoidal categories are a generalisation of monoidal categories in that the tensor need not be functorial in both arguments jointly. This implies abandoning the problematic Equation 1.6, allowing direct categorical models that respect inequalities caused by evaluation order. Roughly speaking, precartesian categories, which we shall define in this thesis, are premonoidal categories with generalised pairing and projections. So, in some sense, premonoidal categories are to monoidal categories what precartesian categories are to categories with finite products.

As computational models, Power and Thielecke introduced *Freyd categories* [PT99]. A Freyd category consists of two categories—a premonoidal category for modelling all expressions, and a category with finite products for modelling values only. (The definition of Freyd categories is part of Section 2.6.) Freyd categories are close relatives of precartesian categories—the main difference is that a precartesian category is really just *one* category. Conceptually, using precartesian categories instead of Freyd categories corresponds to abandoning a syntactic notion of value (see Section 2.6).

κ-categories The categorical models that we shall introduce in this thesis take each sequent $(\Gamma \vdash M : A)$ to a morphism $\Gamma \longrightarrow A$ in some category with extra structure. There is a second approach, which is based on indexed categories with extra structure (for an application to type theory, see [Jac91]). The slogan is that environments Γ are indices for the categories in which the expressions definable in Γ are modelled: An expression of type A in an environment Γ is modelled by an element $1 \longrightarrow A$ in the fibre of the indexed category over Γ. For modelling call-by-value, Power and Thielecke [PT99] consider a weak version of indexed category with extra structure, called a κ-category, which is implicit in some earlier work of Masahito Hasegawa [Has95]. As the main result in [PT99], Power and Thielecke proved that there is an equivalence between the category of κ-categories and the category of Freyd categories. Thus the two fundamental ways of modelling programming languages are closely related. While I consider indexed models to be important, they do not occur in this thesis, because I did not want to overload it.

Categorical models of continuations My work on this thesis started with studying Hayo Thielecke's $\otimes\neg$-*categories* ('tensor-not-categories') [Thi97a, Thi97b]. With hindsight, these are precartesian categories with extra structure that model the source language of a call-by-value *CPS transform* (CPS stands for 'continuation-passing style'). I obtained precartesian categories by extracting from $\otimes\neg$-categories those features that are not specific to continuations. We shall discuss the CPS transform and precartesian models of continuations in Chapter 8.

Models of partiality With hindsight, many well-known models of partiality are special precartesian categories: Concrete examples are categories of sets and partial functions, pointed cpo's and strict continuous functions, and so on. These form precartesian categories such that every morphism is central and copyable (i.e. Equations 1.6 and 1.4 hold). The total maps coincide with the discardable maps (i.e. those that satisfy Equation 1.5). Abstract definitions of such special precartesian categories have been given again and again: pre-dht-categories [Hoe77], p-categories [Ros86], copy categories (Cockett), g-monoidal categories [CG99]. According to Robinson and Rosolini [RR88], Curien and Obtulowicz [CO86] defined 'precartesian categories', which are equivalent to Rosolini's p-categories with a 'one-element object' (i.e. a tensor unit which is terminal in the subcategory of total maps). I used the name 'precartesian category' before I heard of this. Luckily, p-categories with a one-element object are a special case of our precartesian categories, as we shall see in Chapter 4, so my unintended re-definition of precartesian categories is benign.

We shall discuss models of partiality as special precartesian categories in Chapter 4.

Effect systems In the effect-systems literature, the *effect* of a program is a description of the program's non-functional behaviour: "Just as types describe what expressions compute, effects describe how expressions compute" [JG91]. Typically, effect systems have judgements like

$$\Gamma \vdash M\ : A\,!\,E$$

where Γ is an environment, M is an expression, A is the type of M, and E is the effect of M (e.g. $E = \mathtt{pure}$ if M is effect free, or $E = \mathtt{write}$ if M writes to the store). Like for types, there are rules for effect inference, for example

$$\frac{\Gamma \vdash M\ : A\,!\,E \qquad \Gamma \vdash N\ : B\,!\,F}{\Gamma \vdash (M, N)\ : A * B\,!\,E \vee F}$$

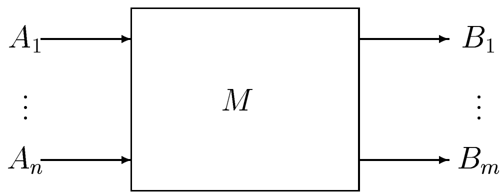
where $E \vee F$ stands for the maximal effect that may happen during the evaluation of (M, N).

The let-calculus, which we shall introduce for deriving program equivalences, has judgements of the form $(\Gamma \vdash M\,!\,E)$, and inference rules for these judgements that resemble those of effect systems. However, in the let-calculus, E ranges over abstract properties like *central*, *copyable*, and *discardable*. These abstract properties are meaningful for *almost every programming language*. For concrete languages, the abstract properties have concrete meanings: For example, in a language whose only non-functional behaviour is accessing a global store, an expression is *central* if it neither reads nor writes, and *discardable* if it does not write. (We shall discuss this in Chapter 6).

A deep study of the connection between the let-calculus and effect systems from the literature still lies in the future.

Linear, affine, and relevant logic As described in Section 1.2.2, considering numbers of free occurrences of variables can help reasoning about programs. In the let-calculus, the properties *affine* and *relevant* form two dimensions of a three-dimensional reasoning principle that I call the *precartesian cube*. Therefore, the let-calculus is related with affine, relevant, and linear logic. However, the third dimension of the precartesian cube, which deals with evaluation order, goes beyond that.

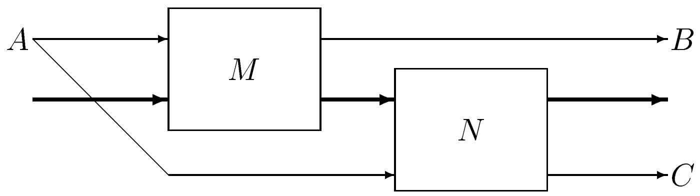
Graphs as abstract syntax Sometimes people use graphs or diagrams as an abstract syntax of programs. For example, a sequent $(x_1 : A_1, \ldots, x_n : A_n \vdash M : B_1 * \cdots * B_m)$ may be represented by a diagram



Graphs work very well as an abstract syntax for denoting morphisms of precartesian categories, and are thus an interesting alternative to the let-calculus. Using graphs in this way has been studied by Alan Jeffrey and Ralf Schweimeier [SJ99, Jef98]. (The categorical models they consider are close relatives of precartesian categories, together with a 'trace' for modelling recursion.) A key feature of their graphs is an extra 'thread' that determines the evaluation order ('control flow'). For example, for sequents $(x : A \vdash M : B)$ and $(x : A \vdash N : C)$, the sequent

$$x : A \vdash \mathit{let}\, y = M\, \mathit{in}\, \mathit{let}\, z = N\, \mathit{in}\, (y, z) : B * C$$

corresponds to the diagram



where the thin arrows indicate the data flow and the thick arrows indicate the control flow.

There is a close connection between isomorphisms of such graphs and program equivalences. Although graph presentations are very interesting, we shall not focus on them in this thesis. However, we shall use some diagrams to prove facts about global state that are otherwise hard to obtain (Chapter 6).

1.5 Overview

A chapter dependency chart is presented in Figure 1.8.

In Chapter 2, we shall introduce precartesian categories, which provide the missing direct semantics described in Section 1.3.3.

In Chapter 3, we shall develop the let-calculus, and establish it as an internal language of precartesian categories.

Chapter 4 contains a simple example of applying precartesian categories: We shall construe well-established models of partiality—that is, concrete and axiomatically given categories of partial maps—as precartesian categories. Accordingly, we shall specialise the let-calculus to the *p-calculus*—a simple calculus which is sound and complete for the models of partiality under consideration. Thus we shall understand the non-termination of programs as a particularly simple computational effect.

In Chapter 5, we shall see how precartesian categories (with extra structure) arise as the Kleisli categories of Moggi's monadic models, in such a way that our direct semantics validates the same program equivalences as Moggi's semantics.

Chapter 6 describes an application of precartesian categories which is more advanced than the study of partiality in Chapter 4: We shall discuss in great detail a way of adding global state to an existing computational system. Mathematically, this corresponds to constructing a new precartesian category from a given one. We shall also describe a language transform that describes this construction. The focus of Chapter 6 is on a mathematical analysis of the new system with global state. This analysis yields insights that seem hard to obtain without using precartesian categories. The main development in Chapter 6 does not involve monads. However, we shall briefly compare our approach with the well-known approach that uses 'side-effects monads', so a small part of Chapter 6 relies on Chapter 5.

In Chapter 7 we shall prove that Moggi's computational lambda-calculus is sound and complete with respect to the precartesian categories with extra structure introduced in Chapter 5. We shall also compare the let-calculus with the computational lambda-calculus and suggest a combination of the two.

Chapter 8 describes another advanced application of precartesian categories: We shall analyse an implementation technique called *continuation-passing-style transform* (CPS transform). Roughly speaking, this transform takes the expressions of a typical programming language to expressions of a simpler language that does not require a call stack[4]. We shall analyse the CPS transform denotationally

[4]This is a very pragmatic and computational description. Continuations have been extensively studied in the literature, with various goals and methods that range from logics to real-life compilers. For example, the compiler of Standard ML of New Jersey used to be based on a CPS transform [App92].

by construing the source language as a precartesian category and using a method similar to the one that we use for global state in Chapter 6. That analysis is based on the pioneering work of Hayo Thielecke [Thi97a].

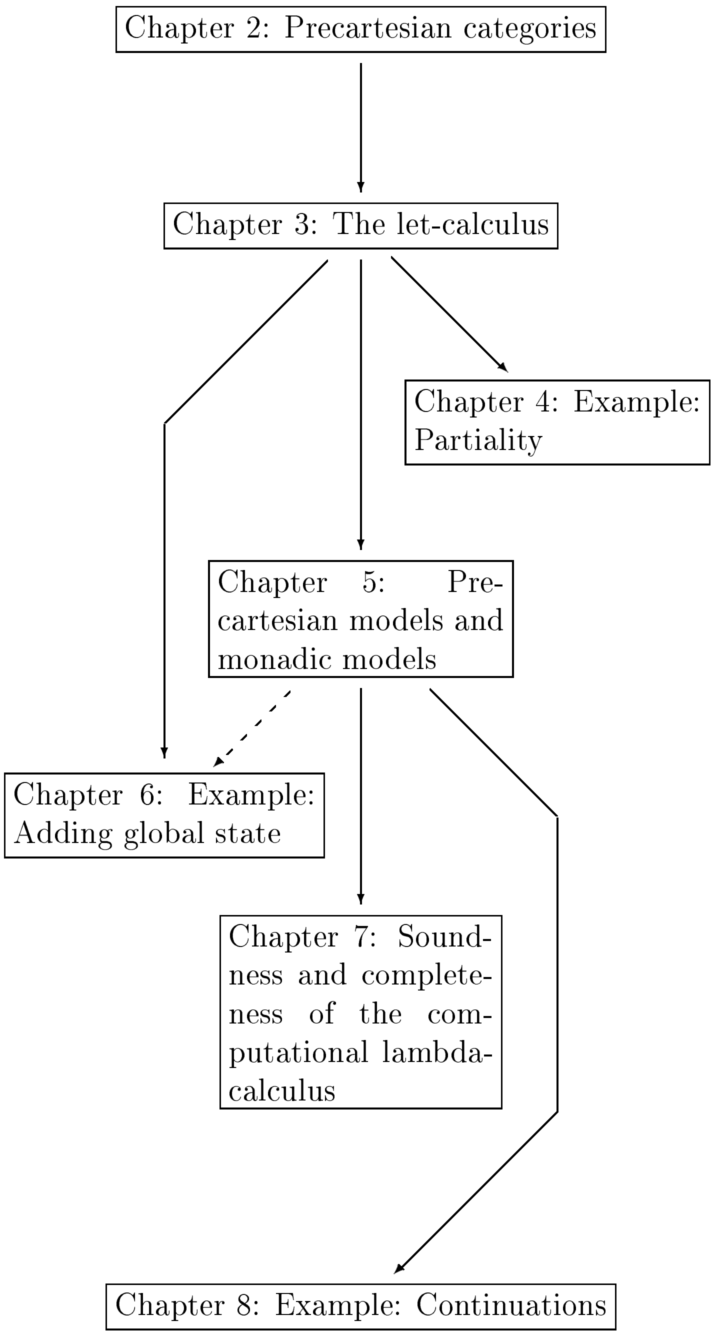


Figure 1.8: Chapter dependency chart

Chapter 2

Precartesian categories

Informally, a precartesian category K is a category together with a tensor $\otimes$ which is a generalised cartesian product, and a tensor unit I which is a generalised terminal object. In particular, K has morphisms $\delta : A \longrightarrow A \otimes A$, $!_A : A \longrightarrow I$, $p : A \otimes B \longrightarrow A$, and $q : A \otimes B \longrightarrow B$ which are generalisations of the evident maps of a category with finite products. The generalisation is such that the tensor need no longer be functorial in both arguments jointly, but only in each argument, and that δ, $!$, p, and q need no longer be natural. This generalisation solves the problems related with the evaluation order and the number of evaluations described in the introduction.

2.1 Preliminary definitions

Although precartesian categories are fundamentally simple, their definition is not. In fact, there are several possible definitions that we shall prove to be equivalent (Proposition 2.1). The definition that we choose uses two auxiliary structures. The first definition is taken from [PR97]:

Definition 2.1. A *binoidal category* is a category K together with

- For each object A, a functor $A \otimes (-) : K \longrightarrow K$
- For each object B, a functor $(-) \otimes B : K \longrightarrow K$

such that for all objects A and B

$$(A \otimes (-))(B) = ((-) \otimes B)(A)$$

For the joint value, we write $A \otimes B$, or short AB.

Definition 2.2. A *pseudocartesian category* is a binoidal category together with an object I and transformations[1] $\delta_A : A \longrightarrow A \otimes A$, $p_{A,B} : A \otimes B \longrightarrow A$, $q_{A,B} : A \otimes B \longrightarrow B$, and $!_A : A : A \longrightarrow I$.

For morphisms $f : A \longrightarrow A'$ and $g : B \longrightarrow B'$ of a pseudocartesian category, let

$$f \otimes g =_{\mathrm{def}} f \otimes B; A' \otimes g \qquad \text{and} \qquad \langle f, g\rangle =_{\mathrm{def}} \delta; f \otimes g$$

We call the transformation $\langle p; p, \langle p; q, q\rangle\rangle : (AB)C \longrightarrow A(BC)$ the *associativity map*, $\langle q, p\rangle : AB \longrightarrow BA$ the *twist map*, and $p : A \otimes I \longrightarrow A$ and $q : I \otimes A \longrightarrow A$ the *neutrality maps*.

Example 2.1. The category *Rel* of sets and relations. The tensor on objects is the cartesian product of sets, I is the one-element set, and (letting A be a set, R a relation, and $*$ the unique element of I)

$$\begin{aligned}
(x, y)(R \otimes A)(x', y') &\Leftrightarrow xRx' \wedge y = y' \\
(x, y)(A \otimes R)(x', y') &\Leftrightarrow x = x' \wedge yRy' \\
x\, \delta\, (y, z) &\Leftrightarrow x = y = z \\
x\, !\, * &\Leftrightarrow \mathit{true} \\
(x, y)\, p\, z &\Leftrightarrow x = z \\
(x, y)\, q\, z &\Leftrightarrow y = z
\end{aligned}$$

Example 2.2. A monoid M, construed as a one-object category. The tensor unit I is the only object, $I \otimes (-)$ and $(-) \otimes I$ are equal to the identity functor, and δ_I, $p_{I,I}$, $q_{I,I}$ and $!_I$ are equal to the neutral element.

2.2 Central, discardable, and copyable morphisms

Recall Diagrams 1.4–1.6 from the introduction. As we have seen, the validity of these diagrams is why categories with finite products may validate too many program equivalences. In a pseudocartesian category, these diagrams need no longer commute. Therefore it makes sense to define the classes of morphisms for which these diagrams *do* commute. These definitions are crucial for the rest of this thesis.

[1] By a transformation from a functor $F : K \longrightarrow K'$ to a functor $G : K \longrightarrow K'$, I mean a map that sends each object A of K to an arrow $FA \longrightarrow GA$

Definition 2.3. A morphism $f : A \longrightarrow A'$ of a binoidal category K is called *central* if for each $g : B \longrightarrow B'$

$$\begin{tikzcd} AB \arrow[r, "fB"] \arrow[d, "Ag"'] & A'B \arrow[d, "A'g"] \\ AB' \arrow[r, "fB'"] & A'B' \end{tikzcd} \qquad \begin{tikzcd} BA \arrow[r, "Bf"] \arrow[d, "gA"'] & BA' \arrow[d, "gA'"] \\ B'A \arrow[r, "B'f"] & B'A' \end{tikzcd}$$

The *centre* ZK of K is defined as the subcategory determined by all objects and the central morphisms.

Example 2.3. In a monoid M construed as a one-object pseudocartesian category, an element $a \in M$ is central if and only if for all $b \in M$ it holds that $ab = ba$. So centrality coincides with the established notion of centrality from algebra.

Example 2.4. In *Rel*, all morphisms are central.

Definition 2.4. A morphism $f : A \longrightarrow B$ in a pseudocartesian category K is called *discardable* if

$$\begin{tikzcd} A \arrow[r, "!"] \arrow[d, "f"'] & I \arrow[d, equal] \\ B \arrow[r, "!"] & I \end{tikzcd}$$

The category $K_!$ is defined as the subcategory of K whose objects are those of K, and whose morphisms are the discardable morphisms of K.

Example 2.5. In a monoid construed as a one-object pseudocartesian category, only the identity morphism is discardable.

Example 2.6. In *Rel*, $R : A \longrightarrow B$ is discardable if and only if R is a total—that is, it relates every $x \in A$ with at least one element of B.

Definition 2.5. A morphism $f : A \longrightarrow B$ of a pseudocartesian category K is called *copyable* if the following two diagrams commute:

$$\begin{tikzcd} A \arrow[r, "\delta"] \arrow[dd, "f"'] & AA \arrow[d, "Af"] \\ & AB \arrow[d, "fB"] \\ B \arrow[r, "\delta"] & BB \end{tikzcd} \qquad \begin{tikzcd} A \arrow[r, "\delta"] \arrow[dd, "f"'] & AA \arrow[d, "fA"] \\ & BA \arrow[d, "Bf"] \\ B \arrow[r, "\delta"] & BB \end{tikzcd}$$

K_δ is defined as the class of copyable morphisms of K.

Example 2.7. In *Rel*, the copyable morphisms are the partial functions.

Example 2.8. In a monoid construed as a one-object pseudocartesian category, the copyable morphisms are the idempotents. In particular, the copyable morphisms do not generally form a category. To see this, let M be the monoid of endofunctions on the set $\{0, 1, 2\}$, and define $f, g \in M$ by

$$\begin{tikzcd}
M \arrow[r, "f"] & M \arrow[r, "g"] & M \\
0 \arrow[r, mapsto] & 0 \arrow[dr, mapsto] & 0 \\
1 \arrow[dr, mapsto] & 1 \arrow[r, mapsto] & 1 \\
2 \arrow[r, mapsto] & 2 \arrow[r, mapsto] & 2
\end{tikzcd}$$

Clearly, f and g are idempotent, but $f; g$ is not.

Definition 2.6. A morphism of a pseudocartesian category is called *focal* if it is central, copyable, and discardable.

Example 2.9. In *Rel*, the focal morphisms are the total functions.

I have the notion of focal morphisms from Peter Selinger, who uses it for models of control [Sel00]. The focal morphisms form a category. To see this, it suffices to check that the composition of focal morphisms is copyable. This is so because for focal morphisms $f : A \longrightarrow B$ and $g : B \longrightarrow C$ we have

$$\begin{aligned}
f; g; \delta &= f; \delta; B \otimes g; g \otimes C \quad \text{(because } g \text{ is copyable)} \\
&= \delta; A \otimes f; f \otimes B; B \otimes g; g \otimes C \quad \text{(because } f \text{ is copyable)} \\
&= \delta; A \otimes f; A \otimes g; f \otimes C; g \otimes C \quad \text{(because } f \text{ and } g \text{ are central)} \\
&= \delta; A \otimes (f; g); (f; g) \otimes C \quad \text{(because } A \otimes (-) \text{ and } (-) \otimes C \text{ are functors)}
\end{aligned}$$

We call the category of focal morphisms the *focus*. With a similar calculation as above we get the following lemma, which we shall need later:

Lemma 2.1. *If in a precartesian category we have n copyable morphisms*

$$A_0 \xrightarrow{f_1} A_1 \xrightarrow{f_2} \cdots \xrightarrow{f_n} A_n$$

such that all or all but one of the f_i are central, then $f_1; \ldots; f_n$ is copyable.

2.3 The main definition

In this section, we shall define precartesian categories and prove an important proposition that provides three alternative definitions.

Definition 2.7. A *precartesian category* is a pseudocartesian category K such that $\otimes$, I, δ, p, q, and $!$ form finite products on the focus, and the associativity map, twist map, and neutrality maps are natural in each argument with respect to *arbitrary* morphisms of K.

This is a very compact definition, so let's spell it out: That $\otimes$, I, δ, p, q, and $!$ form finite products on the focus implies in particular that the focus is closed under $\otimes$, and that δ, p, q, and $!$ are focal. 'Natural in each argument with respect to *arbitrary* morphisms' means the following: For example, letting $\alpha_{A,B,C}$: $(AB)C \longrightarrow A(BC)$ be the associativity map it must hold for *all* morphisms $f : A \longrightarrow A'$, $g : B \longrightarrow B'$, $h : C \longrightarrow C'$ of K (not only for focal morphisms) that

$$\begin{tikzcd}
(AB)C \arrow[r, "\alpha"] \arrow[d, "(fB)C"'] & A(BC) \arrow[d, "f(BC)"] \\
(A'B)C \arrow[r, "\alpha"'] & A'(BC)
\end{tikzcd}$$

$$\begin{tikzcd}
(AB)C \arrow[r, "\alpha"] \arrow[d, "(Ag)C"'] & A(BC) \arrow[d, "A(gC)"] \\
(AB')C \arrow[r, "\alpha"'] & A(B'C)
\end{tikzcd}$$

$$\begin{tikzcd}
(AB)C \arrow[r, "\alpha"] \arrow[d, "(AB)h"'] & A(BC) \arrow[d, "A(Bh)"] \\
(AB)C' \arrow[r, "\alpha"'] & A(BC')
\end{tikzcd}$$

This does not follow from the finite products on the focus, which imply only the naturality of α with respect to focal morphisms. Similar remarks apply to the twist map and the neutrality maps.

Remark 2.1. If f is morphism of a precartesian category, then

$$f; \delta = \delta; idf; fid \qquad \text{if and only if} \qquad f; \delta = \delta; fid; idf$$

(So, to see that f is copyable, it suffices to check one of the two equations.) To see this, let τ be the twist map, and suppose that $f; \delta = \delta; idf; fid$. Then

$$f; \delta = f; \delta; \tau = \delta; idf; fid; \tau = \delta; idf; \tau; idf = \delta; \tau; fid; idf = \delta; fid; idf$$

The converse follows symmetrically. However, there exist precartesian categories where

$$\delta; idf; fid \neq \delta; fid; idf$$

—for example, the Kleisli category of a continuations monad, which we shall study in Chapter 8.

The following proposition provides four equivalent descriptions of precartesian categories. Condition 1 corresponds to our definition of a precartesian category. Condition 2 will be our most-used way of checking that we have a precartesian category. The point of Condition 3 is that it shows (after some contemplation) that all we need to define precartesian categories are equations which are universally quantified over objects and morphisms.

Proposition 2.1. *Let K be a pseudocartesian category such that the associativity map, twist map, and neutrality maps are natural in each argument with respect to arbitrary morphisms of K. Then the following are equivalent:*

1. *$\otimes$, I, δ, p, q, and $!$ form finite products on the focus.*

2. *The central maps, the discardable maps, and the copyable maps, respectively, are closed under $\otimes$, all components of δ, p, q, and $!$ are focal, and*

$$\begin{aligned} \delta; p \otimes q &= id \\ \delta; p &= id \\ \delta; q &= id \\ !_I &= id_I \\ p_{A,B} &= A \otimes !; p_{A,I} \\ q_{A,B} &= ! \otimes B; q_{I,B} \end{aligned}$$

3. *All components of δ, p, q, and $!$ are focal, the equations from Condition 2 hold, the twist map is self-inverse, and the associativity maps $(AB)C \longrightarrow A(BC)$ and $A(BC) \longrightarrow (AB)C$ are copyable and inverse to each other.*

4. $\otimes$*,* I*,* δ*,* p*,* q*, and* $!$ *form finite products on some subcategory of the centre.*

For the proof, which is quite technical, see Appendix C.

Lemma 2.2. *In any precartesian category, the projections are natural in the non-discarded argument—that is for all morphisms* f *(not only focal* f*) it holds that* $f \otimes id; p = p; f$ *and* $id \otimes f; q = q; f$*.*

Proof. For any object B and morphism $f : A \longrightarrow A'$, we have $fB; p_{A',B} = fB; A'!; p_{A',I} = A!; fI; p_{A',I} = A!; p_{A,I}; f = p_{A,B}; f$ $\square$

2.4 A revised semantics of the let-language

Next we shall adapt the semantics of the let-language (i.e. the first-order fragment of the computational lambda-calculus), which we presented in Figure 1.5, to precartesian categories. Let Γ^n stand for $\Gamma \otimes \cdots \otimes \Gamma$, where Γ occurs n times, and let Δ stand for the evident n-fold diagonal $\Gamma \longrightarrow \Gamma^n$. (It is harmless to omit brackets in the n-ary tensor product, because the associativity map is a focal, and in the focus, which is a category with finite products, the associativity and neutrality maps satisfy the usual coherence laws known from monoidal categories (see [Lan71]).) The revised semantics is presented in Figure 2.1. In the case where the precartesian category is a category with finite products, this semantics agrees with the one in Figure 1.5. The only change consists in clarifying the rules for pairs and constants with respect to the evaluation order, which is chosen to be from left to right.

2.5 Strong precartesian functors

When two languages are modelled by precartesian categories, we may want to compare the models. Obviously, this requires some notion of morphism between precartesian categories. For this purpose, we shall define *strong precartesian functors*. The hurried reader may want to skip this section and return to it on a call-by-need basis.

Naively, we could define a morphism $K \longrightarrow K'$ between precartesian categories as a functor that preserves all precartesian structure on the nose (i.e. the tensor an tensor unit, the diagonal, discard map, and the projections). However, 'on the nose' can be too strict, and we have to use 'up to isomorphism' instead. This leads to the definition of a *strong precartesian functor*:

Rule	Syntax	Semantics
var	$x_1 : A_1, \ldots, x_n : A_n \vdash x_i : A_i$	$= A_1 \cdots A_n \xrightarrow{\pi_i} A_i$
let	$\dfrac{\Gamma \vdash M : A \qquad \Gamma, x : A \vdash N : B}{\Gamma \vdash \mathit{let}\, x = M\ \mathit{in}\ N : B}$	$= \Gamma \xrightarrow{f} A$ $= \Gamma A \xrightarrow{g} B$ $= \Gamma \xrightarrow{\delta} \Gamma\Gamma \xrightarrow{\Gamma f} \Gamma A \xrightarrow{g} B$
$()$	$\Gamma \vdash () : \mathit{unit}$	$= \Gamma \xrightarrow{!} I$
$(-,-)$	$\dfrac{\Gamma \vdash M : A \qquad \Gamma \vdash N : B}{\Gamma \vdash (M, N) : A * B}$	$= \Gamma \xrightarrow{f} A$ $= \Gamma \xrightarrow{g} B$ $= \Gamma \xrightarrow{\delta} \Gamma\Gamma \xrightarrow{f\Gamma} A\Gamma \xrightarrow{Ag} AB$
π_i	$\dfrac{\Gamma \vdash M : A_1 * A_2}{\Gamma \vdash \pi_i(M) : A_i}$	$= \Gamma \xrightarrow{f} A_1 A_2$ $= \Gamma \xrightarrow{f} A_1 A_2 \xrightarrow{\pi_i} A_i$
$f : A_1, \ldots, A_n \longrightarrow B$	$\dfrac{\Gamma \vdash M_i : A}{\Gamma \vdash f(M_1, \ldots, M_n) : B}$	$= \Gamma \xrightarrow{g_i} A_i$ $= \Gamma \xrightarrow{\Delta} \Gamma^n \xrightarrow{g_1 \Gamma^{n-1}} A_1 \Gamma^{n-1} \xrightarrow{A_1 g_2 \Gamma^{n-2}} \cdots \xrightarrow{A_1 \cdots A_{n-1} g_n} A_1 \cdots A_n \xrightarrow{f} B$

Figure 2.1: Semantics of the let-language in a precartesian category

$$\begin{tikzcd} FA \arrow[r, phantom, "="] \arrow[d, "\delta'"'] & FA \arrow[d, "F\delta"] \\ (FA) \otimes (FA) \arrow[r, "F_2"] & F(A \otimes A) \end{tikzcd} \tag{2.1}$$

$$\begin{tikzcd} FA \arrow[r, phantom, "="] \arrow[d, "!'"'] & FA \arrow[d, "F!"] \\ I' \arrow[r, "F_0"] & FI \end{tikzcd} \tag{2.2}$$

$$\begin{tikzcd} FA \otimes FB \arrow[r, "p'"] \arrow[d, "q'"'] \arrow[dr, "F_2"] & FA \\ FB & F(AB) \arrow[l, "Fq"'] \arrow[u, "Fp"'] \end{tikzcd} \tag{2.3}$$

Figure 2.2: Conditions stating that precartesian functors preserve structure up to F_2 and F_0

Definition 2.8. A *strong precartesian functor* $F : K \longrightarrow K'$ between precartesian categories consists of a functor $F : K \longrightarrow K'$, a natural isomorphism

$$F_2(A, B) : (FA) \otimes (FB) \cong F(A \otimes B)$$

(natural separately in A and B), and an isomorphism

$$F_0 : I' \cong FI$$

such that for all objects A, B, and C of K, the diagrams in Figures 2.2 and 2.3 commute. A strong precartesian functor is called *strict* if F_0 and F_2 are identities.

For strong precartesian functors $F : K \longrightarrow K'$ and $G : K' \longrightarrow K''$, let

$$(GF)_2(A, B) =_{\text{def}} \left(GFA \otimes GFB \xrightarrow{G_2} G(FA \otimes FB) \xrightarrow{GF_2} GF(A \otimes B)\right) \tag{2.8}$$

$$(GF)_0 =_{\text{def}} \left(I'' \xrightarrow{G_0} GI' \xrightarrow{GF_0} GFI\right) \tag{2.9}$$

With this composition, strong precartesian functors and precartesian categories form a category.

Remark 2.2. As we shall see in Remarks 2.4 and 2.5, there are potential improvements to our definition of a strong precartesian functor (thanks to John Power for

$$
\begin{tikzcd}
(FA \otimes FB) \otimes FC \arrow[r, "\alpha'"] \arrow[d, "F_2 \otimes FC"'] & FA \otimes (FB \otimes FC) \arrow[d, "FA \otimes F_2"] \\
F(A \otimes B) \otimes FC \arrow[d, "F_2"'] & FA \otimes F(B \otimes C) \arrow[d, "F_2"] \\
F((A \otimes B) \otimes C) \arrow[r, "\alpha"] & F(A \otimes (B \otimes C))
\end{tikzcd}
\tag{2.4}
$$

$$
\begin{tikzcd}
FB \otimes I \arrow[r, "p'"] \arrow[d, "FB \otimes F_0"'] & FB \\
FB \otimes FI \arrow[r, "F_2"] & F(B \otimes I) \arrow[u, "Fp"']
\end{tikzcd}
\qquad
\begin{tikzcd}
I \otimes FB \arrow[r, "q'"] \arrow[d, "F_0 \otimes FB"'] & FB \\
FI \otimes FB \arrow[r, "F_2"] & F(I \otimes B) \arrow[u, "Fq"']
\end{tikzcd}
\tag{2.7}
$$

Figure 2.3: Coherence conditions for strong precartesian functors

drawing my attention to this). However, all these improvements lead to special cases of Definition 2.8, so we are playing safe in that everything that we prove about the strong precartesian functors in the sense of Definition 2.8 certainly holds for functors in the sense of the improved definitions.

The precartesian properties (i.e. centrality, copyability, and discardability) will be our major interest. So it is important that strong precartesian functors 'behave well' with respect to these properties. In particular, strong precartesian functors should preserve and reflect these properties to the same extent as strict precartesian functors. Remarkably, this is so although F_2 and F_0 may be not focal:

Proposition 2.2. *Let $F : K \longrightarrow K'$ be a strong precartesian functor. Then*

- *F preserves copyable morphisms and discardable morphisms.*
- *If F is full and for each object C of K' there is a central iso $C \cong FB$ for some object B of K, then F preserves central morphisms.*
- *If F is faithful, then it reflects central morphisms, copyable morphisms, and discardable morphisms.*

Proof. First, let F be full, and for each object C of K' let there be a central iso $C \cong FB$ for some object B of K. To see that F preserves central morphisms, let

$f \in K(A, A')$ be central, let $g \in K'(B, B')$, and consider

$$\begin{tikzcd}
FA \otimes FB \arrow[rrr, "Ff \otimes FB"] \arrow[dr, "F_2"] \arrow[ddd, "FA \otimes Fg"'] & & & FA' \otimes FB \arrow[dl, "F_2"'] \arrow[ddd, "FA' \otimes Fg"] \\
& F(A \otimes B) \arrow[r, "F(f \otimes B)"] \arrow[d, "F(A \otimes g)"'] & F(A' \otimes B) \arrow[d, "F(A' \otimes g)"] & \\
& F(A \otimes B') \arrow[r, "F(f \otimes B')"] & F(A' \otimes B') & \\
FA \otimes FB' \arrow[rrr, "Ff \otimes FB'"] \arrow[ur, "F_2"] & & & FA' \otimes FB' \arrow[ul, "F_2"']
\end{tikzcd}$$

The inner square commutes because f is central in K. The outer square commutes because the iso F_2 is natural. Now let $h \in K'(C, C')$, let B, B' be such that there are central isos $j : C \cong FB$ and $j' : C' \cong FB'$, and let $Fg = j^{-1}; h; j'$. Consider

$$\begin{tikzcd}
FA \otimes C \arrow[rrr, "Ff \otimes C"] \arrow[dr, "FA \oplus j"] \arrow[ddd, "FA \otimes h"'] & & & FA' \otimes C \arrow[dl, "FA' \oplus j"'] \arrow[ddd, "FA' \otimes h"] \\
& FA \otimes FB \arrow[r, "Ff \otimes FB"] \arrow[d, "FA \otimes Fg"'] & FA' \otimes FB \arrow[d, "FA' \otimes Fg"] & \\
& FA \otimes FB' \arrow[r, "Ff \otimes FB'"] & FA' \otimes FB' & \\
FA \otimes C' \arrow[rrr, "Ff \otimes C'"'] \arrow[ur, "FA \oplus j'"] & & & FA' \otimes C' \arrow[ul, "FA' \oplus j'"']
\end{tikzcd}$$

The left and right squares commute because $FA \otimes (-)$ and $FA' \otimes (-)$ are functors. The top and bottom squares commute because j and j' are central. Therefore the whole diagram commutes, so Ff is central.

Proving that a strong precartesian functor F preserves copyable and discardable morphisms is easy. Now suppose that F is faithful. Let $f \in K(A, A')$ such that Ff is central in K'. To see that f is central in K, let $g \in K(B, B')$ and consider

$$\begin{tikzcd}
F(A \otimes B) \arrow[rrr, "F(f \otimes B)"] \arrow[ddd, "F(A \otimes g)"'] & & & F(A' \otimes B) \arrow[ddd, "F(A' \otimes g)"] \\
& FA \otimes FB \arrow[ul, "F_2"'] \arrow[r, "Ff \otimes FB"] \arrow[d, "FA \otimes Fg"'] & FA' \otimes FB \arrow[ur, "F_2"] \arrow[d, "FA' \otimes Fg"] & \\
& FA \otimes FB' \arrow[dl, "F_2"] \arrow[r, "Ff \otimes FB'"] & FA' \otimes FB' \arrow[dr, "F_2"'] & \\
F(A \otimes B') \arrow[rrr, "F(f \otimes B')"] & & & F(A' \otimes B')
\end{tikzcd}$$

The diagram commutes because Ff is central and F_2 is natural. Because F is faithful, we can cancel F from the outer square. The resulting square commutes for all g, which is saying that f is central. Proving that F reflects copyable morphisms and discardable morphisms is straightforward. $\square$

Remark 2.3. There are examples of strict precartesian functors that are relevant for computer science and do not preserve central morphisms (see Proposition 6.11).

Remark 2.4. It is natural to ask whether we should require F_2 and F_0 to be focal. One can prove that if they are focal, the conditions in Figure 2.2 imply the coherence conditions in Figure 2.3, and not having to check those coherence conditions would certainly be nice. Alas, the focality of F_2 can cause a problem with Definition 2.8: If G does not preserve central morphisms (and there are realistic G that don't), the components of $(GF)_2$ may be not central. To solve this problem, we could follow the following recipe:

- Instead of precartesian categories, consider pairs (K, S) where K is a precartesian category and S is a subcategory of the focus of K that has all objects of K.

- Change Definition 2.8 in such a way that a strong precartesian functor $F : (K, S) \longrightarrow (K', S')$ must send morphisms of S to S', and all components of F_2 and F_0 must be in S'.

(The idea of introducing a subcategory S like above is essentially due to John Power, who equipped premonoidal categories with a subcategory of the centre [Pow99b].) This solves the problem with Definition 2.8. For any K, there is a greatest S—the focus—and a least S—the discrete category determined by all objects of K. It is a future challenge to find criteria for choosing a suitable S in the case where K models a programming language. A good choice of S should be a category of (syntactic) 'values' (e.g. freely generated by constants, pairing, the lambda-operator, and so on).

Corollary 2.3. *Strong precartesian functors that are isomorphisms of categories preserve and reflect central morphisms, copyable morphisms, and discardable morphisms.*

Remark 2.5. We cannot simply replace 'isomorphism of categories' by 'equivalence of categories' in Corollary 2.3. To see this, let $F : K \longrightarrow K'$ and $G : K' \longrightarrow K$ be precartesian functors that form an equivalence of categories. To conclude that F preserves central morphisms, we would need the extra condition that every component of the natural isomorphism $Id_{K'} \cong FG$ is central. However, there is a way out of this dilemma if we follow the recipe in Remark 2.4: In that case, we could require natural transformations $F \longrightarrow G : (K, S) \longrightarrow (K', S')$ to have components in S'. With such natural transformations as 2-cells, we would get a 2-category with 0-cells (K, S) such that equivalences $(K, S) \simeq (K', S')$ in that 2-category preserve centrality. However, we shall not use this 2-categorical approach in this thesis.

2.6 About the origin of precartesian categories

This section is a brief review of the literature that inspired my definition of precartesian categories. In a technical sense, the remainder of this thesis does not depend on the contents of this section.

As explained in Section 1.1.2, it matters in which order the arguments of a procedure are evaluated. Earlier in this chapter, we tackled this problem by removing the condition that the tensor of a categorical model has to be functorial in both arguments jointly. To my knowledge, this generalisation was first made when John Power and Stuart Anderson introduced *premonoidal categories* in the early 90's, for modelling finite non-determinism [AP97]. Roughly speaking, symmetric premonoidal categories are generalised symmetric monoidal categories in that the tensor need not be a functorial in both arguments jointly, but only in each argument. Here is a precise definition, taken from [PR97]:

Definition 2.9. A *symmetric premonoidal category* is

- A binoidal category C

- An object I of C
- Four natural isomorphisms $A(BC) \cong (AB)C$, $IA \cong A$, $AI \cong A$, and $AB \cong BA$, with central components that satisfy the coherence conditions known from symmetric monoidal categories.

Symmetric monoidal categories are exactly those symmetric premonoidal categories that have only central morphisms. Obviously, every precartesian category forms a symmetric premonoidal category, where the structural isomorphisms are those of the focus construed as a category with finite products.

Along with symmetric premonoidal categories came symmetric premonoidal functors:

Definition 2.10. A *strict symmetric premonoidal functor* F between symmetric premonoidal categories K and K' is a functor that preserves the tensor, the tensor unit, and the structural isomorphisms on the nose, and sends central morphisms to central morphisms.

Obviously, every strict precartesian functor that preserves central morphisms is a strict symmetric premonoidal functor. (As explained in Section 2.5, to cover certain realistic examples in computer science, we had to remove the condition that functors preserve central morphisms.)

In the Power's framework, copying and discarding come into play in the guise of *Freyd categories* [PT99]:

Definition 2.11. A *Freyd category* consists of a category C with finite products, a symmetric premonoidal category K with the same objects as C, and an identity-on-objects strict symmetric premonoidal functor $F : C \longrightarrow K$.

Obviously, every precartesian category K forms a Freyd category, where C is the focus of K, and F is the inclusion functor. There is also an opposite construction:

Proposition 2.4. *Let $F : C \longrightarrow K$ be a Freyd category, where δ is the diagonal of C, p and q are the projections of C, and $!$ is the discard map of C. Then $(K, F\delta, Fp, Fq, F!)$ is a precartesian category. Moreover, all morphisms in the image of F are focal.*

Typically, when a Freyd category $F : C \longrightarrow K$ is used for modelling a programming language, K is a category for modelling all expressions, C is a category for modelling syntactically-defined 'values' only, and F is faithful. While all morphisms in the image of F are focal, there may be more focal morphisms— that is, there may be focal morphisms that do not denote syntactic values.

Chapter 3

The let-calculus

In this chapter we shall develop the *let-calculus*—a calculus for reasoning about program equivalences. Its design is inspired by the notions *central*, *copyable*, and *discardable*, which are associated with precartesian categories. Its judgements have form

$$\begin{array}{ll} \Gamma \vdash M \equiv N : A & \\ \Gamma \vdash M \,!\, E & \text{where } E \in \{\mathit{central}, \mathit{copyable}, \mathit{discardable}, \mathit{focal}, \ldots\} \\ \Gamma \vdash M \,/\, e\, x & \text{where } e \in \{\mathit{linear}, \mathit{relevant}, \mathit{affine}, \mathit{arbitrary}, \ldots\} \end{array}$$

where $(\Gamma \vdash M : A)$ and $(\Gamma \vdash N : A)$ are sequents of the let-language. (Filling in the dots is part of this section.)

3.1 Exploiting categorical closure properties

The key observation that starts our development is stated by the following proposition:

Proposition 3.1. *In a precartesian category,*

- *The central morphisms are closed under all operations (i.e. they form a subcategory which is closed under tensor and contains all components of δ, p, q, and $!$).*
- *The discardable morphisms are closed under all operations.*
- *The copyable morphisms are closed under all operations except the composition of the category.*

Proof. It is obvious that the central morphisms and the discardable morphisms, respectively, form a subcategory. By the definition of a precartesian category, δ,

p, q, and ! are focal and therefore central, copyable, and discardable. By Condition 2 of Proposition 2.1, the central morphisms, the copyable morphisms, and the discardable morphisms, respectively, are closed under tensor. Clearly, identities are copyable. That the copyable morphisms are not closed under composition we know from Example 2.8. $\square$

The key observation now is the following: Because in every precartesian category the central, copyable, and discardable morphisms, respectively, are closed under all operations (almost all in the case of copyable morphisms), the sequents $(\Gamma \vdash M : A)$ for which $(\Gamma \vdash M \,!\, E)$ holds are closed under all (almost all) sequent formation rules.

Example 3.1. Suppose that we have

$$(\Gamma \vdash M \,!\, \mathit{central}) \qquad \text{and} \qquad (\Gamma \vdash N \,!\, \mathit{central})$$

and let central morphisms $f : \Gamma \longrightarrow A$ and $g : \Gamma \longrightarrow B$ be the denotations of $(\Gamma \vdash M : A)$ and $(\Gamma \vdash N : B)$, respectively. The denotation of $(\Gamma \vdash (M, N) : A * B)$ is $\delta; f\,id; id\,g$. This is central because the centre is closed under all operations, and therefore we should be able to derive

$$\Gamma \vdash (M, N) \,!\, \mathit{central}$$

The only catch is that the copyable morphisms are not closed under composition, but we shall find a way to deal with that problem.

3.2 The precartesian cube

It is helpful to arrange the properties *central*, *copyable*, *discardable*, and all their intersections, in a three-dimensional boolean lattice as in Figure 3.1—let's call it the *precartesian cube*. Let $\vee$ and $\wedge$ stand for the least upper bound and the greatest lower bound, respectively, in the precartesian cube. With these operators we can refine Example 3.1:

Example 3.2. Suppose that in a precartesian category K it holds that

$$(\Gamma \vdash M \,!\, \mathit{central} \wedge \mathit{discardable}) \qquad \text{and} \qquad (\Gamma \vdash N \,!\, \mathit{central} \wedge \mathit{copyable})$$

Because the morphisms denoted by $(\Gamma \vdash M : A)$ and $(\Gamma \vdash N : B)$ are central, by Proposition 3.1 the morphism denoted by $(\Gamma \vdash (M, N) : A * B)$ is central

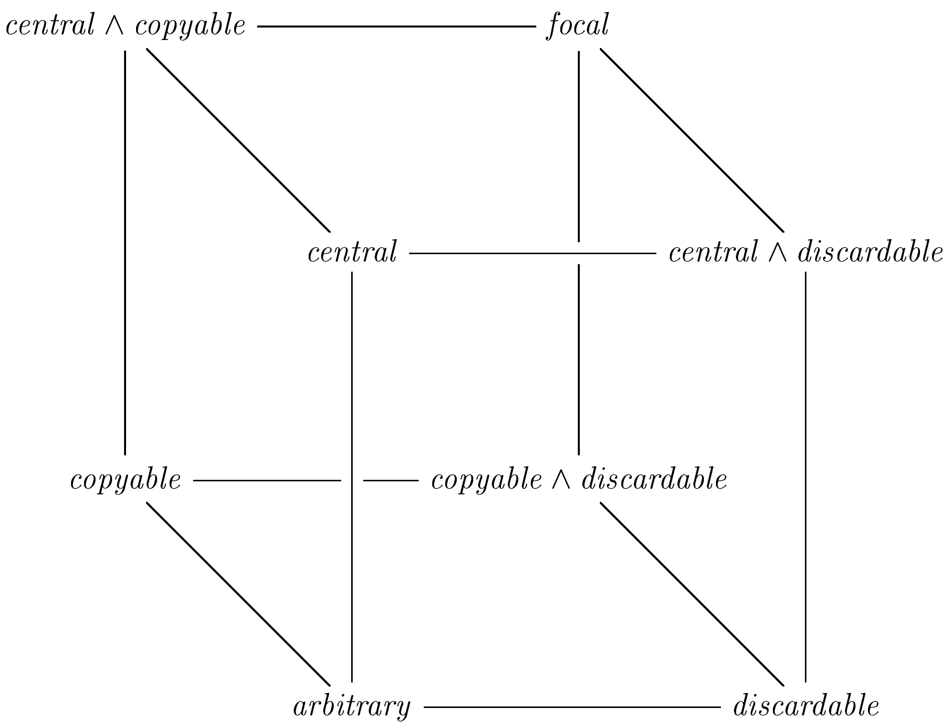


Figure 3.1: The precartesian cube

too. Therefore, it holds in K that $(\Gamma \vdash (M, N) \,!\, \mathit{central})$. Observe that in the precartesian cube, we have

$$(\mathit{central} \wedge \mathit{discardable}) \vee (\mathit{central} \wedge \mathit{copyable}) = \mathit{central}$$

More generally, this suggests that the following inference rule sound:

$$\frac{\Gamma \vdash M \,!\, E \qquad \Gamma \vdash N \,!\, F}{\Gamma \vdash (M, N) \,!\, E \vee F}$$

Rules like this should apply to all kinds of let-language expressions. This leads to the inference system in Figure 3.2. The use of max instead of $\vee$ is necessary only because the copyable morphisms are not generally closed under composition. With max, the Rules in Figure 3.2 are sound because, by Lemma 2.1, for $n+1$ copyable morphisms $A_0 \xrightarrow{f_0} A_1 \xrightarrow{f_1} \cdots \xrightarrow{f_n} A_n$ such that at least n of the f_i are central, the morphism $f_0; \ldots; f_n$ is copyable.

$$\max\{E_1,\ldots,E_n\} = \begin{cases} E_1 \vee \ldots \vee E_n \vee \overline{\mathit{copyable}} & \text{if } E_i \not\leq \mathit{central} \text{ for more than one } i \\ E_1 \vee \ldots \vee E_n & \text{otherwise} \end{cases}$$

$$x_1 : A_1, \ldots, x_n : A_n \vdash x_i \,!\, \mathit{focal} \qquad \frac{\Gamma \vdash M \,!\, E \qquad \Gamma, x : A \vdash N \,!\, F}{\Gamma \vdash \mathit{let}\, x = M \,\mathit{in}\, N \,!\, \max\{E, F\}}$$

$$\Gamma \vdash () \,!\, \mathit{focal} \qquad \frac{\Gamma \vdash M \,!\, E \qquad \Gamma \vdash N \,!\, F}{\Gamma \vdash (M, N) \,!\, \max\{E, F\}} \qquad \frac{\Gamma \vdash M \,!\, E}{\Gamma \vdash \pi_i(M) \,!\, E}$$

$$\frac{\begin{array}{c}\Gamma \vdash M_1 \,!\, E_1 \quad \ldots \quad \Gamma \vdash M_n \,!\, E_n \\ y_1 : A_1, \ldots, y_n : A_n \vdash f(y_1, \ldots, y_n) \,!\, E\end{array}}{\Gamma \vdash f(M_1, \ldots, M_n) \,!\, \max\{E, E_1, \ldots, E_n\}} \quad \begin{array}{l}\text{for every constant} \\ f : A_1, \ldots, A_n \longrightarrow B\end{array}$$

$$\frac{\Gamma \vdash M \,!\, E}{\Gamma \vdash M \,!\, E'} \quad \text{If } E \leq E'$$

Figure 3.2: Rules for the precartesian cube

3.3 Exploiting the precartesian cube

3.3.1 Number of evaluations

What do we gain when we derive $(\Gamma \vdash M \,!\, E)$? For example, suppose that $(\Gamma \vdash M \,!\, \mathit{copyable})$ holds in a precartesian category. Then it also holds that

$$\Gamma \vdash \mathit{let}\, x = M\ \mathit{in}\, (x, x) \equiv (M, M) : A * A$$

because that equation just states that the denotation of $(\Gamma \vdash M : A)$ is copyable. It is reasonable to guess that the equation

$$\Gamma \vdash (\mathit{let}\, x = M\ \mathit{in}\, N) \equiv N[x := M] : B \qquad (\text{let}.\beta)$$

holds for every sequent $(\Gamma, x : A \vdash N : B)$ where N has one or more free occurrences of x. However, this is not true in every precartesian category, because if the denotation of $(\Gamma \vdash M : A)$ is not central, then it is unsound to swap M with subexpressions of N with respect to the evaluation order. For example, the equation

$$\Gamma \vdash (\mathit{let}\, x = M\ \mathit{in}\ \mathit{let}\, y = P\ \mathit{in}\, x) \equiv (\mathit{let}\, y = P\ \mathit{in}\, M) : B$$

may be false if the denotation of P is not central. However, Equation let.β is true for N that have at least one free occurrence of x if the denotation of $(\Gamma \vdash M : A)$ is copyable *and central*. (The proof, which is easy, is part of the soundness proof in Appendix B.) Similarly, Equation let.β is true if the denotation of $(\Gamma \vdash M : A)$ is discardable and central and N has at most one free occurrence of x, and also if denotation of $(\Gamma \vdash M : A)$ is central and N has exactly one free occurrence of x. The rules in Figure 3.3 help phrasing these results systematically.

Remark 3.1. Why do we internalise properties e like *relevant* and *affine* in the calculus, although they are simple and do not depend on the other two kinds of judgements? The answer is that when we deal with evaluation order in the next section, we shall have to introduce a property e such that the judgements of the form $(\Gamma \vdash M \,/\, e\,)$ and the other two kinds of judgements depend on each other.

Now let

$$\begin{aligned}
\Phi(\mathit{central} \wedge \mathit{copyable}) &= \mathit{relevant} \\
\Phi(\mathit{central} \wedge \mathit{discardable}) &= \mathit{affine} \\
\Phi(\mathit{central}) &= \mathit{linear} \\
\Phi(\mathit{focal}) &= \mathit{arbitrary}
\end{aligned}$$

$$\frac{\Gamma \vdash N : A}{\Gamma \vdash N \,/\, \mathit{relevant}\; x} \text{ if } N \text{ has at most one free occurrence of } x$$

$$\frac{\Gamma \vdash N : A}{\Gamma \vdash N \,/\, \mathit{affine}\; x} \text{ if } N \text{ has at least one free occurrence of } x$$

$$\Gamma \vdash N \,/\, \mathit{arbitrary}\; x$$

$$\frac{\Gamma \vdash M \,/\, e\, x \qquad \Gamma \vdash M \,/\, e'\, x}{\Gamma \vdash M \,/\, (e \wedge e')\, x}$$

$$\frac{\Gamma \vdash M \,/\, e\, x}{\Gamma \vdash M \,/\, e'\, x} \quad \text{if } e \leq e'$$

$$(\mathit{linear} =_{\mathrm{def}} \mathit{relevant} \wedge \mathit{affine})$$

Figure 3.3: Rules for the number of free occurrences of variables

Our results mean that for all sequents $(\Gamma \vdash M : A)$ and $(\Gamma, x : A \vdash N : B)$ of the let-calculus, and for each of the above four arguments E of Φ, the following rule holds in every precartesian category:

$$\frac{\Gamma \vdash M \,!\, E \qquad \Gamma, x : A \vdash N \,/\, \Phi(E)\, x}{\Gamma \vdash (\mathit{let}\; x = M \;\mathit{in}\; N) \equiv N[x := M] : B} \tag{3.1}$$

3.3.2 Evaluation order

Now consider Figure 3.4. Let's call the properties in the bottom cube the *expression properties.* To each of the precartesian properties that are lesser or equal then *central* (i.e. to the properties of the top face of the precartesian cube) Φ assigns a suitable expression property from the set $\{\mathit{relevant}, \mathit{affine}, \mathit{central}, \mathit{arbitrary}\}$ such that Rule 3.1 is sound. It is now an obvious step to look for four expression properties that correspond to the bottom face of the precartesian cube. Letting

$$X = \Phi(\mathit{copyable} \wedge \mathit{discardable})$$

we should be able to fill in the three remaining corners of the expression-property cube. So what is X?

$$
\begin{tikzcd}
\mathit{central} \wedge \mathit{copyable} \arrow[rr, dash] \arrow[dr, dash] \arrow[dd, dash] & & \mathit{focal} \arrow[dr, dash] \arrow[dd, dash] & \\
& \mathit{central} \arrow[rr, dash] \arrow[dd, dash] & & \mathit{central} \wedge \mathit{discardable} \arrow[dd, dash] \\
\mathit{copyable} \arrow[rr, dash] \arrow[dr, dash] & & \mathit{copyable} \wedge \mathit{discardable} \arrow[dr, dash] & \\
& \mathit{arbitrary} \arrow[rr, dash] & & \mathit{discardable}
\end{tikzcd}
$$

$\downarrow \Phi$ (order-reversing lattice iso)

$$
\begin{tikzcd}
\mathit{relevant} \arrow[rr, dash] \arrow[dr, dash] \arrow[dd, dash] & & \mathit{arbitrary} \arrow[dr, dash] \arrow[dd, dash] & \\
& \mathit{linear} \arrow[rr, dash] \arrow[dd, dash] & & \mathit{affine} \arrow[dd, dash] \\
X \wedge \mathit{relevant} \arrow[rr, dash] \arrow[dr, dash] & & X \arrow[dr, dash] & \\
& X \wedge \mathit{linear} \arrow[rr, dash] & & X \wedge \mathit{affine}
\end{tikzcd}
$$

Figure 3.4: The missing generator X

Clearly, X should have to do with the evaluation order. Suppose that we have $(\Gamma \vdash M\,!\,\mathit{copyable} \wedge \mathit{discardable})$. Now for example, let

$$N = (\mathit{let}\, y = (P, f_1(x))\ \mathit{in}\ f_2(Q, x))$$

where x occurs free neither in P nor in Q. If Equation let.β is to be true, then the substitution must get M past P to substitute M for the first occurrence of x. Moreover, the substitution must get M past $(P, f_1(x))$ and Q to substitute M for the second occurrence of x. We do not know whether M is central, but the substitution is certainly valid if P, $(P, f_1(x))$, and Q are central. The idea is now to determine for each variable x and expression N the set $b_x(N)$ of subexpressions of N that might obstruct the substitution. In our example, we have

$$b_x(N) = \{P, (P, f_1(x)), Q\}$$

The letter b stands for 'before', because $b_x(N)$ determines all problematic subexpressions of N that occur before free occurrences of x. ('Before' is to be understood with respect to the evaluation order, which is from left to right in our denotational semantics. A different evaluation order would imply a different definition of b.)

Now we ensure that Equation let.β is true by requiring that all elements of $b_x(N)$ be central. To get things completely right, we must define b_x on sequents instead of expressions, because the precartesian properties cannot be made properties of expressions[1]. However, because the uniqueness of the type of an expression M in an environment Γ, we can allow ourselves to omit the types. This leads to the definition of $b_x(N)$ as in Figure 3.5. Note that forming $b_x(N)$ is quite intuitive for a human.

Now we can define the missing expression property X, which we shall call *clear*:

$$\frac{\Gamma \vdash M\,!\,\mathit{central} \quad \forall (\Gamma \vdash M) \in b_x(\Delta \vdash N)}{\Delta \vdash N \,/\, \mathit{clear}\, x} \tag{3.2}$$

This completes the picture. For the record, the cube of expression properties is presented in Figure 3.6.

3.4 Summary of the let-calculus

We shall now summarize the rules of the let-calculus. Some unsurprising rules remain to be introduced—they are presented in Figure 3.7 (omitting environments

[1]For example, for every $(\Gamma \vdash M : A)$ in *Rel* it holds that $(x : 0, y : A \vdash M\,!\,\mathit{discardable})$ if 0 denotes the empty set, which is the initial object. But it can obviously be false that $(y : A \vdash M\,!\,\mathit{discardable})$, for example if $(x : A \vdash M : B)$ denotes the empty relation.

$$
\begin{aligned}
b_x(\Gamma \vdash y) &= \emptyset \\
b_x(\Gamma \vdash \mathit{let}\; y = M \;\mathit{in}\; N) &= \begin{cases} \emptyset & \text{if } x = y \\ b_x(\Gamma \vdash M) & \text{if } x \notin FV(N) \cup \{y\} \\ b_x(\Gamma \vdash M) \cup \{\Gamma \vdash M\} & \\ \cup b_x(\Gamma, y \vdash N) & \text{if } x \in FV(N) - \{y\} \end{cases} \\
b_x(\Gamma \vdash ()) &= \emptyset \\
b_x(\Gamma \vdash (M, N)) &= \begin{cases} b_x(\Gamma \vdash M) & \text{if } x \notin FV(N) \\ b_x(\Gamma \vdash M) \cup \{\Gamma \vdash M\} \cup b_x(\Gamma \vdash N) & \text{if } x \in FV(N) \end{cases} \\
b_x(\Gamma \vdash \pi_i(M)) &= b_x(\Gamma \vdash M) \\
b_x(\Gamma \vdash f(M_1, \ldots, M_n)) &= \begin{cases} (b_x(\Gamma \vdash M_1) \cup \{\Gamma \vdash M_1\}) \cup \ldots \cup (b_x(\Gamma \vdash M_{i-1}) \\ \cup \{\Gamma \vdash M_{i-1}\}) \cup b_x(\Gamma \vdash M_i) \text{ where } M_i \text{ is the right-} \\ \text{most of the } M_j \text{ such that } x \in FV(M_j) \end{cases}
\end{aligned}
$$

Figure 3.5: Definition of $b_x(N)$

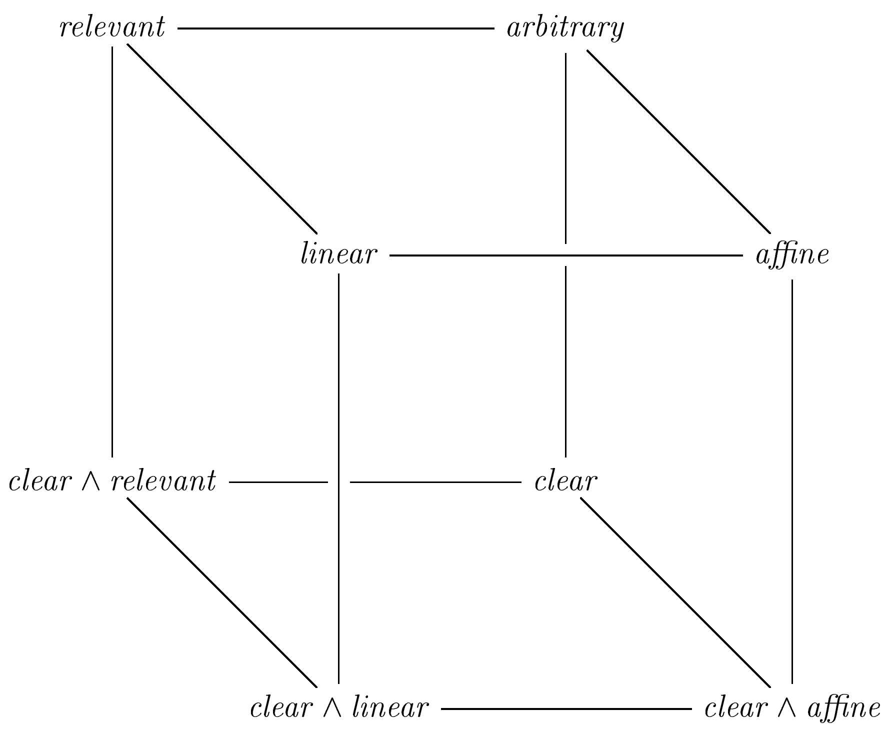


Figure 3.6: The cube of expression properties

$$\equiv \text{ is a congruence}$$

$$
\begin{aligned}
(let\, y = (let\, x = M\ in\, N)\ in\, O) &\equiv (let\, x = M\ in\ let\, y = N\ in\, O) && \text{(comp)}\\
() &\equiv x && (1.\eta)\\
\pi_i(x_1, x_2) &\equiv x_i && (\times.\beta)\\
(\pi_1(x), \pi_2(x)) &\equiv x && (\times.\eta)
\end{aligned}
$$

$$\frac{M \equiv N \qquad M \,!\, E}{N \,!\, E}$$

$$\frac{(let\, x = M\ in\ (x, x)) \equiv (M, M)}{M \,!\ copyable} \qquad \frac{(let\, x = M\ in\ ()) \equiv ()}{M \,!\ discardable}$$

Figure 3.7: The remaining rules of the let-calculus

and types). (Why there are the two rules involving *copyable* and *discardable*, but no rule for *central*, will be explained in the discussion of Conjecture 3.1 at the end of Appendix B.)

Definition 3.1. The *let-calculus* is defined as follows: Its judgements are those of the form

- $(\Gamma \vdash M \equiv N : A)$
- $(\Gamma \vdash M \,!\, E)$ where E is a property of the precartesian cube
- $(\Gamma \vdash M \,/\, e\, x)$ where x is a variable, and e is a property of the expression cube

(where $(\Gamma \vdash M : A)$ and $(\Gamma \vdash N : A)$ are sequents of the let-language.) The derivation rules are the ones summarised in Appendix A.

3.5 A caveat

The let-calculus has the rule

$$\frac{\Gamma \vdash M \,!\, E \qquad \Gamma \vdash M \equiv N : A}{\Gamma \vdash N \,!\, E}$$

which is sound for precartesian categories. However, the following rule is not sound:

$$\frac{\Gamma \vdash M \,/\, e\, x \qquad \Gamma \vdash M \equiv N : A}{\Gamma \vdash N \,/\, e\, x} \tag{3.3}$$

To see this, let

$$
\begin{aligned}
\Gamma &=_{\text{def}} (x : B) \\
M &=_{\text{def}} (\mathit{let}\, y = x\; \mathit{in}\, ()) \\
N &=_{\text{def}} ()
\end{aligned}
\tag{3.4}
$$

Because $(x : B \vdash x\,!\,\mathit{focal})$, by Rule 3.1 we have $(x : B \vdash M \equiv N : \mathit{unit})$. Because we can derive $(x : B \vdash M \,/\, (\mathit{relevant} \wedge \mathit{clear})x\,)$, with Rule 3.3 we could derive $(x : B \vdash N \,/\, (\mathit{relevant} \wedge \mathit{clear})x\,)$. By Rule 3.1, for every $(\vdash M' : B)$ such that $(\vdash M'\,!\,\mathit{copyable})$ we could derive

$$\vdash (\mathit{let}\, x = M'\; \mathit{in}\, ()) \equiv () : \mathit{unit} \tag{3.5}$$

In *Rel*, the copyable morphisms are the partial functions. But if $(\vdash M' : B)$ denotes the empty function, then the left side of Equation 3.5 denotes the empty function, and therefore the equation is false!

Similarly it follows that, given and expression property e and a variable x, the sequents $(\Gamma \vdash M : A)$ for which $(\Gamma \vdash M \,/\, e\, x)$ is derivable cannot denote some class of morphisms in such a way that the semantics is sound an complete for precartesian categories. For suppose there was such a class. With Γ, M, N as in Definition 3.4, the denotations of $(\Gamma \vdash M : A)$ and $(\Gamma \vdash N : A)$ are the same. By soundness, the denotation of $(\Gamma \vdash M : A)$ is in the class denoted by $(\Gamma \vdash - \,/\, (\mathit{relevant} \wedge \mathit{clear})\, x)$, and therefore the denotation of $(\Gamma \vdash N : A)$ too is in that class. By completeness, we get $(\Gamma \vdash N \,/\, (\mathit{relevant} \wedge \mathit{clear})\, x)$, which violates soundness, as shown for *Rel*.

3.6 Soundness and completeness

In this section we shall state soundness and completeness for the let-calculus. First, we need a few definitions:

Definition 3.2 (Precartesian signature). For a collection $\mathcal{B}$ of base types, the *precartesian types over* $\mathcal{B}$ are defined inductively by

$$A = A_1 * A_2 \mid \mathit{unit} \mid \mathcal{B}$$

A *constant over* $\mathcal{B}$ is defined as a formal arrow $f : A_1, \ldots, A_n \longrightarrow B$ where the A_i and B are precartesian types over $\mathcal{B}$. A *precartesian signature* is defined as a pair $\Sigma = (\mathcal{B}, \mathcal{K})$ where $\mathcal{B}$ is a collection of base types, and $\mathcal{K}$ is a collection of constants over $\mathcal{B}$.

$$K[\![\Gamma \vdash M \equiv N : A]\!] = \begin{cases} \mathit{true} & \text{if } K[\![\Gamma \vdash M : A]\!] = K[\![\Gamma \vdash N : A]\!] \\ \mathit{false} & \text{otherwise} \end{cases}$$

$$\begin{aligned}
K[\![\Gamma \vdash M \,!\, \mathit{central}]\!] &= \begin{cases} \mathit{true} & \text{if } K[\![\Gamma \vdash M : A]\!] \text{ is central} \\ \mathit{false} & \text{otherwise} \end{cases} \\
K[\![\Gamma \vdash M \,!\, \mathit{copyable}]\!] &= \begin{cases} \mathit{true} & \text{if } K[\![\Gamma \vdash M : A]\!] \text{ is copyable} \\ \mathit{false} & \text{otherwise} \end{cases} \\
K[\![\Gamma \vdash M \,!\, \mathit{discardable}]\!] &= \begin{cases} \mathit{true} & \text{if } K[\![\Gamma \vdash M : A]\!] \text{ is discardable} \\ \mathit{false} & \text{otherwise} \end{cases} \\
K[\![\Gamma \vdash M \,!\, \mathit{arbitrary}]\!] &= \mathit{true} \\
K[\![\Gamma \vdash M \,!\, E \wedge F]\!] &= K[\![\Gamma \vdash M \,!\, E]\!] \wedge K[\![\Gamma \vdash M \,!\, F]\!]
\end{aligned}$$

$$\begin{aligned}
K[\![\Gamma \vdash M \,/\, \mathit{affine}\, x]\!] &= \begin{cases} \mathit{true} & \text{if M has at most one free occurrence of } x \\ \mathit{false} & \text{otherwise} \end{cases} \\
K[\![\Gamma \vdash M \,/\, \mathit{relevant}\, x]\!] &= \begin{cases} \mathit{true} & \text{if M has at least one free occurrence of } x \\ \mathit{false} & \text{otherwise} \end{cases} \\
K[\![\Gamma \vdash M \,/\, \mathit{clear}\, x]\!] &= \begin{cases} \mathit{true} & \text{if } K[\![\Delta \vdash N : B]\!] \text{ is central for all} \\ & (\Delta \vdash N : B) \in b_x(\Gamma \vdash M) \\ \mathit{false} & \text{otherwise} \end{cases} \\
K[\![\Gamma \vdash M \,/\, \mathit{arbitrary}\,]\!] &= \mathit{true} \\
K[\![\Gamma \vdash M \,/\, e_1 \wedge e_2\,]\!] &= K[\![\Gamma \vdash M \,/\, e_1\,]\!] \wedge K[\![\Gamma \vdash M \,/\, e_2\,]\!]
\end{aligned}$$

Figure 3.8: Interpretation of the let-calculus in a precartesian category K

Definition 3.3. A *precartesian interpretation* of a precartesian signature $\Sigma = (\mathcal{B}, \mathcal{K})$ is a precartesian category K together with, for each base type $A \in \mathcal{B}$, an object $[\![A]\!] \in \mathit{Ob}(K)$, and for each constant $(f : A_1, \ldots, A_n \longrightarrow B) \in \mathcal{K}$, a morphism $[\![f]\!] \in K([\![A_1]\!] \otimes \cdots \otimes [\![A_n]\!], [\![B]\!])$.

For a precartesian category K, let Σ_K be the precartesian signature whose base types are the objects of K and whose constants are the morphisms of K.

A precartesian interpretation K of a precartesian signature Σ assigns truth values to judgements of the let-calculus over Σ as in Figure 3.8.

Definition 3.4. A *let-theory* over a precartesian signature Σ is defined as a collection $\mathcal{T}$ of judgements over Σ of the form $(\Gamma \vdash M \equiv N : A)$, $(\Gamma \vdash M \,!\, E)$, or $(\Gamma \vdash M \,/\, e\, x)$, such that $\mathcal{T}$ is closed under the deduction rules of the let-calculus, and

- If $(\Gamma \vdash M \,/\, \mathit{affine}\, x) \in \mathcal{T}$, then M has at most one free occurrence of x.
- If $(\Gamma \vdash M \,/\, \mathit{relevant}\, x) \in \mathcal{T}$, then M has at least one free occurrence of x.
- If $(\Gamma \vdash M \,/\, \mathit{clear}\, x) \in \mathcal{T}$, then for every $(\Delta \vdash N) \in b_x(\Gamma \vdash N)$ it is true that $(\Delta \vdash N \,!\, \mathit{central}) \in \mathcal{T}$.

Definition 3.5. A *precartesian model* of a let-theory $\mathcal{T}$ over a precartesian signature Σ is a precartesian interpretation of Σ that validates all judgements of $\mathcal{T}$.

Theorem 3.2 (Soundness). *If K is a precartesian interpretation of a precartesian signature Σ, then the judgements over Σ that hold in K form a let-theory.*

Proof. See Appendix B. □

Theorem 3.3 (Completeness). *Let $\mathcal{T}$ be a let-theory over a precartesian signature Σ. If a judgement of the form $(\Gamma \vdash M \equiv N : A)$ holds in all models of $\mathcal{T}$, then it is a theorem of $\mathcal{T}$.*

Proof. See Appendix B. □

Conjecture 3.1. Completeness of the let-calculus holds also for the non-equational judgements. That is, if $\mathcal{T}$ is a let-theory over a precartesian signature Σ, and a judgement of the form $(\Gamma \vdash M \,!\, E)$ or $(\Gamma \vdash M \,/\, e\, x)$ holds in all models of $\mathcal{T}$, then it is a theorem of $\mathcal{T}$.

A discussion of the completeness issue for non-equational judgements, including a strong argument in favour of Conjecture 3.1, is given at the end of Appendix B.

Definition 3.6. Let K and K' be interpretations of a precartesian signature Σ. Then a *morphism of interpretations of* Σ from K to K' is a strict morphism of precartesian categories from K to K' that preserves all base types and constants of Σ.

Theorem 3.4 (Initiality). *Let $\mathcal{T}$ be a let-theory over a precartesian signature Σ. Then $\mathcal{T}$ has a precartesian (term-)model $K_{\mathcal{T}}$ such that for every precartesian model K of $\mathcal{T}$ there is a unique morphism $H : K_{\mathcal{T}} \longrightarrow K$ of interpretations of Σ.*

Proof. See Appendix B. □

3.6.1 Empirical soundness

Is the let-calculus sound for realistic call-by-value programming languages? Or are there phenomena other than number and order of evaluations that we have not taken into account? One can prove that all judgements of the let-calculus can be derived (via longer and less intuitive proofs) in Moggi's computational lambda-calculus. (The converse is an open problem.) In the abstract of [Mog88] Moggi's claims that the computational let-calculus provides

> *a correct basis for proving equivalence of programs, independent from any specific computational model.*

This seems to be empirically true—nobody seems to have found *non-pathological* examples where a derivable equation is operationally false. (A pathological example would be a language that can observe the syntax of the running program (e.g. by examining the store that contains the program). In that case, only identical expressions would be operationally equivalent.)

3.7 Notation

Let's conclude this chapter by fixing some notation that we shall use in the remainder of this thesis.

For $(\Gamma \vdash M : A)$ we may write $(\Gamma \vdash M)$. (This is unambiguous, because Γ and M determine A.) We may even write M for $(\Gamma \vdash M : A)$ when we know Γ. For types $A_1, \ldots, A_n$, let

$$A_1 * A_2 * \cdots A_{n-1} * A_n =_{\text{def}} \begin{cases} \mathit{unit} & \text{if } n = 0 \\ A_1 & \text{if } n = 1 \\ (\ldots(A_1 * A_2) * \cdots * A_{n-1}) * A_n & \text{if } n \geq 2 \end{cases}$$

For expressions $M_1, \ldots, M_n$, let

$$(M_1, M_2, \ldots, M_{n-1}, M_n) =_{\text{def}} \begin{cases} () & \text{if } n = 0 \\ M_1 & \text{if } n = 1 \\ (\ldots(M_1, M_2), \ldots, M_{n-1}), M_n) & \text{if } n \geq 2 \end{cases}$$

For a sequent M of type $A_1 * \cdots * A_n$ and $i \in \{1, \ldots, n\}$, let

$$p_i(M) =_{\text{def}} \pi_{i_1}(\pi_{i_2}(\ldots \pi_{i_k}(M)))$$

where the $i_j \in \{1, 2\}$ are the obvious indices leading to A_i. For expressions $M_1, \ldots, M_n$ and variables $x_1, \ldots, x_n$, let

$$(\mathit{let}\, x_1 = M_1 \ldots x_n = M_n\ \mathit{in}\ N) =_{\text{def}} (\mathit{let}\, x_1 = M_1\ \mathit{in}\ \ldots \mathit{let}\, x_1 = M_1\ \mathit{in}\ N)$$

For variables $x_1, \ldots, x_n$ and expressions $M_1, \ldots, M_n$, let $N[x_1 := M_1, \ldots, x := M_n]$ be the result of the simultaneous substitution of the free occurrences of the x_i by the corresponding M_i (avoiding variable capture).

When during a proof or discussion we introduce a variable that has not been mentioned, we assume that this variable is *fresh*—that is, it occurs neither free nor bound in any of the expressions under discussion.

The *factorisation* of a type A is defined as the sequence $A_1, \ldots, A_n$, where none of the A_i is a product type or the unit type, such that A is the product of the A_i up to associativity and neutrality. For example, the type $A*((B*I)*C)$ has the factorisation (A, B, C) if A, B, and C are not product types. If $(\Gamma \vdash M : A)$ and $(y_1 : A_1, \ldots, y_n : A_n \vdash N : B)$ are sequents such that $A_1, \ldots, A_n$ is the factorisation of A, then let

$$(\mathit{let}\, y_1, \ldots, y_n = M \;\mathit{in}\; N) =_{\mathrm{def}} (\mathit{let}\, z = M \;\mathit{in}\; \mathit{let}\, y_1 = p_1(z) \ldots y_n = p_n(z) \;\mathit{in}\; N)$$

Chapter 4

Example: Partiality

In this chapter, we shall fit categorical models of partiality into our precartesian framework. With hindsight, many well-known models of partiality are special precartesian categories: Concrete examples are categories of sets and partial functions, pointed cpo's and strict continuous functions, and so on. These form precartesian categories such that every morphism is central and copyable. The total maps coincide with the discardable maps. Abstract definitions of such special precartesian categories have been given again and again: pre-dht-categories [Hoe77], p-categories [Ros86], copy categories (Cockett), g-monoidal categories [CG99]. According to Robinson and Rosolini [RR88], Curien and Obtulowicz [CO86] defined categories that they called 'precartesian categories', which are equivalent to Rosolini's p-categories with a 'one-element object' (i.e. a tensor unit which is terminal in the subcategory of total maps). I used the name 'precartesian category' before I heard of this. Luckily, p-categories with a one-element object are a special case of our precartesian categories, as we shall see in this chapter, so my unintended re-definition of precartesian categories is rather benign.

We shall begin this chapter by recalling p-categories and how they arise naturally from *dominions*. Next, we shall specialise the let-calculus to p-categories, obtaining the *p-calculus*. Finally, we shall prove soundness and completeness of the p-calculus with respect to p-categories, and show that completeness holds even for the class of p-categories that arise from dominions.

4.1 Dominions and p-categories

P-categories arise naturally from *dominions*.

Definition 4.1. A *dominion* $\mathcal{M}$ on a category C is a collection of monos that contains all isos and is closed under composition, such that for every $m \in \mathcal{M}$ and

$f : A \longrightarrow B$, the pullback

$$\begin{tikzcd}
A' \arrow[r, "f'"] \arrow[d, hook, "m'"'] \arrow[dr, phantom, "\lrcorner", very near start] & B' \arrow[d, hook, "m"] \\
A \arrow[r, "f"] & B
\end{tikzcd}$$

exists and has $m' \in \mathcal{M}$. We call a category C with a dominion is called a *dominion category.*

A dominion category C is deemed to be a category of total maps. From it we can construct the category $Par(C)$ of partial maps as follows:

Objects: Those of C

Arrows (from A to B): Equivalence classes of spans

$$\begin{tikzcd}
 & A' \arrow[dl, hook', "m"'] \arrow[dr, "f"] & \\
A & & B
\end{tikzcd}$$

where $m \in \mathcal{M}$, $f \in C(A', B)$, and the equivalence is given by considering A' up to isomorphism. We write $[m, f]$ for the equivalence class of the above span.

Composition: Using pullbacks:

$$\begin{tikzcd}
 & & A'' \arrow[dl, hook', "m''"'] \arrow[dr, "f'"] \arrow[dd, phantom, "\vee", very near start] & & \\
 & A' \arrow[dl, hook', "m"'] \arrow[dr, "f"] & & B' \arrow[dl, hook', "m'"'] \arrow[dr, "g"] & \\
A & & B & & C
\end{tikzcd}$$

Identities: Spans $[id_A, id_A]$.

Note that the closure properties in the definition of a dominion arise directly from trying to define $Par(C)$ as above.

There is an obvious faithful identity-on-objects functor $J : C \longrightarrow Par(C)$ that sends a total map $f : A \longrightarrow B$ to the span $[id_A, f]$.

Now suppose that C has finite products. As is well known (see e.g. [RR88]), $Par(C)$ with the definitions in Figure 4.1 forms a *p-category* as introduced in [Ros86]. Here is our version of the definition of a p-category:

$$
\begin{aligned}
A \otimes B &= A \times B \\
I &= 1 \\
[m, f] \otimes [m', f'] &= [m \times m', f \times f'] \\
\delta'_A &= J(\delta_A) \\
p'_{A,B} &= J(p_{A,B}) \\
q'_{A,B} &= J(q_{A,B}) \\
!'_A &= J(!_A)
\end{aligned}
$$

Figure 4.1: Defining a p-category from a dominion category with finite products

Definition 4.2. A *p-category* is a precartesian category such that every morphism is central and copyable.

Our definition agrees with Rosolini's except that we have not only projections, but also a tensor unit and discard map. Therefore, our p-categories should be equivalent to the 'precartesian categories' of Curien and Obtulowicz, as mentioned in the introduction of this chapter.

A concrete example of a p-category is the category *Pfn* of sets and partial functions. This follows immediately from our result that *Rel* is a precartesian category and that a relation is a partial function if and only if it is copyable (see Chapter 2).

Proposition 4.1. *If C is a dominion category with finite products, then $Par(C)$ is a p-category.*

4.2 The p-calculus

Obviously it is sound for p-categories to augment the let-calculus with the rules

$$\Gamma \vdash M \,!\, \mathit{central} \qquad\qquad \Gamma \vdash M \,!\, \mathit{copyable}$$

Therefore, the equation $(\mathit{let}\, x = M\, \mathit{in}\, N) \equiv M[x := N]$ can be false only if the denotation of M is not discardable. In that case it suffices that $(\Gamma, x \vdash N \,/\, x\, \mathit{relevant})$ is true. Therefore, we can abandon judgements that involve *central*, *copyable*, *clear*, and *affine* and replace Rule 3.1 by the following two (where *total* is an alias for *discardable*):

$$\frac{\Gamma, x : A \vdash N \,/\, \mathit{relevant}\, x}{\Gamma \vdash (\mathit{let}\, x = M\ \mathit{in}\ N) \equiv N[x := M] : B} \tag{4.1}$$

$$\frac{\Gamma \vdash M\,!\,\mathit{total}}{\Gamma \vdash (\mathit{let}\, x = M\ \mathit{in}\ N) \equiv N[x := M] : B} \tag{4.2}$$

We can also reduce the rules for precartesian properties to the ones in Figure 4.2. Of the rules for judgements of the form $(\Gamma \vdash M \,/\, e\, x)$ we need only the following:

$$\frac{\Gamma \vdash N : A}{\Gamma \vdash N \,/\, \mathit{relevant}\, x} \text{ if } N \text{ has at most one free occurrence of } x \tag{4.3}$$

To achieve completeness for judgements of the form $(\Gamma \vdash M\,!\,\mathit{total})$, we add the rule

$$\frac{\Gamma \vdash (\mathit{let}\, x = M\ \mathit{in}\ ()) \equiv () : A}{\Gamma \vdash M\,!\,\mathit{total}} \tag{4.4}$$

which is obviously sound. (As explained in Section 3.6, we could not add such rules to the let-calculus, because the one for centrality is unsound.) Thus we arrive at the p-calculus:

Definition 4.3. The *p-calculus* over a precartesian signature Σ is defined as follows: Its judgements are those of the form $(\Gamma \vdash M \equiv N : A)$, $(\Gamma \vdash M\,!\,\mathit{total})$, or $(\Gamma \vdash M \,/\, \mathit{relevant}\, x)$ (where $(\Gamma \vdash M : A)$ and $(\Gamma \vdash N : A)$ are sequents of the let-language over Σ). The rules are those in Figures 4.2 and 3.7 (with $E = \mathit{total}$), and Rules 4.1, 4.2, 4.3, and 4.4.

4.3 Soundness and completeness

In this section, we shall prove that the p-calculus is sound and complete for p-categories *with respect to all three kinds of judgements.* Moreover, we shall prove completeness even for p-categories of the form $\mathit{Par}(C)$, where C is a dominion category with finite products—let's call such p-categories *dominical.*

Definition 4.4. A *p-theory* over a precartesian signature Σ is defined as a collection $\mathcal{T}$ of judgements of the p-calculus over Σ such that $\mathcal{T}$ is closed under the deduction rules of the p-calculus, and such that if $(\Gamma \vdash M \,/\, \mathit{relevant}\, x) \in \mathcal{T}$, then M has at least one free occurrence of x.

Definition 4.5. A p-interpretation of a p-theory $\mathcal{T}$ is a precartesian interpretation K of $\mathcal{T}$ which is a p-category.

Definition 4.6. A *p-model* of a p-theory $\mathcal{T}$ over a precartesian signature Σ is a p-interpretation of Σ that validates all judgements of $\mathcal{T}$.

$$x_1 : A_1, \ldots, x_n : A_n \vdash x_i \,!\, \mathit{total}$$

$$\frac{\Gamma \vdash M \,!\, E \qquad \Gamma, x : A \vdash N \,!\, \mathit{total}}{\Gamma \vdash \mathit{let}\, x = M \;\mathit{in}\, N \,!\, \mathit{total}}$$

$$\Gamma \vdash () \,!\, \mathit{total}$$

$$\frac{\Gamma \vdash M \,!\, \mathit{total} \qquad \Gamma \vdash N \,!\, \mathit{total}}{\Gamma \vdash (M, N) \,!\, \mathit{total}}$$

$$\frac{\Gamma \vdash M \,!\, \mathit{total}}{\Gamma \vdash \pi_i(M) \,!\, \mathit{total}}$$

$$\frac{\begin{array}{c}\Gamma \vdash M_1 \,!\, \mathit{total} \quad \ldots \quad \Gamma \vdash M_n \,!\, \mathit{total} \\ y_1 : A_1, \ldots, y_n : A_n \vdash f(y_1, \ldots, y_n) \,!\, \mathit{total}\end{array}}{\Gamma \vdash f(M_1, \ldots, M_n) \,!\, \mathit{total}} \quad \begin{array}{l}\text{for every constant} \\ f : A_1, \ldots, A_n \longrightarrow B\end{array}$$

Figure 4.2: Inference rules for totality

Theorem 4.2 (Soundness). *If K is a p-interpretation of a precartesian signature Σ, then the judgements over Σ that hold in K form a p-theory.*

Proof. Let $\mathcal{T}$ be the class of judgements over Σ that hold in K. By Theorem 3.2, $\mathcal{T}$ is a let-theory. Because all judgements of the form $(\Gamma \vdash M\,!\,\mathit{central})$ and $\Gamma \vdash M\,!\,\mathit{copyable})$ hold in K, $\mathcal{T}$ is also a p-theory. $\square$

Theorem 4.3 (Initiality). *Let $\mathcal{T}$ be a p-theory over a precartesian signature Σ. Then $\mathcal{T}$ has a (term-) p-model $K_{\mathcal{T}}$ such that for every precartesian model K of $\mathcal{T}$ there is a unique morphism $H : K_{\mathcal{T}} \longrightarrow K$ of interpretations of Σ.*

Proof. Let's construe $\mathcal{T}$ as a let-theory that has all judgements of the form $(\Gamma \vdash M\,!\,\mathit{central})$ and $(\Gamma \vdash M\,!\,\mathit{copyable})$. By Theorem 3.4, $\mathcal{T}$ has a precartesian (term-)model $K_{\mathcal{T}}$ such that for every precartesian model K of $\mathcal{T}$ there is a unique morphism $H : K_{\mathcal{T}} \longrightarrow K$ of interpretations of Σ. Because every morphism of $K_{\mathcal{T}}$ is denotable, the judgements of the form $(\Gamma \vdash M\,!\,\mathit{central})$ and $(\Gamma \vdash M\,!\,\mathit{copyable})$ imply that $K_{\mathcal{T}}$ is a p-category. $\square$

Theorem 4.4 (Completeness). *Let $\mathcal{T}$ be a p-theory over a precartesian signature Σ. If a judgement over Σ holds in all p-models of $\mathcal{T}$, then it is a theorem of $\mathcal{T}$.*

Proof. Suppose that $(\Gamma \vdash M \equiv N : A)$ holds in all p-models of $\mathcal{T}$. Let K be a precartesian model of $\mathcal{T}$. All morphisms of K that are denotable by sequents of the p-calculus over Σ are copyable and central. The morphisms of K that are copyable and central form a p-category. Therefore $(\Gamma \vdash M \equiv N : A)$ holds in all precartesian models of $\mathcal{T}$. By Theorem 3.3, we have $(\Gamma \vdash M \equiv N : A) \in \mathcal{T}$. Now suppose that $(\Gamma \vdash M\,!\,\mathit{total})$ holds in all p-models of $\mathcal{T}$. Therefore, it holds in all precartesian models of $\mathcal{T}$. Because $(\Gamma \vdash M\,!\,\mathit{total})$ holds in the term model $K_{\mathcal{T}}$, we have $(\Gamma \vdash (\mathit{let}\, x = M\, \mathit{in}\, ()) \equiv () : A) \in \mathcal{T}$. By Rule 4.4, we have $(\Gamma \vdash M\,!\,\mathit{total}) \in \mathcal{T}$. If $(\Gamma \vdash M \,/\, \mathit{relevant}\, x)$ holds in all p-models of $\mathcal{T}$, then it holds in $K_{\mathcal{T}}$. Therefore M has at least one free occurrence of x. Therefore $(\Gamma \vdash M \,/\, \mathit{relevant}\, x) \in \mathcal{T}$. $\square$

The next proposition is the key to proving completeness for dominical p-categories:

Proposition 4.5. *Every p-category can be fully embedded into a dominical p-category by a strict precartesian functor.*

This is well known (see e.g. Theorem 1.6 of [RR88]).

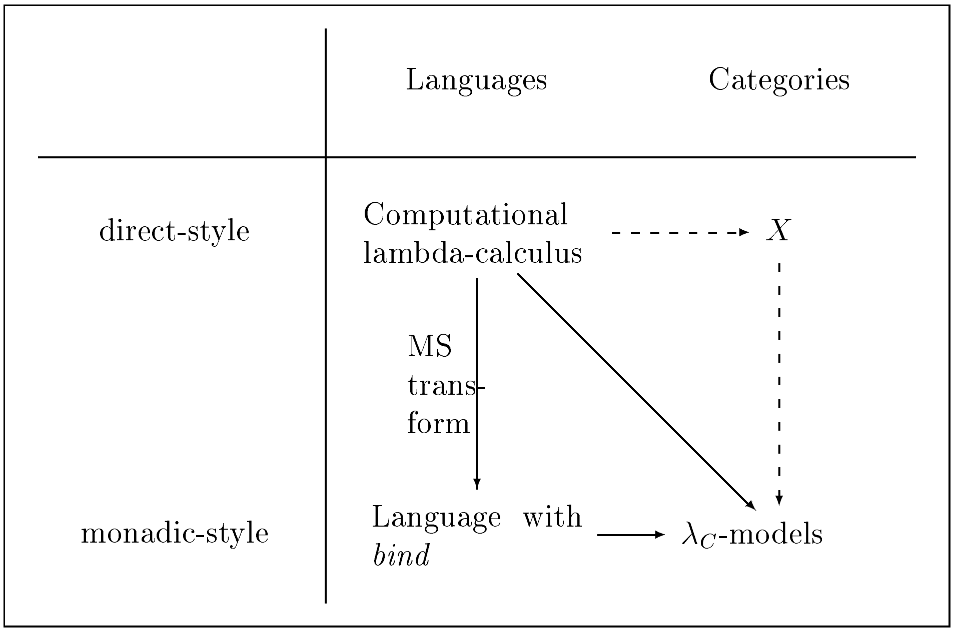
Theorem 4.6 (Completeness). *Let $\mathcal{T}$ be a p-theory over a precartesian signature Σ. If a judgement over Σ holds in all dominical p-models of $\mathcal{T}$, then it is a theorem of $\mathcal{T}$.*

Proof. Let $(\Gamma \vdash M \equiv N : A)$ be a judgement over Σ that holds in all dominical p-models of $\mathcal{T}$. Let K be a p-model of $\mathcal{T}$, and let C be a dominion category with finite products such that K is embedded into $Par(C)$ by a strict precartesian functor. In $Par(C)$ it holds that $(\Gamma \vdash M \equiv N : A)$. Because of the embedding, it holds in K too. So $(\Gamma \vdash M \equiv N : A)$ holds in all precartesian models of $\mathcal{T}$, and by Theorem 4.4 we have $(\Gamma \vdash M \equiv N : A) \in \mathcal{T}$. $\square$

Chapter 5

Precartesian models and monadic models

As explained in the introduction of this thesis, Eugenio Moggi's semantics of the computational lambda-calculus assigns to each sequent $(x_1 : A_1, \ldots, x_n : A_n \vdash M : A)$ a morphism $A_1 \times \ldots \times A_n \longrightarrow TB$, where $\times$ denotes a cartesian product, and T is the endofunctor of a *monad*. By contrast, the precartesian semantics in Figure 2.1 is *direct* in that it assigns to $(x_1 : A_1, \ldots, x_n : A_n \vdash M : A)$ a morphism $A_1 \otimes \ldots \otimes A_n \longrightarrow B$. In the introduction, I claimed that precartesian categories provide the solution X in the diagram



where the diagonal arrow stands for Moggi's semantics. In this chapter, we shall prove this using a construction that takes monadic models to direct models. (We shall construct the direct model by defining extra structure on the Kleisli category of the monad.) In particular, we shall extend the semantics in Figure 2.1 to higher-order operators, obtaining a semantics of the whole computational lambda-calculus.

Translating monadic models into precartesian models enables us discuss central, copyable, and discardable morphisms. Such a discussion can be illuminating, as we shall see in Chapters 6 and 8.

5.1 Monadic models of the computational lambda-calculus

In this section we shall recall some basic definitions and observations about monads, introduce Moggi's *cartesian computational models* and λ_C*-models*, and fix the notation.

Definition 5.1. A *monad* $T = (T, \eta, \mu)$ in a category K consists of a functor $T : K \longrightarrow K$ and two natural transformations

$$\eta : Id_K \longrightarrow T, \qquad\qquad \mu : T^2 \longrightarrow T$$

which make the following diagrams commute

$$\begin{tikzcd} T^3 \arrow[r, "T\mu"] \arrow[d, "\mu T"'] & T^2 \arrow[d, "\mu"] \\ T^2 \arrow[r, "\mu"] & T \end{tikzcd} \qquad\qquad \begin{tikzcd} IdT \arrow[r, "\eta T"] \arrow[dr, "id"'] & T^2 \arrow[d, "\mu"] & TId \arrow[l, "T\eta"'] \arrow[dl, "id"] \\ & T & \end{tikzcd}$$

(The square is called the *associativity law* of the monad, and the two triangles are called the left and right *neutrality laws*, respectively.)

Definition 5.2. The *Kleisli category* K_T of a monad T in a category K is defined as follows (where semicolon denotes the composition of K_T, and colon denotes the composition of K):

$$\begin{aligned} Ob(K_T) &= Ob(K) \\ K_T(A, B) &= K(A, TB) \\ f; g &= f : Tg : \mu \\ id_A &= \eta_A \end{aligned}$$

Definition 5.3. Let K_T be the Kleisli category of a monad T in a category K. Then $F_T : K \longrightarrow K_T$ is defined as the identity-on-objects functor that sends a morphism f to $(f : \eta)$. Moreover, $G_T : K_T \longrightarrow K$ is defined as the functor that sends an object A to TA, and a morphism f of K_T to $(Tf : \mu)$.

Observation 5.1. Let K_T be the Kleisli category of a monad T in a category K. Then there is an adjunction

$$\begin{tikzcd} K \arrow[r, shift left=1.5, "F_T"] \arrow[r, phantom, "\perp"] & K_T \arrow[l, shift left=1.5, "G_T"] \end{tikzcd}$$

such that $G_T F_T = T$, the adjunction's unit is η, the counit is given by $\varepsilon_A =_{\text{def}} id_{TA}$, and it holds that $G_T \varepsilon F_T = \mu$.

Next we shall recall the definition of a strength. The notion of strength is usually defined for monads on (symmetric) monoidal categories. We generalise it to precartesian categories.

Definition 5.4. A *strength* for a monad T on a precartesian category K is a family of arrows

$$t_{A,B} : A \otimes TB \longrightarrow T(A \otimes B)$$

which is natural in A and B *separately*, such that the equations in Figure 5.1 hold. A monad together with a strength is called a *strong monad*.

Let t' stand for the symmetric dual of the strength—that is,

$$(t' : (TA) \otimes B \longrightarrow T(A \otimes B)) =_{\text{def}} \tau; t; T\tau$$

where τ is the twist map.

Next we shall present in our terminology Moggi's definitions of a *cartesian computational model* and a *λ_C-model*. These structures provide the monadic semantics of the computational lambda-calculus.

Definition 5.5. A *precartesian computational model* is a precartesian category together with a strong monad T in K. A *cartesian computational model* (in the sense of Moggi) is a precartesian computational model (K, T) where K is a category with finite products.

Definition 5.6. A monad T in a category K is said to have *T-exponentials* if for all objects A and B of K there is an exponential object $(TB)^A$.

This means that for all objects A and B, there is an arrow $ev : (TB)^A \times A \longrightarrow TB$ such that for all $f : C \times A \longrightarrow B$ there is a unique morphism $\lambda f : C \longrightarrow (TB)^A$ such that

$$\begin{tikzcd} (TB)^A \times A \arrow[r, "ev"] & B \\ C \times A \arrow[u, "\lambda f \times A"] \arrow[ur, "f"'] & \end{tikzcd}$$

$$\begin{tikzcd}
I \otimes TA \arrow[r, "t"] \arrow[dr, "q"'] & T(I \otimes A) \arrow[d, "Tq"] \\
& TA
\end{tikzcd} \tag{5.1}$$

$$\begin{tikzcd}
(A \otimes B) \otimes TC \arrow[rr, "t"] \arrow[d, "\alpha"'] & & T((A \otimes B) \otimes C) \arrow[d, "T\alpha"] \\
A \otimes (B \otimes TC) \arrow[r, "A \otimes t"] & A \otimes T(B \otimes C) \arrow[r, "t"] & T(A \otimes (B \otimes C))
\end{tikzcd} \tag{5.2}$$

$$\begin{tikzcd}
A \otimes B \arrow[d, "A \otimes \eta"'] \arrow[dr, "\eta"] & \\
A \otimes TB \arrow[r, "t"] & T(A \otimes B)
\end{tikzcd} \tag{5.3}$$

$$\begin{tikzcd}
A \otimes TB \arrow[rr, "t"] & & T(A \otimes B) \\
A \otimes T^2B \arrow[u, "A \otimes \mu"] \arrow[r, "t"] & T(A \otimes TB) \arrow[r, "Tt"] & T^2(A \otimes B) \arrow[u, "\mu"']
\end{tikzcd} \tag{5.4}$$

Figure 5.1: Axioms for a strength for a monad on a precartesian category

Rule	Syntax	Semantics
var	$x_1 : A_1, \ldots, x_n : A_n \vdash x_i : A_i$	$= A_1 \times \cdots \times A_n \xrightarrow{\pi_i} A_i \xrightarrow{\eta} TA_i$
let	$\dfrac{\Gamma \vdash M : A \qquad \Gamma, x : A \vdash N : B}{\Gamma \vdash let\, x = M\, in\, N : B}$	$= \Gamma \xrightarrow{f} TA$ $= \Gamma \times A \xrightarrow{g} TB$ $= \Gamma \xrightarrow{\langle id, f\rangle} \Gamma \times TA \xrightarrow{t} T(\Gamma \times A) \xrightarrow{Tg} TTB \xrightarrow{\mu} TB$
()	$\Gamma \vdash () : unit$	$= \Gamma \xrightarrow{!} 1 \xrightarrow{\eta} T1$
$(-,-)$	$\dfrac{\Gamma \vdash M : A \qquad \Gamma \vdash N : B}{\Gamma \vdash (M, N) : A * B}$	$= \Gamma \xrightarrow{f} TA$ $= \Gamma \xrightarrow{g} TB$ $= \Gamma \xrightarrow{\langle f, g\rangle} TA \times TB \xrightarrow{\psi} T(A \times B)$
π_i	$\dfrac{\Gamma \vdash M : A_1 * A_2}{\Gamma \vdash \pi_i(M) : A_i}$	$= \Gamma \xrightarrow{f} T(A_1 \times A_2)$ $= \Gamma \xrightarrow{f} T(A_1 \times A_2) \xrightarrow{T\pi_i} TA_i$

Figure 5.2: Moggi's semantics of the computational lambda-calculus: first-order fragment

Definition 5.7. A *precartesian λ_C-model* is a precartesian computational model (K, T) with T-exponentials. A *λ_C-model* (in the sense of Moggi) is a precartesian λ_C-model (K, T) where K is a category with finite products.

Moggi's semantics of the computational lambda-calculus using a λ_C-model (K, T) is presented in Figures 5.2–5.4, where $\psi =_{\mathrm{def}} t'; Tt; \mu$ and

$$app =_{\mathrm{def}} \left(T((TB)^A) \times TA \xrightarrow{\psi} T((TB)^A \times A) \xrightarrow{Tev} T^2B \xrightarrow{\mu} TB\right)$$

Rule	Syntax	Monadic semantics
$[-]$	$\dfrac{\Gamma \vdash M : A}{\Gamma \vdash [M] : TA}$	$= \Gamma \xrightarrow{f} TA$ $= \Gamma \xrightarrow{f} TA \xrightarrow{\eta} TTA$
μ	$\dfrac{\Gamma \vdash M : TA}{\Gamma \vdash \mu(M) : A}$	$= \Gamma \xrightarrow{f} TTA$ $= \Gamma \xrightarrow{f} TTA \xrightarrow{\mu} TA$

Figure 5.3: Moggi's semantics of the computational lambda-calculus: $\mu(M)$ and $[M]$

Rule	Syntax	Semantics
λ	$\dfrac{\Gamma, x : A \vdash M : B}{\Gamma \vdash \lambda x : A.M : A \rightharpoonup B}$	$= \Gamma \times A \xrightarrow{f} TB$ $= \Gamma \xrightarrow{\lambda f} (TB)^A \xrightarrow{\eta} T((TB)^A)$
app	$\dfrac{\Gamma \vdash M : A \rightharpoonup B \qquad \Gamma \vdash N : A}{\Gamma \vdash MN : B}$	$= \Gamma \xrightarrow{f} T((TB)^A)$ $= \Gamma \xrightarrow{g} TA$ $= \Gamma \xrightarrow{\langle f,g \rangle} T((TB)^A) \times TA \xrightarrow{app} TB$

Figure 5.4: Moggi's semantics of the computational lambda-calculus: higher-order structure

5.2 Precartesian models of the computational lambda-calculus

We shall define the direct models of the computational lambda-calculus using three layers:

Computational abstract Kleisli-categories
Precartesian abstract Kleisli-categories
Abstract Kleisli-categories

The hierarchy of direct models corresponds to the following hierarchy of monadic models:

λ_C-models
Cartesian computational models
Monads

5.2.1 Closure of precartesian categories under the Kleisli construction

In this section, we shall see how to construct a precartesian category K_T as the Kleisli category of a strong monad T on a precartesian category K.

Theorem 5.2. *Let $K = (K, \times, 1, \delta, p, q, !, T)$ be a precartesian computational model. For objects A, B, C, and morphisms $f \in K_T(A, B)$, let*

$$\begin{aligned} A \otimes B &=_{\text{def}} A \times B \\ C \otimes f &=_{\text{def}} \Big(C \times A \xrightarrow{C \times f} C \times TB \xrightarrow{t} T(C \times B)\Big) \\ f \otimes C &=_{\text{def}} \Big(A \times C \xrightarrow{f \times C} B \times TC \xrightarrow{t'} T(B \times C)\Big) \end{aligned}$$

Then $(K_T, \otimes, 1, F_T\delta, F_Tp, F_Tq, F_T!)$ is a precartesian category. Moreover, F_T is a strict precartesian functor that sends central morphisms to central morphisms.

The proof, which is rather technical, can be found in Appendix C.

Remark 5.1. Because of Theorem 5.2, the notion of strong monads now applies to K_T. Because we generalised Moggi's framework by allowing K to be an arbitrary precartesian category (as opposed to a category with finite products), we can now construct yet another precartesian category from any strong monad on K_T. This means for programming-languages that we can now try to add computational effects successively by iterating the Kleisli construction rather than using 'monad transformers' (we shall discuss this in Section 9.2.4).

Remark 5.2. The construction of a premonoidal tensor on the Kleisli-category of a strong monad in a (symmetric) monoidal category is by now well known—see [PR97]. In fact, Power and Robinson proved a more general theorem (Theorem 4.2 in [PR99]) stating that the Kleisli category of a *premonoidal dyad* (a kind of generalised strong monad) on a *pre*monoidal category forms a premonoidal category. So the only novelty of our Theorem 5.2 is that it deals with precartesian rather than just premonoidal structure.

Proposition 5.3. *Let (K, T) be a precartesian computational model, and let K_T be the precartesian category that arises as the Kleisli category. Then the precartesian semantics (Figure 2.1) in K_T agrees with the monadic semantics (Figure 5.2) in (K, T).*

Proof. Straightforward. □

5.2.2 Abstract Kleisli-categories

In this section, we shall give the direct semantics of the operators $\mu(-)$ and $[-]$. This semantics can be understood even in the absence of a strength and a tensor. The key observation is that for a category K with a monad T, the adjunction $F_T \dashv G_T : K_T \longrightarrow K$ induces some structure on the Kleisli category: Let

$$L =_{\mathrm{def}} F_T G_T$$
$$\vartheta_A =_{\mathrm{def}} F_T \eta_A$$

and let ε be the counit of the adjunction. Then L is and endofunctor on K_T, ε is a natural isomorphism $L \longrightarrow Id$, and for each object A, we have $\vartheta_A : A \longrightarrow LA$. However, ϑ is (fortunately) not generally a natural transformation $Id \longrightarrow L$, as we shall see in Example 5.2. Of course, L, ϑ, and ε satisfy certain equations. This motivates the following definition:

Definition 5.8. An *abstract Kleisli-category* is

- A category K
- A functor $L : K \longrightarrow K$
- A transformation $\vartheta : Id \longrightarrow L$
- A natural transformation $\varepsilon : L \stackrel{\cdot}{\longrightarrow} Id$

such that ϑL is a natural transformation $L \stackrel{\cdot}{\longrightarrow} L^2$, and

$$\begin{tikzcd} Id \arrow[r, "\vartheta"] \arrow[d, "\vartheta"'] & L \arrow[d, "L\vartheta"] \\ L \arrow[r, "\vartheta L"] & L^2 \end{tikzcd} \qquad \begin{tikzcd} Id \arrow[r, "\vartheta"] \arrow[dr, "id"'] & L \arrow[d, "\varepsilon"] \\ & Id \end{tikzcd} \qquad \begin{tikzcd} L \arrow[r, "\vartheta L"] \arrow[dr, "id"'] & L^2 \arrow[d, "L\varepsilon"] \\ & L \end{tikzcd}$$

Let's pronounce ϑ 'thunk' and ε 'force', agreeing with Hayo Thielecke's terminology for $\otimes\neg$-categories, which turned out to be abstract Kleisli-categories with extra structure, as we shall see in Chapter 8.

Example 5.1. *Rel* (which is isomorphic to the Kleisli category of the covariant powerset monad (see e.g. [Jac94])). For a set A, let LA be the powerset of A. For a relation $R : A \longrightarrow B$, and sets $X \subseteq A$ and $Y \subseteq B$, let (where $R[X]$ is the image of X under R)

$$\begin{aligned} X(LR)Y &\Leftrightarrow Y = R[X] \\ x\vartheta X &\Leftrightarrow X = \{x\} \\ X\varepsilon\, x &\Leftrightarrow x \in X \end{aligned}$$

Proposition 5.4. *Let K be a category, and let T be a monad on K. Let $L = F_T G_T$, let $\vartheta_A = F_T \eta_A$, and let ε be the counit of the adjunction $F_T \dashv G_T$. Then $(K_T, L, \vartheta, \varepsilon)$ is an abstract Kleisli-category.*

Proof. Let F stand for F_T, and G for G_T. The square in the definition of an abstract Kleisli-category, and the naturality of ϑL, follow from applying F to the naturality squares

$$\begin{tikzcd} A \arrow[r, "\eta_A"] \arrow[d, "\eta_A"'] & GFA \arrow[d, "GF\eta"] \\ GFA \arrow[r, "\eta_{FGA}"] & GFGFA \end{tikzcd} \quad \text{and} \quad \begin{tikzcd} GA \arrow[r, "\eta_{GA}"] \arrow[d, "Gf"'] & GFGA \arrow[d, "GFGf"] \\ GB \arrow[r, "\eta_{GB}"] & GFGB \end{tikzcd}$$

respectively. The left triangle in the definition of an abstract Kleisli-category is

$$\begin{tikzcd} FA \arrow[r, "F\eta_A"] \arrow[dr, "id_{FA}"'] & FGFA \arrow[d, "\varepsilon_{FA}"] \\ & FA \end{tikzcd}$$

The right triangle follows from applying F to

$$\begin{tikzcd} GA \arrow[r, "\eta_{GA}"] \arrow[dr, "id_{GA}"'] & GFGA \arrow[d, "G\varepsilon_A"] \\ & GA \end{tikzcd}$$

□

Remark 5.3. For every abstract Kleisli-category, L forms a comonad on K with comultiplication ϑL and counit ε. (However, this does not seem to matter for our discussion of semantics. In particular, we shall not need to construct the co-Kleisli-category in this thesis.)

Before we shall rephrase the semantics of $\mu(-)$ and $[-]$ in terms of abstract Kleisli-categories, let's make some important definitions and observations:

Definition 5.9. A morphism $f : A \longrightarrow B$ of an abstract Kleisli-category K is called *thunkable* if

$$\begin{tikzcd} A \arrow[r, "\vartheta"] \arrow[d, "f"'] & LA \arrow[d, "Lf"] \\ B \arrow[r, "\vartheta"] & LB \end{tikzcd}$$

The category K_ϑ is defined as the subcategory of K which is determined by all objects and the thunkable morphisms.

Example 5.2. In *Rel*, the thunkable morphisms are the total functions. (That is, they happen to coincide with the focal morphisms.)

Observation 5.5. If K_T is the abstract Kleisli-category of a monad T on a category K, then a morphisms f of K_T is thunkable if and only if in K it holds that

$$f; \eta_T = f; T\eta$$

For a morphism f of an abstract Kleisli-category, let

$$[f] =_{\text{def}} \vartheta; Lf$$

Let $incl : K_\vartheta \hookrightarrow K$ be the inclusion functor.

Observation 5.6. For every abstract Kleisli-category K, there is adjunction

$$[-] : K(incl(A), B) \cong (K_\vartheta)(A, LB)$$

with unit ϑ and counit ε.

Rule	Syntax	Monadic semantics	Abstract Kleisli semantics
$[-]$	$\dfrac{\Gamma \vdash M : A}{\Gamma \vdash [M] : TA}$	$= \Gamma \xrightarrow{f} TA$ $= \Gamma \xrightarrow{f} TA \xrightarrow{\eta} TTA$	$= \Gamma \xrightarrow{f} A$ $= \Gamma \xrightarrow{[f]} LA$
μ	$\dfrac{\Gamma \vdash M : TA}{\Gamma \vdash \mu(M) : A}$	$= \Gamma \xrightarrow{f} TTA$ $= \Gamma \xrightarrow{f} TTA \xrightarrow{\mu} TA$	$= \Gamma \xrightarrow{f} LA$ $= \Gamma \xrightarrow{f} LA \xrightarrow{\varepsilon} A$

Figure 5.5: Monadic semantics and abstract Kleisli semantics of $\mu(M)$ and $[M]$

Note that ϑ plays a double rôle in that it determines K_ϑ *and* the adjunction unit!

Now let T be a monad on a category K, and let K_T be the arising abstract Kleisli-category. Then Moggi's semantics of expressions of the form $\mu(M)$ and $[M]$ can be rephrased as in Figure 5.5.

Remark 5.4. A syntactic construction which amounts to almost the same as our construction of the subcategory of thunkable morphisms is given in Definition 2.8 of [Mog91]: Given a theory $\mathcal{T}$ of the 'simple programming language' (which is a precursor of the computational lambda-calculus), Moggi defines a category $\mathcal{F}(\mathcal{T})$ whose objects are the types, and whose morphisms from A to B are equivalence classes $[x : A \vdash M : B]_\mathcal{T}$ of sequents such that $(x : A \vdash M \downarrow B) \in \mathcal{T}$. According to Moggi's semantics, $(x : A \vdash M \downarrow B)$ states that for the denotation $f : A \longrightarrow TB$ of $(x : A \vdash M : B)$ there is a $g : A \longrightarrow B$ such that in K it holds that $f = g; \eta$. This implies that f is thunkable in K_T. However, it is generally false that every thunkable morphism f of K_T factors through η in K. Therefore $\mathcal{F}(\mathcal{T})$ corresponds to a subcategory of K_ϑ that may be smaller than K_ϑ. A syntactic construction that corresponds precisely to our construction of K_ϑ would use equivalence classes $[x : A \vdash M : B]_\mathcal{T}$ of sequents such that

$$(x : A \vdash (\mathit{let}\, x = M\, \mathit{in}\, [x]) \equiv [M] : TB) \in \mathcal{T}$$

because that equation states that the denotation of $(x : A \vdash M : B)$ is thunkable.

5.2.3 Precartesian abstract Kleisli-categories

As we have seen in the two preceding sections, if T is a strong monad on a category K, then K_T has a precartesian structure and an abstract Kleisli-structure. In this section we shall see how these two structures interact.

Definition 5.10. A *precartesian abstract Kleisli-category* K is

- A precartesian category K together with
- Operators L, ϑ, and ε such that $(K, L, \vartheta, \varepsilon)$ is an abstract Kleisli-category

such that the thunkable morphisms are closed under tensor, and all components of δ, p, q, and $!$ are thunkable.

Now we are aiming to prove that the Kleisli category K_T of a strong monad T on a precartesian category K forms a precartesian abstract Kleisli-category. First, an important lemma:

Lemma 5.1. *Let K be a precartesian category, let L, ϑ, and ε be such that $(K, L, \vartheta, \varepsilon)$ is an abstract Kleisli category. If all morphisms of the form $A \otimes [f]$ or $[f] \otimes A$ are thunkable, then the thunkable morphisms are closed under tensor.*

Proof. Let $f : A \longrightarrow A'$ be a thunkable morphism. We have

$$\begin{aligned}
A \otimes f; \vartheta = A \otimes f; [id] &= A \otimes f; [A \otimes [id]; A \otimes \varepsilon] \\
&= A \otimes f; A \otimes [id]; [A \otimes \varepsilon] && \text{because } A \otimes [id] \text{ is thunkable} \\
&= A \otimes [f]; [A \otimes \varepsilon] && \text{because } f \text{ is thunkable} \\
&= [A \otimes [f]; A \otimes \varepsilon] && \text{because } A \otimes [f] \text{ is thunkable} \\
&= [A \otimes f] = \vartheta; L(A \otimes f)
\end{aligned}$$

□

Remark 5.5. Lemma 5.1 is important in two ways: First, it helps checking that a structure is a precartesian abstract Kleisli-category. Second, it replaces a *conditional* requirement (‘*if* f is thunkable, then so is $A \otimes f$’) by an unconditional one (‘for all f, $A \otimes [f]$ is thunkable’). Therefore, like precartesian categories, precartesian abstract Kleisli-categories have an axiomatisation that consists only of equations that are universally quantified over objects and morphisms.

Proposition 5.7. *Let (K, T) be a precartesian computational model, and let L, ϑ, ε be the abstract Kleisli structure that arises from T. Then $(K_T, L, \vartheta, \varepsilon)$ is a precartesian abstract Kleisli-category. Moreover, every morphism in the image of F_T is thunkable.*

Proof. Let F stand for F_T, and G for G_T. Let colon stand for the composition of K, and semicolon for the composition of K_T. Every morphism in the image of F is thunkable because

$$\begin{aligned} Ff;\vartheta &= Ff : G\vartheta = f : \eta : G\vartheta = f : \eta : GF\eta \\ &= f : \eta : \eta = F(f : \eta) = F(\eta : GFf) = F\eta; FGFf = \vartheta; LFf \end{aligned}$$

Therefore, all components of δ, p, q, and $!$ of K_T are thunkable. It remains to prove that the thunkable morphisms of K_T are closed under tensor. By Lemma 5.1, it suffices to prove that in K_T all morphisms of the form $A \otimes [f]$ and $[f] \otimes A$ are thunkable. This is so because

$$A \otimes [f] = A \otimes F_T f = F_T(A \times f)$$

$\square$

Next we shall prove that, if (K, T) is a cartesian computational model, then all thunkable morphisms of K_T are focal. This implies that it in the computational lambda-calculus we can substitute expressions of the form $[M]$ for any variable (of the same type).

Lemma 5.2. *In a precartesian abstract Kleisli-category, if all morphisms of the form $[f]$ are focal, then all thunkable morphisms are focal.*

Proof. Suppose that all morphisms of the form $[f]$ are focal. Let $f : A \longrightarrow B$ be thunkable. It holds that f is central, because for all $g : A' \longrightarrow B'$ we have

$$\begin{aligned} A \otimes g; f \otimes B' &= A \otimes g; [f] \otimes B'; \varepsilon \otimes B' \\ &= [f] \otimes A'; LB \otimes g; \varepsilon \otimes B' && \text{because } [f] \text{ is central} \\ &= f \otimes A'; \vartheta \otimes A'; LB \otimes g; \varepsilon \otimes B' && \text{because } f \text{ is thunkable} \\ &= f \otimes A'; B \otimes g; \vartheta \otimes B'; \varepsilon \otimes B' && \text{because } \vartheta = [id] \text{ is central} \\ &= f \otimes A'; B \otimes g \end{aligned}$$

To see that f is copyable, consider

$$\begin{aligned} f;\delta &= f;\delta;\vartheta \otimes \vartheta; id \otimes \varepsilon; \varepsilon \otimes id && \text{because } \vartheta = [id] \text{ is central} \\ &= f;\vartheta;\delta; id \otimes \varepsilon; \varepsilon \otimes id && \text{because } \vartheta \text{ is copyable} \\ &= [f];\delta; id \otimes \varepsilon; \varepsilon \otimes id && \text{because } f \text{ is thunkable} \\ &= \delta; [f] \otimes [f]; id \otimes \varepsilon; \varepsilon \otimes id && \text{because } [f] \text{ is copyable and central} \\ &= \delta; [f] \otimes f; \varepsilon \otimes id \\ &= \delta; id \otimes f; f \otimes id \end{aligned}$$

$$
\begin{aligned}
A \rightharpoonup B &=_{\text{def}} (TB)^A \\
(\mathit{apply} \in K_T((A \rightharpoonup B) \otimes A, B)) &=_{\text{def}} \big(ev \in K\,\big((TB)^A \times A, TB\big)\big) \\
\Lambda(f \in K_T(A \otimes B, C)) &=_{\text{def}} F_T(\lambda(f \in K(A \times B, TC))
\end{aligned}
$$

Figure 5.6: Higher-order structure on the precartesian abstract Kleisli-category of a precartesian λ_C-model

It holds that f is discardable because

$$
\begin{aligned}
f;! &= f;\vartheta;! && \text{because } \vartheta \text{ is discardable} \\
&= [f];! && \text{because } [f] \text{ is thunkable} \\
&=! && \text{because } [f] \text{ is discardable}
\end{aligned}
$$

□

Proposition 5.8. *In the precartesian abstract Kleisli-category that arises from a cartesian computational model, all thunkable morphisms are focal.*

Proof. By Lemma 5.2 it suffices to prove that all morphisms of the Kleisli category of the form $[f]$ are focal. We have $[f] = F_T f$. All morphisms of the cartesian computational model are focal. By Proposition 5.2, F_T is a strict precartesian functor that sends central morphisms to central morphisms. By Proposition 2.2, F_T preserves copyable and discardable morphisms. Therefore, $F_T f$ is focal. □

5.2.4 Higher-order structure

In this section, we shall provide the direct semantics of the higher-order operators of the computational lambda-calculus. Let K_T be the precartesian abstract Kleisli-category of a precartesian λ_C-model (K, T), and define *apply* and Λ as in Figure 5.6. Figure 5.7 shows how to rephrase Moggi's semantics in terms of the new operators on K_T. As we shall see, K_T is *closed* in the sense of the following definition:

Definition 5.11. A precartesian abstract Kleisli-category K is called *closed* if for each object A the functor $(-) \otimes A : K_\vartheta \longrightarrow K$ has a right adjoint. For this right adjoint, we shall write $A \rightharpoonup (-)$.

Example 5.3. The category *Rel*. For sets A and B, let $(A \rightharpoonup B)$ be the set of relations $R \in A \times B$, and let

$$
\begin{aligned}
(R, x)\,\mathit{apply}\,y &\Leftrightarrow_{\text{def}} xRy \\
x(\Lambda R)S &\Leftrightarrow_{\text{def}} S = \{(y, z) : (x, y)Rz\}
\end{aligned}
$$

Rule	Syntax	Semantics
λ	$\dfrac{\Gamma, x : A \vdash M : B}{\Gamma \vdash \lambda x : A.M : A \rightharpoonup B}$	$= \Gamma A \xrightarrow{f} B$ $= \Gamma \xrightarrow{\Lambda f} (A \rightharpoonup B)$
app	$\dfrac{\Gamma \vdash M : A \rightharpoonup B \qquad \Gamma \vdash N : A}{\Gamma \vdash MN : B}$	$= \Gamma \xrightarrow{f} (A \rightharpoonup B)$ $= \Gamma \xrightarrow{g} A$ $= \Gamma \xrightarrow{\langle f,g \rangle} (A \rightharpoonup B) \otimes A \xrightarrow{apply} B$

Figure 5.7: Precartesian semantics of the computational lambda-calculus: higher-order structure

Remark 5.6. As explained in Observation 5.6, the inclusion functor $K_\vartheta \hookrightarrow K$ has the right adjoint $L : K \longrightarrow K_\vartheta$. Because the inclusion functor is naturally isomorphic to $(-) \otimes I : K_\vartheta \longrightarrow K$, the functor $L : K \longrightarrow K_\vartheta$ is naturally isomorphic to $I \rightharpoonup (-)$. This implies that semantically $[M]$ is essentially $\lambda x : unit.M$, and $\mu(M)$ is essentially $M()$.

Next we shall prove that the precartesian abstract Kleisli-category of a precartesian λ_C-model is closed. First, a lemma:

Lemma 5.3. *If (K, T) is a precartesian λ_C-model, then for all objects A, the functor $(F_T-) \otimes A : K \longrightarrow K_T$ has a right adjoint.*

Proof. Let $(A \rightharpoonup B)$ and *apply* be as in Figure 5.6. We shall prove that for all $f \in K_T(C \otimes A, B)$ there is a unique solution g of the equation

$$\begin{tikzcd}
(A \rightharpoonup B) \otimes A \arrow[r, "apply"] & B \\
C \otimes A \arrow[u, "(F_T g) \otimes A"] \arrow[ur, "f"'] &
\end{tikzcd}$$

Let's write semicolon for the composition of K_T, and colon for the composition of K. The required g is λf, because

$$\begin{aligned}(F_T g) \otimes A; app &= (F_T g) \otimes A : G(ev) = F_T(g \times A) : G(ev) \\ &= g \times A : \eta : Gev = g \times A : ev\end{aligned}$$

$\square$

Proposition 5.9. *If (K, T) is a precartesian λ_C-model, then the precartesian abstract Kleisli-category K_T is closed.*

Proof. Let F stand for F_T, and G for G_T. By Lemma 5.3, $(F-) \otimes A$ has a right adjoint. Let $A \rightharpoonup (-)$ be this right adjoint, and let *apply* be the counit of the adjunction. For each morphism $f \in K_T(C \otimes A, B)$ we shall find a unique thunkable morphism $h \in K_T(C, A \rightharpoonup B)$ such that

$$\begin{tikzcd}
(A \rightharpoonup B) \otimes A \arrow[r, "apply"] & B \\
C \otimes A \arrow[u, "h \otimes A"] \arrow[ur, "f"'] &
\end{tikzcd}$$

Let λ be the adjunction isomorphism $K_T((FA) \otimes B) \cong K(A, B \rightharpoonup C)$. Because $F\lambda f$ is a solution h, it remains to prove that every thunkable solution h is equal to $F\lambda f$. Consider

$$\begin{aligned}
F\lambda f &= F\lambda(h \otimes A; apply) = F\lambda([h] \otimes A; \varepsilon \otimes A; apply) \\
&= F\lambda((Fh) \otimes A; \varepsilon \otimes A; apply) \\
&= F(h : \lambda(\varepsilon \otimes A; apply)) && \text{naturality of } \lambda \\
&= Fh; F\lambda(\varepsilon \otimes A; apply) = [h]; F\lambda(\varepsilon \otimes A; apply) \\
&= h; \vartheta; F\lambda(\varepsilon \otimes A; apply) && \text{because } h \text{ is thunkable} \\
&= h; F\eta; F\lambda(\varepsilon \otimes A; apply) \\
&= h; F(\eta : \lambda(\varepsilon \otimes A; apply)) \\
&= h; F(\lambda(F\eta \otimes A; \varepsilon \otimes A; apply)) && \text{naturality of } \lambda \\
&= h; F(\lambda(\vartheta \otimes A; \varepsilon \otimes A; apply)) \\
&= h; F(\lambda apply) = h; Fid = h
\end{aligned}$$

□

By Proposition 5.8, if (K, T) is a λ_C-model, then K_T forms a closed precartesian abstract Kleisli-category such that all thunkable morphisms are focal. This is quite a mouthful. Because in Chapter 7 we shall prove soundness and completeness of the computational lambda-calculus with respect to these models, let's give them a shorter name:

Definition 5.12. A *computational abstract Kleisli-category* is a closed precartesian abstract Kleisli-category such that all thunkable morphisms are focal.

5.3 Kleisli categories of commutative, affine, and relevant monads

Next we shall recall what it means for a monad to be commutative, affine, and relevant, and we shall find a striking connection between these three notions and the notions of centrality, copyability, and discardability. The first three notions are due to Kock: Commutative monads were first defined in [Koc70], and affine and relevant ones in [Koc71]—however, this use of the word 'relevant' was introduced only later by Jacobs [Jac94].

Definition 5.13. A strong monad (T, μ, η, t) is on a precartesian category $K = (K, \times, 1, \delta, p, q, !)$ is called

- *commutative* if for all objects A and B, the two maps below agree:

$$\psi = \left(TA \times TB \xrightarrow{t'} T(A \times TB) \xrightarrow{Tt} T^2(A \times B) \xrightarrow{\mu} T(A \times B)\right)$$
$$\psi' = \left(TA \times TB \xrightarrow{t} T(TA \times B) \xrightarrow{Tt'} T^2(A \times B) \xrightarrow{\mu} T(A \times B)\right)$$

- *affine* if $\eta_1 : 1 \longrightarrow T1$ is an isomorphism
- *relevant* if the following diagram commutes for all objects A:

$$\begin{tikzcd} TA \arrow[r, "\delta"] \arrow[dr, "T\delta"'] & TA \times TA \arrow[d, "\psi"] \\ & T(A \times A) \end{tikzcd} \tag{5.5}$$

(There are alternative definitions for affine and relevant monads—for more, see [Jac94].)

Proposition 5.10. *A strong monad T on a category K with finite products is*

- *commutative if and only if every morphism of K_T is central.*
- *relevant if and only if every morphism of K_T is copyable.*
- *affine if and only if every morphism of K_T is discardable.*

We shall prove Proposition 5.10 with the help of the following result, which is interesting in its own right:

Proposition 5.11. *In a precartesian abstract Kleisli-category K_T that arises from a strong monad T on a category K with finite products,*

- *If for all objects A and B the diagram*

$$\begin{tikzcd} LA \otimes LB \arrow[r, "\varepsilon \otimes LB"] \arrow[d, "LA \otimes \varepsilon"'] & A \otimes LB \arrow[d, "A \otimes \varepsilon"] \\ LA \otimes B \arrow[r, "\varepsilon \otimes B"] & A \otimes B \end{tikzcd} \tag{5.6}$$

  commutes, then every morphism is central.

- *If ε_A is copyable for every A, then every morphism is copyable.*

- *If ε_A is discardable for every A, then every morphism is discardable.*

Proof. For every morphism $f \in K_T(A, A')$, it holds that $f = [f]; \varepsilon_{A'}$. Because $[f]$ is thunkable, it follows with Proposition 5.8 that $[f]$ is also focal. Now suppose that Equation 5.6 holds. For arbitrary morphisms $f : A \longrightarrow A'$ and $g : B \longrightarrow B'$ it holds that

$$\begin{aligned} f \otimes id; id \otimes g &= ([f]; \varepsilon) \otimes id; id \otimes ([g]; \varepsilon) \\ &= [f] \otimes id; \varepsilon \otimes id; id \otimes [g]; id \otimes \varepsilon \\ &= [f] \otimes [g]; \varepsilon \otimes id; id \otimes \varepsilon && \text{by centrality of } [g] \end{aligned}$$

Symmetrically, we get

$$id \otimes g; f \otimes id = [f] \otimes [g]; id \otimes \varepsilon; \varepsilon \otimes id$$

So Equation 5.6 implies that $f \otimes id; id \otimes g = id \otimes g; f \otimes id$, and therefore every morphism is central. The equation $f = [f]; \varepsilon_{A'}$ also imples that f is copyable (or discardable) if $\varepsilon_{A'}$ is. $\square$

Proof of Proposition 5.10. It holds that $\psi, \psi' : TA \times TB \longrightarrow T(A \times B)$ are equal to the top-right path and the left-bottom path of Diagram 5.6, respectively. So the Equation $\psi = \psi'$ stating that T is commutative is equivalent to Equation 5.6. Equation 5.5 is equivalent to the equation stating that ε_A is copyable. That η_1 is an isomorphism is equivalent to 1 being terminal in K_T, which is the case if and only if every morphism of K_T is discardable. $\square$

5.4 The monadic-style transform

The construction of the Kleisli category K_T of a precartesian computational model (K, T) expresses the morphisms of K_T in terms of the structure of K and the strong monad T. As explained in the introduction, this construction is the semantic counterpart of the monadic-style transform. In this section, we shall turn this claim into a precise proposition.

Let K be a λ_C-model. Let L be the let-language of K, together with the bind-construct described in Section 1.3.2, types of the form $(TB)^A$ denoting T-exponentials, and terms of the form $\lambda x.M$ and (MN) denoting the corresponding lambda-operator and application. (These differ conceptually from the lambda and application operator of the computational lambda-calculus!) Let L_T be the computational lambda-calculus of K_T. Let $L(\Gamma, A)$ be the collection of sequents $(\Gamma \vdash M : A)$ of L, and let $L_T(\Gamma, A)$ is the collection of sequents $(\Gamma \vdash M : A)$ of L_T. The monadic-style transform $(-)^\sharp$ takes every sequent

$$(x_1 : A_1, \dots, x_n : A_n \vdash M : A) \in L_T$$

to a sequent

$$(x_1 : A_1^\sharp, \dots, x_n : A_n^\sharp \vdash M^\sharp : T(A^\sharp)) \in L$$

of L.

In the introduction, we only presented the monadic-style transform for the first-order fragment of the computational lambda-calculus. The complete version is presented in Figures 5.9 and 5.8. So for each environment Γ of L_T and each type A of L_T, we have a translation

$$(-)^\sharp : L_T(\Gamma, A) \longrightarrow L(\Gamma^\sharp, T(A^\sharp))$$

Before we state the main proposition of this section, note that the identity

$$K_T(\Gamma, A) = K(A, T(A^\sharp))$$

is the defining isomorphism of the adjunction $F_T \dashv G_T : K_T \longrightarrow K$. In agreement with standard categorical notation, we shall call this isomorphism $(-)^\sharp$, where the overloading of $(-)^\sharp$ is intended.

Proposition 5.12. *Let (K, T) be a precartesian λ_C-model. Then the following diagram commutes:*

$$\begin{tikzcd} L_T(\Gamma, A) \arrow[r, "K_T[\![-]\!]"] \arrow[d, "(-)^\sharp"'] & K_T(\Gamma, A) \arrow[d, "(-)^\sharp"] \\ L(\Gamma^\sharp, T(A^\sharp)) \arrow[r, "K[\![-]\!]"] & K(\Gamma^\sharp, T(A^\sharp)) \end{tikzcd} \tag{5.7}$$

Proof. By induction over $(x_1 : A_1, \dots, x_n : A_n \vdash M : A)$. $\square$

$$
\begin{array}{rcl}
(A * B)^\sharp & = & A^\sharp * B^\sharp \\
unit^\sharp & = & unit \\
(A \rightharpoonup B)^\sharp & = & \left(T\left(B^\sharp\right)\right)^{\left(A^\sharp\right)} \\
(TA)^\sharp & = & T\left(A^\sharp\right)
\end{array}
$$

Figure 5.8: Monadic-style transform: types

$$
\begin{array}{rcl}
x^\sharp & = & \eta(x) \\
(let\, x = M\ in\ N)^\sharp & = & \left(bind\, x \Leftarrow M^\sharp\ in\ N^\sharp\right) \\
()^\sharp & = & \eta() \\
(M, N)^\sharp & = & \left(bind\, x \Leftarrow M^\sharp\ in\ bind\, y \Leftarrow N^\sharp\ in\ \eta(x, y)\right) \\
(\pi_i(M))^\sharp & = & \left(bind\, x \Leftarrow M^\sharp\ in\, \eta(\pi_i(x))\right) \\
(f(M_1, \ldots, M_n))^\sharp & = & \left(bind\, x_1 \Leftarrow M_1^\sharp\ in\ \ldots bind\, x_n \Leftarrow M_n^\sharp\ in\ f(x_1, \ldots, x_n)\right) \\
[M]^\sharp & = & \eta\left(M^\sharp\right) \\
(\mu(M))^\sharp & = & \left(bind\, x \Leftarrow M^\sharp\ in\ x\right) \\
(\lambda x : A.M)^\sharp & = & \left[\lambda x : A^\sharp. \left(M^\sharp\right)\right] \\
(MN)^\sharp & = & \left(bind\, f \Leftarrow M^\sharp\ in\ bind\, y \Leftarrow N^\sharp\ in\ (fy)\right)
\end{array}
$$

Figure 5.9: Monadic-style transform: terms

So it does not matter whether we first apply the precartesian semantics and then the adjunction isomorphism, or we first apply the monadic-style transform and then the precartesian semantics. In other words, the monadic-style transform is the syntactic realisation of the adjunction isomorphism.

Proposition 5.13. *Moggi's semantics of the computational lambda-calculus in a precartesian λ_C-model (as presented in Figures 5.4, 5.3, and 5.2) agrees with the diagonal of Diagram 5.7.*

So Moggi's semantics gives meanings to expressions of the source language of the monadic-style transform using the model of the target language. In a sense, it performs a compilation on the fly.

Chapter 6

Example: Adding global state

In this chapter, we shall discuss a special way of constructing a new precartesian category from a given precartesian category K: Given an object S of K, we define a new category K_S such that $K_S(A, B) = K(A \times S, B \times S)$ and K_S inherits the composition from K. The idea is that S represents a universe of global states, and a morphism $f \in K_S(A, B)$ takes a value of type A, and depending on the state, produces an output an goes into a new state.

With this simple example, we demonstrate a general method of analysing implementations of new language features. This method consists of characterising the central, copyable, and discardable morphisms (or programs) of the new system in terms the original system. (It is crucial that the original tensor need not be a cartesian product. Thus we avoid the restriction that the original language has to be call-by-name or free of computational effects.)

For global state, we shall describe the central, copyable, and discardable morphisms in terms of reading and writing to the store. (For exceptions, they can be explained in terms of raising, for continuations in terms of jumping, and so on.) Although the statements of these results are simple, some of the proofs are almost mind-boggling. It is fascinating, but also sobering, that even in a case as simple as global state, some straightforward questions are so hard to answer.

The construction $K \mapsto K_S$ is related with the well-known 'side-effects' monad $TX = (X \times S)^S$ on a cartesian-closed category K: Because T is strong, (K, T) forms a precartesian computational model. So by Theorem 5.2, K_T forms a precartesian category. We have $K_T(A, B) = K(A, (B \times S)^S) \cong K(A \times S, B \times S) = K_S(A, B)$. One can check that this forms a strict precartesian functor $K_T \longrightarrow K_S$ which is an isomorphism. However, T requires K to have exponentials, which corresponds to requiring that the original language has a lambda operator. By contrast, this is unnecessary for the construction $K \mapsto K_S$.

Moreover, the construction $K \mapsto K_S$ is a special case of the construction of the

Kleisli category of a *dyad* on K. Dyads where introduced by Fokkinga [Fok94] as a "least common generalisation of monads and comonads", and generalised by Power and Robinson [PR99]. Dyads allow a substantial analysis of the combinability of computational effects. (For more on dyads, see Section 9.2.4.) By contrast, this chapter deals with global state only, and none of our results except Proposition 6.1 follow from the general theory of dyads.

6.1 Constructing the new system

In this section, we shall construct the precartesian category K_S from a precartesian category K and an object S of K. We shall also express this construction in terms of the let-calculus.

Proposition 6.1. *Let $K = (K, \times, I, \delta, p, q, !)$ be a precartesian category, and let S be an object of K. Let K_S be the category with the same objects as K that has $K_S(A, B) = K(A \times S, B \times S)$ and inherits composition and identities from K. For objects A, B, C, and a morphism $f : A \longrightarrow B$, let*

$$\begin{aligned}
&A \otimes B = A \times B \\
&C \otimes f = \Big((C \times A) \times S \cong C \times (A \times S) \xrightarrow{C\times f} C \times (B \times S) \cong (C \times B) \times S\Big) \\
&f \otimes C = \Big((A \times C) \times S \cong (A \times S) \times C \xrightarrow{f\times C} (B \times S) \times C \cong (B \times C) \times S\Big)
\end{aligned}$$

Let $F_S : K \longrightarrow K_S$ be the identity-on-objects functor that sends a morphism f to $f \times S$. Then $(K_S, \otimes, 1, F_S\delta, F_Sp, F_Sq, F_S!)$ is a precartesian category. Moreover, F_S is a strict precartesian functor and sends central morphisms to central morphisms.

Proof. Using Condition 3 of Proposition 2.1. □

Next we shall express the construction of K_S in terms of the internal languages (i.e. the let-calculi) of K_S and K. Let s and s_i be variables of type S. Figure 6.1 presents a translation of sequents of the let-calculus of K_S into the let-calculus of K (omitting environments and types). Up to syntax, the target language of this translation can be almost any programming language. In particular, it can be a call-by-value language with recursion and computational effects like jumps, exceptions, and so on. The notion of state on the source language is only limited by the capacity of the type S of the target language. If the state is kept small in size, the translation might be reasonable; It is certainly impossible if the state is the complete state of a realistic computer. At any rate, we are not trying to sell a compiler—we are analysing the scope of an established mathematical method.

$$
\begin{aligned}
x^s &= (x, s)\\
(let\, x = M\ in\ N)^s &= (let\, (x, s') = M^s\ in\ N^{s'})\\
()^s &= ((), s)\\
(M, N)^s &= (let\, (x_1, s_1) = M_1^s\ in\ let\, (x_2, s_2) = M_2^{s_1}\ in\ ((x_1, x_2), s_2))\\
\pi_i(M)^s &= (let\, (x, s') = M^s\ in\ (\pi_i(x), s'))\\
f(M_1, \ldots, M_n)^s &= (let\, (x_1, s_1) = M_1^s\ in\ let\, (x_2, s_2) = M_2^{s_1}\ in\\
&\qquad \ldots let\, (x_n, s_n) = M_n^{s_{n-1}}\ in\ f(x_1, \ldots, x_n, s_n))
\end{aligned}
$$

Figure 6.1: The state-passing style transform

Proposition 6.2. *Let K be a precartesian category, and let S be an object of K. Then for every sequent $(\Gamma \vdash M : A)$ of the let-language for K_S, it holds that*

$$K_S[\![\Gamma \vdash M : A]\!] = K[\![\Gamma, s : S \vdash M^s : A * S]\!]$$

Proof. By induction over $(\Gamma \vdash M : A)$. ∎

Letting L and L_S be the let-languages of K and K_S, respectively, Proposition 6.2 states that the following diagram commutes:

$$
\begin{tikzcd}
L_S(\Gamma \vdash A) \arrow[r, "K_S[\![-]\!]"] \arrow[d, "(-)^s"'] & K_S(\Gamma, A) \arrow[d, equal] \\
L(\Gamma, s : S \vdash A * S) \arrow[r, "K[\![-]\!]"] & K(\Gamma \times S, A \times S)
\end{tikzcd}
$$

So the language transform $(-)^s$ describes syntactically the construction of K_S from K.

6.2 The precartesian properties of the new system

In this section, we characterise *all* central morphisms, copyable morphisms, and discardable morphisms of K_S. We shall see that there is a fascinating connection between these three classes and the ways in which expressions can access the store.

6.2.1 Discardable

Characterising discardability in K_S is easier than characterising centrality and copyability, so we shall use it to warm up.

Proposition 6.3. *Let K be a precartesian category. A morphism $f \in K_S(A, B)$ is discardable in K_S if and only if in K it holds that*

$$f; q = q \tag{6.1}$$

Proof.

$$\begin{aligned} f; F! = F! &\Leftrightarrow f; ! \times S =! \times S \\ &\Leftrightarrow f; ! \times S; q =! \times S; q \qquad \text{because } q : 1 \times S \longrightarrow S \text{ is an iso} \\ &\Leftrightarrow f; q = q \end{aligned}$$

□

In the let-calculus for K, Equation 6.1 is

$$\pi_2(f(x, s)) \equiv s \tag{6.2}$$

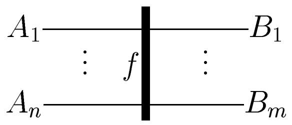
This implies that f does not write to the store. It also implies that $\pi_2(f(x, s))$ is focal in K. For example, f is must not raise an exception or diverge.

6.2.2 Copyable

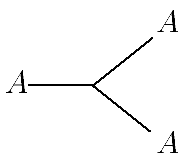
Next we characterise the copyable morphisms of K_S. Here it becomes obvious that it can be much easier to present a morphism

$$f : A_1 \times \cdots \times A_n \longrightarrow B_1 \times \cdots \times B_m$$

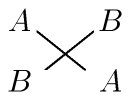
of K (not K_S) by a diagram



The composition of K is presented by the evident horizontal gluing. This is a simplification of Jeffrey's and Schweimeier's graph presentation described in Section 1.4, in that we indicate evaluation order in K by the left-to-right order of the building blocks, rather that using an extra thread that indicates the control flow. The diagonal $\delta : A \longrightarrow A \times A$ is presented by

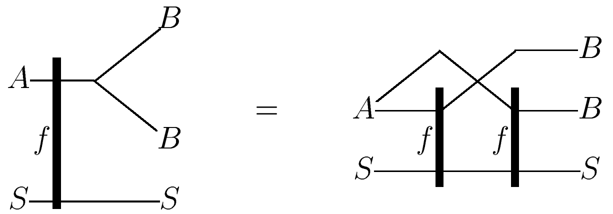


Projections are presented by 'wires' that do not reach the rightmost side of the diagram. The twist map $A \times B \longrightarrow B \times A$ is presented by



We shall not bother with a formal semantics of these diagrams, because intuition should be enough to reconstruct from them the standard categorical notation or the let-calculus notation.

Proposition 6.4. *Let K be a precartesian category. A morphism $f \in K_S(A, B)$ is copyable if and only if in K it holds that*



(6.3)

Proof. By expressing the K_S-equation $f; F\delta = F\delta; A\otimes f; f\otimes B$ with the operators of K. ∎

Note that, for $A = B = I$, Equation 6.3 states that f is an idempotent on S in K. In the let-calculus for K, Equation 6.3 is

$$\begin{aligned} &(\mathit{let}\ (y, s') = f(x, s)\ \mathit{in}\ (y, (y, s'))) \\ &\equiv (\mathit{let}\ (y, s') = f(x, s)\ \mathit{in}\ (y, f(x, s'))) \end{aligned} \tag{6.4}$$

Obviously, copyability is considerably more complicated than discardability. The following proposition helps clarifying the situation:

Proposition 6.5. *Let K be a precartesian category, and let $f \in K_S(A, B)$ be copyable in K. If f is discardable in K_S, then it is copyable in K_S.*

Proof. First observe that

$$\begin{aligned} f(x, s) &\equiv (\mathit{let}\ y = f(x, s)\ \mathit{in}\ y) \\ &\equiv (\mathit{let}\ y = f(x, s)\ \mathit{in}\ (\pi_1(y), \pi_2(y))) \\ &= (\pi_1(f(x, s)), \pi_2(f(x, s))) && \text{because } f \text{ is copyable in } K \\ &\equiv (\pi_1(f(x, s)), s) && \text{by Proposition 6.3} \end{aligned}$$

Now we check Equation 6.4:

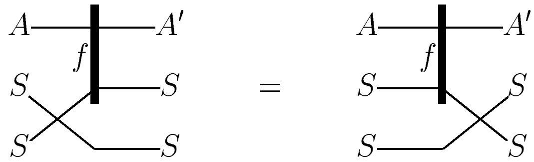
$$\begin{aligned} &(\mathit{let}\ (y, s') = f(x, s)\ \mathit{in}\ (y, (y, s'))) \\ &\equiv (\mathit{let}\ (y, s') = (\pi_1(f(x, s)), s)\ \mathit{in}\ (y, (y, s'))) \end{aligned}$$

$$
\begin{aligned}
&\equiv (\mathit{let}\, y = \pi_1(f(x,s))\ \mathit{in}\ (y,(y,s)))\\
&\equiv (\pi_1(f(x,s)),(\pi_1(f(x,s)),s)) \qquad \text{(because } \pi_1(f(x,s)) \text{ is copyable)}\\
&\equiv (\mathit{let}\, y = \pi_1(f(x,s))\ \mathit{in}\ \mathit{let}\, y' = \pi_1(f(x,s))\ \mathit{in}\ (y,(y',s)))\\
&\equiv (\mathit{let}\,(y,s') = (\pi_1(f(x,s)),s)\ \mathit{in}\\
&\quad \mathit{let}\,(y',s'') = (\pi_1(f(x,s')),s')\ \mathit{in}\ (y,(y',s'')))\\
&\equiv (\mathit{let}\,(y,s') = f(x,s)\ \mathit{in}\ \mathit{let}\,(y',s'') = f(x,s')\ \mathit{in}\ (y,(y',s'')))\\
&\equiv (\mathit{let}\,(y,s') = f(x,s)\ \mathit{in}\ (y,f(x,s')))
\end{aligned}
$$
□

Corollary 6.6. *If K is a category with finite products, then in K_S all discardable morphisms are copyable.*

6.2.3 Central

Proposition 6.7. *Let K be a precartesian category. Then a morphism $f \in K_S(A, A')$ is central in K_S if and only if f is central in K and it holds in K that*

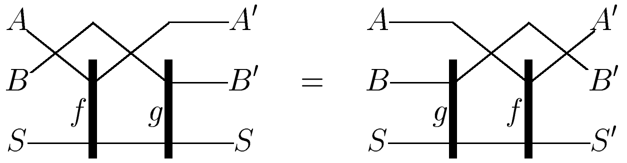


(6.5)

Remark 6.1. In the let-calculus for K, Equation 6.5 is

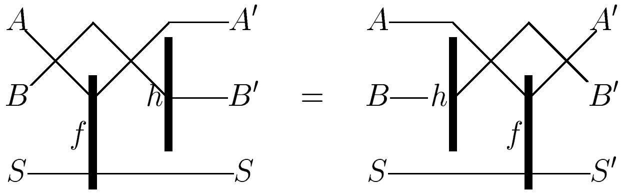
$$(f(x,s'),s) \equiv (\mathit{let}\,(y,s'') = f(x,s)\ \mathit{in}\ ((y,s'),s'')) \qquad (6.6)$$

Proof. f is central in K_S if and only if for all $g \in K_S(B, B')$ it holds in K that

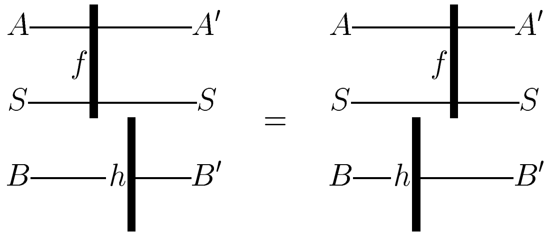


(6.7)

For the 'only if', suppose that f is central in K_S. To see that f is central in K, and let $h \in K(B, B')$. Letting $g = h \times S$ in Equation 6.7, we get



This holds if and only if



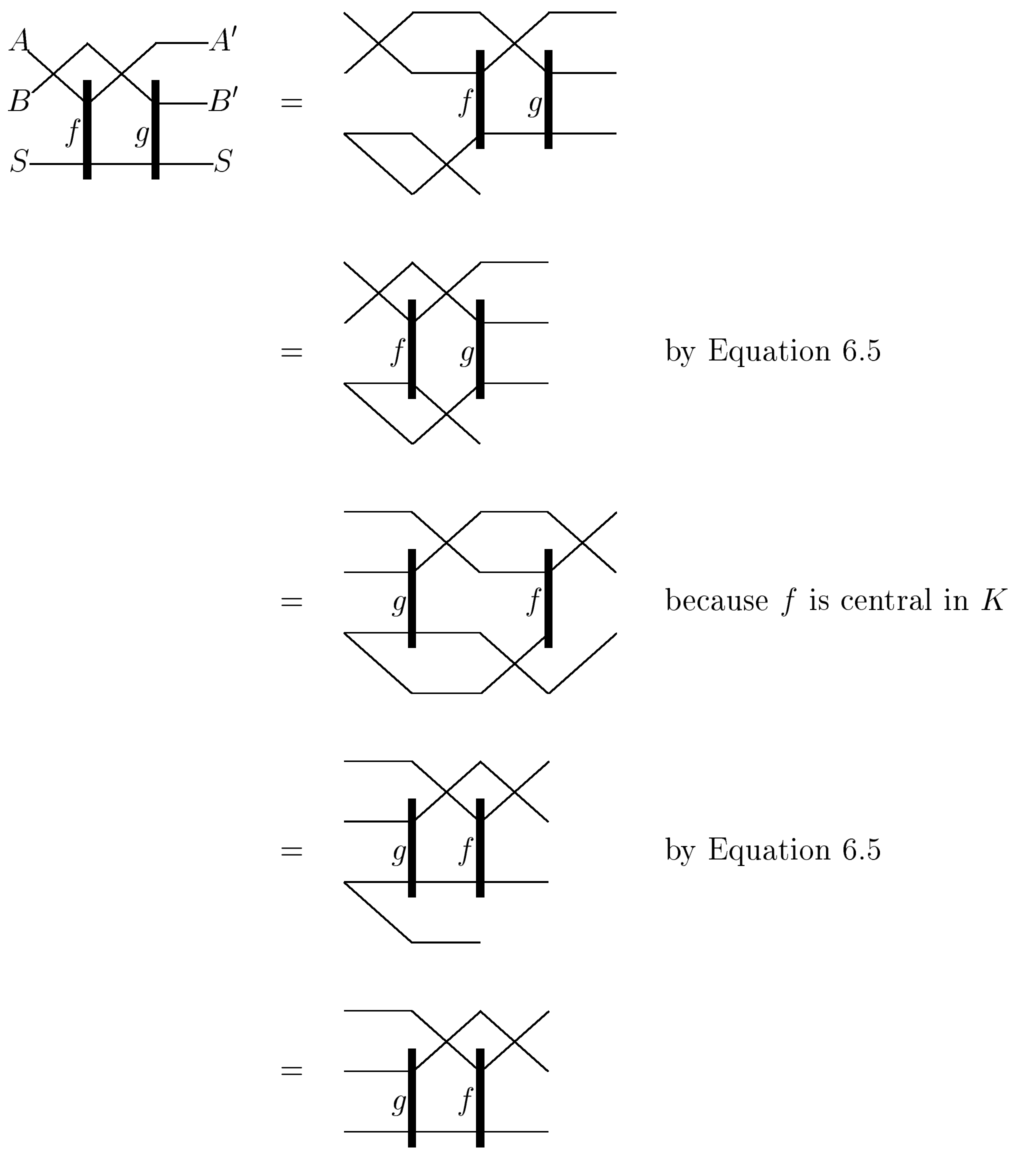


Figure 6.2: Proof of the 'if' of Proposition 6.7

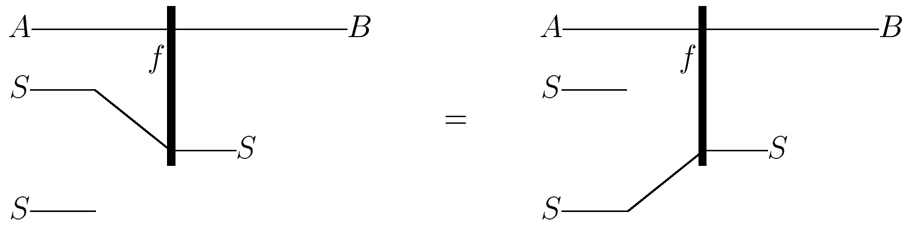
for all $h : B \longrightarrow B'$, which means that f is central in K. To get Equation 6.5, let $g = \tau_{S,S}$ in Equation 6.7. Thus we get

Equation 6.5 follows from the naturality of τ. For the 'if', let f be central in K, let $g \in K_S(B, B')$ and consider Figure 6.2. $\square$

Next we shall explain what centrality has to do with reading the store and writing to the store. Let $f \in K_S(A, B)$. Intuitively, the value of f does not depend on the store if in K it holds that

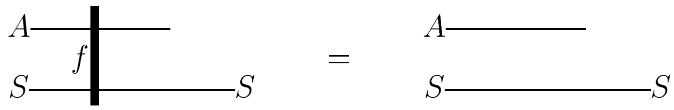
$$x : A, s : S, s' : S \vdash \pi_1(f(x, s)) \equiv \pi_1(f(x, s')) : B$$

As a diagram, this is

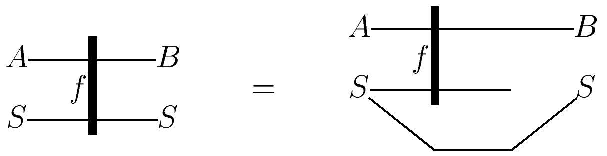


(6.8)

With Equation 6.2, which characterises discardability in K_S, we had found a condition that seems to imply that a morphism 'does not write to the store'. The diagram version of Equation 6.2 is



However, there is another Equation seeming to imply that f does not write:



(6.9)

In fact Equation 6.9 seems to be a good description of what it means for f not to write, because here f is neither copied nor discarded (by contrast to Equation 6.2, where f is discarded, which means that Equation 6.2 is about more than just writing). However, it is straightforward to check that Equations 6.2 and 6.9 are equivalent if the precartesian structure of K is a finite-product structure.

Proposition 6.8. *Let K be a precartesian category, and let $f \in K_S(A, A')$. Then Equation 6.5 holds if and only if Equations 6.8 and 6.9 hold.*

Proof. The proof for the 'if' is presented in Figure 6.3. The proof for the 'only if' is presented in Figure 6.4. ☐

6.2.4 Summary

The special case where K is a category with finite products is summarised in the following proposition:

Proposition 6.9. *Let K be a category with finite products. A morphism $f \in K_S(A, B)$ is*

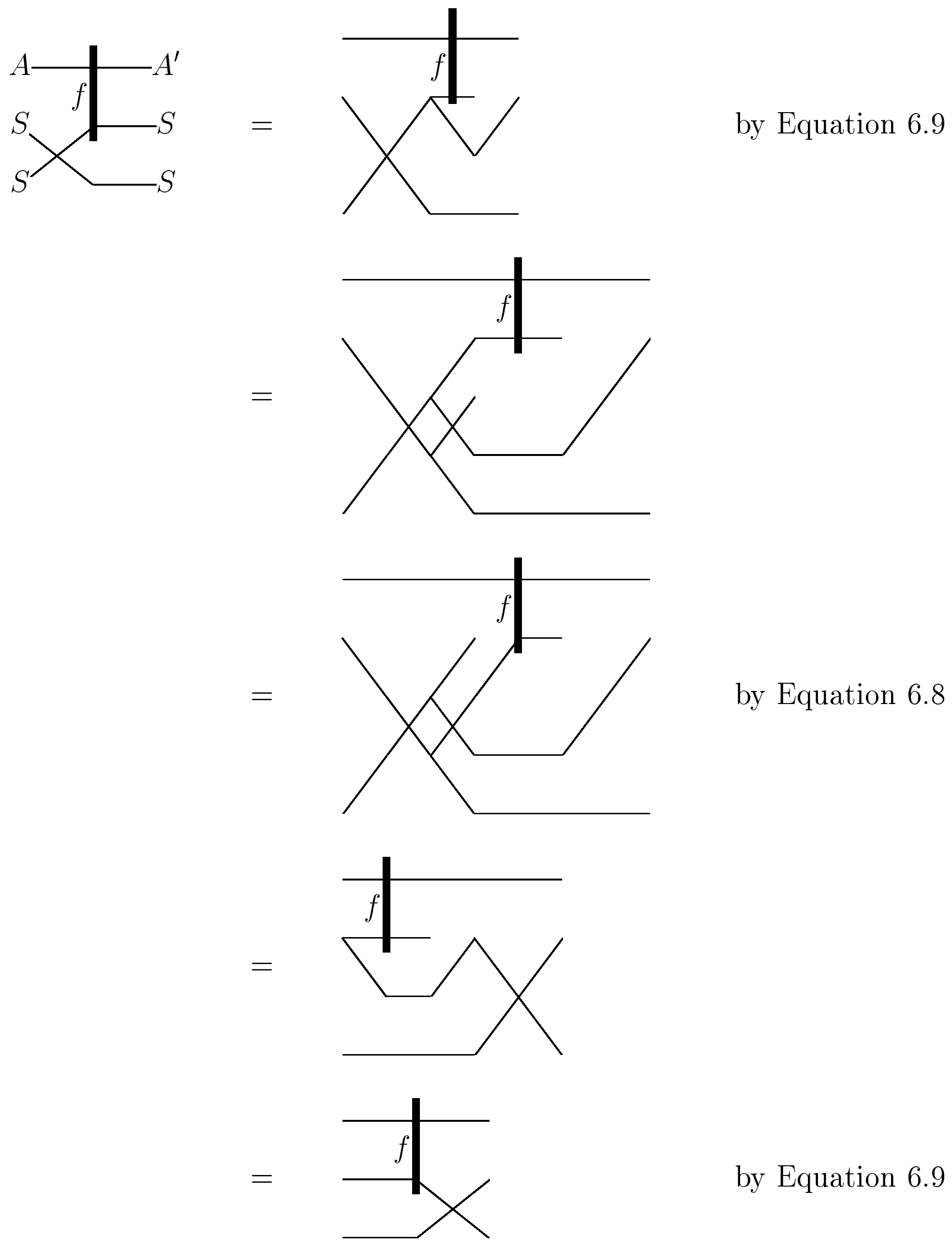


Figure 6.3: Proof for the ‘if’ of Proposition 6.8

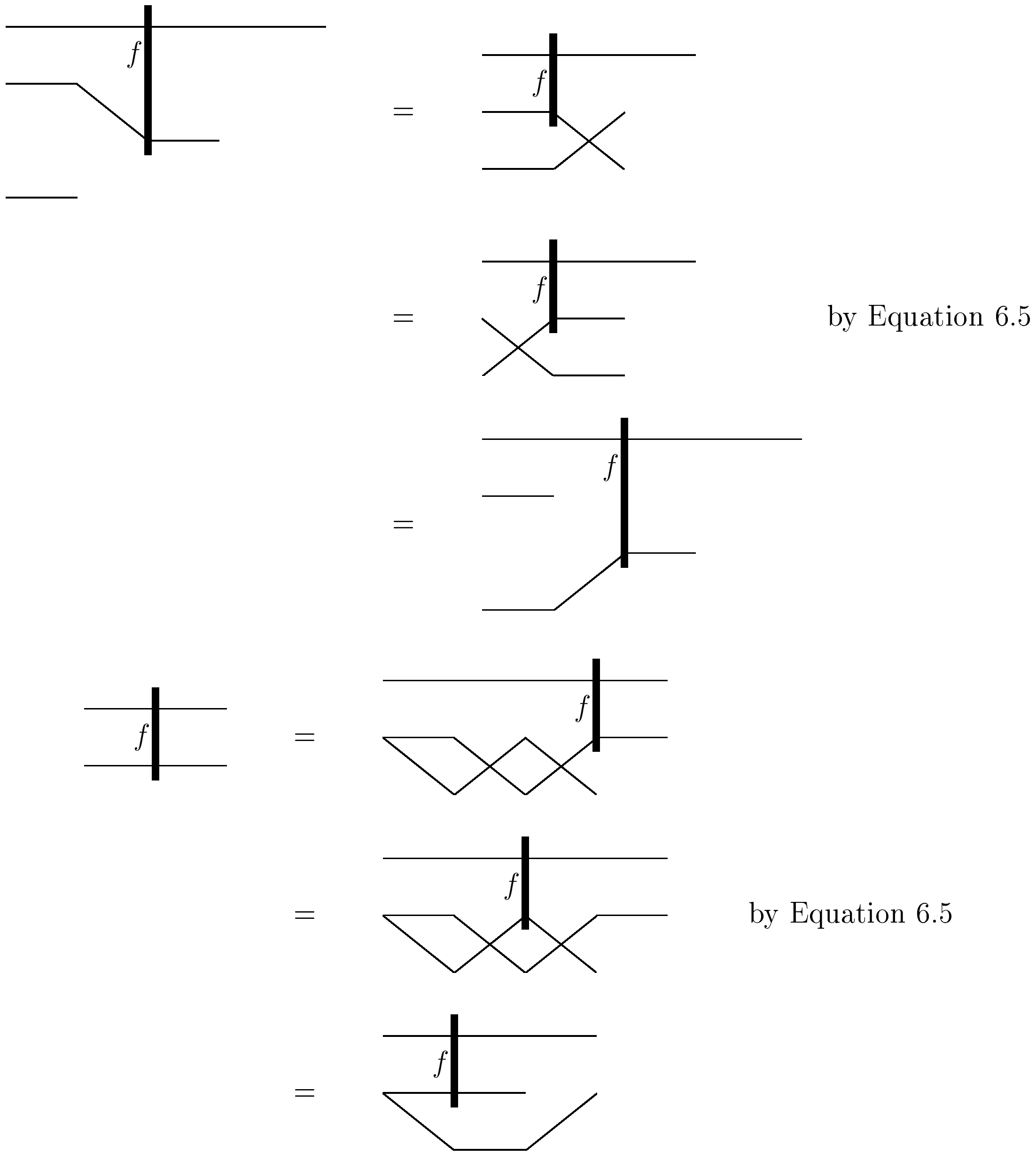


Figure 6.4: Proof of the ‘only if’ of Proposition 6.8

- *central in K_S if and only if it is discardable in K_S and Equation 6.8 holds*
- *copyable in K_S if and only if in K it holds that*

$$x : A, s : S \vdash f(x,s) \equiv f(x, \pi_2(f(x,s))) : B * S$$

(in particular, if f is discardable in K_S)

Proof. The claim for 'central' follows from Proposition 6.7 (using that every morphism of K is central) and Proposition 6.8 (using that, K has finite products, Equation 6.9 holds if and only if f is discardable in K_S). Now for the claim about 'copyable'. The left side of Equation 6.4 is equal to

$$\begin{aligned}
&(\mathit{let}\,(y,s') = f(x,s)\;\mathit{in}\,(y,(y,s')))\\
&\equiv (\mathit{let}\,(y,s') = (\pi_1(f(x,s)),\pi_2(f(x,s)))\;\mathit{in}\,(y,(y,s')))\\
&\quad\text{(because } f \text{ is copyable in } K)\\
&\equiv (\mathit{let}\,y = \pi_1(f(x,s))\;\mathit{in}\;\mathit{let}\,s' = \pi_2(f(x,s))\;\mathit{in}\,(y,(y,s')))\\
&\equiv (\mathit{let}\,y = \pi_1(f(x,s))\;\mathit{in}\,(y,(y,\pi_2(f(x,s)))))\\
&\equiv (\pi_1(f(x,s)),(\pi_1(f(x,s)),\pi_2(f(x,s))))\\
&\quad\text{(because } \pi_1(f(x,s)) \text{ is copyable in } K)\\
&\equiv (\pi_1(f(x,s)), f(x,s)) \qquad \text{(because } f \text{ is copyable in } K)
\end{aligned}$$

The right side of Equation 6.4 is equal to

$$\begin{aligned}
&(\mathit{let}\,(y,s') = f(x,s)\;\mathit{in}\,(y,f(x,s')))\\
&\equiv (\mathit{let}\,(y,s') = (\pi_1(f(x,s)),\pi_2(f(x,s)))\;\mathit{in}\,(y,f(x,s')))\\
&\quad\text{(because } f \text{ is copyable in } K)\\
&\equiv (\pi_1(f(x,s)), f(x,\pi_2(f(x,s))))
\end{aligned}$$

So if $f(x,s) \equiv f(x,\pi_2(f(x,s)))$ then Equation 6.4 holds. The converse follows from sending the Equation 6.4 through $\pi_2(-)$ (it is easiest to do this with the diagram version, Equation 6.3). □

The situation in Proposition 6.9 is summarized in Figure 6.5. In fact, all intersections of areas in Figure 6.5 are inhabited—examples can be found for $K = \mathit{Set}$. This is left as an entertaining exercise.

Open proplem 6.1. Are the 'no-read' morphisms closed under composition and tensor? If so, we could add a property *no-read* to the let-calculus and infer that property like *central* or *discardable*. How could we exploit *no-read* with respect to substitution—that his, how would we have to extend Rule 3.1?

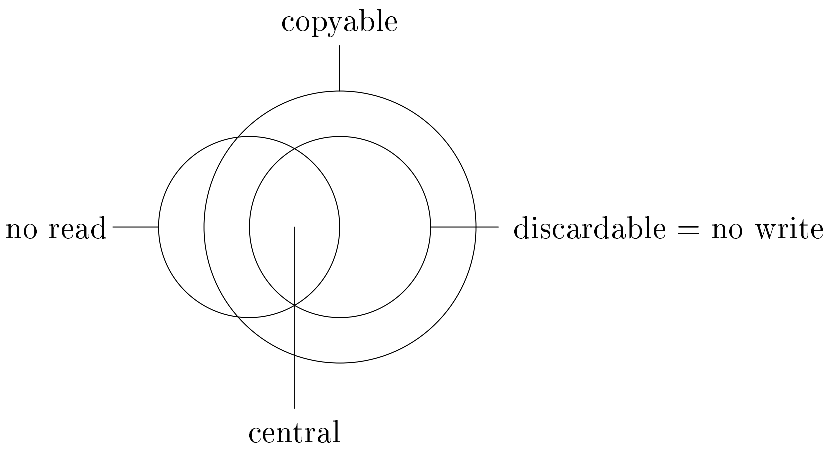


Figure 6.5: Global state over a category with finite products

6.3 A special embedding

We shall conclude this chapter with an important observation. Construing the expressions that do not write to the store (in the sense of 'discardable') as a sublanguage is a natural thing to do. Mathematically, this corresponds to considering the embedding from the subcategory $(K_S)_!$ of discardable morphisms of K_S into K_S. By Proposition 3.1, the discardable morphisms form a precartesian category. Trivially, the embedding is a strict precartesian functor. In this section, we show that the embedding does not preserve the central morphisms. This contributes to justifying the generality of our definition of strong precartesian functors. First, an observation about $(K_S)_!$:

Proposition 6.10. *If K is a category with finite products and S is an object of K_S, then $(K_S)_!$ has finite products.*

Proof. Proposition 2.1 makes clear that it suffices to prove that all morphisms of $(K_S)_!$ are central and copyable in $(K_S)_!$. This is intuitively clear: Expressions that only read should commute with each other with respect to the evaluation order, and it should not matter if reading happens once or twice. To see that all morphisms of $(K_S)_!$ are central in $(K_S)_!$, we prove that for all $f \in (K_S)_!(A, A')$ and $g \in (K_S)_!(B, B')$ it holds in K_S that $f \otimes B; A' \otimes g = A \otimes g; f \otimes B'$. In K, that equation is Diagram 6.7. In the let-language of K, let $f_i(x, s) = \pi_i(f(x, s))$ and $g_i(y, s) \equiv \pi_i(g(y, s))$. Using that K has finite products, Diagram 6.7 states

$$(f_1(x, s), g_1(y, f_2(x, s)), g_2(y, f_2(x, s))) \equiv (f_1(x, g_2(y, s)), g_1(y, s), f_2(x, g_2(y, s)))$$

This equation holds because by Proposition 6.3 we have $f_2(x, s') \equiv s'$ and $g_2(y, s') \equiv s'$. Next we prove that every morphism of $(K_S)_!$ is copyable in $(K_S)_!$. Let $f \in (K_S)_!(A, A')$. Using that K has finite products, by Proposition 6.9 f is copyable if

$$(f_1(x, s), f_2(x, s)) \equiv (f_1(x, f_2(x, s)), f_2(x, f_2(x, s)))$$

This equation holds because by Proposition 6.3 we have $f_2(a, s) \equiv s$. $\square$

Proposition 6.11. *If S is a set with at least two elements,then the embedding $(Set_S)_! \hookrightarrow Set_S$ does not preserve central morphisms.*

Proof. Let *read* $\in K_S(1, S)$ and *write* $\in K_S(S, 1)$ be defined by

$$\begin{aligned} read((), s) &= (s, s) \\ write(s, s') &= ((), s) \end{aligned}$$

By Proposition 6.3, *read* is discardable in K_S, so by Proposition 6.10 it is central in $(K_S)_!$. Obviously, *read* does no commute with *write*, and therefore *read* is not central in K_S. $\square$

Chapter 7

Soundness and completeness of the computational lambda-calculus

As described in Chapter 5, every λ_C-model (K,T) induces a computational abstract Kleisli-category K_T. By Proposition 5.13, Moggi's semantics of the computational lambda-calculus in (K,T) agrees with the direct semantics in K_T. The computational lambda-calculus is known to be sound and complete for λ_C-models (see [Mog88]). So it is complete for computational abstract Kleisli-categories too, because if a judgement holds in all computational abstract Kleisli-categories, then in particular it holds in those that arise from λ_C-models, and therefore it is derivable.

In Section 7.1, we shall prove that the computational lambda-calculus is also sound for computational abstract Kleisli-categories. To obtain soundness, we shall prove that every computational abstract Kleisli-category is isomorphic to the computational abstract Kleisli-category K_T that arises from a λ_C-model (K,T). So the deduction rules of the computational lambda-calculus hold in every computational abstract Kleisli-category K, because they hold in the λ_C-model of which K is the Kleisli category.

Section 7.1 contains the proof the soundness result. The remaining two sections deal with issues resulting from Section 7.1: In Section 7.2 we shall compare the computational lambda-calculus with the let-calculus, and in Section 7.3 we shall discuss the 'equalizing requirement' for monads.

7.1 Monadic representation of abstract Kleisli-categories

In this section, we shall extract a λ_C-model (C, T) from an arbitrary computational abstract Kleisli category K and prove that K is isomorphic to the computational abstract Kleisli-category C_T. As we shall see, we can define (C, T) for every abstract Kleisli category K—we need the precartesian structure on K only if we want finite products on C and a strength on T. Apart from its significance for semantics, this extraction of (C, T) from an abstract Kleisli category K seems to be an interesting piece of category theory.

Proposition 7.1. *If $K = (K, L, \vartheta, \varepsilon)$ is an abstract Kleisli-category, then there is a monad on K_ϑ such that the endofunctor $K_\vartheta \longrightarrow K_\vartheta$ is given by L, the unit is ϑ, and the multiplication is $L\varepsilon$.*

Proof. Because ϑL is a natural transformation $L \longrightarrow L^2$, all morphisms in the image of L are thunkable. Therefore, there is an endofunctor on K_ϑ that agrees with L. Let's overload L to stand for this endofunctor too. By definition of K_ϑ, ϑ forms a natural transformation $Id_{K_\vartheta} \stackrel{\cdot}{\longrightarrow} L$. The associativity law of the monad follows from the naturality of ε. The left neutrality law is the statement $\vartheta; L\varepsilon = id$, and the right neutrality law follows from sending the equation $\vartheta; \varepsilon = id$ through L. □

The goal is now to prove that the Kleisli category $(K_\vartheta)_L$ of the monad $(L, \vartheta, L\varepsilon)$ on K_ϑ is isomorphic to K. For proving soundness, this isomorphism must preserve all available structure—that is, L, ϑ, and ε. For this purpose, we shall now define *morphisms of abstract Kleisli categories*. Like for strong precartesian functors (see Section 2.5), we require morphisms of abstract Kleisli categories to preserve the structure only up to natural isomorphism.

Definition 7.1. A *morphism of abstract Kleisli-categories* from $K = (K, L, \vartheta, \varepsilon)$ to $K' = (K', L', \vartheta', \varepsilon')$ is defined as a functor $F : K \longrightarrow K'$ together with a natural isomorphism $F_1 : L'F \cong FL$ such that the following diagram commutes:

$$\begin{tikzcd}
L'F \arrow[r, "\varepsilon' F"] \arrow[dr, "F_1"] & F \\
F \arrow[u, "\vartheta' F"] \arrow[r, "F\vartheta"'] & FL \arrow[u, "F\varepsilon"']
\end{tikzcd}$$

F is called *strict* if F_1 is the identity.

Proposition 7.2. *Morphisms of abstract Kleisli-categories preserve thunkable morphisms. Faithful morphisms of abstract Kleisli-categories reflect thunkable morphisms.*

Proof. Easy, left to the reader. □

Theorem 7.3. *For every abstract Kleisli-category K there is a strict isomorphism $i_K : K \cong (K_\vartheta)_L$ of abstract Kleisli-categories.*

Proof. The isomorphism i_K is the identity-on-objects functor that sends a morphism f to $[f]$. Its inverse sends a morphism g to $g;\varepsilon$. □

Proposition 7.4. *If K is a precartesian abstract Kleisli-category such that every thunkable morphism is focal, then K_ϑ together with the monad $(L, \vartheta, L\varepsilon)$ forms a cartesian computational model with the strength*

$$t_{A,B} = [A \otimes \varepsilon_B] : A \otimes LB \longrightarrow L(A \otimes B)$$

Proposition 7.5. *For every precartesian abstract Kleisli-category K such that every thunkable morphism is focal, the isomorphism $i_K : K \cong (K_\vartheta)_L$ is a strict precartesian functor.*

Proposition 7.6. *If K is a computational abstract Kleisli-category, then the cartesian computational model (K_ϑ, L) forms a λ_C-model with*

$$\begin{aligned} (LB)^A &= (A \rightharpoonup B) \\ (ev \in K_\vartheta((LB)^A \otimes A, LB)) &= [apply \in K((A \rightharpoonup B) \otimes A, B)] \\ \lambda(f \in K_\vartheta(A \otimes B, LC)) &= \Lambda(f;\varepsilon) \end{aligned}$$

Proposition 7.7. *For every computational abstract Kleisli-category K, the isomorphism $i_K : K \cong (K_\vartheta)_L$ preserves Λ and apply on the nose.*

Theorem 7.8. *The computational lambda-calculus is sound and complete for computational abstract Kleisli categories.*

7.2 Conservative extension?

The computational lambda-calculus has all types and terms of the let-language (plus higher order types—that is TA and $A \rightharpoonup B$, and higher-order terms—that is $\lambda x.M$, (MN), $\mu(M)$, and $[M]$). In particular, every well-formed equation judgement $(\Gamma \vdash M \equiv N : A)$ of the let-language is well-formed in the computational lambda-calculus too. Therefore, we can compare the derivability relations for equation judgements of the two calculi. Formally, given a precartesian signature Σ, let's define a derivability relation $\vdash_{let,\Sigma}$ such that for well-formed equation judgements $X_1, \ldots, X_n$ and Y over Σ it holds that

$$\{X_1, \ldots, X_n\} \vdash_{let,\Sigma} Y$$

if Y is derivable from $X_1, \ldots, X_n$ in the let-calculus, and

$$\{X_1, \ldots, X_n\} \vdash_{\lambda_C,\Sigma} Y$$

if Y is derivable from $X_1, \ldots, X_n$ in the computational lambda-calculus.

Proposition 7.9. *The computational lambda-calculus is an extension of the let-calculus in that for every precartesian signature Σ it holds that*

$$\vdash_{let,\Sigma} \quad \subseteq \quad \vdash_{\lambda_C,\Sigma}$$

Proof. Suppose that $\{X_1, \ldots, X_n\} \vdash_{let,\Sigma} Y$. Because the let-calculus is sound for precartesian categories, Y holds in every precartesian model of $\{X_1, \ldots, X_n\}$. In particular, Y holds in every precartesian model of $\{X_1, \ldots, X_n\}$ which is given by a computational abstract Kleisli-category. By completeness of the computational lambda-calculus, it holds that $\{X_1, \ldots, X_n\} \vdash_{\lambda_C,\Sigma} Y$. □

Is this extension conservative—that is, does $\vdash_{let,\Sigma}$ agree with the the restriction of $\vdash_{\lambda_C,\Sigma}$ to first-order expressions (i.e. expressions of the let-language over Σ)? We shall now reduce that question into a semantic one.

Proposition 7.10. *If every precartesian model can be embedded into a computational abstract Kleisli-category by a strong precartesian functor F with focal F_2 and F_0, then the extension $\vdash_{let,\Sigma} \subseteq \vdash_{\lambda_C,\Sigma}$ of calculi is conservative.*

For the proof of Proposition 7.10 we shall use the following lemma:

Lemma 7.1. *Let K be a precartesian interpretation of a precartesian signature Σ such that the precartesian category K can be embedded into a computational abstract Kleisli-category K' by a strong precartesian functor F with focal F_2 and F_0. Let FK be the evident interpretation of Σ that results from applying F after the interpretation K. Then the extension $(FK)[\![-]\!]$ of FK to the let-language over Σ is isomorphic to the map $F(K[\![-]\!])$ in the following sense: For each precartesian type A over Σ, there is a focal isomorphism*

$$\varphi_A : (FK)[\![A]\!] \cong F(K[\![A]\!])$$

which is natural in that for every sequent $(\Gamma \vdash M : A)$ *of the let-language over* Σ *it holds that*

$$\begin{tikzcd}
(FK)[\![\Gamma]\!] \arrow[r, "\varphi_\Gamma"] \arrow[d, "(FK)[\![M]\!]"'] & F(K[\![\Gamma]\!]) \arrow[d, "F(K[\![M]\!])"] \\
(FK)[\![A]\!] \arrow[r, "\varphi_A"] & F(K[\![A]\!])
\end{tikzcd}$$

Proof. We define φ inductively by

$$\begin{aligned}
\varphi_A &= id_{F(K[\![A]\!])} \quad \text{if } A \text{ is a base type} \\
\varphi_{unit} &= F_0 : I \longrightarrow FI \\
\varphi_{A*B} &= (FK)[\![A]\!] \otimes (FK)[\![B]\!] \xrightarrow{\varphi_A \otimes \varphi_B} F(K[\![A]\!]) \otimes F(K[\![B]\!]) \xrightarrow{F_2} F(K[\![A]\!] \otimes K[\![B]\!])
\end{aligned}$$

Because F_2 and F_2 are focal isomorphisms, so is every component of φ. The naturality of φ follows from induction over $(\Gamma \vdash M : A)$, using the focality of φ. □

Proof of Proposition 7.10. Suppose that for equation judgements $X_1, \ldots X_n$ and Y between first-order expression over Σ it holds that $X_1, \ldots, X_n \vdash_{\lambda_C, \Sigma} Y$. Now let K be a precartesian model of $X_1, \ldots, X_n$, let K' be a computational abstract Kleisli-category, and let $F : K \longrightarrow K'$ be a faithful strong precartesian functor with focal F_2 and F_0. Because of the natural iso φ, the interpretation FK is a model of $X_1, \ldots, X_n$. Because the denotational category K' of the interpretation FK is a computational abstract Kleisli-category, Y holds under FK (by soundness of the computational lambda-calculus). Because of the natural iso φ, and because F is faithful, Y holds in K too. So we have proved that Y holds in every precartesian model of $X_1, \ldots, X_n$. By completeness of the let-calculus for precartesian categories, we have $X_1, \ldots, X_n \vdash_{let, \Sigma} Y$. □

In summary, we have reduced the conservativity problem for the extension $\vdash_{let,\Sigma} \subseteq \vdash_{\lambda_C,\Sigma}$ to the following open problem:

Open proplem 7.1. Can every precartesian model be embedded into a computational abstract Kleisli-category by a strong precartesian functor F with focal F_2 and F_0?

Perhaps this question can be answered positively with the help of Kan extensions—trying this is left as a future challenge.

Remark 7.1. For equations $X_1, \ldots, X_n$ and Y between first-order expressions, let

$$\{X_1, \ldots, X_n\} \vdash_{\lambda fo, \Sigma} Y$$

if Y is derivable from $X_1, \ldots, X_n$ in the computational lambda-calculus in such a way that the derivations uses only first-order expressions. Moggi raised the question whether the extension $\vdash_{\lambda fo,\Sigma} \subseteq \vdash_{\lambda_C,\Sigma}$ is conservative—in in his terminology (see [Mog91], top of p. 24):

> *We do not know... whether $\lambda_C PL$ is a conservative extension of PL.*

Me neither. However, it should hold that

$$\vdash_{\lambda fo,\Sigma} \quad = \quad \vdash_{let,\Sigma}$$

(This should be a rouitine proof, but admittedly, I have not checked.) If so, then Moggi's question and our question about the conservativity of the extension $\vdash_{let,\Sigma} \subseteq \vdash_{\lambda_C,\Sigma}$ are equivalent.

7.3 The equalizing requirement

I would like to conclude this chapter with a little mathematical observation. In some of his articles (e.g. [Mog91]), Moggi discusses the following two properties:

Definition 7.2. A monad $T = (T, \eta, \mu)$ is said to satisfy the *mono requirement* if every component of η is a mono. T is said to satisfy the *equalizing requirement* if, for each object A, η_A is an equalizer of η_{TA} and $T\eta_A$.

Because all equalizers are monos, every monad T in C that satisfies the equalizing requirement satisfies the mono requirement. The converse holds if η_A is regular for every object A—that is, η_A is an equaliser of *some* pair of monos (see Lemma 6 on page 110 of [BW85]). All monos of a topos, for example, are regular.

Abstract Kleisli categories make clear how the equalizing requirement enters semantics:

Theorem 7.11. *If K is an abstract Kleisli category, then the monad $(L, \vartheta, L\varepsilon)$ on K_ϑ satisfies the equalizer requirement. In particular, every abstract Kleisli category is isomorphic to one $(K_\vartheta)_L$ that arises from a monad that satisfies the equalizer requirement.*

Proof. We prove that in K_ϑ the morphism $\vartheta : A \longrightarrow LA$ is an equalizer of $LA \underset{L\vartheta_A}{\overset{\vartheta_{LA}}{\rightrightarrows}} L^2A$. Clearly, it holds that $\vartheta_A; \vartheta_{LA} = \vartheta_A; L\vartheta_A$. Now let $f \in K_\vartheta(B, LA)$ such that

$$\begin{tikzcd} B \arrow[r, "f"] \arrow[d, "f"'] & LA \arrow[d, "\vartheta_{LA}"] \\ LA \arrow[r, "L\vartheta_A"] & L^2A \end{tikzcd}$$

commutes. It remains to find a unique $g \in K_\vartheta(B, A)$ such that $g; \vartheta_A = f$. Appending ε_A to both sides of the equation makes clear that we must have $g = f; \varepsilon_A$. And g is indeed a solution, because $g; \vartheta_A = f; \varepsilon_A; \vartheta_A = f; L\vartheta_A; \varepsilon_{LA} = f; \vartheta_{LA}; \varepsilon_{LA} = f$. □

Moggi proved (Proposition 2.9 of [Mog91]) that the monad in the syntactically defined category $\mathcal{F}(\mathcal{T})$ (see Remark 5.4) satisfies the mono requirement. Because $\mathcal{F}(\mathcal{T})$ can be smaller that the category of thunkable morphisms (as explained in Remark 5.4), it was not possible for Moggi to prove the equalizing requirement.

Chapter 8

Example: Continuations

A good introduction to continuations is describing how they help transforming a typical programming language into a simpler one. Hayo Thielecke reviewed this nicely in [Thi99a]:

> *...a function call is transformed into a jump with arguments to the callee, such that one of the arguments is the return address, i.e., a continuation that enables the callee to jump back to the caller. To match this, all function definitions need to be transformed to take the return address as an extra argument. The systematic addition of return addresses is described in an early paper by van Wijngaarden [vW64], which Reynolds [Rey93] credits with the earliest use of continuation-passing style. Van Wijngaarden (cited in [Rey93]) describes the introduction of continuation parameters thus:*
>
> *'Provide each procedure declaration with an extra formal parameter—specified* `label`*—and insert at the end of its body a* `goto` *statement leading to that formal parameter. Correspondingly, label the statement following a procedure statement, if not labeled already, and provide that label as the corresponding extra actual parameter.'*
>
> *In modern terminology, this additional formal parameter is called a continuation, and the transform that introduces these continuation parameters is called a continuation-passing style (CPS) transform [Ste78].*

In this chapter, we study the CPS transform denotationally by construing the source language as a precartesian category.

8.1 A CPS-transform

Importantly, the target language of a CPS transform can be simpler than the original language in that its execution does not need a call stack. Thielecke describes his version of the target language thus [Thi99a]:

> *... we use as the target language a small calculus idealizing the intermediate language from Appel's account of the Standard ML of New Jersey compiler [App92, Thi97a]. A CPS term M consists of a jump $kx_1 \ldots x_n$ to k with arguments $x_1 \ldots x_n$, together with some bindings of the form* `where` $f x_1 \ldots x_n = M'$. *The grammar is given by the single rule,*
>
> $$M ::= xx^*(\texttt{where}\, xx^* = M)^*$$
>
> *To sketch the intended meaning, it suffices to say here that a jump should be reduced by fetching the term jumped to, and substituting the actual parameters for the formal parameters (avoiding name capture):*
>
> $$\begin{array}{c} f x_1 \ldots x_n \ldots \texttt{where}\, f y_1 \ldots y_n = M \ldots \\ \to M[y_1 := x_1, \ldots, y_n := x_n] \ldots \texttt{where}\, f y_1 \ldots y_n = M \ldots \end{array}$$

This section describes a CPS-transform of the let-language. As the target language, we use Thielecke's CPS language—for the sake of discussion, a typed sequent-style version, which is essentially the one described in Thielecke's thesis [Thi97a]. The types of the CPS-language are as follows:

$$A ::= \overline{A_1 * \cdots * A_n} \mid unit \mid int \mid bool \mid \ldots \qquad \text{where } n \geq 0$$

Here $\overline{A_1 * \cdots * A_n}$ is the type of a continuation k that can be jumped to with arguments of types $A_1, \ldots, A_n$. The term formation rules are:

$$\Gamma, k : \overline{A_1 * \cdots * A_n}, x_1 : A_1, \ldots, x_n : A_n \vdash k\langle x_1, \ldots, x_n \rangle$$

$$\frac{\Gamma, k : \overline{A_1 * \cdots * A_n} \vdash M \qquad \Gamma, x_1 : A_1, \ldots, x_n : A_n \vdash N}{\Gamma \vdash M\, where\ k\langle x_1, \ldots, x_n \rangle = N}$$

The expressions have no return type, because they do not return—they are run only for their effect.

Our CPS transform takes a sequent $(\Gamma \vdash M : A)$ of the let-language and a CPS variable k of type $\overline{A}$ to a sequent $(\Gamma, k : \overline{A} \vdash M^k)$ of the CPS language. The variable k is deemed to be the default continuation passed by the caller—the *current continuation*. We define

$$\begin{aligned} x^k &= k\langle x \rangle \\ (let\ x = M\ in\ N)^k &= (M^l where\ l\langle x \rangle = N^k) \\ ()^k &= k\langle\rangle \end{aligned}$$

Because in the let-calculus it holds that

$$\pi_i(M) \equiv (let\, x = M\ in\ \pi_i(x))$$

and

$$(M, N) \equiv (let\, x = M\ in\ let\, y = N\ in\ (x, y))$$

it suffices explain the transforms of pairs and projections whose arguments are variables. Let

$$\begin{aligned} (x, y)^k &= k\langle x, y\rangle \\ \pi_i(x)^k &= k\langle \pi_i(x)\rangle \end{aligned}$$

The term $k\langle \pi_i(x)\rangle$ is not part of the CPS calculus as we have defined it. The expression $\pi_i(x)$ looks like a procedure call, which the CPS calculus is not supposed to have. However, it is reasonable to assume that π_i exists in the CPS language as a primitive command that needs no calling and returning (e.g. a machine command picking the right register). We remedy this by augmenting the CPS language with rules

$$\begin{gathered} x_1 : A_1, \ldots, x_n : A_n \vdash x_i : A_i \\ \Gamma, x : A_1 * A_2 \vdash \pi_i(x) : A_i \end{gathered}$$

and replacing the rule for jumps by

$$\frac{\Gamma \vdash N_1 : A_1 \quad \ldots \quad \Gamma \vdash N_n : A_n}{\Gamma, k : \overline{A_1 * \cdots * A_n} \vdash k\langle N_1, \ldots, N_n\rangle}$$

If the source language of the CPS transform has function types $A \rightharpoonup B$, like the computational lambda-calculus, then the transform sends $A \rightharpoonup B$ to $\overline{A * \overline{\overline{B}}}$. The idea is that a function $f : A \rightharpoonup B$ corresponds to a continuation that can jumped to with an argument of type A and a B-accepting default continuation. The CPS-transforms of lambda- and application expressions are

$$\begin{aligned} (\lambda x.M)^k &= (k\langle f\rangle\, where\ f\langle x, l\rangle = M^l) \\ (fx)^k &= f\langle x, k\rangle \end{aligned}$$

(In analogy to the transforms of pairs and projections, we have $(MN)^k = (let\, f = M\ in\ let\, x = N\ in\ fx)^k$.) The CPS-transform for the whole computational lambda-calculus is summarised in Figure 8.1.

$$
\begin{aligned}
x^k &= k\langle x\rangle \\
(let\, x = M\ in\ N)^k &= (M^l where\ l\langle x\rangle = N^k) \\
()^k &= k\langle\rangle \\
(x,y)^k &= k\langle x,y\rangle \\
\pi_i(x)^k &= k\langle \pi_i(x)\rangle \\
(\lambda x.M)^k &= (k\langle f\rangle where\ f\langle x,l\rangle = M^l) \\
(fx)^k &= f\langle x,k\rangle \\
[M]^k &= (k\langle f\rangle where\ f\langle l\rangle = M^l) \\
\mu(f)^k &= f\langle k\rangle
\end{aligned}
$$

Figure 8.1: The CPS transform of the computational lambda-calculus

8.2 *callcc* and *throw*

With the CPS transform we can also implement fascinating operators that only few programming languages have. The most prominent of these operators are *callcc* and *throw* , which are part of Scheme [ADH+98] and the `SMLofNJ` library of Standard ML of New Jersey [TL].

To add *callcc* and *throw* to the computational lambda-calculus, we add types of the form $A\,cont$, where A is a type, and sequents

$$
\frac{\Gamma \vdash M : A\,cont \rightharpoonup A}{\Gamma \vdash callcc\, M : A} \qquad \frac{\Gamma \vdash M : A\,cont \quad \Gamma \vdash N : A}{\Gamma \vdash throw\ \ MN : B}\ \text{for any type } B
$$

The idea is that $(callcc(\lambda k : A\,cont.M'))$ binds the current continuation to the variable k in M'. Then M' can access k using an expression of the form $(throw\ \ k\,N)$. For example, the program

$$19 + callcc(\lambda k.8 + throw\ \ k\,69)$$

returns 88, because k stands for $19+(-)$. The return type B of *throw* is arbitrary, because $(throw\ \ MN)$ does not return and can therefore stand at any position of an expression.

The CPS transform makes the semantics of *callcc* and *throw* precise: It sends $A\,cont$ to $\overline{A}$, and

$$
\begin{aligned}
(callcc f)^k &= f\langle k,k\rangle \\
(throw\ \ lx)^k &= l\langle x\rangle
\end{aligned}
$$

So *callcc* copies the current continuation k. The second copy of k becomes the default continuation of f, whereas the first copy becomes an ordinary argument. By contrast, *throw* forgets the current continuation k and invokes the continuation l which came as an ordinary argument.

8.3 Lambda semantics of the CPS language

As described in the introduction of this chapter, the CPS language has an intuitive operational semantics. However, we are aiming for a *denotational* analysis of the CPS transform. One way goes via a denotational semantics of the CPS language. Observe that there is an obvious translation of the CPS language into lambda-expressions:

$$(k\langle N_1, \ldots, N_n\rangle)^\lambda = k(N_1, \ldots, N_n)$$
$$(M \mathit{where}\ k\langle x_1, \ldots, x_n\rangle = N)^\lambda = (\lambda k.(M^\lambda))(\lambda x_1, \ldots, x_n : (N^\lambda))$$

Now let R be a type, and define

$$\left(\overline{A_1 * \cdots * A_n}\right)^\lambda = A_1 * \cdots * A_n \to R$$

If $(x_1 : A_1, \ldots, x_n : A_n \vdash M)$ is a sequent of the CPS language, then $(x_1 : A_1^\lambda, \ldots, x_n : A_n^\lambda \vdash M^\lambda : R)$ is a sequent of the lambda calculus. Intuitively, R stands for all possible effects of CPS expressions—it is commonly called the *answer type*. We define the denotational semantics of the CPS-language as the translation $(-)^\lambda$, followed by the denotational semantics of the lambda-calculus. The lambda-calculus is typically modelled with a cartesian-closed category. However, the translation $(-)^\lambda$ does not need all exponentials of the form B^A, but only those of the form R^A. This leads to the following definition:

Definition 8.1. A *response category* consists of a category K with finite products, an object R of K, and, for each object A of C, an exponential R^A of R by A.

(The term 'response category' was suggested to me by Peter Selinger, who also considered these structures [Sel00].) The object R is commonly called the *answer object*. For each object A, let $ev_A : R^A \times A \longrightarrow R$ stand for the evaluation map. For a morphism $f : A \times B \longrightarrow R$, let $\lambda f : A \longrightarrow R^B$ stand for the curried version of f.

Definition 8.2. The denotational semantics of a CPS-sequent $(\Gamma \vdash M)$ in a response category K is defined as the evident semantics of $(x_1 : A_1^\lambda, \ldots, x_n : A_n^\lambda \vdash M^\lambda : R)$ in K.

Remark 8.1. Requiring a response category to have finite products, as opposed to an arbitrary precartesian tensor, stands in contrast to our previous approach to target-language semantics, because finite products are sound only for languages that are call-by-name or effect-free. Here we require finite products only to avoid mathematical complications. We shall discuss the treatment of more general target languages in Section 8.7.

8.4 Continuations monads

In this section, we shall construe a response category as a cartesian computational model.

In a response category K, the map $R^{(-)}$ forms a functor $K^{op} \longrightarrow K$, where $R^f = \lambda(R^B \times f; ev)$—so we have

$$\begin{tikzcd}
R^A \times A \arrow[r, "ev"] & R \\
R^B \times A \arrow[u, "R^f \times A"] \arrow[r, "R^B \times f"] & R^B \times B \arrow[u, "ev"']
\end{tikzcd}$$

Remark 8.2. Proving that $R^{(-)}$ is a functor seems to require that λg is central for all g. This is one of the reasons why we require $\times$ to be the cartesian product—an arbitrary precartesian tensor complicates finding a suitable definition of exponentials.

Let's overload $\overline{(-)}$ to stand for $R^{(-)}$. In the lambda-calculus of K, the definition of the morphism part of $\overline{(-)}$ looks like this:

$$\overline{f}\, k^{\overline{A}} x^B = k(fx)$$

Proposition 8.1. *A response category K forms a λ_C-model with the following definitions (where Ev and Λ stand for evaluation and Currying for the T-exponentials).*

$$T = \overline{\overline{(-)}} \qquad \eta_A x^A k^{\overline{A}} = kx \qquad \mu_A = \overline{\eta_{\overline{A}}}$$

$$t_{A,B}\left(x^A, y^{\overline{\overline{B}}}\right) k^{\overline{AB}} = y\left(\lambda z^B.k(x,z)\right)$$

$$(TB)^A = \overline{A \times \overline{B}} \qquad \Lambda f x^A(y^B, k^{\overline{C}}) = f(x,y)k \qquad Ev(f^{\overline{A\overline{B}}}, x^A)k^{\overline{B}} = f(x,k)$$

Proof. A routine check in the $\lambda_{\beta\eta}$-calculus. $\square$

The monad above is called the *continuations monad.*

8.5 Comparing the CPS transform with the monadic-style transform

The remainder of this chapter is a denotational analysis of the source language of the CPS transform (here, Moggi's computational lambda-calculus). There are four (!) denotational semantics in a response category K:

The first is the CPS transform $(-)^k$, followed by the denotational semantics $(-)^\lambda$ of the CPS language. The other three use the λ_C-model (K, T) that arises from K. To see this, let's recall Diagram 5.7:

$$\begin{tikzcd}
L_T(\Gamma, A) \arrow[r, "K_T[\![-]\!]"] \arrow[d, "(-)^\sharp"'] & K_T(\Gamma, A) \arrow[d, "(-)^\sharp"] \\
L(\Gamma^\sharp, T(A^\sharp)) \arrow[r, "K[\![-]\!]"] & K(\Gamma^\sharp, T(A^\sharp))
\end{tikzcd}$$

L_T is the computational lambda-calculus of K_T, and L is the language with the bind-construct. The map $(-)^\sharp$ on the left side is the monadic-style transform (Figure 5.9). $K_T[\![-]\!]$ is the precartesian semantics of L_T (Figures 2.1, 5.5, and 5.7). $K[\![-]\!]$ is the semantics of L in the λ_C-model (K, T) (Figures 5.2–5.4). The map $(-)^\sharp$ on the right side is the isomorphism of the adjunction $F_T \dashv G_T$. Proposition 5.12 states that the diagram commutes. Proposition 5.13 states that Moggi's semantics is the diagonal of the diagram.

Next we shall see that the CPS semantics agrees with the other three, up to the bijection

$$K(\Gamma \times \overline{A}, R) \cong K(\Gamma, \overline{\overline{A}})$$

This follows from comparing the CPS-transform with the left-bottom path of the diagram.

Proposition 8.2. *Let K be a response category, let (K, T) be the arising λ_C-model, and let M be an expression of the computational lambda-calculus of K_T. Then it holds in K that*

$$(M^k)^\lambda = M^\sharp k$$

Proof. In K, we have

$$(\mathit{bind}\, x \Leftarrow M_1 \;\mathit{in}\; M_2) = \lambda k.M_1(\lambda x.M_2 k)$$

With this, the claim follows from induction over M and performing routine calculations in the $\lambda_{\beta\eta}$-calculus of K. $\square$

8.6 Precartesian analysis of continuations

We shall now analyse the precartesian structure at the source end of the CPS transform. For comparison, let's recall the steps we took for state (Chapter 6):

1. We constructed a new precartesian category K_S from an original precartesian category K together with an object S, and we expressed this construction in terms of a language transform the 'state-passing style transform').
2. Next, we characterised what central, copyable, and discardable means in the new system.
3. We explained what central, copyable, and discardable in K_S have to do with reading the store and writing to the store.
4. Finally, we presented a blob diagram displaying the relation between central, copyable, and discardable in K_S in the simple case where K has finite products.

For continuations, we have already taken the analogue of the first step by introducing the CPS transform and the response category K, and constructing the Kleisli category K_T from the λ_C-model (K, T). (However, to limit the mathematical complexity, we have been more modest than for state in that we required K to have finite products, as opposed to an arbitrary precartesian tensor.) In this section, we shall take the remaining three steps for continuations.

8.6.1 Discardable

That a sequent $(\Gamma \vdash M : A)$ of the internal language L_T is discardable in K_T is stated by the equation

$$(let\, y = M\ in\ ()) \equiv ()$$

Sending this equation through the CPS transform $(-)^k$ yields

$$(M^l where\ \ l\langle y\rangle = k\langle\rangle) \equiv k\langle\rangle \tag{8.1}$$

An example of a non-discardable expression is

$$M = (throw\ \ m\, 42)$$

where m is a variable of type *int cont*. To see this, let E be the context

$$E[M] = callcc(\lambda m.let\, x = M\ in\ 0)$$

Then the program $E[()]$ returns 0, and $E[\mathit{throw}\ \ m\,42]$ returns 42. (This experiment can be made in Scheme or Standard ML of New Jersey.) The CPS transform confirms the difference: The left side of Equation 8.1 is

$$(m\langle 42\rangle\,\mathit{where}\ \ l\langle y\rangle = k\langle\rangle) \equiv m\langle 42\rangle$$

which is obviously not $\beta\eta$-equivalent to $k\langle\rangle$.

8.6.2 Copyable

That a sequent $(\Gamma \vdash M : A)$ of L_T is copyable in K_T is stated by the equation

$$(\mathit{let}\, y = M\ \mathit{in}\, (y, y)) \equiv (M, M) \tag{8.2}$$

Sending this equation through the CPS transform $(-)^k$ yields

$$(M^l\mathit{where}\ \ l\langle y\rangle = k\langle y, y\rangle) \equiv (M^l\mathit{where}\ \ l\langle y\rangle = (M^m\mathit{where}\ \ m\langle z\rangle = k\langle y, z\rangle)) \tag{8.3}$$

Intuitively, if M^l invokes its default continuation l with y, then left side of the equation proceeds by passing y twice to the continuation k. By contrast, the right side proceeds by running M^m again and passes the two values y and v from the first and second run, respectively, to k.

An example of a non-copyable expression is $\mathit{twicecc}\,(x, h)$ where x is a variable of type A, h is a variable of type $A\ \mathit{cont}$, and

$$\mathit{twicecc}\,(x, h)^k =_{\text{def}} (k\langle x, l\rangle\,\mathit{where}\ \ l\langle y\rangle = k\langle y, h\rangle)$$

In fact, *twicecc* can also be expressed in terms of *callcc* and *throw* (see [Thi97a]). To see that *twicecc* is not generally copyable, let $M = \mathit{twicecc}\,(x, h)$ in Equation 8.3, apply $(-)^\lambda$ to the resulting equation and observe that the two sides are not $\beta\eta$-equivalent. Another way to prove this is letting $M = \mathit{twicecc}\,(x, h)$ in Equation 8.2 and finding a source language context that distinguishes the two sides. Thielecke did this for Standard ML of New Jersey and Scheme (see [Thi97a]).

However, $\mathit{twicecc}\,(x, h)$ is discardable: Letting $M = \mathit{twicecc}\,(x, h)$ in Equation 8.1 and applying $(-)^\lambda$, the left side reduces to $k\langle\rangle$.

Remark 8.3. Thielecke dedicated a whole article [Thi99b] to the implications of using a continuation twice. In that article, he discusses a source-language expression *arg-fc* (‘argument of first call’) which—like *twicecc*—is discardable, but not copyable. Compared with *twicecc*, *arg-fc* is easier to understand in the source language, but looks more complicated after it has been sent through the CPS-transform.

By contrast to *twicecc*, the non-discardable expression *throw* $m42$ is copyable—this follows immediately from letting $M = (\mathit{throw}\ m\,42)$ in Equation 8.3.

An expression which is neither discardable nor copyable is $\mu(x)$ (which denotes the morphism ε). To see this, let $M = \mu(x)$ in Equations 8.1 and 8.3. The resulting two equations, interpreted by $(-)^{\lambda}$, don't hold in the $\lambda_{\beta\eta}$-calculus.

8.6.3 Central and thunkable

Stating in L_T that the denotation of a sequent $(\Gamma \vdash M : A)$ is central in K_T is saying that for all $(\Delta \vdash N : B) \in L_T$ such that Γ and Δ share no variables, it holds in K_T that

$$\begin{aligned}&(\mathit{let}\, x = M\ \mathit{in}\ \mathit{let}\, y = N\ \mathit{in}\,(x, y))\\ \equiv&(\mathit{let}\, y = N\ \mathit{in}\ \mathit{let}\, x = M\ \mathit{in}\,(x, y))\end{aligned}$$

Sending this equation through the CPS transform $(-)^k$ yields

$$\begin{aligned}&(M^l\mathit{where}\ l\langle x\rangle = (N^m\mathit{where}\ m\langle y\rangle = k\langle x, y\rangle))\\ \equiv&(N^m\mathit{where}\ m\langle y\rangle = (M^l\mathit{where}\ l\langle x\rangle = k\langle x, y\rangle))\end{aligned} \tag{8.4}$$

Next we shall get rid of the universal quantifier over N.

Proposition 8.3. *The denotation of a sequent $(\Gamma \vdash M : A) \in L_T$ is central in K_T if and only if for a fresh variable z of type $\overline{\overline{B}}$ it holds in K that*

$$\begin{aligned}&(M^l\mathit{where}\ l\langle x\rangle = (z\langle m\rangle\mathit{where}\ m\langle y\rangle = k\langle x, y\rangle))\\ \equiv&(z\langle m\rangle\mathit{where}\ m\langle y\rangle = (M^l\mathit{where}\ l\langle x\rangle = k\langle x, y\rangle))\end{aligned} \tag{8.5}$$

Proof. The 'only if' follows from letting $N = \mu(z)$ in Equation 8.4. The 'if' holds because sending both sides of Equation 8.5 through $(\ldots\ \mathit{where}\ z\langle m\rangle = N^m)$ yields Equation 8.4. □

Equation 8.5 means that M^l commutes with a jump to an arbitrary z. It is obvious that M^l should have no side-effect, because if z discards m, then the right side of Equation 8.5 does not run M^l at all. In particular, M^l should exit via its default continuation l, because a different exit would mean jumping away before running $z\langle m\rangle$. So centrality for continuations is an extremely strong condition.

That the denotation of a sequent $(\Gamma \vdash M : A) \in L_T$ is thunkable in K_T is stated by

$$(\mathit{let}\, x = M\ \mathit{in}\,[x]) \equiv [M]$$

Sending this through $(-)^k$ yields

$$(M^l \textit{where } l\langle x\rangle = (k\langle f\rangle \textit{where } f\langle m\rangle = m\langle x\rangle)) \equiv (k\langle g\rangle \textit{where } g\langle l\rangle = M^l) \quad (8.6)$$

Proposition 8.4. *If the denotation of* $(\Gamma \vdash M : A) \in L_T$ *is central in* K_T*, then it is thunkable.*

Proof. Sending Equation 8.5 through $(\ldots \textit{where } k\langle x, y\rangle = y\langle x\rangle)$ yields

$$\begin{aligned} &(M^l \textit{where } l\langle x\rangle = (z\langle m\rangle \textit{where } m\langle y\rangle = y\langle x\rangle)) \\ \equiv &(z\langle m\rangle \textit{where } m\langle y\rangle = M^y) \end{aligned} \quad (8.7)$$

Up to renaming variables, this is Equation 8.6. □

We shall complete the precartesian analysis by proving that all thunkable morphisms are focal—that is, K_T is a computational abstract Kleisli-category (see Section 5.2.4). Although this follows directly from Proposition 5.8, it may be more general to prove this with the CPS transform, because it is not clear if for 'impure' target languages we can still use a continuations monad (see the discussion at the end of this chapter).

Lemma 8.1. *In* K_T*, all morphisms of the form* $[f]$ *are focal.*

Proof. Morphisms of the form $[f]$ are denoted by sequents of the form $(\Gamma \vdash [N] : TA)$. The claim follows from letting $M = [N]$ in Equations 8.1, 8.3, and 8.5, and straightforward calculations in the CPS-calculus (interpreted by $(-)^\lambda$). □

Proposition 8.5. *All thunkable morphisms of* K_T *are focal.*

Proof. Follows directly from Lemmas 5.2 and 8.1. (Lemma 5.2 relies on the fact that K_T is a precartesian abstract Kleisli-category, which is true here where we started with a response category. I conjecture that Lemma 5.2 still holds if we construct K_T using the CPS transform into an impure target language). □

The precartesian analysis is summarised in Figure 8.2.

8.7 Effect-full target languages

Roughly speaking, the CPS language may be implemented on a real machine by translating jumps of the CPS language to machine jumps, keeping arguments in machine registers. Therefore, it is reasonable to allow the CPS language to have commands like

`store` *address,value*

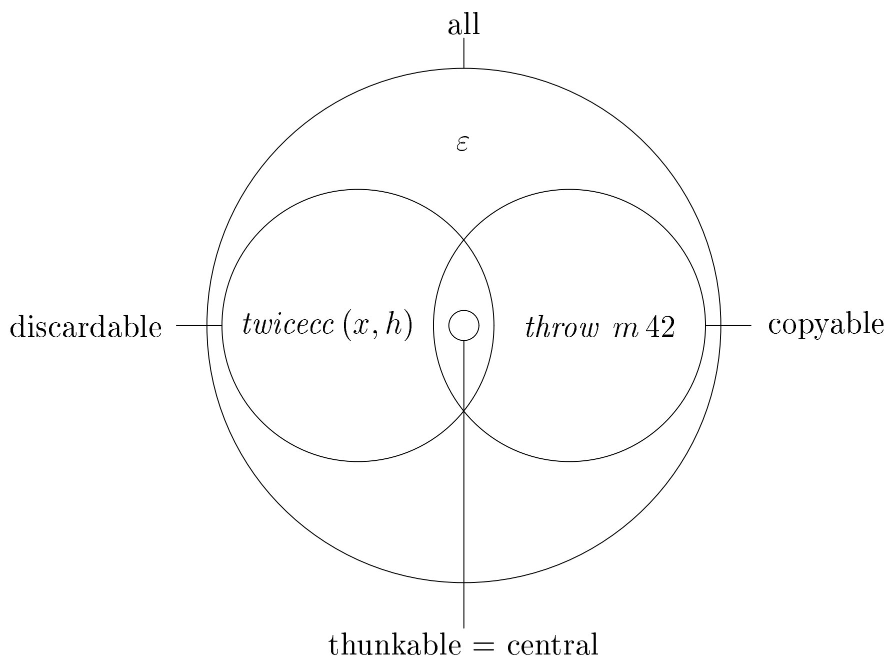


Figure 8.2: Precartesian analysis for continuations

which writes *value* into the machine's memory at *address*. Such commands could then be used to implement effects of the source language of the CPS transform. In the CPS language, `store` would have the type $address \times int \longrightarrow 1$. (Because the command returns on the spot, it has a return type and needs no default continuation.) Obviously, `store` is not discardable.

The finite products of a response category seem to rule out commands like `store`, because every morphism must be discardable. (Similar examples exist for 'copyable' and 'central'.) However, if we allowed K to be an arbitrary precartesian category, problems arise—for example, for a realistic lambda operator, the unit of the continuations monad is no longer natural: For a morphism $f \in K(A, B)$ we have

$$k : \overline{B} \vdash \overline{f}(k) \equiv \lambda x.k(f(x)) : \overline{A}$$
$$x : A \vdash \eta(x) \equiv \lambda k.kx : \overline{\overline{A}}$$

Expressing the naturality equation $f; \eta_B = \eta_A; \overline{\overline{f}}$ in the let-language of K, using the β rule $(\lambda x.M)N = M[x := N]$ only in the safe cases where N is a variable or a lambda expression, yields

$$x : A \vdash (let\, y = fx\; in\; \lambda k.ky) \equiv \lambda k.k(fx) : \overline{\overline{B}}$$

In a realistic target language, this equation can be false, because the left side of the equation runs f, whereas the right side does not. So η is not natural, and therefore our attempt to define a continuations monad fails. One may hope that, despite this failure, K_T is still a category, but it is not (for example, $id; f \neq f$).

But these problems may have a solution: It is reasonable to assume that denotations in K of target-language lambda expressions are focal. So let's assume that all such denotations are in a subcategory S with finite products of the focus of K. (For example, K might be a computational abstract Kleisli-category.) Then the data in Proposition 8.1 forms a λ_C-model on S. From this we could construct the new system as the Kleisli category S_T. This may look like cheating, because the denotation of `store` *address*,*value* is not in S. But $\lambda k.k(\texttt{store}\ address, value)$ is!

Another way of constructing a precartesian category at the source end is letting a morphism $K_T(A, B)$ be an equivalence of CPS terms of the form $(x : A, k : \overline{B} \vdash M)$, where the equivalence is up to realistic equations (e.g. restricted β and η laws). Such a construction, the 'CPS term model', is part of Thielecke's thesis [Thi97a].

Also in his thesis, Thielecke defined *$\otimes\neg$-categories*, which in our terminology are precartesian categories with some extra operators and axioms. One axiom, for example, states that all central morphisms are thunkable. (In this Chapter, we *proved* this for K_T.) Thielecke proved that, if certain equations hold for the CPS language, then the CPS term model is a $\otimes\neg$-category. I proved [Füh98] that every $\otimes\neg$-category arises from a response category (via an extended version of Theorem 7.3 [1].

If $\otimes\neg$-categories are to have a good conceptual status, their axioms must hold even under modest (i.e. realistic assumptions) about the equational laws of the CPS language. Finding out how robust $\otimes\neg$-categories are remains, at least for me, a future challenge.

[1] A very similar result was proved independently by Peter Selinger (Theorem 2.16 in [Sel00])). Although Selinger's theorem is for a call-by-name (modelled by 'control categories'), by a duality result in [Sel00] it can be translated into a theorem for call-by-value (modelled by 'co-control categories', which are essentially $\otimes\neg$-categories with finite sums).

Chapter 9

Conclusions and further research

9.1 Conclusions

We have classified and solved the problems caused by procedure calls in direct-style reasoning about call-by-value programs. One problem was the relevance of the number of evaluations of procedure arguments. The other problem was the relevance of the arguments' evaluation order.

As a semantic solution, we introduced precartesian categories, which we found by generalising categories with finite products in such a way that equations that are not valid for call-by-value need no longer hold. Precartesian categories inspired the let-calculus, a new kind of calculus for proving equivalences of call-by-value programs.

The most innovative aspects of the let-calculus seem to be (1) the properties *copyable*, *discardable*, and *central*, and their inference rules, which resemble effect systems known from the literature, and (2) the treatment of the evaluation order—in particular, the property *clear*. (By contrast, the properties *affine* and *relevant* are well-known.)

It is remarkable that a suitable generalisation of categories with finite products has not been made long ago—in particular given the fact that Rosolini's p-categories and their precursors, which deal with the special case of partiality, showed the direction. Apparently, the difficulty was seeing the need for generalised monoidal tensors that are not functorial in both arguments jointly. This generalisation came only in the early 90's, when John Power introduced premonoidal categories.

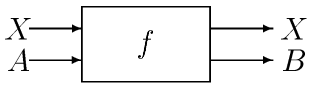
9.2 Directions for further research

9.2.1 Recursion and non-natural traces

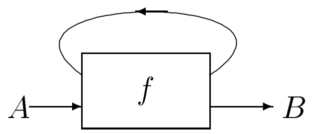
An important topic that we have not addressed is recursion. To model recursion in the absence of computational effects (other than non-termination), Hasegawa [Has97] suggested to use *traced symmetric monoidal categories.* A *trace* one a symmetric monoidal category K is as family of functions

$$Tr^X_{A,B} : K(A \otimes X, B \otimes X) \longrightarrow K(A, B)$$

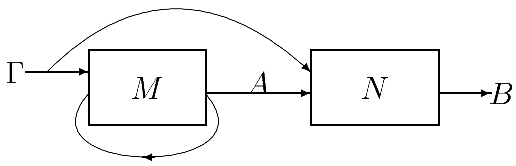
which is natural in X, A, and B, and satisfies certain equations. Using a graphical presentation, sending a morphism



through $Tr^X_{A,B}(-)$ yields



For sequents $(\Gamma, x : A \vdash M : A)$ and $(\Gamma, x : A \vdash N : B)$, the semantics of $(\Gamma \vdash letrec\, x = M\, in\, N : B)$ is presented by



In the conclusion of his (award-winning) thesis [Has99], Hasegawa suggested to generalise traces to symmetric premonoidal categories. This was carried out in detail by Jeffrey and Schweimeier [Jef98].

The future investigation that I suggest stems from with the breakdown of the naturality of $Tr(-)^X_{A,B}$ in the presence of computational effects: As explained in [Has97], the naturality in A, B, and X, respectively, is stated by the equations in Figures 9.1, 9.2, and 9.3. In the presence of computational effects, these equations can be false: In the cases of left tightening and right tightening, on the left side of the equation, f is evaluated as often as g, whereas on the right side, f is evaluated only once. In the case of sliding, on the left side of the equation, g is evaluated first, whereas on the right side, f is evaluated first.

I suggest to study what tightening and sliding have to do with the properties *copyable* and *central*. If tightening and sliding cannot be completely understood in terms of *copyable* and *central*, then it may be useful to introduce *tightenable* and *slideable* morphisms, and study their categorical closure properties. These could inspire useful inference rules that extend the let-calculus to a *recursive let-calculus*.

9.2.2 Jeffrey's premonoidal-copyable morphisms

The elegance of our precartesian framework is slightly spoiled by the fact that copyable morphisms do not generally form a category. However, this is not a flaw of precartesian categories, but reflects the fact of life that the sequential composite of two copyable programs is not generally copyable. During our development of the let-calculus, the missing closure property forced the unpleasant definition

$$\max\{E_1, \ldots, E_n\} = \begin{cases} E_1 \vee \ldots \vee E_n \vee \overline{copyable} & \text{if } E_i \not\leq \textit{central} \text{ for more than one } i \\ E_1 \vee \ldots \vee E_n & \text{otherwise} \end{cases}$$

It is natural to ask whether there is a property that provides an alternative to *copyable* such that the morphisms with that property are closed under composition. Alan Jeffrey (private communications) suggested to me the following definition:

Definition 9.1 (Alan Jeffrey). A morphism $f : A \longrightarrow B$ of a precartesian is called *premonoidal-copyable* if for all morphisms $g : BC \longrightarrow D$ the following diagram commutes:

$$\begin{tikzcd}
AC \arrow[r, "fC"] \arrow[d, "\delta C"'] & BC \arrow[r, "\delta C"] & B^2C \arrow[r, "Bg"] & BD \arrow[d, "id"] \\
A^2C \arrow[r, "AfC"] & ABC \arrow[r, "Ag"] & AD \arrow[r, "fD"] & BD
\end{tikzcd}$$

In the let-calculus, if $(x : A \vdash M : B)$ denotes f and $(z : B, y : C \vdash N : D)$ denotes g, this diagram corresponds to the equation

$$\begin{aligned} x : A, y : C \vdash &(let\, z = M\, in\, (z, N)) \equiv \\ &(let\, z = M\, in\, let\, v = N\, in\, (M, v)) : B * D \end{aligned}$$

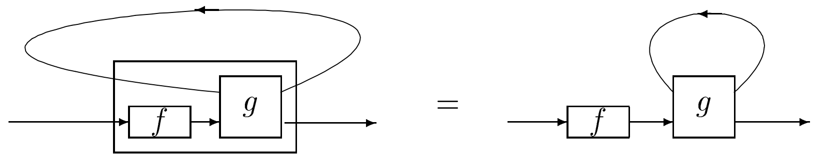


Figure 9.1: ‘Left tightening’

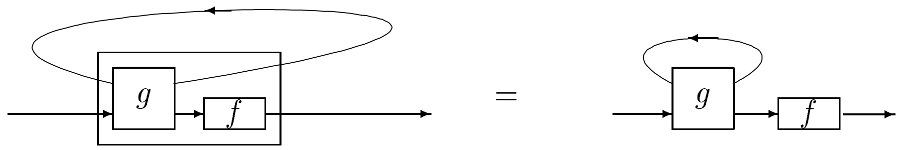


Figure 9.2: ‘Right tightening’

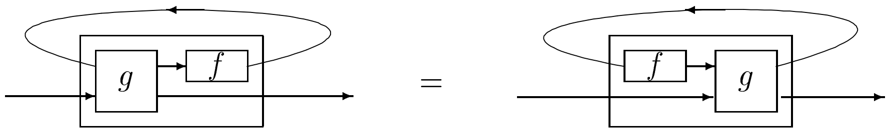


Figure 9.3: ‘Sliding’

Letting $C = I$ and $g = id_B$ shows that every premonoidal-copyable morphism is copyable. Crucially, Alan Jeffrey proved that the premonoidal-copyable morphisms are closed under composition (and also under tensor).

Unfortunately, premonoidal-copyable morphisms have two drawbacks: First, their definition involves both evaluation order and evaluation frequency (in the above equation, M is copied, and it is also made to commute with N in a certain sense). This stands in contrast to our conceptual separation of evaluation order and evaluation frequency. Second, one can prove that morphisms that are premonoidal-copyable and discardable are focal. Therefore, replacing *copyable* by *premonoidal-copyable* would cause a partial collapse of the precartesian cube—in particular, there would be no precartesian property that corresponds to the well-motivated property *clear*.

However, it could be very useful, to *add* the property *premonoidal-copyable* to the let-calculus. This would raise the challenge to find out about the substitution properties of *premonoidal-copyable* expressions. (Certainly, we can substitute them for variables which are both *relevant* and *clear*, but can we do better?)

9.2.3 Adding higher-order to the let-calculus

I suggest to add higher-order operators to the let-calculus, model it with computational abstract Kleisli-categories, and extend the precartesian cube with a fourth dimension *thunkable*, where judgements of the form $(\Gamma \vdash M \,!\, \mathit{thunkable})$ take over the rôle of judgements $(\Gamma \vdash M \downarrow A)$ of the computational lambda-calculus.

In every computational abstract Kleisli-category it holds that,

$$\frac{\Gamma \vdash M \,!\, \mathit{thunkable}}{\Gamma \vdash M \,!\, \mathit{focal}} \tag{9.1}$$

The naturality of the isomorphism

$$\Lambda : K(\mathit{incl}(A) \otimes B, C) \cong K_\vartheta(A, B \rightharpoonup C)$$

implies that for every thunkable f it holds for all morphisms g that

$$f; \Lambda g = \Lambda(f \otimes id; g)$$

For the calculus this implies that thunkable expressions can be substituted for variables under lambda-bindings—therefore, they can be substituted for any variable. By contrast, the converse of Rule 9.1 does not hold in every computational abstract Kleisli-category, and therefore focal expressions cannot generally be substituted for variables under lambda-bindings.

The suggested extension of the let-calculus, with all details worked out, should deepen our understanding of higher-order operators for call-by-value languages.

9.2.4 Precartesian transformers

The basic idea Each of the following constructions takes precartesian categories with extra structure to precartesian categories (with or without extra structure):

- The construction of the p-category $Par(K)$ from a category K with finite products and a dominion (see Chapter 4).
- The construction of the global-state category K_S from a precartesian category K and an object S of K (see Chapter 6).
- The construction of the Kleisli category K_T of a strong monad T on a precartesian category K (see Theorem 5.2).
- The construction of the Kleisli category of a premonoidal dyad[1] on a precartesian category (see Theorem 4.2 in [PR99]).
- The construction of a new precartesian computational model (K, T') from a given precartesian computational model (K, T) by applying a monad transformer (see below) to T.

To bring all these construction into one framework, I suggest to introduce the notion of *precartesian transformers*—that is, functions that map precartesian categories with extra structure to precartesian categories. (Ideally, such a function should be a functor on some category whose morphisms are strong precartesian functors.)

Modularity and dyads Any general account of computational effects raises the question if and how computational effects can be combined. This problems falls into the area "Modularity in denotational semantics".

Since the early 90's, it has been tried to combine features by using *monad transformers* (originally called 'monad constructions', see [Mog89]). The idea is simple: A monad constructor is a function that takes monads to monads. For example, the monad construction $(-)_{seff}$ for adding global state takes a monad T to a certain monad that has the endofunctor $T_{seff}(-) = (T((-) \times S))^S$. The monad construction $(-)_{excp}$ for adding exceptions takes a monad T to a certain monad that has the endofunctor $T_{excp}(-) = T((-)+E)$. To add both global state and exceptions, one would start with the identity monad on the original system and apply $(-)_{excp}$ and $(-)_{seff}$ in the preferred order. (The two results differ, but

[1] a kind of generalised strong monad, see also Remark 5.2 and the introduction of Chapter 6

this is realistic for computer science, and not a *mathematical* problem.) The final model is provided by the Kleisli category of the resulting monad. The scope of monad transformers is limited, because the do not change the category (i.e. a monad T on C sent through a monad transformer is still a monad on C.)

Expressed in the framework of precartesian transformers, a typical modularity problem would be the following: Let K be a response category with answer object R like in Chapter 8. (So on K we have the continuations monad, whose functor is $T = R^{R^-} : K \longrightarrow K$.) Moreover, let S be an object of K. Luckily, we know that we can form the precartesian category K_T and then the global state category $(K_T)_S$. But what if we want to add state and continuations in the opposite order? Adding state gives K_S, and to proceed, K_S must be a response category, or at least have some structure that enables the construction of a new precartesian category for continuations. So the challenge here is to 'lift' the continuations structure on K along the state construction to K_S.

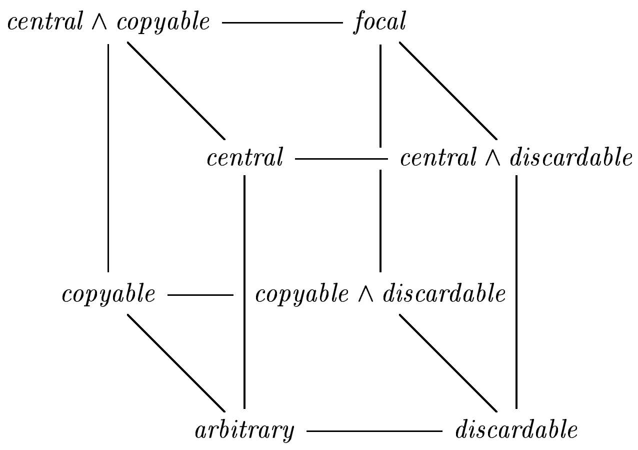
Such structure lifting is not addressed by monad transformers, because they do not explain how to lift one monad along another monad's Kleisli construction. By contrast, structural lifting has been studied in a more general framework by Power, Rosolini, and Robinson [Pow99a, PR98, PR99], where—in the most recent and general version—dyads are lifted along the Kleisli constructions of dyads.

Most generally, a precartesian transformer P should be considered *modular with respect to extra structure of type* X if, informally, for each X-structure on a precartesian category K, P creates an X-structure on PK.

I suggest to study the use of precartesian transformers for dealing with modularity issues.

Appendix A

Overview of the let-calculus



$\downarrow \Phi$ (order-reversing lattice iso)

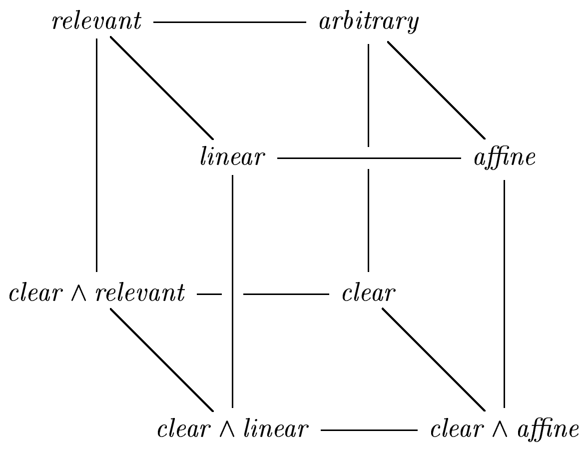


Figure A.1: The precartesian cube (top) and the cube of expression properties (bottom)

$$\max\{E_1,\ldots,E_n\} = \begin{cases} E_1 \vee \ldots \vee E_n \vee \overline{\mathit{copyable}} & \text{if } E_i \not\leq \mathit{central} \text{ for more than one } i \\ E_1 \vee \ldots \vee E_n & \text{otherwise} \end{cases}$$

$$x_1 : A_1, \ldots, x_n : A_n \vdash x_i \,!\, \mathit{focal}$$

$$\frac{\Gamma \vdash M \,!\, E \qquad \Gamma, x : A \vdash N \,!\, F}{\Gamma \vdash \mathit{let}\ x = M\ \mathit{in}\ N \,!\, \max\{E, F\}}$$

$$\Gamma \vdash () \,!\, \mathit{focal}$$

$$\frac{\Gamma \vdash M \,!\, E \qquad \Gamma \vdash N \,!\, F}{\Gamma \vdash (M, N) \,!\, \max\{E, F\}}$$

$$\frac{\Gamma \vdash M \,!\, E}{\Gamma \vdash \pi_i(M) \,!\, E}$$

$$\frac{\begin{array}{c}\Gamma \vdash M_1 \,!\, E_1 \quad \ldots \quad \Gamma \vdash M_n \,!\, E_n \\ y_1 : A_1, \ldots, y_n : A_n \vdash f(y_1, \ldots, y_n) \,!\, E\end{array}}{\Gamma \vdash f(M_1, \ldots, M_n) \,!\, \max\{E, E_1, \ldots, E_n\}} \quad \begin{array}{l}\text{for every constant} \\ f : A_1, \ldots, A_n \longrightarrow B\end{array}$$

$$\frac{\Gamma \vdash M \,!\, E}{\Gamma \vdash M \,!\, E'} \quad \text{If } E \leq E'$$

Figure A.2: Rules for the precartesian cube

$$b_x(\Gamma \vdash y) = \emptyset$$

$$b_x(\Gamma \vdash let\, y = M\, in\, N) = \begin{cases} \emptyset & \text{if } x = y \\ b_x(\Gamma \vdash M) & \text{if } x \notin FV(N) \cup \{y\} \\ b_x(\Gamma \vdash M) \cup \{\Gamma \vdash M\} & \\ \cup b_x(\Gamma, y \vdash N) & \text{if } x \in FV(N) - \{y\} \end{cases}$$

$$b_x(\Gamma \vdash ()) = \emptyset$$

$$b_x(\Gamma \vdash (M, N)) = \begin{cases} b_x(\Gamma \vdash M) & \text{if } x \notin FV(N) \\ b_x(\Gamma \vdash M) \cup \{\Gamma \vdash M\} \cup b_x(\Gamma \vdash N) & \text{if } x \in FV(N) \end{cases}$$

$$b_x(\Gamma \vdash \pi_i(M)) = b_x(\Gamma \vdash M)$$

$$b_x(\Gamma \vdash f(M_1, \ldots, M_n)) = \begin{cases} (b_x(\Gamma \vdash M_1) \cup \{\Gamma \vdash M_1\}) \cup \ldots \cup (b_x(\Gamma \vdash M_{i-1}) \\ \cup \{\Gamma \vdash M_{i-1}\}) \cup b_x(\Gamma \vdash M_i) \text{ where } M_i \text{ is the right-} \\ \text{most of the } M_j \text{ such that } x \in FV(M_j) \end{cases}$$

Figure A.3: Definition of $b_x(N)$

$$\frac{\Gamma \vdash N : A}{\Gamma \vdash N \,/\, relevant\, x} \text{ if } N \text{ has at most one free occurrence of } x$$

$$\frac{\Gamma \vdash N : A}{\Gamma \vdash N \,/\, affine\, x} \text{ if } N \text{ has at least one free occurrence of } x$$

$$\frac{\Gamma \vdash M\, !\, central \quad \forall(\Gamma \vdash M) \in b_x(\Delta \vdash N)}{\Delta \vdash N \,/\, clear\, x}$$

$$\Gamma \vdash N \,/\, arbitrary\, x$$

$$\frac{\Gamma \vdash M \,/\, e\, x \qquad \Gamma \vdash M \,/\, e'\, x}{\Gamma \vdash M \,/\, (e \wedge e')\, x}$$

$$\frac{\Gamma \vdash M \,/\, e\, x}{\Gamma \vdash M \,/\, e'\, x} \quad \text{If } e \leq e'$$

$$(linear =_{\mathrm{def}} relevant \wedge affine)$$

Figure A.4: Rules for the expression properties

$\equiv$ is a congruence

$$
\begin{aligned}
(let\, y = (let\, x = M\ in\, N)\ in\, O) &\equiv (let\, x = M\ in\ let\, y = N\ in\, O) && \text{(comp)} \\
() &\equiv x && (1.\eta) \\
\pi_i(x_1, x_2) &\equiv x_i && (\times.\beta) \\
(\pi_1(x), \pi_2(x)) &\equiv x && (\times.\eta)
\end{aligned}
$$

$$\frac{M \equiv N \qquad M\,!\,E}{N\,!\,E}$$

$$\frac{\Gamma \vdash M\,!\,E \qquad \Gamma, x : A \vdash N \,/\, \Phi(E)\, x}{\Gamma \vdash (let\, x = M\ in\, N) \equiv N[x := M] : B}$$

$$\frac{(let\, x = M\ in\, (x, x)) \equiv (M, M)}{M\,!\,copyable} \qquad \frac{(let\, x = M\ in\, ()) \equiv ()}{M\,!\,discardable}$$

Figure A.5: Remaining rules

Appendix B

The let-calculus as an internal language

This chapter contains the proof of the theorems in Section 3.6.

B.1 Proof of Theorem 3.2 (soundness)

Because of Rule 3.1, the let-calculus is very powerful in that it enables very short proofs. Therefore, proving soundness is harder than proving completeness. However, reading the soundness proof should be worthwhile, because it improves fluency in reasoning about realistic call-by-value languages.

Lemma B.1. *The rules in Figure 3.2 hold in every precartesian category.*

Proof. This follows immediately from the closure properties of the classes of central morphisms, copyable morphisms, discardable morphisms, and their intersections (see Lemma 2.1 and Proposition 3.1). □

Lemma B.2. *Let K be a precartesian category, let $(x_1 : A_1, \ldots x_n : A_n \vdash M : B)$ be a sequent of the let-language, let $(y_1 : B_1, \ldots, y_m : B_m)$ be an environment, and let $s : \{1, \ldots, n\} \longrightarrow \{1, \ldots, m\}$ be a function such that $B_{s_i} = A_i$. Then in K it holds that*

$$\begin{aligned}&[\![y_1 : B_1, \ldots y_m : B_n \vdash M[x_1 := y_{s_1}, \ldots, x_n := y_{s_n}] : A]\!]\\ &= \left(B_1 \ldots B_n \xrightarrow{h_s} A_1 \ldots A_n \xrightarrow{[\![x_1:A_1,\ldots,x_n:A_n \vdash M:B]\!]} B\right)\end{aligned}$$

where h_s is the evident morphism built from diagonals and projections.

Proof. By induction over $(x_1 : A_1, \ldots x_n : A_n \vdash M : B)$. □

Corollary B.1 (Semantics of weakening). *Let K be a precartesian category, let $(\Gamma \vdash M : A)$ be a sequent, let Δ be an environment that contains all variables* *of Γ. Then in K it holds that*

$$[\![\Delta \vdash M : A]\!] = \left(\Delta \xrightarrow{p} \Gamma \xrightarrow{[\![\Gamma \vdash M:A]\!]} A\right)$$

where $p : \Delta \longrightarrow \Gamma$ is the evident morphism built from projections.

Lemma B.3. *In every precartesian category, for constants f of fitting type, it holds that:*

$$\begin{aligned}&f(x_1, \ldots, x_k, M_1, \ldots, M_l, N_1, \ldots, N_m)\\ &\equiv (\mathit{let}\, y_1 = M_1 \ldots y_l = M_l\, \mathit{in}\, f(x_1, \ldots, x_k, y_1, \ldots, y_l, N_1, \ldots, N_m))\end{aligned}$$

Proof. By induction over l. □

Lemma B.4. *The rules in Figure 3.7, as well as the following rule, hold in every precartesian category.*

$$M \equiv (\mathit{let}\, x = M\, \mathit{in}\, x) \qquad (\mathit{id})$$

Proof. Straightforward. □

It remains to prove that Rule 3.1 is sound. First we shall collect another few lemmas:

Lemma B.5. *Let K be a precartesian category, and let $(\Gamma \vdash M : A)$ be a sequent whose denotation is central. Then for all sequents $(\Gamma \vdash N : B)$ and $(\Gamma, x : A, y : B \vdash O : C)$ it holds in K that*

$$\Gamma \vdash (\mathit{let}\, x = M\, \mathit{in}\, \mathit{let}\, y = N\, \mathit{in}\, O) \equiv (\mathit{let}\, y = N\, \mathit{in}\, \mathit{let}\, x = M\, \mathit{in}\, O) : C$$

Proof. Let

$$\begin{aligned}(f : \Gamma \longrightarrow A) &=_{\mathrm{def}} K[\![(\Gamma \vdash M : A)]\!]\\ (g : \Gamma \longrightarrow B) &=_{\mathrm{def}} K[\![(\Gamma \vdash N : B)]\!]\\ (h : \Gamma AB \longrightarrow C) &=_{\mathrm{def}} K[\![\Gamma, x : A, y : B \vdash O : C]\!]\end{aligned}$$

Let $\Delta : \Gamma \longrightarrow \Gamma\Gamma\Gamma$ be the obvious diagonal morphism. Then the two morphisms

$$K[\![\Gamma \vdash (\mathit{let}\, x = M\, \mathit{in}\, \mathit{let}\, y = N\, \mathit{in}\, O) : C]\!]$$
$$K[\![\Gamma \vdash (\mathit{let}\, y = N\, \mathit{in}\, \mathit{let}\, x = M\, \mathit{in}\, O) : C]\!]$$

are equal to, respectively,

$$\Gamma \xrightarrow{\Delta} \Gamma\Gamma\Gamma \xrightarrow{\Gamma f \Gamma} \Gamma A \Gamma \xrightarrow{\Gamma A g} \Gamma AB \xrightarrow{h} C$$
$$\Gamma \xrightarrow{\Delta} \Gamma\Gamma\Gamma \xrightarrow{\Gamma\Gamma g} \Gamma\Gamma B \xrightarrow{\Gamma f B} \Gamma AB \xrightarrow{h} C$$

Obviously, the two agree if f is central. □

Lemma B.6. *In every precartesian category it holds that*

$$(\mathit{let}\, x = y\, \mathit{in}\, M) \equiv M[x := y]$$

Proof. This follows easily from Lemma B.2. □

Lemma B.7. *In every precartesian category, if $z \notin FV(M)$, it holds that*

$$\begin{aligned}&(\mathit{let}\, z = (y_1, \ldots, y_n)\, \mathit{in}\, \mathit{let}\, z_1 = p_1(z) \ldots z_n = p_n(z)\, \mathit{in}\, M)\\ &\equiv (\mathit{let}\, z_1 = y_1 \ldots z_n = y_1\, \mathit{in}\, M)\end{aligned}$$

Proof. Straightforward (using Corollary B.1). □

Lemma B.8. *In every precartesian category, if the denotation of M is copyable and $n \geq 1$, it holds that*

$$(\mathit{let}\, x = M\, \mathit{in}\, \underbrace{(x, \ldots, x)}_{n\,\text{times}}) \equiv (M, \ldots, M)$$

Proof. $n = 1$: By Rule (id).
$n \rightsquigarrow n + 1$:

$$\begin{aligned}
&(\mathit{let}\, x = M\, \mathit{in}\, ((x, \ldots, x), x))\\
&\equiv (\mathit{let}\, x = M\, \mathit{in}\, \mathit{let}\, y = (x, \ldots, x)\, \mathit{in}\, (y, x)) \qquad \text{(Lemma B.3)}\\
&\equiv (\mathit{let}\, x = M\, \mathit{in}\, \mathit{let}\, r = x\, \mathit{in}\, \mathit{let}\, s = x\, \mathit{in}\, \mathit{let}\, y = (r, \ldots, r)\, \mathit{in}\, (y, s))\\
&\quad \text{(Lemma B.6)}\\
&\equiv (\mathit{let}\, x = M\, \mathit{in}\, \mathit{let}\, z = (x, x)\, \mathit{in}\, \mathit{let}\, r = \pi_1(z)\, \mathit{in}\, \mathit{let}\, s = \pi_2(z)\, \mathit{in}\\
&\quad \mathit{let}\, y = (r, \ldots, r)\, \mathit{in}\, (y, s)) \qquad \text{(Lemma B.7)}\\
&\equiv (\mathit{let}\, z = (\mathit{let}\, x = M\, \mathit{in}\, (x, x))\, \mathit{in}\, \mathit{let}\, r = \pi_1(z)\, \mathit{in}\, \mathit{let}\, s = \pi_2(z)\, \mathit{in}\\
&\quad \mathit{let}\, y = (r, \ldots, r)\, \mathit{in}\, (y, s)) \qquad \text{(Rule (comp))}\\
&\equiv (\mathit{let}\, z = (M, M)\, \mathit{in}\, \mathit{let}\, r = \pi_1(z)\, \mathit{in}\, \mathit{let}\, s = \pi_2(z)\, \mathit{in}\\
&\quad \mathit{let}\, y = (r, \ldots, r)\, \mathit{in}\, (y, s))\\
&\quad \text{(because the denotation of } M \text{ is copyable)}\\
&\equiv (\mathit{let}\, z = (\mathit{let}\, u = M\, \mathit{in}\, \mathit{let}\, v = M\, \mathit{in}\, (u, v))\, \mathit{in}\, \mathit{let}\, r = \pi_1(z)\, \mathit{in}\\
&\quad \mathit{let}\, s = \pi_2(z)\, \mathit{in}\, \mathit{let}\, y = (r, \ldots, r)\, \mathit{in}\, (y, s)) \qquad \text{(Lemma B.3)}\\
&\equiv (\mathit{let}\, u = M\, \mathit{in}\, \mathit{let}\, v = M\, \mathit{in}\, \mathit{let}\, z = (u, v)\, \mathit{in}\, \mathit{let}\, r = \pi_1(z)\, \mathit{in}\\
&\quad \mathit{let}\, s = \pi_2(z)\, \mathit{in}\, \mathit{let}\, y = (r, \ldots, r)\, \mathit{in}\, (y, s)) \qquad \text{(Rule (comp))}\\
&\equiv (\mathit{let}\, u = M\, \mathit{in}\, \mathit{let}\, v = M\, \mathit{in}\, \mathit{let}\, r = u\, \mathit{in}\\
&\quad \mathit{let}\, s = v\, \mathit{in}\, \mathit{let}\, y = (r, \ldots, r)\, \mathit{in}\, (y, s)) \qquad \text{(Lemma B.7)}\\
&\equiv (\mathit{let}\, u = M\, \mathit{in}\, \mathit{let}\, v = M\, \mathit{in}\, \mathit{let}\, y = (u, \ldots, u)\, \mathit{in}\, (y, v))\\
&\quad \text{(Lemma B.6)}\\
&\equiv (\mathit{let}\, u = M\, \mathit{in}\, \mathit{let}\, y = (u, \ldots, u)\, \mathit{in}\, \mathit{let}\, v = M\, \mathit{in}\, (y, v))\\
&\quad \text{(because the denotation of } (u, \ldots, u) \text{ is central)}
\end{aligned}$$

$\equiv (\mathit{let}\, y = (\mathit{let}\, u = M \;\mathit{in}\; (u, \ldots, u))\;\mathit{in}\;\mathit{let}\, v = M \;\mathit{in}\; (y, v))$
(Rule (comp))
$\equiv (\mathit{let}\, y = (M, \ldots, M)\;\mathit{in}\;\mathit{let}\, v = M \;\mathit{in}\; (y, v))$
(by induction hypothesis)
$\equiv ((M, \ldots, M), M)$ (Lemma B.3) □

Now finally the soundness of Rule 3.1. We have to check eight cases, one for each corner of the precartesian cube. Although the eight checks have similarities I avoided cross-references to keep reading easy.

Lemma B.9. *Rule 3.1 holds in every precartesian category.*

Proof. $\boxed{E = \mathit{arbitrary},\, e = \mathit{clear} \wedge \mathit{linear}:}$
$N = x$: Because of Rule (id).
$N = y \neq x$: Because $(\Gamma, x \vdash y \,/\, (\mathit{clear} \wedge \mathit{linear})\, x)$ is false.
$N = ()$: Because $(\Gamma, x \vdash () \,/\, (\mathit{clear} \wedge \mathit{linear})\, x)$ is false.
$N = (\mathit{let}\, y = P \;\mathit{in}\; Q)$:

If $x \notin FV(Q)$:
$(\mathit{let}\, x = M \;\mathit{in}\;\mathit{let}\, y = P \;\mathit{in}\; Q)$
$\equiv (\mathit{let}\, y = (\mathit{let}\, x = M \;\mathit{in}\; P)\;\mathit{in}\; Q))$ (by Rule (comp))
$\equiv (\mathit{let}\, y = P[x := M]\;\mathit{in}\; Q)$
(because $(\Gamma, x \vdash P \,/\, x\, (\mathit{clear} \wedge \mathit{linear}))$ is true)
$\equiv (\mathit{let}\, y = P \;\mathit{in}\; Q)[x := M]$

If $x \in FV(Q)$ (and therefore $x \notin FV(P)$):
$(\mathit{let}\, x = M \;\mathit{in}\;\mathit{let}\, y = P \;\mathit{in}\; Q)$
$\equiv (\mathit{let}\, y = P \;\mathit{in}\;\mathit{let}\, x = M \;\mathit{in}\; Q)$
(because the denotation of P is central)
$\equiv (\mathit{let}\, y = P \;\mathit{in}\; (Q[x := M]))$
(because $(\Gamma, x, y \vdash Q \,/\, x\, (\mathit{clear} \wedge \mathit{linear}))$ is true)
$\equiv (\mathit{let}\, y = P \;\mathit{in}\; Q)[x := M]$

$N = f(M_1, \ldots, M_n)$:

Let i be the j such that $x \in FV(M_j)$. Then
$(\mathit{let}\, x = M \;\mathit{in}\; f(M_1, \ldots, M_n))$
$\equiv (\mathit{let}\, x = M \;\mathit{in}\;\mathit{let}\, x_1 = M_1 \ldots x_i = M_i \;\mathit{in}$
$f(x_1, \ldots, x_i, M_{i+1}, \ldots, M_n))$ (by Lemma B.3)
$\equiv (\mathit{let}\, x_1 = M_1 \ldots x_{i-1} = M_{i-1} \;\mathit{in}\;\mathit{let}\, x = M \;\mathit{in}\;\mathit{let}\, x_i = M_i \;\mathit{in}$
$f(x_1, \ldots, x_i, M_{i+1}, \ldots, M_n))$
(because $(\Gamma, x \vdash M_j \,!\, \mathit{central})$ is true for $j < i$.)
$\equiv (\mathit{let}\, x_1 = M_1 \ldots x_{i-1} = M_{i-1} \;\mathit{in}\;\mathit{let}\, x_i = (\mathit{let}\, x = M \;\mathit{in}\; M_i)\;\mathit{in}$

$f(x_1, \ldots, x_i, M_{i+1}, \ldots, M_n))$ (by Rule (comp))

$\equiv f(M_1, \ldots, M_{i-1}, (\mathit{let}\, x = M \mathit{\ in}\ M_i), M_{i+1}, \ldots, M_n)$ (by Lemma B.3)

$\equiv f(M_1, \ldots, M_{i-1}, M_i[x := M], M_{i+1}, \ldots, M_n)$

(because $(\Gamma, x \vdash M_i \,/\, x\, (\mathit{clear} \wedge \mathit{linear}))$ is true)

$\equiv f(M_1, \ldots, M_{i-1}, M_i, M_{i+1}, \ldots, M_n)[x := M]$

$N = (M_1, M_2)$ and $N = \pi_i(P)$: Special cases of $N = f(M_1, \ldots, M_n)$.

$\boxed{E = \mathit{discardable},\ e = \mathit{clear} \wedge \mathit{affine}:}$

$N = ()$: Because the denotation of M is discardable.

$N = x$: Because of Rule (id).

$N = y \neq x$:

$(\mathit{let}\, x = M \mathit{\ in}\ y)$

$\equiv (\mathit{let}\, x = M \mathit{\ in\ let}\, z = () \mathit{\ in}\ y)$

$\equiv (\mathit{let}\, z = (\mathit{let}\, x = M \mathit{\ in}\ ()) \mathit{\ in}\ y)$ (Rule (comp))

$\equiv (\mathit{let}\, z = () \mathit{\ in}\ y)$ (because the denotation of M is discardable)

$\equiv y$

$N = (\mathit{let}\, y = P \mathit{\ in}\ Q)$:

If $x \notin FV(Q)$:

$(\mathit{let}\, x = M \mathit{\ in\ let}\, y = P \mathit{\ in}\ Q)$

$\equiv (\mathit{let}\, y = (\mathit{let}\, x = M \mathit{\ in}\ P) \mathit{\ in}\ Q))$ (by Rule (comp))

$\equiv (\mathit{let}\, y = P[x := M] \mathit{\ in}\ Q)$

(because $(\Gamma, x \vdash P \,/\, x\, (\mathit{clear} \wedge \mathit{affine}))$ is true)

$\equiv (\mathit{let}\, y = P \mathit{\ in}\ Q)[x := M]$

If $x \in FV(Q)$ (and therefore $x \notin FV(P)$):

$(\mathit{let}\, x = M \mathit{\ in\ let}\, y = P \mathit{\ in}\ Q)$

$\equiv (\mathit{let}\, y = P \mathit{\ in\ let}\, x = M \mathit{\ in}\ Q)$

(because the denotation of P is central)

$\equiv (\mathit{let}\, y = P \mathit{\ in}\ (Q[x := M]))$

(because $(\Gamma, x, y \vdash Q \,/\, x\, (\mathit{clear} \wedge \mathit{affine}))$ is true)

$\equiv (\mathit{let}\, y = P \mathit{\ in}\ Q)[x := M]$

$N = f(M_1, \ldots, M_n)$:

Suppose that $x \in FV(M_i)$. Then

$(\mathit{let}\, x = M \mathit{\ in}\ f(M_1, \ldots, M_n))$

$\equiv (\mathit{let}\, x = M \mathit{\ in\ let}\, x_1 = M_1 \ldots x_i = M_i \mathit{\ in}$

$f(x_1, \ldots, x_i, M_{i+1}, \ldots, M_n))$ (by Lemma B.3)

$\equiv (\mathit{let}\, x_1 = M_1 \ldots x_{i-1} = M_{i-1} \mathit{\ in\ let}\, x = M \mathit{\ in\ let}\, x_i = M_i \mathit{\ in}$

$f(x_1, \ldots, x_i, M_{i+1}, \ldots, M_n))$

(because $(\Gamma, x \vdash M_j \,!\, \mathit{central})$ is true for $j < i$.)

$\equiv (\mathit{let}\, x_1 = M_1 \ldots x_{i-1} = M_{i-1} \;\mathit{in}\; \mathit{let}\, x_i = (\mathit{let}\, x = M \;\mathit{in}\; M_i) \;\mathit{in}$

$f(x_1, \ldots, x_i, M_{i+1}, \ldots, M_n))$ (by Rule (comp))

$\equiv f(M_1, \ldots, M_{i-1}, (\mathit{let}\, x = M \;\mathit{in}\; M_i), M_{i+1}, \ldots, M_n)$ (by Lemma B.3)

$\equiv f(M_1, \ldots, M_{i-1}, M_i[x := M], M_{i+1}, \ldots, M_n)$

(because $(\Gamma, x \vdash M_i \,/\, x\, (\mathit{clear} \wedge \mathit{affine}))$ is true)

$\equiv f(M_1, \ldots, M_{i-1}, M_i, M_{i+1}, \ldots, M_n)[x := M]$

Suppose that $\forall i. x \notin FV(M_i)$. Then

$(\mathit{let}\, x = M \;\mathit{in}\; f(M_1, \ldots, M_n))$

$\equiv (\mathit{let}\, x = M \;\mathit{in}\; \mathit{let}\, z = () \;\mathit{in}\; f(M_1, \ldots, M_n))$

$\equiv (\mathit{let}\, z = (\mathit{let}\, x = M \;\mathit{in}\; ()) \;\mathit{in}\; f(M_1, \ldots, M_n))$ (Rule (comp))

$\equiv (\mathit{let}\, z = () \;\mathit{in}\; f(M_1, \ldots, M_n))$

(because the denotation of M is discardable)

$\equiv f(M_1, \ldots, M_n)$

$N = (M_1, M_2)$ and $N = \pi_i(P)$: Special cases of $N = f(M_1, \ldots, M_n)$.

$\boxed{E = \mathit{central},\ e = \mathit{linear}:}$

$N = x$: Because of Rule (id).

$N = y \neq x$: Because $(\Gamma, x \vdash y \,/\, \mathit{linear}\, x)$ is false.

$N = ()$: Because $(\Gamma, x \vdash () \,/\, \mathit{linear}\, x)$ is false.

$N = (\mathit{let}\, y = P \;\mathit{in}\; Q)$:

If $x \notin FV(Q)$:

$(\mathit{let}\, x = M \;\mathit{in}\; \mathit{let}\, y = P \;\mathit{in}\; Q)$

$\equiv (\mathit{let}\, y = (\mathit{let}\, x = M \;\mathit{in}\; P) \;\mathit{in}\; Q))$ (by Rule (comp))

$\equiv (\mathit{let}\, y = P[x := M] \;\mathit{in}\; Q)$ (because $(\Gamma, x \vdash P \,/\, x\; \mathit{linear})$ is true)

$\equiv (\mathit{let}\, y = P \;\mathit{in}\; Q)[x := M]$

If $x \in FV(Q)$ (and therefore $x \notin FV(P)$):

$(\mathit{let}\, x = M \;\mathit{in}\; \mathit{let}\, y = P \;\mathit{in}\; Q)$

$\equiv (\mathit{let}\, y = P \;\mathit{in}\; \mathit{let}\, x = M \;\mathit{in}\; Q)$

(because the denotation of M is central)

$\equiv (\mathit{let}\, y = P \;\mathit{in}\; (Q[x := M]))$ (because $(\Gamma, x, y \vdash Q \,/\, x\; \mathit{linear})$ is true)

$\equiv (\mathit{let}\, y = P \;\mathit{in}\; Q)[x := M]$

$N = f(M_1, \ldots, M_n)$:

Suppose that $x \in FV(M_i)$ (and therefore not in any other $FV(M_j)$). Then

$(\mathit{let}\, x = M \;\mathit{in}\; f(M_1, \ldots, M_n)$

$\equiv (\mathit{let}\, x = M \;\mathit{in}\; \mathit{let}\, x_1 = M_1 \ldots x_i = M_i \;\mathit{in}$

$f(x_1, \ldots, x_i, M_{i+1}, \ldots, M_n))$ (by Lemma B.3)

$\equiv (\mathit{let}\, x_1 = M_1 \ldots x_{i-1} = M_{i-1}\, \mathit{in}\, \mathit{let}\, x = M\, \mathit{in}\, \mathit{let}\, x_i = M_i\, \mathit{in}$
$f(x_1, \ldots, x_i, M_{i+1}, \ldots, M_n))$
(because the denotation of M is central)
$\equiv (\mathit{let}\, x_1 = M_1 \ldots x_{i-1} = M_{i-1}\, \mathit{in}\, \mathit{let}\, x_i = (\mathit{let}\, x = M\, \mathit{in}\, M_i)\, \mathit{in}$
$f(x_1, \ldots, x_i, M_{i+1}, \ldots, M_n))$ (by Rule (comp))
$\equiv f(M_1, \ldots, M_{i-1}, \mathit{let}\, x = M\, \mathit{in}\, M_i, M_{i+1}, \ldots, M_n)$ (by Lemma B.3)
$\equiv f(M_1, \ldots, M_{i-1}, M_i[x := M], M_{i+1}, \ldots, M_n)$
(because $(\Gamma, x \vdash M_i \,/\, x\, \mathit{linear}$ is true)
$\equiv f(M_1, \ldots, M_{i-1}, M_i, M_{i+1}, \ldots, M_n)[x := M]$
$N = (M_1, M_2)$ and $N = \pi_i(P)$: Special cases of $N = f(M_1, \ldots, M_n)$.

$\boxed{E = \mathit{copyable},\, e = \mathit{clear} \wedge \mathit{relevant}\text{:}}$
$N = x$: By Rule (id).
$N = y \neq x$: because $(\Gamma, x \vdash y \,/\, x\, (\mathit{clear} \wedge \mathit{relevant}))$ is false.
$N = ()$: because $(\Gamma, x \vdash () \,/\, x\, (\mathit{clear} \wedge \mathit{relevant}))$ is false.
$N = (\mathit{let}\, y = P\, \mathit{in}\, Q)$:
If $x \notin FV(Q)$:
$(\mathit{let}\, x = M\, \mathit{in}\, \mathit{let}\, y = P\, \mathit{in}\, Q)$
$\equiv (\mathit{let}\, y = (\mathit{let}\, x = M\, \mathit{in}\, P)\, \mathit{in}\, Q))$ (by Rule (comp))
$\equiv (\mathit{let}\, y = P[x := M]\, \mathit{in}\, Q)$
(because $(\Gamma, x \vdash P \,/\, x\, (\mathit{clear} \wedge \mathit{relevant}))$ is true)
$\equiv (\mathit{let}\, y = P\, \mathit{in}\, Q)[x := M]$
If $x \notin FV(P)$:
$(\mathit{let}\, x = M\, \mathit{in}\, \mathit{let}\, y = P\, \mathit{in}\, Q)$
$\equiv (\mathit{let}\, y = P\, \mathit{in}\, \mathit{let}\, x = M\, \mathit{in}\, Q)$
(because the denotation of P is central)
$\equiv (\mathit{let}\, y = P\, \mathit{in}\, (Q[x := M]))$
(because $(\Gamma, x, y \vdash Q \,/\, x\, (\mathit{clear} \wedge \mathit{relevant}))$ is true)
$\equiv (\mathit{let}\, y = P\, \mathit{in}\, Q)[x := M]$
If $x \in FV(P)$ and $x \in FV(Q)$:
$(\mathit{let}\, x = M\, \mathit{in}\, \mathit{let}\, y = P\, \mathit{in}\, Q)$
$\equiv (\mathit{let}\, x = M\, \mathit{in}\, \mathit{let}\, r = x\, \mathit{in}\, \mathit{let}\, s = x\, \mathit{in}$
$\mathit{let}\, y = P[x := r]\, \mathit{in}\, (Q[x := s]))$ (Lemma B.6)
$\equiv (\mathit{let}\, x = M\, \mathit{in}\, \mathit{let}\, z = (x, x)\, \mathit{in}\, \mathit{let}\, r = \pi_1(z)\, \mathit{in}\, \mathit{let}\, s = \pi_2(z)\, \mathit{in}$
$\mathit{let}\, y = P[x := r]\, \mathit{in}\, (Q[x := s]))$ (Lemma B.7)
$\equiv (\mathit{let}\, z = (\mathit{let}\, x = M\, \mathit{in}\, (x, x))\, \mathit{in}\, \mathit{let}\, r = \pi_1(z)\, \mathit{in}\, \mathit{let}\, s = \pi_2(z)\, \mathit{in}$
$\mathit{let}\, y = P[x := r]\, \mathit{in}\, (Q[x := s]))$ (by Rule (comp))

$\equiv (let\, z = (M, M)\; in\; let\, r = \pi_1(z)\; in\; let\, s = \pi_2(z)\; in$
$let\, y = P[x := r]\; in\; (Q[x := s]))$
(because the denotation of M is copyable)

$\equiv (let\, z = (let\, u = M\; in\; let\, v = M\; in\; (u, v))\; in\; let\, r = \pi_1(z)\; in$
$let\, s = \pi_2(z)\; in\; let\, y = P[x := r]\; in\; (Q[x := s]))$ (by Lemma B.3)

$\equiv (let\, u = M\; in\; let\, v = M\; in\; let\, z = (u, v)\; in\; let\, r = \pi_1(z)\; in$
$let\, s = \pi_2(z)\; in\; let\, y = P[x := r]\; in\; (Q[x := s]))$ (by Rule (comp))

$\equiv (let\, u = M\; in\; let\, v = M\; in\; let\, r = u\; in\; let\, s = v\; in$
$let\, y = P[x := r]\; in\; (Q[x := s]))$ (Lemma B.7)

$\equiv (let\, u = M\; in\; let\, v = M\; in\; let\, y = P[x := u]\; in\; (Q[x := v]))$
(Lemma B.6)

$\equiv (let\, u = M\; in\; let\, y = P[x := u]\; in\; let\, v = M\; in\; (Q[x := v]))$
(because the denotation of P is central, and therefore the denotation of $P[x := u]$ is central too)

$\equiv (let\, y = (let\, u = M\; in\; (P[x := u]))\; in\; let\, v = M\; in\; (Q[x := v]))$
(by Rule (comp))

$\equiv (let\, y = (let\, x = M\; in\; P)\; in\; let\, x = M\; in\; Q)$

$\equiv (let\, y = P[x := M]\; in\; Q[x := M])$
(because $(\Gamma, x \vdash P \,/\, x\, (clear \wedge relevant)$ is true and $(\Gamma, x, y \vdash Q \,/\, x\, (clear \wedge relevant))$ is true)

$\equiv (let\, y = P\; in\; Q)[x := M]$

$N = f(M_1, \ldots, M_n)$:

Let $j_1, \ldots, j_k$ be the indices i such that $x \in FV(M_i)$. Then

$(let\, x = M\; in\; f(M_1, \ldots, M_n)$

$\equiv (let\, x = M\; in\; let\, x_1 = M_1 \ldots x_n = M_n\; in$
$f(x_1, \ldots, x_n))$ (by Lemma B.3)

$\equiv (let\, x = M\; in\; let\, y_1 = x \ldots y_k = x\; in\; let\, x_1 = M'_1 \ldots x_n = M'_n\; in$
$f(x_1, \ldots, x_n))$
(where $M'_l = M_l[x := y_i]$ if $l = j_i$ and $M'_l = M_l$ otherwise; holds by Lemma B.6)

$\equiv (let\, x = M\; in\; let\, z = (x, \ldots, x)\; in\; let\, y_1 = p_1(z) \ldots y_k = p_k(z)\; in$
$let\, x_1 = M'_1 \ldots x_n = M'_n\; in\; f(x_1, \ldots, x_n))$ (Lemma B.7)

$\equiv (let\, z = (let\, x = M\; in\; (x, \ldots, x))\; in\; let\, y_1 = p_1(z) \ldots y_k = p_k(z)\; in$
$let\, x_1 = M'_1 \ldots x_n = M'_n\; in\; f(x_1, \ldots, x_n))$ (by Rule (comp))

$\equiv (let\, z = (M, \ldots, M)\; in\; let\, y_1 = p_1(z) \ldots y_k = p_k(z)\; in$
$let\, x_1 = M'_1 \ldots x_n = M'_n\; in\; f(x_1, \ldots, x_n))$ (Lemma B.8)

$\equiv (let\, z = (let\, z_1 = M \ldots z_k = M\; in\; (z_1, \ldots, z_k))\; in$
$let\, y_1 = p_1(z) \ldots y_k = p_k(z)\; in$
$let\, x_1 = M'_1 \ldots x_n = M'_n\; in\; f(x_1, \ldots, x_n))$ (Lemma B.3)

$\equiv (let\, z_1 = M \ldots z_k = M\ in\ let\, z = (z_1, \ldots, z_k)\ in$
$let\, y_1 = p_1(z) \ldots y_k = p_k(z)\ in$
$let\, x_1 = M'_1 \ldots x_n = M'_n\ in\ f(x_1, \ldots, x_n))$ (Rule (comp))

$\equiv (let\, z_1 = M \ldots z_k = M\ in\ let\, y_1 = z_1 \ldots y_k = z_k\ in$
$let\, x_1 = M'_1 \ldots x_n = M'_n\ in\ f(x_1, \ldots, x_n))$ (Lemma B.7)

$\equiv (let\, z_1 = M \ldots z_k = M\ in\ let\, x_1 = M'_1[y_1 := z_1, \ldots y_k := z_k] \ldots$
$x_n = M'_n[y_1 := z_1, \ldots y_k := z_k]\ in\ f(x_1, \ldots, x_n))$ (Lemma B.6)

$\equiv (let\, y_1 = M \ldots y_k = M\ in\ let\, x_1 = M'_1 \ldots x_n = M'_n\ in\ f(x_1, \ldots, x_n))$

$\equiv (let\, x_1 = M''_1 \ldots x_n = M''_n\ in\ f(x_1, \ldots, x_n))$
(where $M''_l = (let\, y_i = M\ in\ M'_l)$ if $l = j_i$ for some i, and $M''_l = M'_l = M_l$ otherwise. This works because the M_l, and therefore the M'_l, are central for $l < j_k$ and we have Rule (comp).)

$\equiv f(M''_1, \ldots, M''_n)$ (Lemma B.3)

$\equiv f(M'''_1, \ldots, M'''_n)$
(where $M'''_l = M'_l[y_i := M]$ if $l = j_i$ for some i, and $M'''_l = M''_l = M'_l = M_l$ otherwise. This works because for $l = j_i$ it is true that $(\Gamma \vdash M_l \,/\, (clear \wedge relevant)\, x)$, and therefore it is true that $(\Gamma \vdash M'_l \,/\, (clear \wedge relevant)\, y_i)$.)

$\equiv f(M_1, \ldots, M_n)[x := M]$
(because $M'_l[y_i := M] = M_l[x := M]$ for $l = j_i$.)

$N = (M_1, M_2)$ and $N = \pi_i(P)$: Special cases of $N = f(M_1, \ldots, M_n)$.

$\boxed{E = central \wedge copyable,\ e = relevant:}$

$N = x$: Because of Rule (comp).

$N = y \neq x$: Because $(\Gamma \vdash y \,/\, x\ relevant)$ is false.

$N = ()$: Because $(\Gamma \vdash () \,/\, x\ relevant)$ is false.

$N = (let\, y = P\ in\ Q)$:

If $x \notin FV(Q)$:

$(let\, x = M\ in\ let\, y = P\ in\ Q)$
$\equiv (let\, y = (let\, x = M\ in\ P)\ in\ Q))$ (by Rule (comp))
$\equiv (let\, y = P[x := M]\ in\ Q)$ (because $(\Gamma, x \vdash P \,/\, x\ relevant)$ is true)
$\equiv (let\, y = P\ in\ Q)[x := M]$

If $x \notin FV(P)$:

$(let\, x = M\ in\ let\, y = P\ in\ Q)$
$\equiv (let\, y = P\ in\ let\, x = M\ in\ Q)$ (because the denotation of M is central)
$\equiv (let\, y = P\ in\ (Q[x := M]))$ (because $(\Gamma, x \vdash Q \,/\, x\ relevant)$ is true)
$\equiv (let\, y = P\ in\ Q)[x := M]$

If $x \in FV(P)$ and $x \in FV(Q)$:

$(let\, x = M\ in\ let\, y = P\ in\ Q)$

$\equiv (let\, x = M\; in\; let\, r = x\; in\; let\, s = x\; in$
$let\, y = P[x := r]\; in\; Q[x := s])$ (Lemma B.6)
$\equiv (let\, x = M\; in\; let\, z = (x, x)\; in\; let\, r = \pi_1(z)\; in\; let\, s = \pi_2(z)\; in$
$let\, y = P[x := r]\; in\; (Q[x := s]))$ (Lemma B.7)
$\equiv (let\, z = (let\, x = M\; in\; (x, x))\; in\; let\, r = \pi_1(z)\; in\; let\, s = \pi_2(z)\; in$
$let\, y = P[x := r]\; in\; (Q[x := s]))$ (by Rule (comp))
$\equiv (let\, z = (M, M)\; in\; let\, r = \pi_1(z)\; in\; let\, s = \pi_2(z)\; in$
$let\, y = P[x := r]\; in\; (Q[x := s]))$
(because the denotation of M is copyable)
$\equiv (let\, z = (let\, u = M\; in\; let\, v = M\; in\; (u, v))\; in\; let\, r = \pi_1(z)\; in$
$let\, s = \pi_2(z)\; in\; let\, y = P[x := r]\; in\; (Q[x := s]))$ (by Rule B.3)
$\equiv (let\, u = M\; in\; let\, v = M\; in\; let\, z = (u, v)\; in\; let\, r = \pi_1(z)\; in$
$let\, s = \pi_2(z)\; in\; let\, y = P[x := r]\; in\; (Q[x := s]))$ (by Rule (comp))
$\equiv (let\, u = M\; in\; let\, v = M\; in\; let\, r = u\; in\; let\, s = v\; in$
$let\, y = P[x := r]\; in\; (Q[x := s]))$ (Lemma B.7)
$\equiv (let\, u = M\; in\; let\, v = M\; in\; let\, y = P[x := u]\; in\; (Q[x := v]))$
(Lemma B.6)
$\equiv (let\, u = M\; in\; let\, y = P[x := u]\; in\; let\, v = M\; in\; (Q[x := v]))$
(because the denotation of M is central)
$\equiv (let\, y = (let\, u = M\; in\; (P[x := u]))\; in\; let\, v = M\; in\; (Q[x := v]))$
(by Rule (comp))
$\equiv (let\, y = (let\, x = M\; in\; P)\; in\; let\, x = M\; in\; Q$
$\equiv (let\, y = P[x := M]\; in\; Q[x := M]$
(because it is true that $(\Gamma, x \vdash P \,/\, x\; relevant)$ and $(\Gamma, x, y \vdash Q \,/\, x\; relevant)$)
$\equiv (let\, y = P\; in\; Q)[x := M]$

$N = f(M_1, \ldots, M_n)$:

Let $j_1, \ldots, j_k$ be the indices i such that $x \in FV(M_i)$. Then
$(let\, x = M\; in\; f(M_1, \ldots, M_n))$
$\equiv (let\, x = M\; in\; let\, x_1 = M_1 \ldots x_n = M_n\; in\; f(x_1, \ldots, x_n))$
(by Lemma B.3)
$\equiv (let\, x = M\; in\; let\, y_1 = x \ldots y_k = x\; in\; let\, x_1 = M'_1 \ldots x_n = M'_n\; in$
$f(x_1, \ldots, x_n))$
(where $M'_l = M_l[x := y_i]$ if $l = j_i$ and $M'_l = M_l$ otherwise; holds by Lemma B.6)
$\equiv (let\, x = M\; in\; let\, z = (x, \ldots, x)\; in\; let\, y_1 = p_1(z) \ldots y_k = p_k(z)\; in$
$let\, x_1 = M'_1 \ldots x_n = M'_n\; in\; f(x_1, \ldots, x_n))$ (Lemma B.7)
$\equiv (let\, z = (let\, x = M\; in\; (x, \ldots, x))\; in\; let\, y_1 = p_1(z) \ldots y_k = p_k(z)\; in$

$let\, x_1 = M_1' \ldots x_n = M_n' \; in\, f(x_1, \ldots, x_n))$ (by Rule (comp))

$\equiv (let\, z = (M, \ldots, M)\; in\; let\, y_1 = p_1(z) \ldots y_k = p_k(z)\; in$
$let\, x_1 = M_1' \ldots x_n = M_n'\; in\, f(x_1, \ldots, x_n))$ (Lemma B.8)

$\equiv (let\, z = (let\, z_1 = M \ldots z_k = M\; in\, (z_1, \ldots, z_k))\; in$
$let\, y_1 = p_1(z) \ldots y_k = p_k(z)\; in\; let\, x_1 = M_1' \ldots x_n = M_n'\; in$
$f(x_1, \ldots, x_n))$ (Lemma B.3)

$\equiv (let\, z_1 = M \ldots z_k = M\; in\; let\, z = (z_1, \ldots, z_k)\; in$
$let\, y_1 = p_1(z) \ldots y_k = p_k(z)\; in\; let\, x_1 = M_1' \ldots x_n = M_n'\; in$
$f(x_1, \ldots, x_n))$ (Rule (comp))

$\equiv (let\, z_1 = M \ldots z_k = M\; in\; let\, y_1 = z_1 \ldots y_k = z_k\; in$
$let\, x_1 = M_1' \ldots x_n = M_n'\; in\, f(x_1, \ldots, x_n))$ (Lemma B.7)

$\equiv (let\, z_1 = M \ldots z_k = M\; in\; let\, x_1 = M_1'[y_1 := z_1, \ldots y_k := z_k] \ldots$
$x_n = M_n'[y_1 := z_1, \ldots y_k := z_k]\; in\, f(x_1, \ldots, x_n))$ (Lemma B.6)

$\equiv (let\, y_1 = M \ldots y_k = M\; in\; let\, x_1 = M_1' \ldots x_n = M_n'\; in\, f(x_1, \ldots, x_n))$

$\equiv (let\, x_1 = M_1'' \ldots x_n = M_n''\; in\, f(x_1, \ldots, x_n))$
(where $M_l'' = (let\, y_i = M\; in\, M_l')$ if $l = j_i$ for some i, and $M_l'' = M_l' = M_l$ otherwise. This works because the denotation of M is central and we have the Rule (comp).)

$\equiv f(M_1'', \ldots, M_n'')$ (Lemma B.3)

$\equiv f(M_1''', \ldots, M_n''')$
(where $M_l''' = (M_l'[y_i := M])$ if $l = j_i$ for some i, and $M_l''' = M_l'' = M_l' = M_l$ otherwise. This works because $(\Gamma \vdash M_l \,/\, x\; relevant)$ is true for $l = j_i$ for some i.)

$\equiv f(M_1, \ldots, M_n)[x := M]$ (because $M_l'[y_i := M] = M_l[x := M]$)

$N = (M_1, M_2)$ and $N = \pi_i(P)$: Special cases of $N = f(M_1, \ldots, M_n)$.

$\boxed{E = central \wedge discardable,\; e = affine:}$

Suppose that $(\Gamma, x \vdash N \,/\, linear\, x)$ is true. Then—as shown—it is true that $(let\, x = M\; in\, N) \equiv N[x := M]$ because M is central. If $x \notin FV(N)$, then

$(let\, x = M\; in\, N)$
$\equiv (let\, x = M\; in\; let\, z = ()\; in\, N)$
$\equiv (let\, z = (let\, x = M\; in\, ())\; in\, N)$
$\equiv (let\, z = ()\; in\, N)$
$\equiv N$
$\equiv N[x := M]$

$\boxed{E = focal,\; e = arbitrary:}$

Suppose that $(\Gamma, x \vdash N \,/\, affine\, x)$ is true. Then—as shown—it is true that $(let\,x = M\,in\,N) \equiv N[x := M]$ because M is central and discardable. If $(\Gamma, x \vdash N\,/\,relevant\,x)$ is true, then—as shown—it is true that $(let\,x = M\,in\,N) \equiv N[x := M]$ because M is central and copyable.

$\boxed{E = copyable \wedge discardable,\ e = clear:}$
Suppose that $(\Gamma, x \vdash N\,/\,(clear \wedge affine)\,x)$ is true. Then—as shown—it is true that $(let\,x = M\,in\,N) \equiv N[x := M]$ because M is discardable. If $(\Gamma, x \vdash N\,/\,(clear \wedge relevant)\,x)$ is true, then—as shown—it is true that $(let\,x = M\,in\,N) \equiv N[x := M]$ because M is copyable. $\square$

B.2 Proof of Theorems 3.3 and 3.4 (completeness and initiality)

In this section, we shall define a translation from *precartesian categorical expressions* like $(A \otimes B) \otimes I$ and $f; g; \delta$ into types and sequents, respectively, of the let-language. This translation is the inverse of the semantics in a sense that we shall make precise. From this we shall prove completeness and thus establish the let-calculus as an internal language.

Definition B.1. The *precartesian categorical expressions* over a precartesian signature $\Sigma = (\mathcal{B}, \mathcal{K})$ are defined inductively by

$$f = id_A \,|\, f; f \,|\, A \otimes f \,|\, f \otimes A \,|\, \delta_A \,|\, p_{A,B} \,|\, q_{A,B} \,|\, !_A \,|\, \mathcal{K}$$

where A and B range over the precartesian types over $\mathcal{B}$, subject to the obvious type constraint for semicolon.

Figure 2.1 (which displays the semantics of the let-language) also has a reading as a syntactic translation from sequents into precartesian categorical expressions. Let's write c for this translation. Now we turn to defining the inverse of c—let's call it c'. For each precartesian signature Σ, the translation c' takes each precartesian categorical expression $f : A \longrightarrow B$ to a sequent $(x_1 : A_1, \ldots, x_n : A_n \vdash M : B)$, where $A_1, \ldots, A_n$ is the factorisation of A. It proceeds by recursion over precartesian categorical expressions as in Figure B.1 (where $\overrightarrow{x}$ stands for the obvious tuple formed by the x_i). Lemmas B.10 and B.14 below state that the translations c and c' are essentially inverse.

Lemma B.10. *In every precartesian category, for every precartesian categorical expression m, letting i be the evident iso built from associativity and neutrality maps, it holds that*

$$c(c'(m)) = i; m$$

$$
\begin{array}{ll}
c'(f : A \longrightarrow B) & = x_1 : A_1 \ldots, x_n : A_n \vdash f(\bar{x}) : A \\
\text{if } f \text{ is a constant} & \\
 & \\
c'(f : A \longrightarrow B) & = \Gamma \vdash M : B \\
c'(g : B \longrightarrow C) & = y_1 : B_1, \ldots, y_n : B_n \vdash N : C \\
\cline{1-1}
c'(f ; g : A \longrightarrow C) & = \Gamma \vdash \mathit{let}\, y_1, \ldots, y_n = M \;\mathit{in}\, N : C \\
 & \\
c'(\mathit{id} : A \longrightarrow A) & = x_1 : A_1 \ldots, x_n : A_n \vdash \overrightarrow{x} : A \\
 & \\
c'(f : A \longrightarrow B) & = \Gamma \vdash M : B \\
\cline{1-1}
c'(C \otimes f : C \otimes A \longrightarrow C \otimes B) & = y_1 : C_1, \ldots, y_n : C_n, \Gamma \vdash (\overrightarrow{y}, M) : C * B \\
 & \\
c'(f : A \longrightarrow B) & = \Gamma \vdash M : B \\
\cline{1-1}
c'(f \otimes C : A \otimes C \longrightarrow B \otimes C) & = \Gamma, y_1 : C_1, \ldots, y_n : C_n \vdash (M, \overrightarrow{y}) : B * C \\
 & \\
c'(\delta : A \longrightarrow A \otimes A) & = x_1 : A_1 \ldots, x_n : A_n \vdash (\overrightarrow{x}, \overrightarrow{x}) : A * A \\
 & \\
c'(p : A \otimes B \longrightarrow A) & = x_1 : A_1 \ldots, x_n : A_n, y_1 : B_1, \ldots, y_n : B_m \vdash \overrightarrow{x} : A \\
 & \\
c'(q : A \otimes B \longrightarrow A) & = x_1 : A_1 \ldots, x_n : A_n, y_1 : B_1, \ldots, y_n : B_m \vdash \overrightarrow{y} : B \\
 & \\
c'(! : A \longrightarrow I) & = x_1 : A_1 \ldots, x_n : A_n \vdash () : \mathit{unit}
\end{array}
$$

Figure B.1: Translating precartesian categorical expressions into the let-language

Proof. By induction over m. □

Lemma B.11. *In the let-calculus it is derivable that*

$$let\, x_1, \dots, x_n = y_1, \dots, y_n \, in\, M \equiv M[x_1 := y_1, \dots, x_n := y_n]$$

Lemma B.12. *In the let-calculus, it is derivable that*

$$\begin{aligned}&(x_1, \dots, x_k, M_1, \dots, M_l, N_1, \dots, N_m)\\ &\equiv (let\, y_1 = M_1 \dots y_l = M_l \, in\, (x_1, \dots, x_k, y_1, \dots, y_l, N_1, \dots, N_m))\end{aligned}$$

Moreover, for all constants f of fitting type, it is derivable that

$$\begin{aligned}&f(x_1, \dots, x_k, M_1, \dots, M_l, N_1, \dots, N_m)\\ &\equiv (let\, y_1 = M_1 \dots y_l = M_l \, in\, f(x_1, \dots, x_k, y_1, \dots, y_l, N_1, \dots, N_m))\end{aligned}$$

Lemma B.13. *Let $y_1, \dots, y_l$ be variables such that y_i has type B_i, and let M have type $B_1 * \dots * B_l$. Then in the let-calculus it is derivable that*

$$\begin{aligned}&(let\, x_1, \dots, x_k, y_1, \dots, y_l, z_1, \dots, z_m = (u_1, \dots, u_k, M, v_1, \dots, v_m) \, in\, N)\\ &\equiv (let\, y_1, \dots, y_l = M \, in\, N[x_1 := u_1, \dots, x_k := u_k, z_1 := v_1, \dots, z_m := v_m])\end{aligned}$$

Proof. Straightforward using Lemma B.12. □

Lemma B.14. *Let $(\Gamma \vdash M : A)$ be a sequent such that, letting $(x_1 : A_1, \dots, x_n : A_n) =_{\mathrm{def}} \Gamma$, none of the A_i is a product type or the unit type. Let M' be defined by*

$$(\Gamma \vdash M' : A) =_{\mathrm{def}} c'(c(\Gamma \vdash M : A))$$

Then it is derivable in the let-calculus that

$$\Gamma \vdash M' \equiv M : A$$

Proof. By induction over $(\Gamma \vdash M : A)$, using Lemmas B.11, B.12, and B.13. □

Definition B.2. For a let-theory $\mathcal{T}$ over a precartesian signature Σ, the binary relation $\approx_{\mathcal{T}}$ on the precartesian categorical expressions over Σ is defined as follows: Let $m \approx_{\mathcal{T}} n$ if and only if, letting $(\Gamma \vdash M : A) = c'(m)$ and $(\Gamma \vdash N : A) = c'(n)$, we have $(\Gamma \vdash M \equiv N : A)$ in $\mathcal{T}$.

Proposition B.2 (Term model). *Let $\mathcal{T}$ be a let-theory over a precartesian signature Σ. Then $\approx_{\mathcal{T}}$ is a congruence, and the congruence classes form a model of $\mathcal{T}$.*

Proof. The relation $\approx$ is a congruence because $\equiv$ is a congruence. So we have a graph whose nodes are the precartesian types over Σ, and whose arrows are $\approx_{\mathcal{T}}$-classes of precartesian categorical expressions over Σ, and on this graph we have well-defined operators like sequential composition of arrows, tensor, and so on. To see that we have a precartesian category, by Condition 3 of Proposition 2.1 it suffices to check the following equations: First, the one stating that we have a binoidal category. Second, the equations stating the naturality of the associativity map, the twist map, and the neutrality maps. Third, the three equations for each of δ, p, q, and ! that state focality. Fourth, the equations from Condition 2 of Proposition 2.1. Fifth and last, the equations stating that the twist map is self-inverse and that the associativity maps are copyable and inverse to one each other. Each such equation $m = n$ is checked by deriving in the let-calculus the equation which is the image of $m = n$ under c'. Checking the details is straightforward (use Lemmas B.11, B.12, and B.13).

Let's call our precartesian category $K_{\mathcal{T}}$. It remains to prove that $K_{\mathcal{T}}$ is a model of $\mathcal{T}$. To see this, let $(\Gamma \vdash M \equiv N : A) \in \mathcal{T}$, and let $m =_{\text{def}} c(\Gamma \vdash M : A)$ and $n =_{\text{def}} c(\Gamma \vdash N : A)$. We need to prove that $m \approx_{\mathcal{T}} n$. So let $(\Gamma \vdash M' : A) =_{\text{def}} c'(m)$ and $(\Gamma \vdash N' : A) =_{\text{def}} c'(n)$. We need $(\Gamma \vdash M' \equiv N' : A) \in \mathcal{T}$. This is so because, by Lemma B.14, we have $(\Gamma \vdash M \equiv M' : A)$ and $(\Gamma \vdash N \equiv N' : A)$ in $\mathcal{T}$, and $\equiv$ is transitive. Now for judgements of the form $(\Gamma \vdash M \,!\, E)$. Let $m =_{\text{def}} c(\Gamma \vdash M : A)$. To start with, let $E = \mathit{central}$. We must prove that m is central in $K_{\mathcal{T}}$. So for all $n : \Delta \longrightarrow B$ the diagram

$$\begin{tikzcd}
\Gamma \otimes \Delta \arrow[r, "m \otimes \Delta"] \arrow[d, "\Gamma \otimes n"'] & A \otimes \Delta \arrow[d, "A \otimes n"] \\
\Gamma \otimes B \arrow[r, "m \otimes B"] & A \otimes B
\end{tikzcd}$$

must commute. Letting $(\Gamma \vdash M' : A) =_{\text{def}} c'(m)$ and $(\Delta \vdash N : B) =_{\text{def}} c'(n)$, this amounts to proving that

$$(\mathit{let}\, x = M' \,\mathit{in}\, \mathit{let}\, y = N \,\mathit{in}\, (x, y)) \equiv (\mathit{let}\, y = N \,\mathit{in}\, \mathit{let}\, x = M' \,\mathit{in}\, (x, y))$$

is in $\mathcal{T}$. By Lemma B.14, $(\Gamma \vdash M \equiv M' : A) \in \mathcal{T}$, and therefore it suffices to prove that

$$(\mathit{let}\, x = M \,\mathit{in}\, \mathit{let}\, y = N \,\mathit{in}\, (x, y)) \equiv (\mathit{let}\, y = N \,\mathit{in}\, \mathit{let}\, x = M \,\mathit{in}\, (x, y))$$

is in $\mathcal{T}$. This follows from $(\Gamma, \Delta, x \vdash \mathit{let}\, y = N \,\mathit{in}\, (x, y) \,/\, \mathit{linear}\, x) \in \mathcal{T}$. Similar arguments work for the remaining seven E. Now for judgements of the form $(\Gamma \vdash M \,/\, e\, x)$. The cases $e = \mathit{arbitrary}$, $e = \mathit{affine}$, and $e = \mathit{relevant}$ are trivial. Now let $e = \mathit{clear}$. If $(\Gamma \vdash M \,/\, \mathit{clear}\, x) \in \mathcal{T}$, then for all $(\Delta \vdash N) \in b_x(\Gamma \vdash M)$ we have $(\Delta \vdash N \,!\, \mathit{central}) \in \mathcal{T}$. Therefore, as shown above, the denotation of $(\Delta \vdash N)$ in $K_{\mathcal{T}}$ is central, and therefore $(\Gamma \vdash M \,/\, \mathit{clear}\, x)$ holds in $K_{\mathcal{T}}$. Now suppose that $(\Gamma \vdash M \,/\, (e_1 \wedge e_2)\, x) \in \mathcal{T}$. Because $e_1 \leq e$ and $e_2 \leq e$, we have $(\Gamma \vdash M \,/\, e_1\, x) \in \mathcal{T}$ and $(\Gamma \vdash M \,/\, e_2\, x) \in \mathcal{T}$. Reasoning inductively, we can assume that $(\Gamma \vdash M \,/\, e_1\, x)$ and $(\Gamma \vdash M \,/\, e_1\, x)$ hold in $K_{\mathcal{T}}$. Therefore $(\Gamma \vdash M \,/\, (e_1 \wedge e_2)\, x)$ too holds in $K_{\mathcal{T}}$. □

Definition B.3. For every let-theory $\mathcal{T}$, let $K_{\mathcal{T}}$ be the model induced by $\approx_{\mathcal{T}}$.

Proof of Theorem 3.3. If $(\Gamma \vdash M \equiv N : A)$ holds in every model of $\mathcal{T}$, then it holds in $K_{\mathcal{T}}$. By definition of $K_{\mathcal{T}}$, letting $m =_{\mathrm{def}} c(\Gamma \vdash M : A)$ and $n =_{\mathrm{def}} c(\Gamma \vdash N : A)$, we have $m \approx_{\mathcal{T}} n$. By definition of $\approx_{\mathcal{T}}$, letting $(\Gamma \vdash M' : A) =_{\mathrm{def}} c'(m)$ and $(\Gamma \vdash N' : A) =_{\mathrm{def}} c'(n)$, we have $(\Gamma \vdash M' \equiv N' : A) \in \mathcal{T}$. By Lemma B.14, we have $(\Gamma \vdash M \equiv M' : A)$ and $(\Gamma \vdash N \equiv N' : A)$ in $\mathcal{T}$. Transitivity of $\equiv$ implies $(\Gamma \vdash M \equiv N : A) \in \mathcal{T}$. □

Proof of Theorem 3.4. Let K be a model of $\mathcal{T}$, and let $H : K_{\mathcal{T}} \longrightarrow K$ be a morphism of interpretations of Σ. For each precartesian categorical expression m over Σ, because $K_{\mathcal{T}}[\![m]\!] = [m]_{\approx_{\mathcal{T}}}$, it holds that $H([m]_{\approx_{\mathcal{T}}}) = K[\![m]\!]$. So it remains to prove that this makes H well defined. To see this, let $m \approx_{\mathcal{T}} n$. By definition of $\approx_{\mathcal{T}}$, letting $(\Gamma \vdash M : A) =_{\mathrm{def}} c'(m)$ and $(\Gamma \vdash N : A) =_{\mathrm{def}} c'(n)$, we have $(\Gamma \vdash M \equiv N : A) \in \mathcal{T}$. Soundness implies $K[\![\Gamma \vdash M : A]\!] = K[\![\Gamma \vdash N : A]\!]$, and therefore $c(c'(m)) = c(c'(n))$ holds in K. By Lemma B.10, in K we have $c(c'(m)) = i; m$ and $c(c'(n)) = i; n$. By transitivity and cancelling i, we have $m = n$ in K. □

Towards a proof of Conjecture 3.1

Here is an argument why Conjecture 3.1 should be true. Suppose that $\mathcal{T}$ is a let-theory over a precartesian signature Σ. Let J be a judgement that holds in every model of $\mathcal{T}$. First, suppose J is of the form $(\Gamma \vdash M \,!\, \mathit{copyable})$ (or $(\Gamma \vdash M \,!\, \mathit{discardable})$). Because J holds in the term model $K_{\mathcal{T}}$, the judgement $(\Gamma \vdash (\mathit{let}\, x = M\, \mathit{in}\, (x, x)) \equiv (M, M) : A * A)$ (or $(\Gamma \vdash (\mathit{let}\, x = M\, \mathit{in}\, ()) \equiv () : \mathit{unit})$) is in $\mathcal{T}$. By one of the two rules at the bottom of Figure 3.7, J is in $\mathcal{T}$.

Now suppose that $J = (\Gamma \vdash M \,!\, \mathit{central})$. To create some awareness, I'll first describe a failing attempt of proving $J \in \mathcal{T}$: Because J holds in $K_{\mathcal{T}}$, the following judgement is in $\mathcal{T}$ for all N with no free occurrence of x:

$$(\mathit{let}\, x = M\, \mathit{in}\, \mathit{let}\, y = N\, \mathit{in}\, (x, y)) \equiv (\mathit{let}\, y = N\, \mathit{in}\, \mathit{let}\, x = M\, \mathit{in}\, (x, y)) \qquad \text{(B.1)}$$

We could deduce $J \in \mathcal{T}$ if we had the rule

$$\frac{(let\,x = M\ in\ let\,y = N\ in\,(x,y)) \equiv (let\,y = N\ in\ let\,x = M\ in\,(x,y)) \quad \text{for all } N \text{ in which } x \text{ is not free}}{M\,!\,central} \tag{B.2}$$

But we didn't include this rule into the let-calculus—wisely so, because it is not sound: Equation B.1 does not guarantee that the denotation of M is central, because a model of $\mathcal{T}$ can have morphisms that are not denotable by any N.

However, completeness for centrality judgements should hold even without Rule B.2. To see this, let $\mathcal{T}'$ be the theory that results from $\mathcal{T}$ by adding a new constant $f : unit \longrightarrow unit$, together with equations

$$(let\,x = f()\ in\ let\,y = N\ in\,(x,y)) \equiv (let\,y = N\ in\ let\,x = f()\ in\,(x,y)) \tag{B.3}$$

for every $(\Gamma \vdash N\,!\,central) \in \mathcal{T}$. The term model $K_{\mathcal{T}'}$ is a model of $\mathcal{T}$. (The key point here is that, if $(\Gamma \vdash N\,!\,central) \in \mathcal{T}$, then the denotation of N in $K_{\mathcal{T}'}$ is indeed central, because of Equation B.3.) In particular, J holds in $K_{\mathcal{T}'}$, and therefore M is central in $K_{\mathcal{T}'}$. So the judgement

$$(let\,x = f()\ in\ let\,y = M\ in\,(x,y)) \equiv (let\,y = M\ in\ let\,x = f()\ in\,(x,y)) \tag{B.4}$$

is in $\mathcal{T}'$. Because f is a new constant, M either contains no constants, or Equation B.4 got into $K_{\mathcal{T}'}$ as one of the Equations B.3. (While this seems obvious, it is not clear to me how to prove it.) In either case, it holds that $(\Gamma \vdash M\,!\,central) \in \mathcal{T}$. Completeness for judgements of the form $(\Gamma \vdash M \,/\, relevant\,x)$ and $(\Gamma \vdash M \,/\, affine\,x)$ is trivial, and completeness for judgements of the form $(\Gamma \vdash M \,/\, clear\,x)$ follows directly from completeness for judgements of the form $(\Gamma \vdash M\,!\,central)$.

Appendix C

Proofs

C.1 Proof of Proposition 2.1

We have 1⇒4 because the focus is a subcategory of the centre. Proving 2⇒1 is easy. To see 4⇒3, let C be the subcategory of the centre like in Condition 4. The components of δ, p, q, and ! are focal because they are morphisms of C and every morphism of C is focal (central because C is a subcategory of the centre; copyable and discardable because of the finite products on C). The equations as in Condition 2 hold because C has finite products. For the same reason we have the equations about the twist map and the associativity maps. Now for 3⇒2. To see that the centre is closed under $\otimes$, let $f : A \longrightarrow A'$ be a central morphism. We prove that for each object B the morphism fB is central. So let $g : C \longrightarrow C'$. The following diagram commutes because the associativity map α is a natural iso and f is central:

$$
\begin{tikzcd}
(AB)C \arrow[rrr, "(fB)C"] \arrow[dr, "\alpha"] \arrow[ddd, "(AB)g"'] & & & (A'B)C \arrow[dl, "\alpha"'] \arrow[ddd, "(A'B)g"] \\
& A(BC) \arrow[r, "f(BC)"] \arrow[d, "A(Bg)"'] & A'(BC) \arrow[d, "A'(Bg)"] & \\
& A(BC') \arrow[r, "f(BC')"] \arrow[dl, "\alpha^{-1}"] & A'(BC') \arrow[dr, "\alpha^{-1}"'] & \\
(AB)C' \arrow[rrr, "(fB)C'"] & & & (A'B)C'
\end{tikzcd}
\tag{C.1}
$$

The next diagram commutes because the twist map τ is a natural iso and Diagram C.1 commutes:

$$
\begin{tikzcd}
C(AB) \arrow[rrr, "C(fB)"] \arrow[ddd, "g(AB)"'] \arrow[dr, "\tau"] & & & C(A'B) \arrow[ddd, "g(A'B)"] \arrow[dl, "\tau"'] \\
& (AB)C \arrow[r, "(fB)C"] \arrow[d, "(AB)g"'] & (A'B)C \arrow[d, "(A'B)g"] & \\
& (AB)C' \arrow[r, "(fB)C'"] \arrow[dl, "\tau^{-1}"'] & (A'B)C' \arrow[dr, "\tau^{-1}"] & \\
C'(AB) \arrow[rrr, "C'(fB)"] & & & C'(A'B)
\end{tikzcd}
$$

Therefore, fB is central. A symmetric argument works for Bf. Next we shall prove that if $f : A \longrightarrow A'$ is discardable, then so is fB. First, note that the transformation p is natural in its first argument, because

$$fB; p_{A',B} = fB; A'!; p_{A',I} = A!; fI; p_{A',I} = A!; p_{A,I}; f = p_{A,B}; f$$

Therefore

$$fB; ! = fB; p; ! = p; f; ! = p; ! = !$$

A symmetric argument works for Bf. Next we shall prove that if $f : A \longrightarrow A'$ is copyable, then so is fB. Consider the following diagram, where

$$
\begin{aligned}
x_{A,A'} = (AA')(BB) &\xrightarrow{\alpha^{-1}} ((AA')B)B \xrightarrow{\alpha B} (A(A'B))B \\
&\xrightarrow{(A\tau)B} (A(BA'))B \xrightarrow{\alpha^{-1}B} ((AB)A')B \xrightarrow{\alpha} (AB)(A'B)
\end{aligned}
\tag{C.2}
$$

$$\begin{tikzcd}
AB \arrow[rrr, "\delta"] \arrow[dr, "\delta B"] \arrow[dddd, "fB"'] & & & (AB)(AB) \arrow[dd, "(AB)(fB)"] \\
& (AA)B \arrow[r, "(AA)\delta"] \arrow[d, "(Af)B"'] & (AA)(BB) \arrow[ur, "x_{A,A}"] \arrow[d, "(Af)(BB)"] & \\
& (AA')B \arrow[r, "(AA')\delta"] \arrow[d, "(fA')B"'] & (AA')(BB) \arrow[r, "x_{A,A'}"] \arrow[d, "(fA')(BB)"] & (AB)(A'B) \arrow[dd, "(fB)(A'B)"] \\
& (A'A')B \arrow[r, "(A'A')\delta"] & (A'A')(BB) \arrow[dr, "x_{A',A'}"] & \\
A'B \arrow[ur, "\delta B"] \arrow[rrr, "\delta"] & & & (A'B)(A'B)
\end{tikzcd} \tag{C.3}$$

The two middle squares commute because δ is central. The left square commutes because f is copyable. The two right squares commute because the associativity and twist maps from which x is built are natural. To see that the upper square commutes, we first collect some intermediate results:

1. The associativity map is focal.
2. A morphism of the form $A(Bf)$ is copyable if and only if $(AB)f$ is copyable.
3. Every morphism of the form qC or Cp is copyable.
4. The twist map is copyable.
5. A morphism of the form fA is copyable if and only if Af is copyable.
6. Every morphism of the form $A\delta$ is copyable.

Claim 1 holds because by assumption α is copyable, and applying the shown closure properties of the discardable and central morphisms to $\alpha = \langle p; p, \langle p; q, q\rangle\rangle$ implies that α is discardable and central. Claim 2 holds because the associativity map is copyable and natural. Claim 3 holds because $q_{A,B}C = \alpha; q_{A,BC}$, and α and q are focal, and a symmetric argument works for Cp. Now for Claim 4. We have $\tau_{AB} = \delta; q_{A,B}(AB); Bp_{A,B}$. By Claim 3, $q_{A,B}(AB)$ and $Bp_{A,B}$ are copyable. So they are focal, and therefore τ is focal, and in particular, copyable. Claim 5 follows from Claim 4 and the naturality of τ. Claim 6 holds because $A\delta_B = \delta_{AB}; (AB)q; \alpha$, and δ, $(AB)q$, and α are focal. Now we divide the map $x_{A,A}$ in the upper square of

Diagram C.3 into five morphisms like in Definition C.2. Using the six intermediate results enumerated above, one can check that the resulting triangles with common vertex AB that divide the upper square of Diagram C.3 commute. The same argument work for the bottom square of Diagram C.3.

C.2 Proof of Theorem 5.2

In this proof, let's write semicolon for the composition of K_T, and colon for the composition of K. Moreover, let F stand for F_T, and G for G_T. We use Condition 3 of Proposition 2.1. First we prove that K_T together with $\otimes$ forms a binoidal category. $A \otimes (-)$ is a functor because

$$\begin{aligned}
A \otimes id_B &= A \otimes \eta_B \\
&= A \times \eta_B : t_{A,B} && \text{by definition of } \otimes \\
&= \eta_{A \times B} && \text{by Equation 5.3} \\
&= id_{A \otimes B}
\end{aligned}$$

and for morphisms $B \xrightarrow{f} C \xrightarrow{g} D$ of K_T it holds that

$$\begin{aligned}
A \otimes f; A \otimes g &= A \otimes f : T(A \otimes g) : \mu \\
&= A \times f : t_{A,C} : T(A \times g) : Tt_{A,D} : \mu && \text{by definition of } \otimes \\
&= A \times f : A \times Tg : t_{A,TD} : Tt_{A,D} : \mu && \text{because } t \text{ is natural} \\
&= A \times f : A \times Tg : id_A \times \mu_D : t_{A,D} && \text{by Equation 5.4} \\
&= A \times (f; g) : t_{A,D} \\
&= A \otimes (f; g) && \text{by definition of } \otimes
\end{aligned}$$

By the dual argument, $(-) \otimes A$ too is a functor. F strictly preserves the tensor because

$$\begin{aligned}
F(A \times f) &= A \times f : \eta_{A \times C} \\
&= A \times f : A \times \eta_C ; t_{A,C} && \text{by Equation 5.3} \\
&= A \times Ff : t_{A,C} \\
&= A \otimes Ff && \text{by definition of } \otimes
\end{aligned}$$

Next we prove that F sends central morphisms to central morphisms. Let $f \in K(A, A')$ be central, and let $g \in K_T(B, B')$. Then

$$\begin{aligned}
(Ff) \otimes B; A' \otimes g &= F(f \times B); A' \otimes g \\
&= f \times B : \eta_{A' \times B} : G(A' \otimes g) \\
&= f \times B : A' \otimes g \\
&= f \times B : A' \times g : t_{A',B'} \\
&= A \times g : f \times TB' : t_{A',B'} && \text{because } f \text{ is central} \\
&= A \times g : t_{A,B'} : T(f \times B') && \text{by naturality of } t \\
&= A \otimes g : T(f \times B') \\
&= A \otimes g : GF(f \times B') \\
&= A \otimes g; F(f \times B') \\
&= A \otimes g; (Ff) \otimes B'
\end{aligned}$$

F preserves copyable morphisms too, because for every copyable $f \in K(A, B)$ it holds that

$$\begin{aligned}
Ff; F\delta &= F(f : \delta) \\
&= F(\delta : A \times f : f \times B) && \text{because } f \text{ is copyable} \\
&= F\delta; F(A \times f); F(f \times B) \\
&= F\delta; A \otimes Ff; Ff \otimes B
\end{aligned}$$

F preserves discardable morphisms, because for every discardable morphism f of K it holds that

$$Ff; F! = F(f :!) = F!$$

Therefore, all components of $F\delta$, Fp, Fq, and $F!$ are focal. The equations in Condition 2 hold because they hold in K and F strictly preserves all structure. Now let $\alpha : (A \times B) \times C \longrightarrow A \times (B \times C)$ and $\alpha' : A \times (B \times C) \longrightarrow (A \times B) \times C$ be the associativity maps of K, and let $\tau : A \times B \longrightarrow B \times A$ be the twist map of K. Because F strictly preserves all operators, the associativity maps of K are equal to $F\alpha$ and $F\alpha'$, respectively, and the twist map of K is equal to $F\tau$. Because α and α' are copyable and inverse to each other, the same holds for $F\alpha$ and $F\alpha'$. Because τ is self-inverse, the same holds for $F\tau$. So it remains to prove that the associativity, neutrality, and twist maps of K_T are natural in each of their arguments. For the neutrality map $Fq : 1 \otimes A \longrightarrow A$, we have

$$\begin{aligned}
1 \otimes f; Fq &= 1 \otimes f : GFq \\
&= 1 \otimes f : Tq \\
&= 1 \times f : t : Tq \\
&= 1 \times f : q && \text{by Equation 5.1} \\
&= q : f && \text{because } q \text{ is natural} \\
&= q : \eta : Gf \\
&= Fq : Gf \\
&= Fq; f
\end{aligned}$$

The naturality of $Fp : A \otimes 1 \longrightarrow A$ follows from the dual argument. For the naturality of $F\tau$, consider

$$\begin{aligned}
f \otimes B; F\tau &= f \otimes B : GF\tau \\
&= f \otimes B : T\tau \\
&= f \times B : t' : T\tau \\
&= f \times B : \tau : t \\
&= \tau : B \times f : t && \text{because } \tau \text{ is natural} \\
&= \tau : B \otimes f \\
&= \tau : \eta : G(B \otimes f) \\
&= F\tau; B \otimes f
\end{aligned}$$

For naturality of $F\alpha$, let $f \in K_T(C, C')$. Then

$$\begin{aligned}
(A \otimes B) \otimes f; F\alpha &= (A \times B) \otimes f : T\alpha \\
&= (A \times B) \times f : t : T\alpha \\
&= (A \times B) \times f : \alpha : A \times t : t && \text{by Equation 5.2} \\
&= \alpha : A \times (B \times f) : A \times t : t && \text{because } \alpha \text{ is natural} \\
&= \alpha : A \times (B \otimes f) : t \\
&= \alpha : A \otimes (B \otimes f) \\
&= \alpha : \eta : G(A \otimes (B \otimes f)) \\
&= F\alpha; A \otimes (B \otimes f)
\end{aligned}$$

The naturality in the first argument follows from the dual argument. For naturality in the second argument, let $f \in K_T(B, B')$ and consider

$$
\begin{array}{ll}
F\alpha; A \otimes (f \otimes C) & \\
= F(\tau \times C : \alpha : B \times \tau : \alpha : \tau); A \otimes (f \otimes C) & \text{by coherence of } K \\
= F\tau \otimes C; F\alpha; B \otimes F\tau; F\alpha; F\tau; A \otimes (f \otimes C) & \\
= F\tau \otimes C; F\alpha; B \otimes F\tau; F\alpha; (f \otimes C) \otimes A; F\tau & \text{by naturality of } F\tau \\
= F\tau \otimes C; F\alpha; B \otimes F\tau; f \otimes (C \otimes A); F\alpha; F\tau & \text{by naturality of } F\alpha \\
 & \text{in the first argument} \\
= F\tau \otimes C; F\alpha; f \otimes (A \otimes C); B' \otimes F\tau; F\alpha; F\tau & \text{by centrality of } F\tau \\
= F\tau \otimes C; (f \otimes A) \otimes C; F\alpha; B' \otimes F\tau; F\alpha; F\tau & \text{by naturality of } F\alpha \\
 & \text{in the first argument} \\
= (A \otimes f) \otimes C; F\tau \otimes C; F\alpha; B' \otimes F\tau; F\alpha; F\tau & \text{by centrality of } F\tau \\
= (A \otimes f) \otimes C; F(\tau \times C : \alpha : B' \times \tau : \alpha : \tau) & \\
= (A \otimes f) \otimes C; F\alpha & \text{by coherence of } K
\end{array}
$$

Bibliography

[ADH$^+$98] H. Abelson, R.K. Dybvig, C.T. Haynes, G.J. Rozas, N.I. Adams, D.P. Friedman, E. Kohlbecker, G.L. Steele, D.H. Bartley, R. Halstead, D. Oxley, G.J. Sussman, G. Brooks, C. Hanson, K.M. Pitman, and M. Wand. Revised^5 report on the algorithmic language scheme. *Higher-order and Symbolic Computation*, 11(1):7–105, 1998.

[AP97] S.O. Anderson and A.J. Power. A representable approach to finite nondeterminism. *Theoretical Computer Science*, 177:3–25, 1997.

[App92] Andrew W. Appel. *Compiling with Continuations*. Cambridge University Press, 1992.

[BFS99] Anna Bucalo, Carsten Führmann, and Alex Simpson. Equational lifting monads. In *Proceedings CTCS'99*, Electronic Notes in Theoretical Computer Science, Edinburgh, 1999. Elsevier.

[BW85] M. Barr and C. Wells. *Toposes, Triples and Theories*, volume 278 of *Grundlehren der mathematische Wissenschaften*. Springer-Verlag, 1985.

[CG99] Andrea Corradini and Fabio Gadducci. A functorial semantics for multi-algebras and partial algebras, 1999.

[CO86] P.-L. Curien and A. Obtulowicz. Partiality and cartesian closedness. typescript, 1986.

[Fok94] Maarten M Fokkinga. Dyads, a generalisation of monads. Memoranda Informatica 94-30, University of Twente, June 1994.

[Füh98] Carsten Führmann. Direct-style and continuation-passing style models of control. Available from Carsten Führmann's homepage, 1998.

[Has95] M. Hasegawa. Decomposing typed lambda calculus into a couple of categorical programming languages. In *Proceedings, 6th International*

Conference on Category Theory and Computer Science (CTCS'95), volume 953 of *Lecture Notes in Computer Science*. Springer-Verlag, 1995.

[Has97] Masahito Hasegawa. Recursion from cyclic sharing: traced monoidal categories and models of cyclic lambda calculi. In *Proc. 6th International Conference on Category Theory and Computer Science (CTCS'95)*, volume 1210 of *LNCS*, Nancy, April 1997. Springer Verlag.

[Has99] Masahito Hasegawa. *Models of Sharing Graphs (A Categorical Semantics of Let and Letrec)*. Distinguished Dissertation Series. Springer-Verlag, 1999.

[HDM93] Robert Harper, Bruce Duba, and David MacQueen. Typing first-class continuations in ML. *Journal of Functional Programming*, 3(4), October 1993.

[Hoe77] H. J. Hoehnke. On partial recursive definitions and programs. In M. Karpinski, editor, *Fundamentals of Computation Theory*, volume 56 of *LNCS*, pages 260–274. Springer Verlag, Berlin, 1977.

[Hof95] Martin Hofmann. Sound and complete axiomatisations of call-by-value control operators. *Mathematical Structures in Computer Science*, 5:461–482, 1995.

[Jac91] Bart Jacobs. *Categorical Type Theory*. PhD thesis, University of Nijmegen, 1991.

[Jac94] Bart Jacobs. Semantics of weakening and contraction. *Annals of Pure and Applied Logic*, 69(1):73–106, 1994.

[Jef98] Alan Jeffrey. Premonoidal categories and a graphical view of programs. Available from http://klee.cs.depaul.edu/premon/, 1998.

[JG91] P. Jouvelot and D.K. Gifford. Algebraic reconstruction of types and effects. In *Proceedings of the 1991 ACM Conference on Principles of Programming Languages*, New York, 1991.

[Koc70] A. Kock. Monads on symmetric monoidal closed categories. *Archiv der Mathematik*, XXI:1–10, 1970.

[Koc71] A. Kock. Bilinearity and cartesian closed monads. *Math. Scand.*, (29):161–174, 1971.

[Lan71] Saunders Mac Lane. *Categories for the Working Mathematician*. Graduate Texts in Mathematics. Springer-Verlag, 1971.

[Mog88] E. Moggi. Computational lambda-calculus and monads. Technical Report ECS-LFCS-88-66, Edinburgh Univ., Dept. of Comp. Sci., 1988.

[Mog89] E. Moggi. An abstract view of programming languages. Technical Report ECS-LFCS-90-113, Edinburgh Univ., Dept. of Comp. Sci., 1989. Lecture Notes for course CS 359, Stanford Univ.

[Mog91] E. Moggi. Notions of computation and monads. *Information and Computation*, 93(1), 1991.

[Nie96] Flemming Nielson. Annotated type and effect systems. *ACM Computing Surveys*, 28(2), 1996. Invited postition statement for the Symposium on Models of Programming Languages and Computation.

[Plo75] G. D. Plotkin. Call-by-name, call-by-value and the λ-calculus. *Theoretical Computer Science*, 1(2):125–159, December 1975.

[Pow99a] John Power. Modularity in denotational semantics. In *Proceedings MFPS XIII*, volume 6 of *Electronic Notes in Theoretical Computer Science*, New Orleans, 1999. Elsevier.

[Pow99b] John Power. Premonoidal categories as categories with algebraic structure. submitted, 1999.

[PR97] John Power and Edmund Robinson. Premonoidal categories and notions of computation. *Mathematical Structures in Computer Science*, 7(5):453–468, October 1997.

[PR98] A. J. Power and G. Rosolini. A modular approach to denotational semantics. In K. G. Larsen, S Skyum, and G. Winskel, editors, *25th International Colloquium on Automata, Languages, and Programming*. Springer-Verlag, 1998.

[PR99] John Power and Edmund Robinson. Modularity and dyads. In *Proceedings MFPS XV*, volume 20 of *Electronic Notes in Theoretical Computer Science*, New Orleans, 1999. Elsevier.

[PT97] John Power and Hayo Thielecke. Environments, continuation semantics and indexed categories. In *Proceedings TACS'97*, volume 1281 of *LNCS*, pages 391–414. Springer Verlag, 1997.

[PT99] John Power and Hayo Thielecke. Closed Freyd- and kappa-categories. In *Proc. ICALP '99*, volume 1644 of *LNCS*. Springer, 1999.

[Rey93] John C. Reynolds. The discoveries of continuations. *Lisp and Symbolic Computation*, 6(3/4):233–247, November 1993.

[Ros86] G. Rosolini. *Continuity and effectiveness in topoi*. D.phil thesis, University of Oxford, 1986.

[RR88] E.P. Robinson and G. Rosolini. Categories of partial maps. *Information and Computation*, 79:95–130, 1988.

[Sel00] Peter Selinger. Control categories and duality: on the categorical semantics of the lambda-mu calculus. To appear in Mathematical Structures in Computer Science, 2000.

[Sim93] Alex Simpson. Towards algebraic semantics of programming languages. Notes for a talk at the LFCS Lab Lunch, March 1993.

[SJ99] Ralf Schweimeier and Alan Jeffrey. A categorical and graphical treatment of closure conversion. In *Proceedings MFPS XV*, volume 20 of *Electronic Notes in Theoretical Computer Science*, New Orleans, 1999. Elsevier. To appear.

[Ste78] Guy Steele. Rabbit: A compiler for Scheme. AI Technical Report 474, MIT, May 1978.

[Thi97a] Hayo Thielecke. *Categorical Structure of Continuation Passing Style*. PhD thesis, University of Edinburgh, 1997.

[Thi97b] Hayo Thielecke. Continuation semantics and self-adjointness. In *Proceedings MFPS XIII*, Electronic Notes in Theoretical Computer Science. Elsevier, 1997.

[Thi99a] Hayo Thielecke. Continuations, functions and jumps. *ACM SIGACT News*, 30(2), 1999.

[Thi99b] Hayo Thielecke. Using a continuation twice and its implications for the expressive power of `call/cc`. *Higher-Order and Symbolic Computation*, 12(1):47–74, 1999.

[TL] Lucent Technologies and Bell Laboratories. Standard ML of New Jersey web pages. Available at http://cm.bell-labs.com/cm/cs/what/smlnj/doc/SMLofNJ/pages/cont.html.

[vW64] Adriaan van Wijngaarden. Recursive definition of syntax and semantics. In *Formal Language Description Languages for Computer Programming, Proceedings of an IFIP Working Conference*, pages 13–24, 1964.