Call-By-Push-Value

Paul Blain Levy

Submitted for the degree of Doctor of Philosophy

Queen Mary and Westfield College, University of London

2001

Call-By-Push-Value

Paul Blain Levy

Abstract

Call-by-push-value (CBPV) is a new programming language paradigm, based on the slogan "a value is, a computation does". We claim that CBPV provides the semantic primitives from which the call-by-value and call-by-name paradigms are built. The primary goal of the thesis is to present the evidence for this claim, which is found in a remarkably wide range of semantics: from operational semantics, in big-step form and in machine form, to denotational models using domains, possible worlds, continuations and games.

In the first part of the thesis, we come to CBPV and its equational theory by looking critically at the call-by-value and call-by-name paradigms in the presence of general computational effects. We give a Felleisen/Friedman-style CK-machine semantics, which explains how CBPV can be understood in terms of push/pop instructions.

In the second part we give simple CBPV models for printing, divergence, global store, errors, erratic choice and control effects, as well as for various combinations of these effects. We develop the store model into a possible world model for cell generation, and (following Steele) we develop the control model into a "jumping implementation" using a continuation language called Jump-With-Argument (JWA).

We present a pointer game model for CBPV, in the style of Hyland and Ong. We see that the game concepts of questioning and answering correspond to the CBPV concepts of forcing and producing respectively. We observe that this game semantics is closely related to the jumping implementation.

In the third part of the thesis, we study the categorical semantics for the CBPV equational theory. We present and compare 3 approaches:

- models using *strong monads*, in the style of Moggi;
- models using *value/producer structures*, in the style of Power and Robinson;
- models using (strong) adjunctions.

All the concrete models in the thesis are seen to be adjunction models.

Submitted for the degree of Doctor of Philosophy

Queen Mary and Westfield College, University of London

2001

Acknowledgements

Firstly, I want to thank Peter O'Hearn, for his help, advice—many of his suggestions have been incorporated into this thesis—and encouragement over four years. He has always been patient and generous with time. I am fortunate to have been supervised by him.

I have benefited also from the expertise of Hayo Thielecke, with whom I shared an office for almost two years, and from Josh Berdine, Adam Eppendahl, Samin Ishtiaq, Paul Taylor and the rest of LFP, the Logic and Foundations of Programming group at Queen Mary. I am grateful to them all for their ideas, macros[1] and friendship.

Further afield, many colleagues have provided enjoyable discussions, including Jim Laird, Michael Marz, Guy McCusker, John Power and Uday Reddy. Julian Gilbey helped me to design the graphical syntax for JWA in Sect. 8.3.2. My examiners, Chris Hankin and Gordon Plotkin, improved the thesis with their useful comments.

Finally, I am grateful to my parents, Ruth and David Levy, for their love and support.

[1] Paul Taylor's prooftree macros are used throughout the thesis, and his diagram macros for some of the diagrams. The others are created with `xfig`.

Contents

List of Figures

Part I

Language

Chapter 1

Introduction

1.1 Computational Effects

Much attention has been devoted to functional languages with divergence (nontermination), such as PCF [Plo77] and FPC [Plo85], and to their models. After all, it is well-known that a language with full recursion must have divergent programs. Yet some of the most fundamental semantic issues involved in these languages are far more general than is often realized, and (as we will argue in Sect. 2.2) can be correctly understood only in the light of this generality. They arise whenever we combine imperative and functional features within a single language.

It may strike the reader as strange to regard divergence as an imperative feature, for what could be more "purely functional" than PCF? But it is a remarkable fact that the same core theory (the subject of Chap. 2) applies when we add to the simply typed λ-calculus any of the following [Mog91]:

- divergence
- reading and assigning to a storage cell
- input and output
- erratic choice
- generating a new name or cell
- halting with an error message
- control effects (we shall explain these in Chap. 6)

Thus, whether or not `diverge` really is a command, it certainly behaves like one. We call all these features *computational effects* or just *effects*. In our exposition, we shall use `print` commands as our leading example of an effect, rather than the more familiar `diverge`, because doing so makes important distinctions clearer (as we shall argue in Sect. 2.2).

The first issue that arises for effectful languages is that—by contrast with the simply typed λ-calculus—order of evaluation matters. Whether a program converges or diverges, whether a program prints `hello` or prints `goodbye`, will depend on the evaluation order that the language uses.

A priori, there are many evaluation orders that could be considered. But two of them have been found to be significant, in the sense that they possess a wide range of elegant semantics[1]:

[1] A third method of evaluation, *call-by-need*, is useful for implementation purposes. But it lacks a clean denotational semantics—at least for effects other than divergence and erratic choice whose special properties are exploited in [Hen80] to provide a call-by-need model. So we shall not consider call-by-need.

call-by-name (CBN) and *call-by-value* (CBV).

1.2 Reconciling CBN and CBV

1.2.1 The Problem Of Two Paradigms

Researchers have developed many semantics for CBN and CBV. But each time a new form of semantics is introduced, each time a technical result is proved, each time an analysis of semantic issues is presented, we have to perform the work twice: once for CBN, once for CBV.

Here are some examples of this situation.

- In many introductory textbooks on the semantics of programming languages (e.g. [Gun95]), we are first shown a CBN language and its operational semantics. We are given a denotational semantics using domains and this is proved adequate. Next, we are shown a CBV language and its operational semantics. We are given a denotational semantics and this too is proved adequate.
- In [HO94, Nic96] a game semantics is presented for a CBN language, and various technical properties are proved. Then, in [AM98a, HY97], a game semantics is presented for a CBV language, and similar technical properties are proved.
- In [SR98], a machine semantics and continuation semantics are presented and their agreement proved: first for a CBV language, then for a CBN language.
- In [Ole82], a functor category semantics for a CBN language is presented. Then, in [Mog90, Sta94] a functor category semantics for a CBV language is presented.

This duplication of work is tiresome. Furthermore, it makes the languages involved seem inherently arbitrary. We would prefer to study a single, canonical language.

1.2.2 A Single Language

How can the situation described in Sect. 1.2.1 be remedied? A first suggestion is as follows.

> Suppose we have a language $\mathfrak{L}$ in which both CBN and CBV programs can be written. Then we need only give semantics for $\mathfrak{L}$, and this automatically provides semantics for CBN and CBV.

There is a basic problem with this approach. As an extreme example, consider that CBN and CBV languages can both be translated into π-calculus [San99] so π-calculus is "a language in which both CBN and CBV programs can be written". But this does not relieve us of the responsibility to provide (say) functional Scott semantics for CBN and CBV, because there is no functional Scott semantics for π-calculus. For this reason, it is essential for $\mathfrak{L}$ to have a functional Scott semantics, a game semantics, a continuation semantics, an operational semantics, etc.

But even where $\mathfrak{L}$ does have all these semantics, there can be a more subtle problem. As an example, consider that CBN can be translated into CBV [HD97], and so CBV itself is "a language in which both CBN and CBV programs can be written". Now in this case $\mathfrak{L}$ certainly has a Scott semantics, a game semantics, a continuation semantics etc., and so, as proposed, we obtain semantics for CBN. But the Scott semantics for CBN thus obtained is not the traditional CBN Scott semantics. (For example, whereas in the traditional semantics λ`x.diverge` and `diverge` have the same denotation, in the new semantics they have different denotations.) So the translation of CBN into CBV does not relieve us of the task of constructing the traditional semantics for CBN.

We see that, in order to remedy the situation described in Sect. 1.2.1, we have to be sure that the Scott semantics for $\mathfrak{L}$ induces the *usual* Scott semantics for CBN and CBV, rather than

a more complicated one; and likewise for game semantics, continuation semantics, operational semantics etc. Another way of saying this is that the translations from CBV and from CBN into $\mathfrak{L}$ must *preserve* Scott semantics, game semantics etc.

Of course, it will be impossible to show that the translations preserve *every* semantics we might wish to study. But the semantics we have mentioned are extremely diverse, and if these are all preserved then this makes a strong case that we do not need to continue investigating CBN and CBV as independent entities.

We say then that the language $\mathfrak{L}$ *subsumes* CBN and CBV, and that the two translations into it are *subsumptive*. Notice that subsumptiveness is not a formal, technical property of a translation (such as full abstraction). Rather, it says, informally, that there is no reason to regard the source language as anything more than an arbitrary fragment of the target language.

1.2.3 "Hasn't This Been Done By ... ?"

There are many claims in the literature that a given language contains both CBN and CBV:

- Moggi's monadic metalanguage [BW96, Mog91] and Filinski's variant [Fil96];
- CBV itself [HD97, SJ98]—in the presence of general effects, Moggi called this the *computational λ-calculus*, or *λ_{c}-calculus* [Mog88];
- π-calculus [San99];
- continuation languages [Plo76];
- languages based on Girard's linear logic [BW96, Gir87];
- SFL and SFPL [Mar00, MRS99].

We explained in Sect. 1.2.2 that CBV itself does not contain CBN in our sense because the equation $\lambda\mathtt{x}.\mathtt{diverge} = \mathtt{diverge}$ is invalidated by the translation from CBN into CBV. The same criticism applies to the translation from CBN to Moggi's metalanguage (although not Filinski's variant). Thus Moggi's language does not subsume CBN in our sense; it is not a solution to our problem.

Filinski's variant does not have this drawback, but both Moggi's language and Filinski's variant have the drawback that they lack operational semantics, essentially because every term is a value. (Our value/computation distinction in CBPV will remedy this.)

We explained in Sect. 1.2.2 that π-calculus does not have functional Scott semantics, so the translations into it are not subsumptive. Continuation calculi do have functional Scott semantics, but the CPS transforms into them do not preserve direct semantics, so the CPS transforms (from source languages without control effects) are not subsumptive. The linear λ-calculus of [BW96] does have Scott semantics, but, as remarked there, the language is based on the assumption that effects are "commutative", so effects such as printing are excluded.

The translation from CBV to the language SFL does not preserve the η-law for sum types (explained in Sect. 2.8). By contrast, the successor language SFPL (which was developed later than CBPV) does indeed subsume CBV and CBN. It is in a sense equivalent to the computation part of CBPV, but it lacks the simplicity of CBPV.

In Chap. 16.2, we make further criticisms of Moggi's decomposition

$$A \to_{\mathbf{CBV}} B = A \to TB$$

and of the linear decomposition

$$A \to_{\mathbf{CBN}} B = !A \multimap B$$

1.3 The Case For Call-By-Push-Value

In this thesis we introduce a new language paradigm, *call-by-push-value* (CBPV). It is based on Filinski's variant of Moggi's "monadic metalanguage" [Fil96, Mog91], but we do not assume familiarity with these calculi (discussed in Sect. 1.2.3 and Sect. 16.2).

We shall see that CBPV satisfies all that was required in Sect. 1.2.2. Consequently, what we previously regarded as the primitives of CBN and CBV can now be seen as just idioms built from the genuine primitives of CBPV.

For this reason, CBPV deserves the attention *even of those who are interested only in CBN* (or only in CBV) and not in the above problem of reconciling CBN and CBV. Such people benefit from using CBPV because the CBN semantics they are studying will exhibit the decomposition of CBN primitives into CBPV primitives, so CBPV is closer to the semantics. We will see numerous examples of this in Part II.

We mention also that, although full abstraction is neither a necessary nor a sufficient condition for subsumptiveness, we prove (Cor. 166 and Cor. 176) that the translations into CBPV are indeed fully abstract. Thus, semanticists initially concerned with fully abstract models for CBV and CBN can obtain them from fully abstract models for CBPV.

1.4 Conventions

1.4.1 Notation and Terminology

We write $V\text{`}M$ for "M applied to V". This operand-first notation has some advantages over the traditional notation of MV. In particular, it allows the "push" reading of Sect. 1.5.1.

For any natural number n, we write $\$n$ for $\{0,\dots,n-1\}$, the canonical set of size n.

We avoid the ambiguous word "variable", using instead the following distinct terms:

- An *identifier* is a syntactic symbol whose binding does not change, as in λ-calculus and predicate logic. The list giving the binding of each identifier is called the *environment*.
- A *cell* is a memory location whose contents can change through time. The list giving the contents of each cell is called the *store*.

This crucial distinction between environment and store, attributed by [TG00] to Park and his contemporaries [Par68], is maintained throughout our treatment.

1.4.2 Equations

In the course of the thesis we will present several equational theories. In order to reduce clutter, we adopt the following conventions for each equation.

1. Like a term, an equation has a context and a type, but we omit these.
2. We assume all the conditions needed to make each side of an equation well-typed, and to make sure that the two sides have the same context and type.
3. For a metasyntactic identifier M ranging over terms, and an identifier $\mathtt{x}$, if M occurs in the scope of an $\mathtt{x}$-binder and also occurs not in the scope of any $\mathtt{x}$-binder, then it is assumed that $\mathtt{x}$ is not in the context of M.

These conventions allow us, for example, to write the η-equation for functions as

$$M = \lambda \mathtt{x}(\mathtt{x}\text{`}M) \tag{1.1}$$

rather than as

$$\frac{\Gamma \vdash M : A \to B}{\Gamma \vdash M = \lambda \mathtt{x}(\mathtt{x}\text{`}M) : A \to B}\ \mathtt{x} \notin \Gamma$$

Since in (1.1) M occurs (in the RHS) in the scope of an `x`-binder and also occurs (in the LHS) not in the scope of any `x`-binder, it is assumed that Γ—the context of M—does not contain `x` . Strictly speaking, M is weakened by `x` : A in the RHS, but we will leave all weakening implicit.

1.5 A CBPV Primer

1.5.1 Basic Features

The aim of this section is to enable the reader to understand a simple example program. For this purpose, it suffices to explain the following basic features.

- CBPV has a simple imperative reading in terms of a stack.
  - λ`x` is understood as the command "pop `x`".
  - V‘ is understood as the command "push V".
- CBPV terms are classified into computations and values. A computation *does* something, whereas a value is something like a boolean or number which can be passed around.

  Slogan A value is, a computation does.
- Only a value can be pushed or popped. In other words, only a value can be an operand. However, we can "freeze" a computation M into a value; this value is called the *thunk* of M [Ing61]. Later, when desired, this thunk can be *forced* (i.e. executed).
- Certain computations produce values; they are called *producers*. If M is a producer, and N another computation, then we can *sequence* them into the computation M `to x`. N. This means: first obey M until finally it produces a value V, then obey N with `x` bound to V.

1.5.2 Example Program

The easiest way to grasp these basic ideas is to see an example program explained very informally. The following program uses the computational effect of printing messages to the screen.

```
print "hello0";
let x be 3.
let y be thunk (
            print "hello1";
            λz.
            print "we just popped "z;
            produce x + z
         ).
print "hello2";
( print "hello3";
  7‘
  print "we just pushed 7";
  force y
) to w.
print "w is bound to "w;
produce w + 5
```

We give a blow-by-blow account of execution—the program executes the commands in order. First it prints `hello0`, then binds `x` to 3, then binds `y` to a thunk. If the word `thunk` were omitted, the program would not typecheck, because an identifier can be bound only to a value—a computation is too active to sit in an environment (list of bindings for identifiers).

Next, the program prints `hello2` and the computation enclosed in parentheses (from `print "hello3"` to `force y`), which as we shall see is a producer, commences execution. This producer first prints `hello3`, pushes 7, and prints `we just pushed 7`. Then it forces the thunk `y`. So it prints `hello1` and pops `z` from the stack; i.e. it removes the top entry (which is 7) from the stack and binds `z` to it. It reports `we just popped 7` and produces $\mathtt{x}+\mathtt{z}$ which is 10.

So the producer enclosed in parentheses (from `print "hello3"` to `force y`) has had the overall effect of printing several messages and producing 10. Thus `w` becomes bound to 10 and the program prints `w is bound to 10`. Finally the program produces $\mathtt{w}+5$ which is 15.

In summary the program outputs as follows

```
hello0
hello2
hello3
we just pushed 7
hello1
we just popped 7
w is bound to 10
```

and finally produces the value 15.

In more familiar terms, `y` is a procedure, 7 is the parameter that is passed to it and it returns 10. ("Returns" and "produces" are synonymous.) But, compared to a typical high-level language, CBPV is unusually liberal in that

- it allows the command `print "we just pushed 7"` to intervene between pushing the parameter 7 and calling the procedure;
- it allows the command `print "hello 3"` to intervene between the start of the procedure and popping the parameter.

In a practical sense, this liberality is of little benefit, for the program would have the same observable behaviour if the lines

```
7‘
print "we just pushed 7"
```

were exchanged, and likewise if the lines

```
print "hello 3"
λz.
```

were exchanged. But it is this flexibility that allows CBPV to give a fine-grain analysis of types.

1.5.3 CBPV Makes Control Flow Explicit

It is worth noticing that there are two lines in this program which cause execution to move to another part of the program, rather than to the next line. These are `force y`, where execution jumps to the body of the thunk that `y` is bound to, and `produce` $\mathtt{x}+\mathtt{z}$, where the value 10 is returned to just after the line `force y`. This illustrates a general phenomenon.

> Only `force` and `produce` cause execution to move to another part of the program.

This kind of information about control flow is much less explicit in CBV and CBN. For this reason, as we shall see in Chap. 8 and Chap. 9, it is beneficial to use CBPV when studying semantics that describes interaction between different parts of a program, such as continuation semantics or game semantics.

Although `force` and `produce` both cause jumps, there is an important difference between the two. A `force` instruction will specify where control moves to: in the example program, to `y`. By contrast, a `produce` instruction will cause a jump to a point at the top of a stack, so it is not specified explicitly. This distinction is apparent in both continuation semantics and game semantics.

1.5.4 Value Types and Computation Types

We said in Sect. 1.5.1 that CBPV has 2 disjoint classes of terms: values and computations. It is therefore unsurprising that it has 2 disjoint classes of types: value types and computation types.

- A value has a value type.
- A computation has a computation type.

For example, `nat` and `bool` are value types.

The two classes of types are given by

$$\begin{array}{lrcl} \text{value types} & A & ::= & U\underline{B} \mid \sum_{i\in I} A_i \mid 1 \mid A \times A \\ \text{computation types} & \underline{B} & ::= & FA \mid \prod_{i\in I} \underline{B}_i \mid A \to \underline{B} \end{array}$$

where each set I of tags is finite. (We will also be concerned with *infinitely wide* CBPV in which I may be countably infinite—see Sect. 5.1 for more discussion.) We underline computation types for clarity.

We explain these types as follows—notice how this explanation maintains the principle "a value is, a computation does".

- A value of type $U\underline{B}$ is a *thunk* of a computation of type $\underline{B}$.
- A value of type $\sum_{i\in I} A_i$ is a pair (i, V), where $i \in I$ and V is a value of type A_i.
- A value of type 1 is an empty tuple $()$.
- A value of type $A \times A'$ is a pair (V, V'), where V is a value of type A and V' is a value of type A'.
- A computation of type FA *produces* a value of type A.
- A computation of type $\prod_{i\in I} \underline{B}_i$ *pops* a tag $i \in I$ from the stack, and then behaves as a computation of type $\underline{B}_i$.
- A computation of type $A \to \underline{B}$ *pops* a value of type A from the stack, and then behaves as a computation of type $\underline{B}$.

We have apparently excluded types such as `bool` and `nat` from this, but `bool` can be recovered as $1 + 1$. If we add type recursion as in Sect. 5.3.2, we can recover `nat` as $\mu\mathtt{X}.(1 + \mathtt{X})$. In infinitely wide CBPV, we can recover `nat` as $\sum_{i\in\mathbb{N}} 1$.

Looking at the example computation of Sect. 1.5.2, we can see that its type is $F\mathtt{nat}$ because it produces a natural number. The identifier `y` has type $U(\mathtt{nat} \to F\mathtt{nat})$. This means that the value to which `y` is bound is a thunk (U) of a computation that pops a natural number (`nat` $\to$) and then produces (F) a natural number (`nat`).

1.6 Structure of Thesis

1.6.1 Goals

This thesis has two goals.

1. The main goal is to argue the claim made in Sect. 1.3—to persuade the reader that CBPV is significant by exhibiting a wide range of elegant semantics, from which CBN and CBV semantics can be recovered. This goal is the subject of Part II; the preliminary work is done in Chap. 3.

2. The second goal is to understand the categorical semantics of the CBPV equational theory, introduced in Chap. 4. This goal is the subject of Part III. We do not claim that this provides any further motivation for CBPV.

1.6.2 Chapter Outline

We begin by describing an intellectual journey that leads to CBPV. We do this briefly—a full technical treatment is given in the Appendix. The journey starts in Chap. 2, where we explain the language design choices that characterize the CBV and CBN paradigms, and we examine the consequences of these choices. We give both operational (big-step) semantics and denotational semantics in the presence of printing, our leading example of a computational effect. (We look at divergence too, because of its familiarity.) We learn, most importantly, that CBV types and CBN types denote different kinds of things—sets and $\mathcal{A}$-sets (which we will define in Sect. 2.7.4) respectively. By considering which equations are valid as observational equivalences, we see that function types behave well in CBN but not in CBV, whereas sum types behave well in CBV but not in CBN.

This critical exploration of CBV and CBN leads us, in Chap. 3, to CBPV. Because of the denotational difference between CBV types and CBN types, it is clear that the subsuming language must have two kinds of type. We give big-step and denotational semantics (for printing and for divergence) and we explain how CBV and CBN are subsumed in CBPV. This completes the journey. In addition, we give another form of operational semantics, the *CK-machine* [FF86], which makes precise the push/pop reading that we illustrated in Sect. 1.5.2.

Whereas function types behave badly in CBV and sum types behave badly in CBN, in CBPV both function types and sum types behave well—all the badness is concentrated in the *producer types*. We see this in Chap. 4 by giving an equational theory for CBPV. To present this theory, we have to extend CBPV with *complex values*. However, as we explain in Sect. 4.6, these complex values can always be removed from computations and from closed values.

Chap. 5, the last chapter of Part I, looks at recursion and infinitary CBPV. There is little original here; it merely sets up material that is needed later in Chap. 7, where we need to understand the semantics of type recursion in order to be able to interpret thunk storage, and in Chap. 9, where we need infinitely deep syntax in order to obtain definability results for types.

Having seen the operational ideas and the equational theory we are ready to give the concrete semantics of the former (Part II) and the categorical semantics of the latter (Part III).

In Chap. 6 we look at denotational semantics for a range of effects: global store, control, errors, erratic choice, printing, divergence and various combinations of these. We learn that a useful heuristic for creating CBPV semantics is to guess the form of the soundness theorem. Again and again, throughout these examples, we see traditional semantics for CBV primitives decomposing naturally into CBPV semantics. As for the corresponding CBN models, some are new while others were known but previously appeared mysterious—the CBPV decomposition makes the structure of these semantics clear.

Both the store model and the control model from Chap. 6 are developed further in the subsequent chapters.

- The store model is developed in Chap. 7 into a possible world model for cell generation, which captures some interesting intuitions about CBPV: in particular, the idea that a thunk is something that can be forced at any future time. This model is surprising in itself, as previous possible world models (with the exception of [Ghi97]) allow only the storage of ground values, whereas this model treats storage of all values. But it also provides a good example of the benefits of CBPV, because in the corresponding model for CBV (suggested independently by O'Hearn) the semantics of functions is unwieldy—CBPV decomposes it into manageable pieces.

- Based on the control model, we introduce in Chap. 8 another language called Jump-With-Argument, equivalent to—but much simpler than—CBPV with control effects. Unlike CBPV, Jump-With-Argument is not really new, as it is essentially Steele's CPS intermediate language [Ste78]. This language enables us to see that the OPS transform (the CBPV analogue of the CPS transform) provides a jumping implementation for CBPV, just as Steele explained for CBV. We see how the explicit control flow described in Sect. 1.5.3 makes CBPV a good starting point for such an analysis.

Our final piece of evidence for the advantages of CBPV is Hyland-Ong-style game semantics, which we discuss in Chap. 9. We see once again that CBPV is much closer to the semantics than the usual CBN languages (PCF and Idealized Algol) because of its explicit control flow described in Sect. 1.5.3. Key notions of game semantics—the question/answer distinction, pointers between moves, the bracketing condition—become clearer from a CBPV viewpoint. For example, we see that "asking a question" corresponds to forcing, while "answering" corresponds to producing.

In Sect. 9.3–9.4 we give a conceptually simple game semantics for Jump-With-Argument and thereby relate the CBPV game semantics in Chap. 9 to the jumping implementation in Chap. 8.

Before we give a categorical account of the CBPV equational theory (Part III) we give a background chapter on categorical semantics of effect-free languages, which is independent of the rest of the thesis. Such a chapter may seem unnecessary, as there is a large literature on the subject and, after all, we are concerned only with simply typed languages. However, there are some points which we wish to look at carefully.

- We examine the nature of the relationship between an equational theory and its categorical semantics—what do we mean when we say, for example, that "a model of a language with finite products is precisely a cartesian category"?

- We look at the use of *locally indexed categories* to give a simple, abstract description (which has appeared in [Jac99]) of the notion of *distributive coproduct*. Later we will use locally indexed categories for CBPV, but we see here their importance even in the effect-free setting.

We are then in a position to give the categorical account of the CBPV equational theory. After a brief overview (Chap. 11), we present 3 approaches, thereby relating CBPV to a variety of research and ideas in the literature.

- In Chap. 12 we look at semantics using *strong monads*, in the style of Moggi. This approach is probably the most familiar to the reader.

- In Chap. 13 we look at semantics using *value/producer structures*, in the style of Power and Robinson. This approach is the closest to the CBPV syntax.

- In Chap. 14 we look at semantics using *adjunctions*. This approach is the most elegant. Furthermore, since adjunction is an important mathematical concept, it is valuable to see how CBPV relates to it. Unfortunately, adjunction models do not agree exactly with CBPV,

in the sense that non-equivalent adjunction models can give the same CBPV model. But all the concrete models from Part II do indeed arise naturally as adjunction models.

We complete our categorical account in Chap. 15, where we relate our various approaches, and see how this gives us several ways of characterizing the notion of *homomorphism* between computation objects.

Finally, in Chap. 16, we look at some problems and possible directions for further work.

1.6.3 Chapter Dependence

The dependence between chapters and sections is shown in Fig. 1.1. A line between two sections

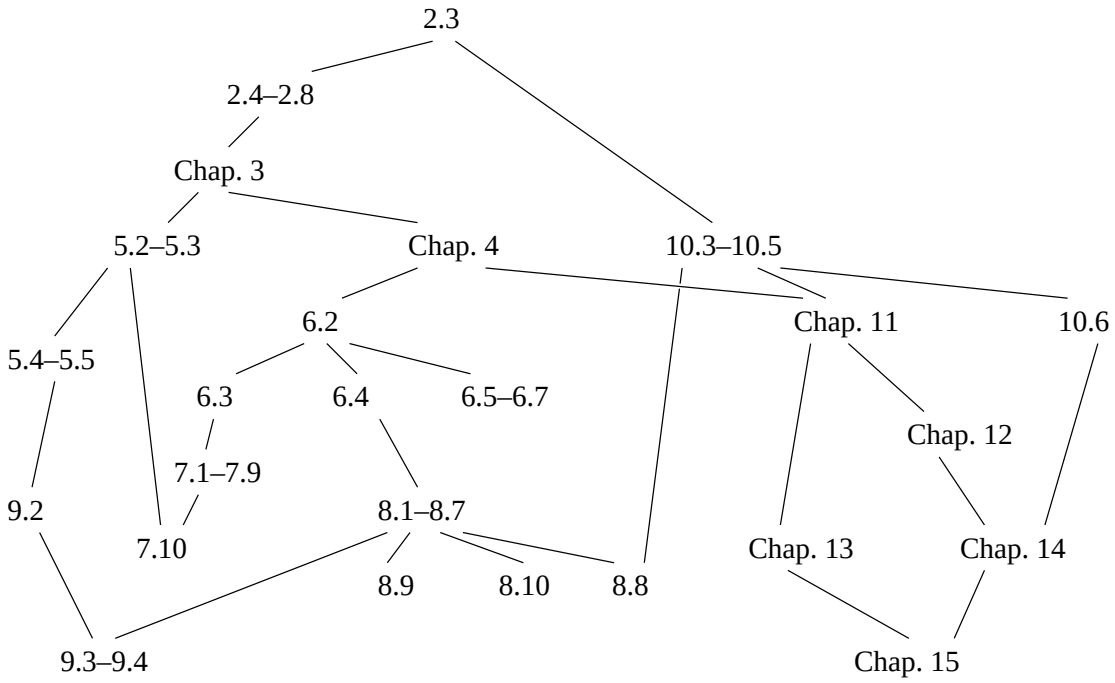


Figure 1.1: Chapter and Section Dependence

or chapters indicates that the lower one depends on the upper one. The diagram does not cover examples, only the main text. Thus, we will sometimes describe an example that depends on a chapter not indicated here.

We group the categorical semantics of Jump-With-Argument (8.8) with the other Jump-With-Argument material (Chap. 8), even though it requires some of the early material from Part III.

Although the language modelled by the pointer game semantics of Sect. 9.2 includes both store and control effects, the reader can understand the semantics and the examples without knowing about these effects, so we have not indicated them as dependences.

1.6.4 Proofs

We have usually omitted proofs which are straightforward inductions, and we have omitted proofs of results for CBV and CBN where we have given corresponding results for CBPV.

Chapter 2

Call-By-Value and Call-By-Name

2.1 Introduction

There is a certain tension in the presentation of CBPV: how much attention shall we devote to CBN and CBV? On the one hand, we are claiming that CBPV "subsumes" CBN and CBV, and to see how that is achieved we have to discuss CBN and CBV, at least to some extent. On the other hand, a thorough study of CBN and CBV would be a waste of effort, since a primary purpose of CBPV is to relieve us of that task— once we have CBPV (which we want to introduce as early as possible), CBV and CBN are seen to be just particular fragments of it.

The CBN/CBV material is therefore organized as follows. In this chapter, we look informally at the key concepts and properties of these paradigms, assuming no prior knowledge of them. This provides background for CBPV which we introduce in Chap. 3. But for the interested reader, we provide in Appendix A a thorough, technical treatment of CBN and CBV and their relationship to CBPV. This chapter can thus be seen as a synopsis of Appendix A.

2.2 The Main Point Of The Chapter

The main point of the chapter is this:

> CBV types and CBN types denote different kinds of things.

This is true both for printing semantics and for Scott semantics:

- In our printing semantics, a CBV type denotes a set whereas a CBN type denotes an $\mathcal{A}$-set (which we will define in Sect. 2.7.4).
- In Scott semantics, a CBV type denotes a cpo whereas a CBN type denotes a pointed cpo.

The importance of this main point is that it makes it clear why CBPV, the subsuming paradigm, will need to have two disjoint classes of type.

Unfortunately, the Scott semantics obscures this main point, because a pointed cpo is a special kind of cpo. By contrast, the class of sets and the class of $\mathcal{A}$-sets are (as will be apparent once we have defined $\mathcal{A}$-sets) disjoint. Thus printing illustrates our main point much better than divergence does. This is why we have chosen printing as our leading example of a computational effect.

2.3 A Simply Typed λ-Calculus

2.3.1 The Language

Before we look at any computational effects, we study an effect-free language. We look at the simply typed λ-calculus whose only ground type is a boolean type, extended with binary sum types; we call this $\lambda\,\mathtt{bool}+$. Its types are

$$A ::= \quad \mathtt{bool} \mid A + A \mid A \to A$$

and its terms are given in Fig. 2.1. We write `pm` for "pattern-match"; and recall that we write ‘ for operand-first application. While `let` is not strictly necessary (`let x be` M. N can be desugared as M‘$\lambda\mathtt{x}N$), it is convenient to include it as a primitive.

$$\frac{}{\Gamma,\mathtt{x}:A,\Gamma'\vdash \mathtt{x}:A} \qquad \frac{\Gamma\vdash M:A \quad \Gamma,\mathtt{x}:A\vdash N:B}{\Gamma\vdash \mathtt{let\ x\ be}\ M.\ N:B}$$

$$\frac{}{\Gamma\vdash \mathtt{true}:\mathtt{bool}} \qquad \frac{}{\Gamma\vdash \mathtt{false}:\mathtt{bool}}$$

$$\frac{\Gamma\vdash M:\mathtt{bool} \quad \Gamma\vdash N:B \quad \Gamma\vdash N':B}{\Gamma\vdash \mathtt{if}\ M\ \mathtt{then}\ N\ \mathtt{else}\ N':B}$$

$$\frac{\Gamma\vdash M:A}{\Gamma\vdash \mathtt{inl}\ M:A+A'} \qquad \frac{\Gamma\vdash M:A'}{\Gamma\vdash \mathtt{inr}\ M:A+A'}$$

$$\frac{\Gamma\vdash M:A+A' \quad \Gamma,\mathtt{x}:A\vdash N:B \quad \Gamma,\mathtt{x}:A'\vdash N':B}{\Gamma\vdash \mathtt{pm}\ M\ \mathtt{as}\ \{\mathtt{inl\ x}.N,\mathtt{inr\ x}.N'\}:B}$$

$$\frac{\Gamma,\mathtt{x}:A\vdash M:B}{\Gamma\vdash \lambda\mathtt{x}.M:A\to B} \qquad \frac{\Gamma\vdash M:A \quad \Gamma\vdash N:A\to B}{\Gamma\vdash M\text{‘}N:B}$$

Figure 2.1: Terms of $\lambda\,\mathtt{bool}+$

$\lambda\,\mathtt{bool}+$ has a straightforward semantics where types denote sets and terms denote functions. A closed term of type `bool` denotes either `true` or `false`; there is an easy decision procedure to find which. We call `bool` the ground type

2.3.2 Product Types

Suppose we want to add product types $A\times A'$ to $\lambda\,\mathtt{bool}+$. It is clear what the introduction rule should be:

$$\frac{\Gamma\vdash M:A \quad \Gamma\vdash M':A'}{\Gamma\vdash (M,M'):A\times A'}$$

But we have a choice as to the form of the elimination rule. Either we can use *projections*:

$$\frac{\Gamma\vdash M:A\times A'}{\Gamma\vdash \pi M:A} \qquad \frac{\Gamma\vdash M:A\times A'}{\Gamma\vdash \pi' M:A'}$$

Or we can use *pattern-matching*:

$$\frac{\Gamma\vdash M:A\times A' \quad \Gamma,\mathtt{x}:A,\mathtt{y}:A'\vdash N:B}{\Gamma\vdash \mathtt{pm}\ M\ \mathtt{as}\ (\mathtt{x},\mathtt{y}).N:B}$$

At first sight, the choice between projections and pattern-matching seems unimportant, because these two forms of elimination rule are equivalent:

$$\begin{array}{rcl} \pi M & = & \mathtt{pm}\ M\ \mathtt{as}\ (\mathtt{x},\mathtt{y}).\ \mathtt{x} \\ \pi' M & = & \mathtt{pm}\ M\ \mathtt{as}\ (\mathtt{x},\mathtt{y}).\ \mathtt{y} \\ \mathtt{pm}\ M\ \mathtt{as}\ (\mathtt{x},\mathtt{y}).N & = & \mathtt{let}\ \mathtt{x}\ \mathtt{be}\ \pi M,\mathtt{y}\ \mathtt{be}\ \pi' M.\ N \end{array}$$

In the presence of effects (CBN and CBV), however, these equations are not necessarily valid and the choice does matter. This is discussed in Appendix A.

Notice

- the resemblance between a pattern-match product and a sum type—each of these types has an elimination rule using pattern-matchingw;
- the resemblance between a projection product and a function type from $\{0,1\}$. For we can think of a tuple (M,M') of projection product type as a function taking 0 to M and 1 to M'. To emphasize this resemblance, we use a novel notation: we write

$$\begin{array}{rcl} (M,M') & \text{as} & \lambda\{0.M, 1.M'\} \\ \pi M & \text{as} & 0\text{`}M \\ \pi' M & \text{as} & 1\text{`}M \end{array}$$

Because of these resemblances, we can understand the key issues in CBV and CBN without having to include products. That is why, in this chapter, we will not consider product types further. They are dealt with fully in Appendix A.

There are more type constructors we could include while remaining simply typed, and in Appendix A we will include them so that our treatment of CBV and CBN there is as thorough as possible. The type system used in this chapter, therefore, provides only a fragment of the full type system that a (simply typed) CBN or CBV language can allow. However, our aim here is just to explain the key ideas, and the types in $\lambda\,\mathtt{bool}{+}$are quite sufficient for that purpose.

2.3.3 Equations

Each type constructor $(\to, \mathtt{bool}, +)$ has two associated equations called the β-law and the η-law.

We look first at the β-laws, which are straightforward.

- The β-law for $A \to B$:

$$M\text{`}\lambda\mathtt{x}.N = N[M/\mathtt{x}]$$

- The β-laws for $\mathtt{bool}$:

$$\begin{array}{c} \mathtt{if}\ \mathtt{true}\ \mathtt{then}\ N\ \mathtt{else}\ N' = N \\ \mathtt{if}\ \mathtt{false}\ \mathtt{then}\ N\ \mathtt{else}\ N' = N' \end{array}$$

- The β-laws for $A + A'$:

$$\begin{array}{c} \mathtt{pm}\ \mathtt{inl}\ M\ \mathtt{as}\ \{\mathtt{inl}\ \mathtt{x}.N, \mathtt{inr}\ \mathtt{x}.N'\} = N[M/\mathtt{x}] \\ \mathtt{pm}\ \mathtt{inr}\ M\ \mathtt{as}\ \{\mathtt{inl}\ \mathtt{x}.N, \mathtt{inr}\ \mathtt{x}.N'\} = N'[M/\mathtt{x}] \end{array}$$

- There is also a β-law for $\mathtt{let}$:

$$\mathtt{let}\ \mathtt{x}\ \mathtt{be}\ M.\ N = N[M/\mathtt{x}]$$

We next turn to the η-laws. They are more subtle than the β-laws, but they are important in understanding the CBV/CBN issues. So we urge the reader to look at them carefully and understand why they are valid in the set semantics.

- The η-law for $A \to B$: any term M of type $A \to B$ can be expanded

$$M = \lambda \mathtt{x}(\mathtt{x}\text{`}M)$$

  where $\mathtt{x}$ does not occur in the context Γ of M.

- The η-law[1] for $\mathtt{bool}$: if M has a free identifier $\mathtt{z} : \mathtt{bool}$ then for any term N of type $\mathtt{bool}$

$$M[N/\mathtt{z}] = \mathtt{if}\ N\ \mathtt{then}\ M[\mathtt{true}/\mathtt{z}]\ \mathtt{else}\ M[\mathtt{false}/\mathtt{z}] \tag{2.3}$$

  Intuitively, this holds because, in a given environment (list of bindings for identifiers), N denotes either true or false.

- The η-law for $A + A'$: if M has a free identifier $\mathtt{z} : A + A'$ then for any term N of type $A + A'$

$$M[N/\mathtt{z}] = \mathtt{pm}\ N\ \mathtt{as}\ \{\mathtt{inl}\ \mathtt{x}.M[\mathtt{inl}\ \mathtt{x}/\mathtt{z}], \mathtt{inr}\ \mathtt{x}.M'[\mathtt{inr}\ \mathtt{x}/\mathtt{z}]\}$$

  where $\mathtt{x}$ does not occur in the context $\Gamma, \mathtt{z} : A$ of M.

Notice the similarity between sum types and $\mathtt{bool}$.

2.3.4 Reversible Derivations

As a consequence of these equations, we have *reversible derivations*, e.g. for $\to$

$$\frac{\Gamma, A \vdash B}{\Gamma \vdash A \to B}$$

This means that from a term of the form $\Gamma, \mathtt{x} : A \vdash M : B$ (assuming $\mathtt{x} \notin \Gamma$) we can construct a term of the form $\Gamma \vdash N : A \to B$ and vice versa and that these operations are inverse up to provable equality. The reversible derivation is given by

- the operation $\theta : M \mapsto \lambda \mathtt{x}.M$, which turns a term $\Gamma, \mathtt{x} : A \vdash M : B$ into a term $\Gamma \vdash N : A \to B$
- the context $\theta^{-1} : N \mapsto \mathtt{x}\text{`}N$, which turns a term $\Gamma \vdash N : A \to B$ into a term $\Gamma, \mathtt{x} : A \vdash M : B$

and it is clear that these operations are inverse up to provable equality, using the β- and η- laws for $\to$.

Furthermore, the operation θ *preserves substitution in* Γ. This means that it satisfies the equation

$$\theta(M[\overrightarrow{N_i/\mathtt{x}_i}]) = \theta(M)[\overrightarrow{N_i}/\mathtt{x}_i]$$

[1] Some authors e.g. [GLT88]give the name "η-law" to the weaker equation

$$M = \mathtt{if}\ M\ \mathtt{then}\ \mathtt{true}\ \mathtt{else}\ \mathtt{false} \tag{2.1}$$

together with the *commuting conversion* law

$$M[\mathtt{if}\ N\ \mathtt{then}\ P\ \mathtt{else}\ P'/\mathtt{z}] = \mathtt{if}\ N\ \mathtt{then}\ M[P/\mathtt{z}]\ \mathtt{else}\ M[P'/\mathtt{z}] \tag{2.2}$$

or a variant of this. (2.1)–(2.2) together are equivalent to our η-law (2.3).

is provable, where $\mathtt{x}_i$ are the identifiers in Γ. (Actually the two sides are the same term, but provable equality is sufficient for our purposes.) It follows that θ^{-1} too preserves substitution in Γ.

The reversible derivations for `bool` and $+$ are

$$\frac{\Gamma \vdash B \quad \Gamma \vdash B}{\Gamma, \mathtt{bool} \vdash B} \qquad \frac{\Gamma, A \vdash B \quad \Gamma, A' \vdash B}{\Gamma, A + A' \vdash B}$$

Like the reversible derivation for $\to$, they preserve substitution in Γ.

Readers familiar with categorical semantics will see that these reversible derivations provide important information about the categorical structure of an equational theory.

2.4 Adding Effects

We now wish to add a computational effect to $\lambda\,\mathtt{bool}{+}$, and we will use output as our example. We suppose that to any term we can prefix a command such as `print 'a'`. For example, the term `print 'p'; true` when evaluated, prints `p` and then produces the result `true`. We write `print "pq"; true` as an abbreviation for `print 'p'; (print 'q'; true)`.

We write $\mathcal{A}$ for the set of characters that can be printed, $\mathcal{A}^*$ for the set of finite strings of characters in $\mathcal{A}$ and $*$ for concatenation. Formally, we add to the term syntax the following rule, for every character $c \in \mathcal{A}$:

$$\frac{\Gamma \vdash M : B}{\Gamma \vdash \mathtt{print}\ c;\ M : B}$$

There are other, equivalent, syntactic possibilities, e.g. in CBV we could let commands have type 1, as in ML. But we prefer commands to be prefixes. This is for the sake of consistency between CBN, CBV, CBPV and the Jump-With-Argument language discussed in Chap. 8.

2.5 The Principles Of Call-By-Value and Call-By-Name

The first consequence of adding effects is that, by contrast with pure $\lambda\,\mathtt{bool}{+}$, the order of evaluation matters. Given a closed term of type `bool`, the output depends on the evaluation order. For some effects the answer too will depend on the evaluation order, but for output that is not the case. The two evaluation orders we will look at are CBV and CBN. We describe the principles of these two paradigms.

Firstly, in both CBV and CBN, we do not evaluate under λ. Thus a λ-abstraction is *terminal*, in the sense that it requires no further evaluation.

Having decided not to evaluate under λ, we have numerous choices still to make.

1. To evaluate the term $\mathtt{let\ x\ be}\ M.\ N$, do we
   (a) evaluate M to T and then evaluate $N[T/\mathtt{x}]$, or
   (b) leave M alone, and just evaluate $N[M/\mathtt{x}]$?
2. To evaluate a term M of sum type, how far do we proceed? Do we
   (a) evaluate M to $\mathtt{inl}\ T$ or $\mathtt{inr}\ T$, where T is terminal, or
   (b) evaluate M to $\mathtt{inl}\ N$ or $\mathtt{inr}\ N$, where N may not be terminal?
3. To evaluate an application such as $M`N$, we certainly need to evaluate N, giving say $\lambda\mathtt{x}.P$. Besides this, do we
   (a) evaluate M to T (either before or after evaluating N) and then evaluate $P[T/\mathtt{x}]$, or

(b) leave M alone, and just evaluate $P[M/\mathtt{x}]$?

In fact, there is one fundamental question whose answer will determine how we answer questions (1)–(3): what may we substitute for an identifier? At one extreme, we allow only a terminal term to replace an identifier—this defines the CBV paradigm. At the other extreme, we allow only a totally unevaluated term to replace an identifier—this defines the CBN paradigm.

To each of questions (1)–(3), CBV requires us to answer (a) and CBN requires us to answer (b). In the case of questions (1) and (3), this is clear because substitution is involved. For question (2), it requires some explanation. Consider a term such as

$$\mathtt{pm}\ M\ \mathtt{as}\ \{\mathtt{inl}\ \mathtt{x}.N, \mathtt{inr}\ \mathtt{y}.N'\}$$

To evaluate this we must first evaluate M. Suppose we evaluate M to $\mathtt{inl}\ M'$. In CBN we must not proceed to evaluate M', because we want to substitute it for $\mathtt{x}$ in N, so it must be completely unevaluated. In CBV we must evaluate M' to a terminal term T, so that we can substitute T for $\mathtt{x}$ in N.

In conclusion, we emphasize the following point:

> The essential difference between CBV and CBN is not (as is often thought) the way that application is evaluated; rather it is what an identifier may be bound to.

Notice that within the CBV paradigm, we have the choice of whether, when evaluating an application $M\text{`}N$, we evaluate M before N or vice versa. In fact, so long as we are consistent, the semantic theory is essentially unaffected. Arbitrarily, we stipulate that we evaluate the operand M before the operator N.

2.6 Call-By-Value

2.6.1 Operational Semantics

In CBV a closed term which is terminal is called a *closed value*. These are given by

$$V ::= \ \mathtt{true} \mid \mathtt{false} \mid \mathtt{inl}\ V \mid \mathtt{inr}\ V \mid \lambda\mathtt{x}.M$$

Every closed term M prints a string of characters m and then produces a closed value V. We write $M \Downarrow m, V$. This is defined inductively in Fig. 2.2.

Proposition 1 For every M there is a unique m, V such that $M \Downarrow m, V$. □

The proof is similar to that of Prop. 9.

2.6.2 Denotational Semantics for `print`

Definition 1

- The following CBV terms are called *values*:

$$V ::= \ \mathtt{x} \mid \mathtt{true} \mid \mathtt{false} \mid \mathtt{inl}\ V \mid \mathtt{inr}\ V \mid \lambda\mathtt{x}.M$$

  (This generalizes the notion of "closed value" we have already used.)

- All CBV terms are called *producers*. (This is because, when evaluated, they produce a value.)

□

Notice that a term is a value iff every closed substitution instance (substituting only closed values) is a closed value.

We now describe and explain a denotational semantics for the CBV printing language. The key principle is that

$$\frac{M \Downarrow m, V \quad N[V/\mathtt{x}] \Downarrow m', W}{\mathtt{let\ x\ be}\ M.\ N \Downarrow m * m', W}$$

$$\frac{}{\mathtt{true} \Downarrow 1, \mathtt{true}} \qquad \frac{}{\mathtt{false} \Downarrow 1, \mathtt{false}}$$

$$\frac{M \Downarrow m, \mathtt{true} \quad N \Downarrow m', V}{\mathtt{if}\ M\ \mathtt{then}\ N\ \mathtt{else}\ N' \Downarrow m * m', V} \qquad \frac{M \Downarrow m, \mathtt{false} \quad N' \Downarrow m', V}{\mathtt{if}\ M\ \mathtt{then}\ N\ \mathtt{else}\ N' \Downarrow m * m', V}$$

$$\frac{M \Downarrow m, V}{\mathtt{inl}\ M \Downarrow m, \mathtt{inl}\ V} \qquad \frac{M \Downarrow m, V}{\mathtt{inr}\ M \Downarrow m, \mathtt{inr}\ V}$$

$$\frac{M \Downarrow m, \mathtt{inl}\ V \quad N[V/\mathtt{x}] \Downarrow m', W}{\mathtt{pm}\ M\ \mathtt{as}\ \{\mathtt{inl\ x}.N, \mathtt{inr\ x}.N'\} \Downarrow m * m', W} \qquad \frac{M \Downarrow m, \mathtt{inr}\ V \quad N'[V/\mathtt{x}] \Downarrow m', W}{\mathtt{pm}\ M\ \mathtt{as}\ \{\mathtt{inl\ x}.N, \mathtt{inr\ x}.N'\} \Downarrow m * m', W}$$

$$\frac{}{\lambda\mathtt{x}.M \Downarrow 1, \lambda\mathtt{x}.M} \qquad \frac{M \Downarrow m, V \quad N \Downarrow m', \lambda\mathtt{x}.N' \quad N'[V/\mathtt{x}] \Downarrow m'', W}{M\text{`}N \Downarrow m * m' * m'', W}$$

$$\frac{M \Downarrow m, V}{\mathtt{print}\ c;\ M \Downarrow c * m, V}$$

Figure 2.2: Big-Step Semantics for CBV with `print`

> each type A denotes a set $[\![A]\!]$ whose elements are the denotations of closed *values* of type A.

Thus the type `bool` denotes the 2-element set $\{\mathsf{true}, \mathsf{false}\}$ because there are two closed values of type `bool`. Likewise the type $A + A'$ denotes $[\![A]\!] + [\![A']\!]$ because a closed value of type $A + A'$ must be either of the form $\mathtt{inl}\ V$, where V is a closed value of type A, or of the form $\mathtt{inr}\ V$, where V is a closed value of type A'. We shall come to $A \to B$ presently.

Given a closed value V of type A, we write $[\![V]\!]^{\mathsf{val}}$ for the element of $[\![A]\!]$ that it denotes. Given a closed producer M of type A, we recall that M prints a string of characters $m \in \mathcal{A}^*$ and then produces a closed value V of type A. So M will denote an element $[\![M]\!]^{\mathsf{prod}}$ of $\mathcal{A}^* \times [\![A]\!]$. Thus a closed value V will have two denotations $[\![V]\!]^{\mathsf{val}}$ and $[\![V]\!]^{\mathsf{prod}}$ related by

$$[\![V]\!]^{\mathsf{prod}} = (1, [\![V]\!]^{\mathsf{val}})$$

A closed value of type $A \to B$ is of the form $\lambda\mathtt{x}.M$. This, when applied to a closed *value* of type A gives a closed *producer* of type B. So $A \to B$ denotes $[\![A]\!] \to (\mathcal{A}^* \times [\![B]\!])$. It is true that the syntax appears to allow us to apply $\lambda\mathtt{x}.M$ to any producer N of type A, not just to a value. But N will be evaluated before it interacts with $\lambda\mathtt{x}.M$, so $\lambda\mathtt{x}.M$ is really only applied to the value that N produces.

Given a context $\Gamma = \mathtt{x}_0 : A_0, \ldots, \mathtt{x}_{n-1} : A_{n-1}$, an environment (list of bindings for identifiers) associates to each $\mathtt{x}_i$ a closed value of type A_i. So the environment denotes an element of $[\![A_0]\!] \times \cdots \times [\![A_{n-1}]\!]$, and we write $[\![\Gamma]\!]$ for this set.

Given a value $\Gamma \vdash^{\mathsf{v}} V : B$, we see that V, together with an environment, gives (by substitution) a closed value of type B. So V denotes a function $[\![V]\!]^{\mathsf{val}}$ from $[\![\Gamma]\!]$ to $[\![B]\!]$.

Given a producer $\Gamma \vdash^{\mathsf{c}} M : B$, we see that M, together with an environment, gives (by substitution) a closed producer of type B. So M denotes a function $[\![M]\!]^{\mathsf{prod}}$ from $[\![\Gamma]\!]$ to $\mathcal{A}^* \times$

$[\![B]\!]$.

Generalizing what we saw for closed values, an arbitrary value V will have two denotations $[\![V]\!]^{\mathsf{val}}$ and $[\![V]\!]^{\mathsf{prod}}$ related by

$$[\![V]\!]^{\mathsf{prod}}\rho = (1, [\![V]\!]^{\mathsf{val}}\rho)$$

for each environment ρ.

In summary, the denotational semantics is organized as follows.

- A type A denotes a set $[\![A]\!]$.
- A context $\mathtt{x}_0 : A_0, \ldots, \mathtt{x}_{n-1} : A_{n-1}$ denotes the set $[\![A_0]\!] \times \cdots \times [\![A_{n-1}]\!]$.
- A value $\Gamma \vdash V : B$ denotes a function $[\![V]\!]^{\mathsf{val}} : [\![\Gamma]\!] \longrightarrow [\![B]\!]$
- A term $\Gamma \vdash M : B$ denotes a function $[\![M]\!]^{\mathsf{prod}} : [\![\Gamma]\!] \longrightarrow \mathcal{A}^* \times [\![B]\!]$

The denotations of types is given by

$$\begin{array}{rcl} [\![\mathtt{bool}]\!] & = & \{\mathsf{true}, \mathsf{false}\} \\ [\![A + A']\!] & = & [\![A]\!] + [\![A']\!] \\ [\![A \to B]\!] & = & [\![A]\!] \to (\mathcal{A}^* \times [\![B]\!]) \end{array}$$

The denotations of values—some example clauses:

$$\begin{array}{rcl} [\![\mathtt{x}]\!]^{\mathsf{val}}(\rho, \mathtt{x} \mapsto x, \rho') & = & x \\ [\![\mathtt{true}]\!]^{\mathsf{val}}\rho & = & \mathsf{true} \\ [\![\mathtt{inl}\ V]\!]^{\mathsf{val}}\rho & = & \mathsf{inl}\ [\![V]\!]^{\mathsf{val}}\rho \\ [\![\lambda\mathtt{x}.M]\!]^{\mathsf{val}}\rho & = & \lambda x.[\![M]\!]^{\mathsf{prod}}(\rho, \mathtt{x} \mapsto x) \end{array}$$

The denotations of producers—some example clauses:

$$\begin{array}{rcl} [\![\mathtt{x}]\!]^{\mathsf{prod}}(\rho, \mathtt{x} \mapsto x, \rho') & = & (1, x) \\ [\![\mathtt{true}]\!]^{\mathsf{prod}}\rho & = & (1, \mathsf{true}) \\ [\![\mathtt{inl}\ M]\!]^{\mathsf{prod}} & = & (m, \mathsf{inl}\ v) \text{ where } [\![M]\!]^{\mathsf{prod}}\rho = (m, v) \\ [\![\mathtt{if}\ M\ \mathtt{then}\ N\ \mathtt{else}\ N']\!]\rho & = & \begin{cases} (m * m', v) \text{ if } [\![M]\!]^{\mathsf{prod}}\rho = (m, \mathsf{true}) \text{ and } [\![N]\!]^{\mathsf{prod}}\rho = (m', v) \\ (m * m', v) \text{ if } [\![M]\!]^{\mathsf{prod}}\rho = (m, \mathsf{false}) \text{ and } [\![N']\!]^{\mathsf{prod}}\rho = (m', v) \end{cases} \\ [\![\lambda\mathtt{x}.M]\!]^{\mathsf{prod}}\rho & = & (1, \lambda x.[\![M]\!]^{\mathsf{prod}}(\rho, \mathtt{x} \mapsto x)) \\ [\![M \text{`} N]\!]^{\mathsf{prod}}\rho & = & (m * m' * m'', w) \text{ where } [\![M]\!]^{\mathsf{prod}}\rho = (m, v) \\ & & \text{and } [\![N]\!]^{\mathsf{prod}}\rho = (m', f) \text{ and } v \text{`} f = (m'', w) \end{array}$$

Notice how strongly these clauses resemble the corresponding clauses of Fig. 2.2.

Proposition 2 (soundness) If $M \Downarrow m, V$ then $[\![M]\!]^{\mathsf{prod}} = (m, [\![V]\!]^{\mathsf{val}})$ □

Corollary 3 (by Prop. 1) For any closed ground producer (i.e. producer of ground type) M, we have $M \Downarrow m, \mathtt{produce}\ n$ iff $[\![M]\!] = (m, n)$. □

2.6.3 Scott Semantics

The reader may be familiar with Scott semantics for CBV. This has appeared in two forms:

1. in the older form, types denote pointed cpos, producers denote strict functions and values denote strict, bottom-reflecting functions;
2. in the more recent form, due to Plotkin [Plo85], types denote (unpointed) cpos, values denote total functions and producers denote partial functions.

Although these two semantics are equivalent, (2) is more natural, and it agrees with the key principle of our printing semantics: the elements of $[\![A]\!]$ are denotations of closed *values* of type A. We outline the Scott semantics in form (2)[2] to make apparent its similarity to the printing semantics.

Definition 2 1. A *cpo* $(X, \leqslant)$ is a poset with joins of all directed subsets.

2. We write **Cpo** for the category of cpos and continuous functions.

□

The semantics is organized as follows.

- A type A denotes a cpo $[\![A]\!]$. Our intention is that a closed value of type A will denote an element of $[\![A]\!]$.
- A context $\Gamma = A_0, \ldots, A_{n-1}$ denotes the cpo $[\![\Gamma]\!] = [\![A_0]\!] \times \cdots \times [\![A_{n-1}]\!]$. Intuitively, this is the set of environments for Γ, because an identifier can be bound only to a closed value.
- For a value $\Gamma \vdash V : A$, we see that given an environment we obtain (by substitution) a closed value. So V denotes a continuous function $[\![V]\!]^{\mathsf{val}}$ from $[\![\Gamma]\!]$ to $[\![A]\!]$.
- For a producer $\Gamma \vdash M : A$, we see that, given an environment, we obtain (by substitution) a closed producer, which either diverges or produces a value of type A. So M denotes a continuous function $[\![M]\!]^{\mathsf{prod}}$ from $[\![\Gamma]\!]$ to $[\![A]\!]_\perp$.

The semantics of types is given as follows:

- `bool` denotes the flat 2-element cpo $\{\mathsf{true}, \mathsf{false}\}$, because a closed value of type `bool` is either `true` or `false`.
- $A + A'$ denotes the disjoint union of $[\![A]\!]$ and $[\![A']\!]$.
- $A \to B$ denotes the cpo of continuous functions from $[\![A]\!]$ to $[\![B]\!]_\perp$, because a closed value of type $A \to B$ is of the form $\lambda \mathtt{x}.M$, and this, when applied to a closed value of type A gives (by substitution) a closed producer of type B.

We omit the semantics of terms.

2.6.4 The Monad Approach

We briefly mention a categorical viewpoint on CBV due to Moggi [Mog91]. Readers unfamiliar with this viewpoint or with category theory may omit this section—a full treatment of the monad approach will be given in Chap. 12.

Suppose we have a category $\mathcal{C}$ with finite products (suitable coproducts are also required but we leave this aside) and a strong monad T. Then, assuming sufficient exponents in $\mathcal{C}$, we obtain

[2] We make the slight modification of using total functions to lifted cpos instead of partial functions.

a CBV model: a CBV value denotes a morphism of $\mathcal{C}$, and a CBV producer denotes a morphism of the Kleisli category $\mathcal{C}_T$.

Both of our denotational models are instances of this situation. For the printing semantics, $\mathcal{C}$ is **Set** and T is the strong monad $\mathcal{A}^* \times -$ (Moggi's "interactive output" monad). For the cpo semantics, $\mathcal{C}$ is **Cpo** (the category of cpos and continuous functions) and T is the lifting monad.

2.6.5 Observational Equivalence

Definition 3 A *context* $\mathcal{C}[]$ is a term with zero or more occurrences of a hole $[]$. If $\mathcal{C}[]$ is of ground type we say it is a *ground context*. □

Definition 4 Given two terms $\Gamma \vdash M, M' : B$, we say that

1. $M \simeq_{\mathsf{ground}} M'$ when for all ground contexts $\mathcal{C}[]$, $\mathcal{C}[M] \Downarrow m, i$ iff $\mathcal{C}[M'] \Downarrow m, i$;
2. $M \simeq_{\mathsf{anytype}} M'$ when for all contexts $\mathcal{C}[]$ of any type, $\mathcal{C}[M] \Downarrow m, V$ for some V iff $\mathcal{C}[M'] \Downarrow m, V$ for some V

□

Proposition 4 The two relations $\simeq_{\mathsf{ground}}$ and $\simeq_{\mathsf{anytype}}$ are the same. □

This is an important feature of CBV: it does not matter whether we allow observation at ground type or every type.

Cor. 3 implies that terms with the same denotaton are observationally equivalent.

2.6.6 Coarse-Grain CBV vs. Fine-Grain CBV

The form of CBV we have looked at is called *coarse-grain CBV*. It suffers from two problems.

1. Our decision to evaluate operand before operator was arbitrary.
2. A value V has two denotations: $[\![V]\!]^{\mathsf{val}}$ and $[\![V]\!]^{\mathsf{prod}}$.

We can eliminate these problems by presenting a more refined calculus called *fine-grain CBV* in which (partially based on [Mog91]) we make a syntactic distinction between values and producers. This calculus is more suitable for formulating an equational theory. Foralthough Moggi in [Mog88] provided a theory for coarse-grain CBV, which he called "λ_{c}-calculus", this theory was not purely equational but required an additional predicate to assert that a term is a value.

We will not trouble to study fine-grain CBV in this chapter, because in the next chapter we will present CBPV, which is even more fine-grain. A treatment of fine-grain CBV and its equational theory is given in the Appendix.

2.7 Call-By-Name

2.7.1 Operational Semantics

In CBN the following terms are terminal:

$$T ::= \quad \mathtt{true} \mid \mathtt{false} \mid \mathtt{inl}\, M \mid \mathtt{inr}\, M \mid \lambda \mathtt{x}.M$$

Every closed term M prints a string of characters and terminates at a terminal term T. We write $M \Downarrow m, T$. This is defined inductively in Fig. 2.3.

Proposition 5 For every M there is a unique m, T such that $M \Downarrow m, T$. □

The proof is similar to that of Prop. 9.

$$\frac{N[M/\mathtt{x}] \Downarrow m, T}{\mathtt{let\ x\ be}\ M.\ N \Downarrow m, T}$$

$$\frac{}{\mathtt{true} \Downarrow 1, \mathtt{true}} \qquad \frac{}{\mathtt{false} \Downarrow 1, \mathtt{false}}$$

$$\frac{M \Downarrow m, \mathtt{true} \quad N \Downarrow m', T}{\mathtt{if}\ M\ \mathtt{then}\ N\ \mathtt{else}\ N' \Downarrow m * m', T} \qquad \frac{M \Downarrow m, \mathtt{false} \quad N' \Downarrow m', T}{\mathtt{if}\ M\ \mathtt{then}\ N\ \mathtt{else}\ N' \Downarrow m * m', T}$$

$$\frac{}{\mathtt{inl}\ M \Downarrow 1, \mathtt{inl}\ M} \qquad \frac{}{\mathtt{inr}\ M \Downarrow 1, \mathtt{inr}\ M}$$

$$\frac{M \Downarrow m, \mathtt{inl}\ M' \quad N[M'/\mathtt{x}] \Downarrow m', T}{\mathtt{pm}\ M\ \mathtt{as}\ \{\mathtt{inl\ x}.N, \mathtt{inr\ x}.N'\} \Downarrow m * m', T} \qquad \frac{M \Downarrow m, \mathtt{inr}\ M' \quad N'[M'/\mathtt{x}] \Downarrow m', T}{\mathtt{pm}\ M\ \mathtt{as}\ \{\mathtt{inl\ x}.N, \mathtt{inr\ x}.N'\} \Downarrow m * m', T}$$

$$\frac{}{\lambda\mathtt{x}.M \Downarrow 1, \lambda\mathtt{x}.M} \qquad \frac{N \Downarrow m, \lambda\mathtt{x}.N' \quad N'[M/\mathtt{x}] \Downarrow m', T}{M`N \Downarrow m * m', T}$$

$$\frac{M \Downarrow m, T}{\mathtt{print}\ c;\ M \Downarrow c * m, T}$$

Figure 2.3: Big-Step Semantics for CBN with `print`

2.7.2 Observational Equivalence

The denotational semantics for CBN is more subtle than that for CBV, so we first look at observational equivalences.

Definition 5 A *context* $\mathcal{C}[]$ is a term with zero or more occurrences of a hole $[]$. If $\mathcal{C}[]$ is of ground type we say it is a *ground context*. □

Definition 6 Given two terms $\Gamma \vdash M, M' : B$, we say that

1. $M \simeq_{\mathsf{ground}} M'$ when for all ground contexts $\mathcal{C}[]$, $\mathcal{C}[M] \Downarrow m, i$ iff $\mathcal{C}[M'] \Downarrow m, i$;
2. $M \simeq_{\mathsf{anytype}} M'$ when for all contexts $\mathcal{C}[]$ of any type, $\mathcal{C}[M] \Downarrow m, T$ for some T iff $\mathcal{C}[M'] \Downarrow m, T$ for some T.

□

By contrast with Prop. 4 we have the following.

Proposition 6 The relation $\simeq_{\mathsf{anytype}}$ is strictly finer than $\simeq_{\mathsf{ground}}$. □

Perhaps the simplest example of this proposition is

$$\mathtt{print\ "hello";}\ \lambda\mathtt{x}.M \simeq_{\mathsf{ground}} \lambda\mathtt{x}.(\mathtt{print\ "hello";}\ M) \tag{2.4}$$

Obviously the trivial context $[]$, which is not ground, distinguishes the two sides. An intuitive explanation (not a rigorous proof) of (2.4) is that, inside a ground CBN term, the only way to cause a subterm of type $A \to B$ to be evaluated is to apply it.

If we use divergence rather than printing, the analogous example is the equivalence

$$\mathtt{diverge} \simeq_{\mathsf{ground}} \lambda \mathtt{x}.\mathtt{diverge}$$

This general CBN phenomenon can be described as "effects commute with λ". Various authors studying CBN languages with only ground types and function types have exploited it by allowing certain features at ground type only, as the corresponding features at function type are then definable: erratic choice [HA80], control effects [Lai97] and conditional branching [Plo77].

For both printing and divergence (indeed for all effects), we have, for similar reasons, the η-law for functions: any term M of type $A \to B$ can be expanded

$$M \simeq_{\mathsf{ground}} \lambda \mathtt{x}.(\mathtt{x}\text{`}M) \tag{2.5}$$

where $\mathtt{x}$ is not in the context Γ of M.

2.7.3 CBN vs. Lazy

Definition 7

- We use the term "CBN" (equations, models etc.) to refer to the CBN operational semantics together with $\simeq_{\mathsf{ground}}$.
- We use the term "lazy" (equations, models etc.) to refer to the CBN operational semantics together with $\simeq_{\mathsf{anytype}}$.

□

Thus in a CBN model the η-law (2.5) must be validated, whereas in a lazy model (2.5) must not be validated.

Our usage of "lazy" follows [Abr90, Ong88]. However, the terminology in the literature is not consistent, and the reader should beware the following.

1. "Lazy" is widely used to describe call-by-need.
2. "Call-by-name" is sometimes used to mean (our sense of) "lazy", especially in the continuation literature [HD97, Plo76] and the monad literature [Mog91].
3. In the *untyped* λ-calculus literature, the phrase "call-by-name" is used with a slightly different meaning from ours, and necessarily so because there is no ground type—there is just one type and it is a function type. In order for (2.5) to hold despite observation at this type, a different operational semantics is used: reduction continues to *head normal form* [Wad76].

The lazy paradigm is treated in Appendix A. We see there that it is subsumed in CBV, so its denotational semantics is straightforward.

By contrast, in the CBN paradigm much of the big-step semantics is not observable. For example, the two sides of (2.4) have different operational behaviour, yet that difference cannot be observed. So a denotational semantics, in order to validate (2.4), must conceal part of the big-step semantics. For this reason, the CBN paradigm is more subtle than CBV.

2.7.4 Denotational Semantics for `print`

Definition 8

1. An $\mathcal{A}$-set $(X, *)$ consists of a set X together with a function $*$ from $\mathcal{A} \times X$ to X. We call X the *carrier* and $*$ the *structure*.
2. An *element* of $(X, *)$ is an element of X.
3. A *function* from a set W to $(X, *)$ is a function from W to X.

□

Each CBN type A denotes an $\mathcal{A}$-set $[\![A]\!] = (X, *)$. Our intention is that a closed term M of type A will then denote an element $[\![M]\!]$ of $(X, *)$, and `print` c; M will denote $c * [\![M]\!]$. Thus $*$ provides a way of "absorbing" the effect into X. Given an $\mathcal{A}$-set $(X, *)$ we can extend $*$ to a function from $\mathcal{A}^* \times X$ to X in the evident way—we call the extension $*$ too. This extension allows us to interpret `print` m; M directly.

Here are some ways of constructing $\mathcal{A}$-sets.

Definition 9 1. For any set X, the *free* $\mathcal{A}$-set on X has carrier $\mathcal{A}^* \times X$ and we set $c * (m, x)$ to be $(c * m, x)$.

2. For an $i \in I$-indexed family of $\mathcal{A}$-sets $(X_i, *)$, we define the $\mathcal{A}$-set $\prod_{i \in I}(X_i, *)$ to have carrier $\prod_{i \in I} X_i$ and structure given pointwise: $\hat{\imath}\text{`}(c * f) = c * (\hat{\imath}\text{`} f)$.

3. For any set X and $\mathcal{A}$-set $(Y, *)$, we define the $\mathcal{A}$-set $X \to (Y, *)$ to have carrier $X \to Y$ and structure given pointwise: $x\text{`}(c * f) = c * (x\text{`} f)$.

□

The semantics of types is as follows:

- `bool` denotes the free $\mathcal{A}$-set on $\{\mathsf{true}, \mathsf{false}\}$. This is because any closed term of type `bool` prints a string and then terminates as `true` or `false`, and this behaviour is observable.

- If $[\![A]\!] = (X, *)$ and $[\![A']\!] = (X', *)$ then $A + A'$ denotes the free $\mathcal{A}$-set on $X + X'$. This is because any closed term of this type prints a string and then terminates as `inl` M or `inr` M, and this behaviour is observable.

- If $[\![A]\!] = (X, *)$ and $[\![B]\!] = (Y, *)$ then $A \to B$ denotes $X \to (Y, *)$. This is because any closed term of this type is equivalent to some $\lambda \mathtt{x}.M$, and prefixing with a `print` command is then given by (2.4).

Given a context $\mathtt{x}_0 : A_0, \ldots, \mathtt{x}_{n-1} : A_{n-1}$, an environment consists of a sequence of closed terms $M_0, \ldots, M_{n-1}$. Writing $(X_i, *)$ for $[\![A_i]\!]$, this environment denotes an element of $X_0 \times \cdots \times X_{n-1}$ We say that the context denotes this set. (There is no need to retain the structure.)

Given a term $\Gamma \vdash M : B$, from any environment we obtain (by substitution) a closed term of type B. So M will denote a function from $[\![\Gamma]\!]$ to $[\![B]\!]$.

Semantics of terms—some example clauses:

$$
\begin{array}{rcl}
[\![\mathtt{x}]\!](\rho, \mathtt{x} \mapsto x, \rho') & = & x \\
[\![\mathtt{let\ x\ be}\ M.\ N]\!] & = & [\![M]\!](\rho, \mathtt{x} \mapsto [\![N]\!]\rho) \\
[\![\mathtt{true}]\!]\rho & = & (1, \mathsf{true}) \\
[\![\mathtt{if}\ M\ \mathtt{then}\ N\ \mathtt{else}\ N']\!] & = & \begin{cases} m * [\![N]\!]\rho \text{ if } [\![M]\!]\rho = (m, \mathsf{true}) \\ m * [\![N']\!]\rho \text{ if } [\![M]\!]\rho = (m, \mathsf{false}) \end{cases} \\
[\![\mathtt{inl}\ M]\!]\rho & = & (1, \mathsf{inl}\ [\![M]\!]\rho) \\
[\![\mathtt{pm}\ M\ \mathtt{as}\ \{\mathtt{inl\ x}.N, \mathtt{inr\ y}.N'\}]\!] & = & \begin{cases} m * [\![N]\!](\rho, \mathtt{x} \mapsto a) \text{ if } [\![M]\!]\rho = (m, \mathsf{inl}\ a) \\ m * [\![N']\!](\rho, \mathtt{x} \mapsto a') \text{ if } [\![M]\!]\rho = (m, \mathsf{inr}\ a') \end{cases} \\
[\![\lambda \mathtt{x}.M]\!] & = & \lambda x.[\![M]\!](\rho, \mathtt{x} \mapsto x) \\
[\![M\text{`}N]\!] & = & ([\![M]\!]\rho)\text{`}([\![N]\!]\rho) \\
[\![\mathtt{print}\ c;\ M]\!]\rho & = & c * [\![M]\!]\rho
\end{array}
$$

Proposition 7 (soundness) If $M \Downarrow m, T$ then $[\![M]\!] = m * [\![T]\!]$. □

Corollary 8 (by Prop. 5) For any closed ground term M (term of ground type), $M \Downarrow m, \texttt{produce}\, n$ iff $[\![M]\!] = (m, n)$. Hence terms with the same denotation are observationally equivalent. □

Notice the sequencing of effects in the semantics of `if` and `pm`. This is characteristic of CBN: pattern-matching is what finally causes evaluation to happen.

We emphasize that we have not used any notion of "homomorphism" between $\mathcal{A}$-sets. But, for use in Sect. 4.7 and Part III, we define this notion here.

Definition 10 1. A *homomorphism* from $(X, *)$ to $(Y, *)$ is a function $x \xrightarrow{f} y$ such that $f(c * x) = c * f(x)$. If f is a bijection, then f^{-1} too is a homomorphism, so we say that f is an *isomorphism*.

2. We can generalize this: a homomorphism from $(X, *)$ to $(Y, *)$ *over* a set Γ is a function $\Gamma \times X \xrightarrow{f} Y$ such that $f(\rho, c * x) = c * f(\rho, x)$.

□

2.7.5 Scott Semantics

The reader may be familiar with Scott semantics for CBN, where types denote pointed cpos and terms denote continuous functions. We recall this semantics here, to make apparent its similarity to the printing semantics.

Definition 11 A cpo is *pointed* iff it has a least element, which we call $\bot$. We use *cppo* as an abbreviation for "pointed cpo". □

Each CBN type A denotes a cppo. Our intention is that a closed term M of type A will denote an element of X, and if it diverges then it will denote $\bot$. (A convergent term also may denote $\bot$.) Thus $\bot$ provides a way of "absorbing" the effect into A.

We recall some familiar ways of constructing cppos:

1. For a cpo X, its *lift* $X_\bot$ consists of X together with an additional element $\bot$, which is below all the elements of X.

2. For an $i \in I$-indexed family of cppos $(X_i, \leqslant, \bot)$ the *product* of this family $\prod_{i \in I}(X_i, \leqslant, \bot)$ is the set $\prod_{i \in I} X_i$, ordered pointwise. Its least element is given pointwise $\hat{\imath}\text{`}\bot = \bot$.

3. Given a cpo $(X, \leqslant)$ and a cppo $(Y, \leqslant, \bot)$ the *exponent* $(X, \leqslant) \to (Y, \leqslant, \bot)$ is the set of continuous functions from $(X, \leqslant)$ to $(Y, \leqslant)$, ordered pointwise. Its least element is given pointwise $\mathtt{x}\text{`}\bot = \bot$.

The semantics of types is given as follows:

- `bool` denotes the lift of the cpo $\{\mathsf{true}, \mathsf{false}\}$. This is because a closed term of type `bool` either diverges or terminates as `true` or `false`, and this behaviour is observable.

- If A denotes $(X, \leqslant, \bot)$ and A' denotes $(X', \leqslant, \bot)$ then $A + A'$ denotes the lift of the cpo $(X, \leqslant) + (X', \leqslant)$. ("CBN sum denotes lifted sum.") This is because every closed term of this type either diverges or terminates as `inl` M or `inr` M, and this behaviour is observable.

- If A denotes $(X, \leqslant, \bot)$ and B denotes $(Y, \leqslant, \bot)$ then $A \to B$ denotes $(X, \leqslant) \to (Y, \leqslant, \bot)$. This is because every closed term M of this type is equivalent to some $\lambda\mathtt{x}.N$, and if M diverges then it is equivalent to $\lambda\mathtt{x}.\mathtt{diverge}$.

Given a context $\mathtt{x}_0 : A_0, \ldots, \mathtt{x}_{n-1} : A_{n-1}$, an environment consists of a sequence of closed terms $M_0, \ldots, M_{n-1}$. Writing $(X_i, \leqslant, \bot)$ for $[\![A_i]\!]$, this environment denotes an element of the cpo $(X_0, \leqslant) \times \cdots \times (X_{n-1}, \leqslant)$. We say that the context denotes this cpo.

Given a term $\Gamma \vdash M : B$, from any environment we obtain (by substitution) a closed term of type B. So M will denote a continuous function from $[\![\Gamma]\!]$ to $[\![\underline{B}]\!]$. We omit the semantics of terms.

We emphasize that we have not used any notion of "strict function" between cppos. But, for use in Sect. 4.7 and Part III, we define this notion here.

Definition 12 1. A *strict continuous function* from a cppo X to a cppo Y is a continuous function $x \xrightarrow{f} y$ such that $f(\bot) = \bot$. If f is a bijection, then f^{-1} too is a strict continuous function, so we say that f is an *isomorphism*.

2. More generally, a strict continuous function from $(X, *)$ to $(Y, *)$ *over* a set Γ is a continuous function $\Gamma \times X \xrightarrow{f} Y$ such that $f(\rho, \bot) = \bot$.

□

2.7.6 Algebras and Plain Maps

We continue the categorical discussion of Sect. 2.6.4. Again, this section may be omitted by readers unfamiliar with category theory.

The two denotational models of CBN that we have seen are both cartesian closed categories:

- The printing semantics for CBN is the cartesian closed category in which an object is an $\mathcal{A}$-set $(X, *)$ and a morphism from $(X, *)$ to $(Y, *)$ is a function from X to Y.
- The cpo semantics for CBN is the cartesian closed category of cppos and continuous functions.

In fact, every model for CBN must be a cartesian closed category, as it must validate the β- and η-laws for functions.

These two cartesian closed categories are instances of a general construction. Suppose, as in Sect. 2.6.4, that we have a cartesian category $\mathcal{C}$ with a strong monad T. Then we form the category[3] of "algebras and plain maps", in which an object is a T-algebra (X, θ) and a morphism from (X, θ) to (Y, ϕ) is *any* $\mathcal{C}$-morphism from X to Y. Assuming sufficient exponents in $\mathcal{C}$ (we will make this precise in Def. 97), this category must be cartesian closed [Sim92].

To see that our two models are instances of this, notice that

- an algebra for the $\mathcal{A}^* \times -$ monad on **Set** is precisely an $\mathcal{A}$-set;
- an algebra for the lifting monad on **Cpo** is precisely a cppo.

We emphasize that in the category of algebras and plain maps, an object is an algebra (X, θ), not just a $\mathcal{C}$-object X on which there exists a strucure map θ. Although this latter definition would give a cartesian closed category equivalent to ours, it would not give a semantics of CBN. To see this, look at the semantics for $\mathtt{pm}$: it uses the specific structure θ. In summary,

> a CBV type denotes an object of $\mathcal{C}$; a CBN type denotes a T-algebra.

We are not claiming that *every* model of CBN arises from a monad in this way, just that the printing model and the Scott model do.

[3] This category can be seen as the co-Kleisli category for the comonad T on the Eilenberg-Moore category $\mathcal{C}^T$.

2.8 Comparing CBV and CBN

Because CBN satisfies the β-law and η-law for functions, it is sometimes suggested that CBN is "mathematically better behaved but practically less useful" than CBV. While the claim about the practical inferiority of CBN is valid, the claim about its mathematical superiority is not valid.

For although the CBN *function type* is mathematically superior to the CBV function type, the CBN *sum type* (and boolean type) is inferior to the CBV sum type. We saw that in the printing semantics the CBV sum simply denotes the sum of sets, while the CBN sum denotes a more complex construction on $\mathcal{A}$-sets. Similarly, in the Scott semantics, the sum of cpos is a much simpler construction than the lifted sum of cppos. In particular, the CBV sum is associative whereas the CBN sum is not.

This situation is seen not just in denotational semantics but also in equations. Consider the following equivalence—a special case of the η-law for `bool`. If M has a free identifier `z` : `bool` then

$$M = \texttt{if z then } M[\texttt{true/z}] \texttt{ else } M[\texttt{false/z}]$$

holds in CBV but not in CBN. It holds in CBV because `z` can be bound only to a value, `true` or `false`. It fails in CBN because `z` can be bound to a term such as `print "hello"; true`.

Thus, the perception of CBN's superiority is actually due to the fact that function types have been considered more important, and hence received more attention, than sum types or even ground types.

A consequence of this bias (towards function types and towards CBN) has been the promotion of cartesian closed categories as a significant structure, in the semantics of programming languages and even in the semantics of intuitionistic logic. The latter is especially inappropriate, because the type theory to which intuitionistic logic corresponds is effect-free rather than CBN or CBV, so its models must be *bicartesian* closed categories.

Chapter 3

Call-By-Push-Value: A Subsuming Paradigm

3.1 Introduction

3.1.1 Aims Of Chapter

In this chapter we present the CBPV language and its operational and denotational semantics for our example effects of printing and divergence, and we show how it contains both CBV and CBN. Operational semantics is presented in two forms: the familiar big-step form and the CK-machine of [FF86].

We will introduce the following vocabulary, all of which will be used in subsequent chapters:

- value, computation, terminal computation;
- value type, computation type;
- thunk, forcing a thunk;
- configuration, inside, outside;
- sequenced computation, producer, consumer.

We suggest the following slogans and mnemonics.

- A value is, a computation does.
- U types are thUnk types, F types are producer types.
- For cpos, U means nUthing, F means "liFt".

3.1.2 CBV And CBN Lead To CBPV

Because we have used printing rather than divergence as our leading example of an effect, our exploration of CBV and CBN leads us naturally to the types of CBPV. As we explained in Sect. 2.2, in the printing semantics, a CBV type denotes a set whereas a CBN type denotes an $\mathcal{A}$-set. Thus we will need two disjoint classes of type in the subsuming language: types denoting sets and types denoting $\mathcal{A}$-sets. These are precisely the value types and computation types (respectively) of CBPV. Furthermore, the subsuming language should provide type constructors corresponding to the various ways we have seen of constructing $\mathcal{A}$-sets, and CBPV indeed does this. For example,

- FA denotes the free $\mathcal{A}$-set on $[\![A]\!]$

- $U\underline{B}$ denotes the carrier of $[\![\underline{B}]\!]$.

Inventing the term calculus of the subsuming language is not as easy as the types, but we can obtain helpful guidelines by recalling equational facts about CBV and CBN and the informal reasons for them.

- In CBV, the η-law for sum types (and boolean type) holds because identifiers are bound to values.
- In CBN, the η-law for function types holds because a term of function type can be made to evaluate only by applying it.

We would like our subsuming language to have both of these properties, and indeed CBPV does.

3.2 Syntax

We recall that the types of CBPV are given by

$$\begin{array}{rcl} A & ::= & U\underline{B} \mid \sum_{i\in I} A_i \mid 1 \mid A \times A \\ \underline{B} & ::= & FA \mid \prod_{i\in I} \underline{B}_i \mid A \to \underline{B} \end{array}$$

where each set I of tags is finite. (We will also be concerned with *infinitely wide* CBPV in which I may be countably infinite—see Sect. 5.1 for more discussion.) We will often write $\hat{\imath}$ for a particular element of I. We usually omit terms, equations etc. for the type 1, because it is just the nullary analogue of $\times$.

Since identifiers can be bound only to values, they must have value type. So we have the following:

Definition 13 A *context* Γ is a finite sequence of identifiers with value types $\mathtt{x}_0 : A_0, \ldots, \mathtt{x}_{n-1} : A_{n-1}$. Sometimes we omit the identifiers and write Γ as a list of value types. □

The calculus has two kinds of judgement

$$\Gamma \vdash^{\mathsf{c}} M : \underline{B} \qquad\qquad \Gamma \vdash^{\mathsf{v}} V : A$$

for computations and values respectively. We emphasize that, in each case, only value types appear on the left of $\vdash$. The terms of basic (i.e. effect-free CBPV) are defined by Fig. 3.1. We will freely add whatever syntax is required for the various computational effects that we look at.

Definition 14

1. A *producer* is a computation of type FA.
2. A *ground type* is a type of the form $\sum_{i\in I} 1$ (such as $\mathtt{bool} = 1+1$).
3. A *ground value* is a value of ground type.
4. A *ground producer* is a computation of type FA, where A is a ground type.

We write $\mathtt{true}$ and $\mathtt{false}$ for the closed values of type $\mathtt{bool} = 1+1$ and $\mathtt{if}\ V\ \mathtt{then}\ M\ \mathtt{else}\ M'$ for the corresponding $\mathtt{pm}$ construct. □

Notice that CBPV has two forms of product. In the terminology of Sect. 2.3.2, the product of value types is a *pattern-match product* whereas the product of computation types is a *projection product*. The reason we do not have a construct πV where V has type $A \times A'$ is that the CBPV operational semantics (presented in Sect. 3.3) exploits the fact that values do not need to be evaluated. So we cannot allow a *complex value* such as $\pi(\mathtt{true},\mathtt{false})$, which needs to be evaluated to $\mathtt{true}$. Complex values are discussed fully in Chap. 4.

$$\frac{}{\Gamma, \mathtt{x} : A, \Gamma' \vdash^{\mathsf{v}} \mathtt{x} : A} \qquad \frac{\Gamma \vdash^{\mathsf{v}} V : A \quad \Gamma, \mathtt{x} : A \vdash^{\mathsf{c}} M : \underline{B}}{\Gamma \vdash^{\mathsf{c}} \mathtt{let}\ \mathtt{x}\ \mathtt{be}\ V.\ M : \underline{B}}$$

$$\frac{\Gamma \vdash^{\mathsf{v}} V : A}{\Gamma \vdash^{\mathsf{c}} \mathtt{produce}\ V : FA} \qquad \frac{\Gamma \vdash^{\mathsf{c}} M : FA \quad \Gamma, \mathtt{x} : A \vdash^{\mathsf{c}} N : \underline{B}}{\Gamma \vdash^{\mathsf{c}} M\ \mathtt{to}\ \mathtt{x}.\ N : \underline{B}}$$

$$\frac{\Gamma \vdash^{\mathsf{c}} M : \underline{B}}{\Gamma \vdash^{\mathsf{v}} \mathtt{thunk}\ M : U\underline{B}} \qquad \frac{\Gamma \vdash^{\mathsf{v}} V : U\underline{B}}{\Gamma \vdash^{\mathsf{c}} \mathtt{force}\ V : \underline{B}}$$

$$\frac{\Gamma \vdash^{\mathsf{v}} V : A_{\hat{\imath}}}{\Gamma \vdash^{\mathsf{v}} (\hat{\imath}, V) : \sum_{i \in I} A_i} \qquad \frac{\Gamma \vdash^{\mathsf{v}} V : \sum_{i \in I} A_i \quad \cdots \quad \Gamma, \mathtt{x} : A_i \vdash^{\mathsf{c}} M_i : \underline{B} \ \cdots}{\Gamma \vdash^{\mathsf{c}} \mathtt{pm}\ V\ \mathtt{as}\ \{\ldots, (i, \mathtt{x}).M_i, \ldots\} : \underline{B}}$$

$$\frac{\Gamma \vdash^{\mathsf{v}} V : A \quad \Gamma \vdash^{\mathsf{v}} V' : A'}{\Gamma \vdash^{\mathsf{v}} (V, V') : A \times A'} \qquad \frac{\Gamma \vdash^{\mathsf{v}} V : A \times A' \quad \Gamma, \mathtt{x} : A, \mathtt{y} : A' \vdash^{\mathsf{c}} M : \underline{B}}{\Gamma \vdash^{\mathsf{c}} \mathtt{pm}\ V\ \mathtt{as}\ (\mathtt{x}, \mathtt{y}).M : \underline{B}}$$

$$\frac{\cdots \ \Gamma \vdash^{\mathsf{c}} M_i : \underline{B}_i \ \cdots}{\Gamma \vdash^{\mathsf{c}} \lambda\{\ldots, i.M_i, \ldots\} : \prod_{i \in I} \underline{B}_i} \qquad \frac{\Gamma \vdash^{\mathsf{c}} M : \prod_{i \in I} \underline{B}_i}{\Gamma \vdash^{\mathsf{c}} \hat{\imath}\text{`}M : \underline{B}_{\hat{\imath}}}$$

$$\frac{\Gamma, \mathtt{x} : A \vdash^{\mathsf{c}} M : \underline{B}}{\Gamma \vdash^{\mathsf{c}} \lambda \mathtt{x}.M : A \to \underline{B}} \qquad \frac{\Gamma \vdash^{\mathsf{v}} V : A \quad \Gamma \vdash^{\mathsf{c}} M : A \to \underline{B}}{\Gamma \vdash^{\mathsf{c}} V\text{`}M : \underline{B}}$$

Figure 3.1: Terms of Basic Language

3.3 Operational Semantics Without Effects

Although at this stage we give the language without effects, we take care that the definitions, propositions and proofs can easily be adapted to various effects where possible.

3.3.1 Big-Step Semantics

The following closed computations are *terminal*:

$$T ::= \quad \mathtt{produce}\ V \ \mid \ \lambda\{\ldots, i.M_i, \ldots\} \ \mid \ \lambda \mathtt{x}.M$$

We write $\mathbb{C}_{\underline{B}}$ for the set of closed computations of type $\underline{B}$, $\mathbb{T}_{\underline{B}}$ for the set of terminal computations of type $\underline{B}$, and $\mathbb{V}_A$ for the set of closed values of type A.

The big-step semantics is given in Fig. 3.2.

Proposition 9 For every closed computation M, there is a unique terminal computation T such that $M \Downarrow T$. □

Proof (in the style of [Tai67]) We define, by mutual induction over types, three families of subsets:

$$\begin{array}{ll} \text{for each } A, & \mathsf{red}^{\mathsf{v}}_A \subseteq \mathbb{V}_A \\ \text{for each } \underline{B}, & \mathsf{red}^{\mathsf{t}}_{\underline{B}} \subseteq \mathbb{T}_{\underline{B}} \\ \text{for each } \underline{B}, & \mathsf{red}^{\mathsf{c}}_{\underline{B}} \subseteq \mathbb{C}_{\underline{B}} \end{array}$$

These definition of these subsets proceeds as follows:

$$\frac{}{\texttt{produce}\, V \Downarrow \texttt{produce}\, V} \qquad \frac{M[V/\mathtt{x}] \Downarrow T}{\texttt{let}\ \mathtt{x}\ \texttt{be}\ V.\ M \Downarrow T} \qquad \frac{M \Downarrow \texttt{produce}\, V \quad N[V/\mathtt{x}] \Downarrow T}{M\ \texttt{to}\ \mathtt{x}.\ N \Downarrow T}$$

$$\frac{M \Downarrow T}{\texttt{force}\,\texttt{thunk}\, M \Downarrow T} \qquad \frac{M_{\hat{\imath}}[V/\mathtt{x}] \Downarrow T}{\texttt{pm}\ (\hat{\imath}, V)\ \texttt{as}\ \{\ldots, (i,\mathtt{x}).M_i, \ldots\} \Downarrow T}$$

$$\frac{M[V/\mathtt{x}, V'/\mathtt{y}] \Downarrow T}{\texttt{pm}\ (V, V')\ \texttt{as}\ (\mathtt{x},\mathtt{y}).M \Downarrow T} \qquad \frac{}{\lambda\{\ldots, i.M_i, \ldots\} \Downarrow \lambda\{\ldots, i.M_i, \ldots\}} \qquad \frac{M \Downarrow \lambda\{\ldots, i.N_i, \ldots\} \quad N_{\hat{\imath}} \Downarrow T}{\hat{\imath}\text{`}M \Downarrow T}$$

$$\frac{}{\lambda\mathtt{x}.M \Downarrow \lambda\mathtt{x}.M} \qquad \frac{M \Downarrow \lambda\mathtt{x}.N \quad N[V/\mathtt{x}] \Downarrow T}{V\text{`}M \Downarrow T}$$

Note that each of these rules is of the form

$$\frac{M_0 \Downarrow T_0 \quad \cdots \quad M_{r-1} \Downarrow T_{r-1}}{M \Downarrow T} \tag{3.1}$$

for some $r \geqslant 0$.

Figure 3.2: Big-Step Semantics for CBPV

$$
\begin{array}{lll}
\mathtt{thunk}\ M \in \mathsf{red}^{\mathsf{v}}_{U\underline{B}} & \text{iff} & M \in \mathsf{red}^{\mathsf{c}}_{\underline{B}} \\
(\hat{\imath}, V) \in \mathsf{red}^{\mathsf{v}}_{\sum_{i\in I} A_i} & \text{iff} & V \in \mathsf{red}^{\mathsf{v}}_{A_{\hat{\imath}}} \\
(V, V') \in \mathsf{red}^{\mathsf{v}}_{A\times A'} & \text{iff} & V \in \mathsf{red}^{\mathsf{v}}_{A} \text{ and } V' \in \mathsf{red}^{\mathsf{v}}_{A'} \\
 & & \\
\mathtt{produce}\ V \in \mathsf{red}^{\mathsf{t}}_{FA} & \text{iff} & V \in \mathsf{red}^{\mathsf{v}}_{A} \\
\lambda\{\ldots, i.M_i, \ldots\} \in \mathsf{red}^{\mathsf{t}}_{\prod_{i\in I}\underline{B}_i} & \text{iff} & M_i \in \mathsf{red}^{\mathsf{c}}\underline{B}_i \text{ for all } i \in I \\
\lambda \mathtt{x}.M \in \mathsf{red}^{\mathsf{t}}_{A\to\underline{B}} & \text{iff} & M[V/\mathtt{x}] \in \mathsf{red}^{\mathsf{c}}_{\underline{B}} \text{ for all } V \in \mathsf{red}^{\mathsf{v}}_{A} \\
 & & \\
M \in \mathsf{red}^{\mathsf{c}}_{\underline{B}} & \text{iff} & M \Downarrow T \text{ for unique } T, \text{ and furthermore } T \in \mathsf{red}^{\mathsf{t}}_{\underline{B}}
\end{array}
$$

Notice that if $T \in \mathbb{T}_{\underline{B}}$, then $T \in \mathsf{red}^{\mathsf{c}}_{\underline{B}}$ iff $T \in \mathsf{red}^{\mathsf{t}}_{\underline{B}}$.

Finally we show that for any computation $A_0, \ldots, A_{n-1} \vdash^{\mathsf{c}} M : \underline{B}$, if $W_i \in \mathsf{red}^{\mathsf{v}}_{A_i}$ for $i = 0, \ldots, n-1$ then $M[\overrightarrow{W_i/\mathtt{x}_i}] \in \mathsf{red}^{\mathsf{c}}_{\underline{B}}$; and similarly for any value $A_0, \ldots, A_{n-1} \vdash^{\mathsf{v}} V : A$. This is shown by mutual induction on M and V, and gives the required result. □

3.3.2 CK-Machine

The CK-machine is a general form of operational semantics that can be used for CBV and CBN as well as for CBPV. It was introduced in [FF86] as a simplification of Landin's SECD machine[Lan64], and there are many similar machines [Bie98, Kri85, SR98]. At any point in time, the machine has configuration M, K when M (the *inside*) is the term we are evaluating and K (the *outside*) is a stack[1]. There is no need for an environment or for closures, because substitution is used.

The machine is summarized in Fig. 3.3. To understand the CK-machine, just think about how we might implement the big-step rules using a stack.

Suppose for example that we are evaluating M `to` $\mathtt{x}.\ N$. The big-step semantics tells us that we must first evaluate M. So we put the context $[]$ `to` $\mathtt{x}.\ N$ onto the stack, because at present we do not need it. Later, having evaluated M to `produce` V, we can remove $[]$ `to` $\mathtt{x}.\ N$ from the stack and proceed to evaluate $N[V/\mathtt{x}]$, as the big-step semantics suggests.

As another example, suppose we are evaluating V`‘`M. The big-step semantics tells us that we must first evaluate M. So we put the operand[2] V onto the stack, because at present we do not need it. Later, having evaluated M to $\lambda\mathtt{x}.N$, we can remove the operand V from the stack and proceed to evaluate $N[V/\mathtt{x}]$, as the big-step semantics suggests.

To evaluate a closed computation M, we start with the configuration M,`nil` and follow the transitions in Fig. 3.3 until we reach a configuration T,`nil` for a terminal computation T. We shall see in Sect. 3.3.4 that this will happen precisely when $M \Downarrow T$. The outside `nil` is often referred to as the *top-level outside*.

Notice that

- the behaviour of V`‘`M is to push V and then evaluate M
- the behaviour of $\lambda\mathtt{x}.M$ is to pop V and then evaluate $M[V/\mathtt{x}]$

From the big-step rules we have thus recovered the push/pop reading described in Sect. 1.5.1.

[1] Readers familiar with similar concepts in a CBV setting may find the usage of "outside" confusing, so we explain as follows. What we call an *outside* is roughly what Felleisen called an *evaluation context*. We use the terminology *current outside* rather than *current continuation* because in CBPV (unlike CBV) not all outsides are continuations: for example, the outside $V :: K$ is a pair, not a continuation. We look at continuation semantics in detail in Sect. 6.4.4.

[2] If we wanted our usage to be strictly consistent, we would put the context V`‘`$[]$ rather than just V onto the stack, but this is unnecessarily complicated.

Initial Configuration

	M	$\mathtt{nil}$

Transitions

	$\mathtt{let}\ \mathtt{x}\ \mathtt{be}\ V.\ M$	K
$\rightsquigarrow$	$M[V/\mathtt{x}]$	K
	$M\ \mathtt{to}\ \mathtt{x}.\ N$	K
$\rightsquigarrow$	M	$[]\ \mathtt{to}\ \mathtt{x}.\ N :: K$
	$\mathtt{produce}\ V$	$[]\ \mathtt{to}\ \mathtt{x}.\ N :: K$
$\rightsquigarrow$	$N[V/\mathtt{x}]$	K
	$\mathtt{force}\ \mathtt{thunk}\ M$	K
$\rightsquigarrow$	M	K
	$\mathtt{pm}\ (\hat{\imath}, V)\ \mathtt{as}\ \{\ldots, (i,\mathtt{x}).M_i, \ldots\}$	K
$\rightsquigarrow$	$M_{\hat{\imath}}[V/\mathtt{x}]$	K
	$\mathtt{pm}\ (V, V')\ \mathtt{as}\ (\mathtt{x},\mathtt{y}).M$	K
$\rightsquigarrow$	$M[V/\mathtt{x}, V'/\mathtt{y}]$	K
	$\hat{\imath}\text{`}M$	K
$\rightsquigarrow$	M	$\hat{\imath} :: K$
	$\lambda\{\ldots, i.M_i, \ldots\}$	$\hat{\imath} :: K$
$\rightsquigarrow$	$M_{\hat{\imath}}$	K
	$V\text{`}M$	K
$\rightsquigarrow$	M	$V :: K$
	$\lambda\mathtt{x}.M$	$V :: K$
$\rightsquigarrow$	$M[V/\mathtt{x}]$	K

Terminal Configurations

$\mathtt{produce}\ V$	$\mathtt{nil}$
$\lambda\{\ldots, i.M_i, \ldots\}$	$\mathtt{nil}$
$\lambda\mathtt{x}.M$	$\mathtt{nil}$

Figure 3.3: CK-Machine For CBPV

3.3.3 Typing the CK-Machine

If we execute the CK-machine from an initial configuration, then each configuration M, K that we reach has 2 types associated with it:

- the type $\underline{B}$ of M;
- the type $\underline{C}$ of the initial computation.

We say that M, K is a *closed*[3] *configuration of type* $\underline{C}$ and that K is an *closed outside from* $\underline{B}$ *to* $\underline{C}$. It is clear that we can form a category whose objects are computation types and whose morphisms are outsides; composition is given by concatenation.

We write $\vdash^{\mathsf{k}}_{\underline{C}} K : \underline{B}$ to mean that K is a closed outside from $\underline{B}$ to $\underline{C}$; in other words, it accompanies an inside of type $\underline{B}$ in the course of evaluating a computation of type $\underline{C}$. More generally, we can include free identifiers Γ and we then write $\Gamma \vdash^{\mathsf{k}}_{\underline{C}} K : \underline{B}$. (This will be useful in Sect. 3.3.5.) The typing rules for this judgement are given in Fig. 3.4. We can then say

Definition 15 A Γ-*configuration of type* $\underline{C}$ consists of

- a computation type $\underline{B}$;
- a pair M, K where $\Gamma \vdash^{\mathsf{c}} M : \underline{B}$ and $\Gamma \vdash^{\mathsf{k}}_{\underline{C}} K : \underline{B}$.

□

$$\frac{}{\Gamma \vdash^{\mathsf{k}}_{\underline{C}} \mathtt{nil} : \underline{C}} \qquad \frac{\Gamma, \mathtt{x} : A \vdash^{\mathsf{c}} M : \underline{B} \quad \Gamma \vdash^{\mathsf{k}}_{\underline{C}} K : \underline{B}}{\Gamma \vdash^{\mathsf{k}}_{\underline{C}} [] \,\mathtt{to}\, \mathtt{x}.\, M :: K : FA}$$

$$\frac{\Gamma \vdash^{\mathsf{k}}_{\underline{C}} K : \underline{B}_{\hat{\imath}}}{\Gamma \vdash^{\mathsf{k}}_{\underline{C}} \hat{\imath} :: K : \prod_{i \in I} \underline{B}_i} \qquad \frac{\Gamma \vdash^{\mathsf{v}} V : A \quad \Gamma \vdash^{\mathsf{k}}_{\underline{C}} K : \underline{B}}{\Gamma \vdash^{\mathsf{k}}_{\underline{C}} V :: K : A \to \underline{B}}$$

Figure 3.4: Typing Outsides

Proposition 10 (deterministic subject reduction) For every closed configuration M, K of type $\underline{C}$, precisely one of the following holds.

1. M, K is not terminal, and $M, K \rightsquigarrow N, L$ for unique N, L. N, L is a closed configuration of type $\underline{C}$.
2. M, K is terminal, and there does not exist N, L such that $M, K \rightsquigarrow N, L$.

□

To complete the CK-machine semantics, we write $\rightsquigarrow^*$ for the transitive closure of $\rightsquigarrow$. More technically:

Definition 16 We define inductively the relation $\rightsquigarrow^*$ on closed configurations:

$$\frac{}{M, K \rightsquigarrow^* M, K} \qquad \frac{M', K' \rightsquigarrow^* N, L}{M, K \rightsquigarrow^* N, L} \, (M, K \rightsquigarrow M', K')$$

□

[3] Since we have defined operational semantics for closed computations only, all configurations that arise during execution are closed. However, this will not be the case in Sect. 3.3.5, where we present operational semantics for non-closed computations.

By analogy with Prop. 9, we can now formulate the following.

Proposition 11 For every closed configuration M, K there is a unique terminal T such that $M, K \rightsquigarrow^* T, \mathtt{nil}$, and there is no infinite sequence of transitions from M, K. □

We defer the proof of this to Sect. 8.6.

Definition 17 (complementary to Def. 14(1)) A *consumer* is an outside from a type FA to any type. □

Such an outside "consumes" the value that a producer produces. For example, $[]\ \mathtt{to}\ \mathtt{x}.\ N :: K$ is a consumer.

3.3.4 Agreement Of Big-Step and CK-Machine Semantics

The sole aim of this section is to prove

Proposition 12 For closed computations M, T of type $\underline{B}$, the following are equivalent:

1. $M \Downarrow T$;
2. $M, K \rightsquigarrow^* T, K$ for all outsides K such that $\vdash^{\mathsf{k}}_{\underline{C}} K : \underline{B}$ for some $\underline{C}$;
3. $M, \mathtt{nil} \rightsquigarrow^* T, \mathtt{nil}$.

□

(1) ⇒ (2) is a straightforward induction. (2) ⇒ (3) is trivial. To prove (3)⇒(1) we first define a mapping from configurations to computations.

Definition 18 Given a closed configuration M, K of type $\underline{C}$ we define $\vdash^{\mathsf{c}} M \bullet K : \underline{C}$ by induction on K:

$$\begin{array}{rcl} M \bullet \mathtt{nil} & = & M \\ M \bullet ([]\ \mathtt{to}\ \mathtt{x}.\ N :: K) & = & (M\ \mathtt{to}\ \mathtt{x}.\ N) \bullet K \\ M \bullet (\hat{\imath} :: K) & = & (\hat{\imath}\text{`}M) \bullet K \\ M \bullet (V :: K) & = & (V\text{`}M) \bullet K \end{array}$$

□

Lemma 13 For all M and N of type $\underline{B}$, if, for all T, $M \Downarrow T$ implies $N \Downarrow T$, then, for every K such that $\vdash^{\mathsf{k}}_{\underline{C}} K : \underline{B}$ for some $\underline{C}$, $M \bullet K \Downarrow T$ implies $N \bullet K \Downarrow T$. □

Proof Induct on K. □

Lemma 14 If $M, K \rightsquigarrow^* T, \mathtt{nil}$ then $M \bullet K \Downarrow T$. □

Proof We induct on the antecedent. As an example clause, suppose that the antecedent is given by

$$\mathtt{produce}\ V, []\ \mathtt{to}\ \mathtt{x}.\ N :: K \rightsquigarrow N[V/\mathtt{x}], K \rightsquigarrow^* T, \mathtt{nil}$$

We know by the inductive hypothesis that $(N[V/\mathtt{x}]) \bullet K \Downarrow T$, so by Lemma 13 we know that $(\mathtt{produce}\ V\ \mathtt{to}\ \mathtt{x}.\ N) \bullet K \Downarrow T$. □

Prop. 12((3)⇒(1)) is an immediate consequence of Lemma 14.

3.3.5 Operational Semantics For Non-Closed Computations

We have presented big-step and CK-machine semantics for closed computations only. We now show how to extend them to non-closed computations on a fixed context Γ. This extension will be advantageous in Chap. 6, especially when we look at control effects, because we will then want to treat the outside `nil` as a free identifier. The only difficulty is when a computation tries to force or pattern-match a free identifier: we must then terminate execution.

Big-Step Semantics

We want to define a relation $M \Downarrow T$ where $\Gamma \vdash^{\mathsf{c}} M : \underline{B}$ and $\Gamma \vdash^{\mathsf{c}} T : \underline{B}$. We require a bigger class of terminal computations than in Sect. 3.3.1:

$$\begin{array}{rl} T ::= & \mathtt{produce}\, V \mid \lambda\{\ldots, i.M_i, \ldots\} \mid \lambda \mathtt{x}.M \\ \mid & \mathtt{force}\, \mathtt{z} \mid \mathtt{pm}\, \mathtt{z}\, \mathtt{as}\, \{\ldots, i.M_i, \ldots\} \mid \mathtt{pm}\, \mathtt{z}\, \mathtt{as}\, (\mathtt{x},\mathtt{y}).M \end{array}$$

We must add to Fig. 3.2 rules for the additional terminal computations:

$$\frac{}{\mathtt{force}\, \mathtt{z} \Downarrow \mathtt{force}\, \mathtt{z}}$$

$$\frac{}{\mathtt{pm}\, \mathtt{z}\, \mathtt{as}\, \{\ldots, i.M_i, \ldots\} \Downarrow \mathtt{pm}\, \mathtt{z}\, \mathtt{as}\, \{\ldots, i.M_i, \ldots\}}$$

$$\frac{}{\mathtt{pm}\, \mathtt{z}\, \mathtt{as}\, (\mathtt{x},\mathtt{y}).M}$$

CK-Machine

The only change required to the CK-machine described in Sect. 3.3.2 is that we use Γ-configurations rather than closed configurations, and so we require a bigger class of terminal configurations:

$\mathtt{produce}\, V$	`nil`
$\lambda\{\ldots, i.M_i, \ldots\}$	`nil`
$\lambda \mathtt{x}.M$	`nil`
$\mathtt{force}\, \mathtt{z}$	K
$\mathtt{pm}\, \mathtt{z}\, \mathtt{as}\, \{\ldots, i.M_i, \ldots\}$	K
$\mathtt{pm}\, \mathtt{z}\, \mathtt{as}\, (\mathtt{x},\mathtt{y}).M$	K

3.4 Operational Semantics for `print`

We now add printing: more precisely, we add to the syntax of CBPV the typing rule

$$\frac{\Gamma \vdash^{\mathsf{c}} M : \underline{B}}{\Gamma \vdash^{\mathsf{c}} \mathtt{print}\, c;\ M : \underline{B}}$$

and we must adapt the operational semantics accordingly.

3.4.1 Big-Step Semantics for `print`

Since we have added only one term constructor to the basic language, we might expect that we need only add one rule to Fig. 3.2. However, this is clearly not possible: whereas for the basic language the big-step relation has the form $M \Downarrow T$, it now has the form $M \Downarrow m, T$. It seems therefore that we must present the big-step semantics from scratch.

Fortunately this is not the case. We simply replace each rule in Fig. 3.2 of the form (3.1) by

$$\frac{M_0 \Downarrow m_0, T_0 \quad \cdots \quad M_{r-1} \Downarrow m_{r-1}, T_{r-1}}{M \Downarrow m_0 * \ldots * m_{r-1}, T}$$

Then we add the big-step rule

$$\frac{M \Downarrow m, T}{\texttt{print}\ c;\ M \Downarrow c * m, T}$$

Proposition 15 For every computation M, there exists unique m, T such that $M \Downarrow m, T$. □

The proof is easily adapted from the proof of Prop. 9.

3.4.2 CK-Machine For `print`

In Fig. 3.3 a transition has the form

$$M \qquad K \qquad \rightsquigarrow \qquad M' \qquad K' \tag{3.2}$$

When we add `print` to the language, we want a transition to have the form

$$M \qquad K \qquad \rightsquigarrow \qquad m \qquad M' \qquad K'$$

for some $m \in \mathcal{A}^*$. We therefore replace each transition (3.2) in Fig. 3.3 by

$$M \qquad K \qquad \rightsquigarrow \qquad 1 \qquad M' \qquad K'$$

and we add the transition

$$\texttt{print}\ c;\ M \qquad K \qquad \rightsquigarrow \qquad c \qquad M \qquad K$$

We replace Def. 16 by the following.

Definition 19 We define the relation $\rightsquigarrow^*$, whose form is $M, K \rightsquigarrow^* m, M', K'$ inductively:

$$\frac{}{M, K \rightsquigarrow^* 1, M, K} \qquad \frac{M', K' \rightsquigarrow^* n, N, L}{M, K \rightsquigarrow^* m * n, N, L}\ (M, K \rightsquigarrow m, M', K')$$

□

We can state and prove analogues of all the results in Sect. 3.3.3–3.3.2. In particular we have

Proposition 16 $M \Downarrow m, T$ iff $M, \texttt{nil} \rightsquigarrow^* m, T, \texttt{nil}$. □

3.5 Observational Equivalence

As with CBV and CBN, we want to define a notion of observational equivalence.

Definition 20

1. A *ground context* is a closed ground producer with zero or more occurrences of a hole which might be a computation or a value.
2. Given two computations $\Gamma \vdash^{\mathsf{c}} M, N : \underline{B}$, we say that $M \simeq N$ when for every ground context $\mathcal{C}[]$ we have that $\mathcal{C}[M] \Downarrow T$ iff $\mathcal{C}[N] \Downarrow T$ (for every T). We similarly define $\simeq$ for values.

□

We modify this to suit the effect being considered. For printing (without divergence) we say that $M \simeq N$ when for every ground context $\mathcal{C}[]$, we have that $\mathcal{C}[M] \Downarrow m, T$ iff $\mathcal{C}[N] \Downarrow m, T$ (for every T).

3.6 Denotational Semantics

In the printing semantics, a value type (and hence a context) denotes a set, and a computation type denotes an $\mathcal{A}$-set. The semantics of types is given by

$$\begin{array}{rcl} [\![U\underline{B}]\!] & = & \text{the carrier of } [\![\underline{B}]\!] \\ [\![\Sigma_{i\in I} A_i]\!] & = & \Sigma_{i\in I}[\![A_i]\!] \\ [\![A \times A']\!] & = & [\![A]\!] \times [\![A']\!] \end{array}$$

$$\begin{array}{rcl} [\![FA]\!] & = & \text{the free } \mathcal{A}\text{-set on } [\![A]\!] \\ [\![\Pi_{i\in I} \underline{B}_i]\!] & = & \Pi_{i\in I}[\![\underline{B}_i]\!] \\ [\![A \to \underline{B}]\!] & = & [\![A]\!] \to [\![\underline{B}]\!] \end{array}$$

Similarly we define the denotation of a context Γ: if Γ is the sequence $A_0, \ldots, A_{n-1}$, then we set $[\![\Gamma]\!]$ to be the set $[\![A_0]\!] \times \cdots \times [\![A_{n-1}]\!]$.

Next we define the semantics of terms. A value $\Gamma \vdash^{\mathsf{v}} V : A$ denotes a function $[\![V]\!]$ from the set $[\![\Gamma]\!]$ to the set $[\![A]\!]$, and a computation $\Gamma \vdash^{\mathsf{c}} M : \underline{B}$ denotes a function from the set $[\![\Gamma]\!]$ to the carrier of the $\mathcal{A}$-set $[\![\underline{B}]\!]$. Here are some example clauses:

$$\begin{array}{rcl} [\![\mathtt{produce}\, V]\!]\rho & = & (1, [\![V]\!]\rho) \\ [\![M \ \mathtt{to}\ \mathtt{x}.\ N]\!]\rho & = & m * [\![N]\!](\rho, \mathtt{x} \mapsto a) \text{ where } [\![M]\!]\rho = (m, a) \\ [\![\mathtt{thunk}\, M]\!]\rho & = & [\![M]\!]\rho \\ [\![\mathtt{force}\, V]\!]\rho & = & [\![V]\!]\rho \\ [\![\lambda \mathtt{x}.M]\!]\rho & = & \lambda x.[\![M]\!](\rho, \mathtt{x} \mapsto x) \\ [\![V \text{`} M]\!]\rho & = & ([\![V]\!]\rho) \text{`} ([\![M]\!]\rho) \\ [\![\mathtt{print}\, m;\ M]\!]\rho & = & m * ([\![M]\!]\rho) \end{array}$$

In the Scott semantics, a value type denotes a cpo and a computation type denotes a pointed cpo. Some example clauses:

$$\begin{array}{rcl} [\![U\underline{B}]\!] & = & [\![\underline{B}]\!] \\ [\![FA]\!] & = & \text{lift of } [\![A]\!] \\ [\![A \to \underline{B}]\!] & = & [\![A]\!] \to [\![\underline{B}]\!] \end{array}$$

Like a value type, a context denotes a cpo, given by $\times$. Then a value $\Gamma \vdash^{\mathsf{v}} V : A$ denotes a continuous function $[\![V]\!]$ from the cpo $[\![\Gamma]\!]$ to the cpo $[\![A]\!]$, and a computation $\Gamma \vdash^{\mathsf{c}} M : \underline{B}$ denotes a continuous function from the cpo $[\![\Gamma]\!]$ to the pointed cpo $[\![\underline{B}]\!]$.

Notice that, in both printing and Scott semantics, the constructs `thunk` and `force` are *invisible* in the sense that

$$\begin{array}{rcl} [\![\mathtt{thunk}\, M]\!] & = & [\![M]\!] \\ [\![\mathtt{force}\, V]\!] & = & [\![V]\!] \end{array}$$

In the Scott semantics, U too is invisible, because $[\![U\underline{B}]\!] = [\![\underline{B}]\!]$. By contrast, in many of the semantics in Chap. 6 `thunk` , `force` and U are all visible.

Proposition 17 (Soundness of Denotational Semantics) For any closed computation M, if $M \Downarrow m, T$ then $[\![M]\!] = m * [\![T]\!]$. □

Corollary 18 (by Prop. 9) For any closed ground producer M, we have $M \Downarrow m, \mathtt{produce}\, n$ iff $[\![M]\!] = (m, n)$. Hence terms with the same denotation are observationally equivalent. □

If we are dealing with non-closed computations as in Sect. 3.3.5, then Prop. 17 extends to

Proposition 19 For any computation $\Gamma \vdash^{\mathsf{c}} M : \underline{B}$ and any environment $\rho \in [\![\Gamma]\!]$ if $M \Downarrow m, T$ then $[\![M]\!]\rho = m * ([\![T]\!]\rho)$. □

3.7 Subsuming CBV and CBN

We give translations from the small fragments of CBV and CBN described in Chap. 2. The full translations and their technical properties (adequacy, full abstraction etc.) are given in Appendix A.

If the reader bears in mind the denotational semantics of CBV and CBN, the translations into CBPV are obvious.

3.7.1 From CBV to CBPV

The big difference between CBV and CBPV is that in CBV $\lambda \mathtt{x}.M$ is a value whereas in CBPV $\lambda \mathtt{x}.M$ is a computation. Thus $\lambda \mathtt{x}$ in CBV decomposes into $\mathtt{thunk}\ \lambda \mathtt{x}$ in CBPV.

More generally, a CBV function from A to B is, from a CBPV perspective, a thunk of a computation that pops a value of type A and produces a value of type B. So we have a decomposition of $\to_{\mathbf{CBV}}$ into CBPV given by

$$A \to_{\mathbf{CBV}} B = U(A \to FB) \tag{3.3}$$

It is important to see that this decomposition respects both of our denotational semantics i.e. the two sides of (3.3) have the same denotation. This is essential if the translation is to be regarded as subsumptive.

- In the printing semantics, $A \to FB$ denotes an $\mathcal{A}$-set whose carrier is the set of functions from $[\![A]\!]$ to $\mathcal{A}^* \times [\![B]\!]$. So the thunk type $U(A \to FB)$ denotes this set, as does $A \to_{\mathbf{CBV}} B$.

- In the Scott semantics $A \to FB$ denotes the cppo of continuous functions from $[\![A]\!]$ to $[\![B]\!]_\perp$. So the thunk type $U(A \to FB)$ denotes this cpo, as does $A \to_{\mathbf{CBV}} B$.

The translation from CBV to CBPV is presented in Fig. 3.5. Just as a CBV value V has two denotations $[\![V]\!]^{\mathsf{val}}$ and $[\![V]\!]^{\mathsf{prod}}$, so it has two translations V^{val} and V^{prod} related by

$$V^{\mathsf{prod}} = \mathtt{produce}\ V^{\mathsf{val}}$$

in the CBPV equational theory of Chap. 4.

As we explained for $\to_{\mathbf{CBV}}$, the translation preserves denotational semantics. Thus, for both printing and Scott semantics, we have the following:

Proposition 20

1. For any CBV type A, $[\![A]\!]_{\mathbf{CBV}} = [\![A^{\mathsf{v}}]\!]_{\mathbf{CBPV}}$.
2. For any CBV value $\Gamma \vdash V : A$, $[\![V]\!]^{\mathsf{val}}_{\mathbf{CBV}} = [\![V^{\mathsf{val}}]\!]_{\mathbf{CBPV}}$.
3. For any CBV producer $\Gamma \vdash M : A$, $[\![M]\!]^{\mathsf{prod}}_{\mathbf{CBV}} = [\![M^{\mathsf{prod}}]\!]_{\mathbf{CBPV}}$.

□

That the translation respects operational semantics (to a certain degree of intensionality) is proved in Appendix A.

C	C^{v} (a value type)
$\mathtt{bool}$	$\mathtt{bool}$ i.e. $1+1$
$A+B$	$A^{\mathsf{v}}+B^{\mathsf{v}}$
$A\to B$	$U(A^{\mathsf{v}}\to FB^{\mathsf{v}})$

$A_0,\dots,A_{n-1}\vdash V:C$	$A_0^{\mathsf{v}},\dots,A_{n-1}^{\mathsf{v}}\vdash^{\mathsf{v}} V^{\mathsf{val}}:C^{\mathsf{v}}$
$\mathtt{x}$	$\mathtt{x}$
$\mathtt{true}$	$\mathtt{true}$
$\mathtt{false}$	$\mathtt{false}$
$\mathtt{inl}\ V$	$\mathtt{inl}\ V^{\mathsf{val}}$
$\mathtt{inr}\ V$	$\mathtt{inr}\ V^{\mathsf{val}}$
$\lambda\mathtt{x}.M$	$\mathtt{thunk}\ \lambda\mathtt{x}.M^{\mathsf{prod}}$

$A_0,\dots,A_{n-1}\vdash M:C$	$A_0^{\mathsf{v}},\dots,A_{n-1}^{\mathsf{v}}\vdash^{\mathsf{c}} M^{\mathsf{prod}}:FC^{\mathsf{v}}$
$\mathtt{x}$	$\mathtt{produce}\ \mathtt{x}$
$\mathtt{let}\ \mathtt{x}\ \mathtt{be}\ M.\ N$	$M^{\mathsf{prod}}\ \mathtt{to}\ \mathtt{x}.\ N^{\mathsf{prod}}$
$\mathtt{true}$	$\mathtt{produce}\ \mathtt{true}$
$\mathtt{false}$	$\mathtt{produce}\ \mathtt{false}$
$\mathtt{if}\ M\ \mathtt{then}\ N\ \mathtt{else}\ N'$	$M^{\mathsf{prod}}\ \mathtt{to}\ \mathtt{z}.\ \mathtt{if}\ \mathtt{z}\ \mathtt{then}\ N^{\mathsf{prod}}\ \mathtt{else}\ N'^{\mathsf{prod}}$
$\mathtt{inl}\ M$	$M^{\mathsf{prod}}\ \mathtt{to}\ \mathtt{z}.\ \mathtt{produce}\ \mathtt{inl}\ \mathtt{z}$
$\mathtt{inr}\ M$	$M^{\mathsf{prod}}\ \mathtt{to}\ \mathtt{z}.\ \mathtt{produce}\ \mathtt{inr}\ \mathtt{z}$
$\mathtt{pm}\ M\ \mathtt{as}\ \{\mathtt{inl}\ \mathtt{x}.N,\mathtt{inr}\ \mathtt{x}.N'\}$	$M^{\mathsf{prod}}\ \mathtt{to}\ \mathtt{z}.\ \mathtt{pm}\ \mathtt{z}\ \mathtt{as}\ \{\mathtt{inl}\ \mathtt{x}.N^{\mathsf{prod}},\mathtt{inr}\ \mathtt{x}.N'^{\mathsf{prod}}\}$
$\lambda\mathtt{x}.M$	$\mathtt{produce}\ \mathtt{thunk}\ \lambda\mathtt{x}.M^{\mathsf{prod}}$
$M\text{`}N$	$M^{\mathsf{prod}}\ \mathtt{to}\ \mathtt{x}.\ N^{\mathsf{prod}}\ \mathtt{to}\ \mathtt{f}.\ \mathtt{x}\text{`}(\mathtt{force}\ \mathtt{f})$
$\mathtt{print}\ c;\ M$	$\mathtt{print}\ c;\ M^{\mathsf{prod}}$

Figure 3.5: Translation of CBV types, values and producers

3.7.2 From CBN to CBPV

The translation from CBN to CBPV is motivated as follows.

- Identifiers in CBN are bound to unevaluated terms, so we regard them (from a CBPV perspective) as bound to thunks.
- Similarly, tuple-components and operands in CBN are unevaluated terms, so we regard them as thunks.
- Consequently, identifiers, tuple-components and operands all have type of the form $U\underline{B}$—a thunk type.

We thus have a decomposition of $\rightarrow_{\mathbf{CBN}}$ into CBPV.

$$\underline{A} \rightarrow_{\mathbf{CBN}} \underline{B} = (U\underline{A}) \rightarrow \underline{B} \tag{3.4}$$

It is important to see that this decomposition respects both of our denotational semantics i.e. both sides of (3.4) have the same denotation. This is essential if the translation is to be regarded as subsumptive.

- In the printing semantics, if $\underline{A}$ denotes the $\mathcal{A}$-set $(X,*)$ and $\underline{B}$ denotes the $\mathcal{A}$-set $(Y,*)$ then $U\underline{A}$ denotes the set X and $(U\underline{A}) \rightarrow \underline{B}$ denotes $X \rightarrow (Y,*)$, as does $\underline{A} \rightarrow_{\mathbf{CBN}} \underline{B}$.
- In the Scott semantics, $(U\underline{A}) \rightarrow \underline{B}$ denotes the cppo of continuous functions from $[\![\underline{A}]\!]$ to $[\![\underline{B}]\!]$, as does $\underline{A} \rightarrow_{\mathbf{CBN}} \underline{B}$.

Similarly we have decompositions

$$\begin{array}{rcl} \mathtt{bool}_{\mathbf{CBN}} & = & F\mathtt{bool} \\ \underline{A} +_{\mathbf{CBN}} \underline{B} & = & F((U\underline{A}) + (U\underline{B})) \end{array}$$

It is easily seen that these also respect denotational semantics, e.g. the Scott semantics for $F((U\underline{A}) + (U\underline{B}))$ is the lifted sum of $[\![\underline{A}]\!]$ and $[\![\underline{B}]\!]$.

As we explained for $\rightarrow_{\mathbf{CBN}}$, the translation preserves denotational semantics. Thus, for both printing and Scott semantics, we have the following:

Proposition 21 1. For any CBN type A, $[\![A]\!]_{\mathbf{CBN}} = [\![A^{\mathsf{n}}]\!]_{\mathbf{CBPV}}$.

2. For any CBN term $\Gamma \vdash M : A$, $[\![M]\!]_{\mathbf{CBN}} = [\![M^{\mathsf{n}}]\!]_{\mathbf{CBPV}}$.

□

That the translation respects operational semantics (to a certain degree of intensionality) is proved in Appendix A.

3.8 CBPV As A Metalanguage

We can use CBPV not just as an object language but as a metalanguage. When we do this, we have to specify which CBPV model the metalanguage is referring to. For example, when talking about the printing model, we use FA to refer to the free $\mathcal{A}$-set on the set A, and $U\underline{B}$ to refer to the carrier of the $\mathcal{A}$-set $\underline{B}$.

Another example is the Scott model.

- If A is a cpo we write FA for its lift, a cppo.
- If $a \in A$ we write $\mathsf{produce}\ a$ for the corresponding element of FA.

C	$C^{\mathtt{n}}$ (a computation type)
$\mathtt{bool}$	$F\mathtt{bool}$ i.e. $F(1+1)$
$A+B$	$F(UA^{\mathtt{n}}+UB^{\mathtt{n}})$
$A\to B$	$(UA^{\mathtt{n}})\to B^{\mathtt{n}}$

$A_0,\dots,A_{n-1}\vdash M:C$	$UA_0^{\mathtt{n}},\dots,UA_{n-1}^{\mathtt{n}}\vdash^{\mathtt{c}} M^{\mathtt{n}}:C^{\mathtt{n}}$
$\mathtt{x}$	$\mathtt{force\ x}$
$\mathtt{let\ x\ be}\ M.\ N$	$\mathtt{let\ x\ be\ thunk}\ M^{\mathtt{n}}.\ M^{\mathtt{n}}$
$\mathtt{true}$	$\mathtt{produce\ true}$
$\mathtt{false}$	$\mathtt{produce\ false}$
$\mathtt{if}\ M\ \mathtt{then}\ N\ \mathtt{else}\ N'$	$M^{\mathtt{n}}\ \mathtt{to\ z.\ if\ z\ then}\ N^{\mathtt{n}}\ \mathtt{else}\ N'^{\mathtt{n}}$
$\mathtt{inl}\ M$	$\mathtt{produce\ inl\ thunk}\ M^{\mathtt{n}}$
$\mathtt{pm}\ M\ \mathtt{as}\ \{\mathtt{inl\ x}.N,\mathtt{inr\ x}.N'\}$	$M^{\mathtt{n}}\ \mathtt{to\ z.\ pm\ z\ as}\ \{\mathtt{inl\ x}.N^{\mathtt{n}},\mathtt{inr\ x}.N'^{\mathtt{n}}\}$
$\lambda\mathtt{x}.M$	$\lambda\mathtt{x}.M^{\mathtt{n}}$
$N\text{`}M$	$(\mathtt{thunk}\ N^{\mathtt{n}})\text{`}M^{\mathtt{n}}$
$\mathtt{print}\ c;\ M$	$\mathtt{print}\ c;\ M^{\mathtt{n}}$

Figure 3.6: Translation of CBN types and terms

- If $b\in FA$ and f is a function from A to a cppo $\underline{B}$, we write $b\ \mathsf{to}\ x.\ f(x)$ to mean $\bot$ if $b=\bot$ and $f(a)$ if $b=\mathsf{produce}\ a$.
- If $\underline{B}$ is a cppo, we write $U\underline{B}$ to mean the cpo $\underline{B}$.
- If $b\in\underline{B}$ we write $\mathsf{thunk}\ b$ for b regarded as an element of $U\underline{B}$.
- If $a\in U\underline{B}$ we write $\mathsf{force}\ a$ for a regarded as an element of $\underline{B}$.

It may seem superfluous to write U, $\mathtt{thunk}$ and force when referring to the Scott model, because they are invisible. But the advantage of doing so is that everything we write in this notation is meaningful not just in the Scott model but in any CBPV model.

3.9 Useful Syntactic Sugar

3.9.1 Pattern-Matching

It is convenient to extend all constructs that bind identifiers to allow pattern-matching. We give some examples.

sugar	unsugared
$M\ \mathtt{to}\ \{\dots,(i,\mathtt{x}).N_i,\dots\}$	$M\ \mathtt{to\ z.\ pm\ z\ as}\ \{\dots,(i,\mathtt{x}).N_i,\dots\}$
$\lambda(\mathtt{x},\mathtt{y}).M$	$\lambda\mathtt{z}.(\mathtt{pm\ z\ as}\ (\mathtt{x},\mathtt{y}).M)$
$\mathtt{pm}\ M\ \mathtt{as}\ (\mathtt{w},(\mathtt{x},\mathtt{y})).N$	$\mathtt{pm}\ M\ \mathtt{as}\ (\mathtt{w},\mathtt{z}).(\mathtt{pm\ z\ as}\ (\mathtt{x},\mathtt{y}).N)$

We use such abbreviations only informally, as it would be complicated to give a precise, general description.

3.9.2 Commands

It is sometimes convenient to have a computation type $\underline{\texttt{comm}}$ of *commands* such as $\texttt{print}\ c$, with the following rules:

$$\frac{}{\Gamma \vdash^{\mathsf{c}} \texttt{skip} : \underline{\texttt{comm}}} \qquad \frac{\Gamma \vdash^{\mathsf{c}} M : \underline{\texttt{comm}} \quad \Gamma \vdash^{\mathsf{c}} N : \underline{B}}{\Gamma \vdash^{\mathsf{c}} M; N : \underline{B}}$$

$$\frac{}{\Gamma \vdash^{\mathsf{c}} \texttt{print}\ c : \underline{\texttt{comm}}}$$

It is also useful to have a computation type $\underline{\texttt{nrcomm}}$ of *non-returning commands* (e.g. `diverge`, $\texttt{error}\ e$), with the following rule:

$$\frac{\Gamma \vdash^{\mathsf{c}} M : \underline{\texttt{nrcomm}}}{\Gamma \vdash^{\mathsf{c}} \texttt{coerce}\ _{\underline{B}} M : \underline{B}}$$

Both can be regarded as sugar:

sugar	unsugared
$\underline{\texttt{comm}}$	$F1$
`skip`	$\texttt{produce}\ ()$
$M; N$	$M\ \texttt{to}\ ().\ N$
$\texttt{print}\ c$	$\texttt{print}\ c;\ \texttt{produce}\ ()$
$\underline{\texttt{nrcomm}}$	$F0$
$\texttt{coerce}\ _{\underline{B}} M$	$M\ \texttt{to}\ \{\}$

In order to give the CK-machine rule for $\texttt{coerce}\ M$, we provide a dummy outside `neverused` for non-returning commands. The machine rule is then

$\texttt{coerce}\ M$	K
M	`neverused`

The typing rule for `neverused` is

$$\frac{}{\Gamma \vdash^{\mathsf{k}}_{\underline{C}} \texttt{neverused} : \underline{\texttt{nrcomm}}}$$

Chapter 4

Complex Values and the CBPV Equational Theory

4.1 Introduction

In this chapter we build the CBPV equational theory and look at its main properties. In order for the equational theory to have reasonable mathematical properties (in particular, categorical semantics), we need to add in Sect. 4.2 some extra terms called *complex values*, although we show in Sect. 4.6 that they can be eliminated to some extent.

As in Sect. 2.3.4, we will look at the reversible derivations present in the equational theory. We will use these derivations to construct isomorphisms between types, and convert types by isomorphism into canonical form.

Finally, we will look at the relationships between the CBPV equational theory and effect-free equational theories.

4.2 Complex Values

Suppose we want to form a value $\mathtt{x} : \mathtt{bool} \vdash^{\mathsf{v}} \mathtt{not}\ \mathtt{x} : \mathtt{bool}$. This seems reasonable: it makes sense denotationally, and a natural way to code it is $\mathtt{if}\ \mathtt{x}\ \mathtt{then}\ \mathtt{false}\ \mathtt{else}\ \mathtt{true}$. But the CBPV syntax as defined in Fig. 3.1 will not allow this, because the rules allow pattern-matching into computations only, not into values. (Recall from Sect. 3.2 that $\mathtt{if}$ is the pattern-match construct for $\mathtt{bool} = 1 + 1$.) A similar problem arises with values such as $\mathtt{w} + 5$ and $\mathtt{x} + \mathtt{y}$, which we used in our example program in Sect. 1.5.2.

For another example—and one which which will be indispensable when we come to equational/ categorical issues, because it gives us a cartesian category of values—try to form a value $\mathtt{x} : \mathtt{bool} \times \mathtt{nat} \vdash^{\mathsf{v}} \pi\mathtt{x} : \mathtt{bool}$. Again, this makes sense denotationally, and a natural way to code it is $\mathtt{pm}\ \mathtt{x}\ \mathtt{as}\ (\mathtt{y},\mathtt{z}).\ \mathtt{y}$. But this too involves pattern-matching into a value.

All these are examples of *complex values*. We incorporate them into the CBPV language by adding the rules

$$\frac{\Gamma \vdash^{\mathsf{v}} V : A \quad \Gamma, \mathtt{x} : A \vdash^{\mathsf{v}} W : B}{\Gamma \vdash^{\mathsf{v}} \mathtt{let}\ \mathtt{x}\ \mathtt{be}\ V.\ W : B}$$

$$\frac{\Gamma \vdash^{\mathsf{v}} V : \textstyle\sum_{i \in I} A_i \quad \cdots \quad \Gamma, \mathtt{x} : A_i \vdash^{\mathsf{v}} W_i : B \quad \cdots}{\Gamma \vdash^{\mathsf{v}} \mathtt{pm}\ V\ \mathtt{as}\ \{\ldots, (i,\mathtt{x}).W_i, \ldots\} : B}$$

$$\frac{\Gamma \vdash^{\mathsf{v}} V : A \times A' \quad \Gamma, \mathtt{x} : A, \mathtt{y} : A' \vdash^{\mathsf{v}} W : B}{\Gamma \vdash^{\mathsf{v}} \mathtt{pm}\ V\ \mathtt{as}\ (\mathtt{x},\mathtt{y}).W : B}$$

All these rules make sense in every denotational model, and, as we mentioned above, they are indispensable when we study equational/categorical issues. The reader may therefore wonder: why did we not include these rules from the outset? The answer is that excluding complex values keeps the operational semantics simple: both our big-step and our machine semantics exploit the fact that values do not need to be evaluated. Furthermore, the range of the transforms from CBN and (coarse-grain) CBV into CBPV does not involve complex values.

It would certainly be possible to extend the operational semantics to include complex values, but at the cost of canonicity. We would have to make an arbitrary decision as to when to evaluate complex values. Since the evaluation of complex values causes no effects, the decision has no semantic significance. We will therefore continue to exclude complex values when treating operational issues, but otherwise we will include them.

We shall see in Sect. 4.6 that complex values add no expressive power to computations or to closed values, because a computation or closed value containing complex values can be converted into one without. For example, `produce` (`if x then false else true`) can be converted into `if x then produce true else produce false`.

4.3 Equations

In Sect. 3.1 we looked at equational properties of CBV and CBN, and the informal reasons for them:

- In CBV, the η-law for sum types (and boolean type) holds because identifiers are bound to values.
- In CBN, the η-law for function types holds because a term of function type can be made to evaluate only by applying it.

We stipulated that a subsuming language should have both of these properties. We can now see that CBPV indeed meets these requirements.

- Identifiers are bound to values, so the η-laws for sum types holds.
- A term of function type can be made to evaluate only by applying it, so the η-law for function types holds.

The equational issue is more immediately apparent from a denotational perspective, whether we look at the printing semantics or the Scott semantics:

- In CBPV, $[\![A + A']\!]$ is precisely the disjoint union of $[\![A]\!]$ and $[\![A']\!]$—like in CBV but unlike in CBN.
- In CBPV $[\![A]\!] \to [\![\underline{B}]\!]$ is precisely the set of functions (or the cpo of continuous functions) from $[\![A]\!]$ to $[\![\underline{B}]\!]$—like in CBN but unlike in CBV.

We thus formulate an equational theory for CBPV (with complex values), given in Fig 4.1. M, N and P range over computations, while V and W range over values.

The equations for `print` and `diverge` are of course specific to our example effects, but there are directly analogous equations for many other effects. The sequencing equations will be used in Prop. 24, and throughout Part III. We call this theory (without the `print` and `diverge` equations) the *CBPV equational theory*.

For both printing and Scott models we have (as a consequence of a substitution lemma, which we omit)

Proposition 22 Provably equal terms have the same denotation. □

β-laws

$$
\begin{array}{rcl}
\texttt{let x be } V.\ M & = & M[V/\texttt{x}] \\
\texttt{let x be } V.\ W & = & W[V/\texttt{x}] \\
(\texttt{produce } V) \texttt{ to x. } M & = & M[V/\texttt{x}] \\
\texttt{force thunk } M & = & M \\
\texttt{pm } (\hat{\imath}, V) \texttt{ as } \{\dots, (i,\texttt{x}).M_i, \dots\} & = & M_{\hat{\imath}}[V/\texttt{x}] \\
\texttt{pm } (\hat{\imath}, V) \texttt{ as } \{\dots, (i,\texttt{x}).W_i, \dots\} & = & W_{\hat{\imath}}[V/\texttt{x}] \\
\texttt{pm } (V, V') \texttt{ as } (\texttt{x},\texttt{y}).M & = & M[V/\texttt{x}, V'/\texttt{y}] \\
\texttt{pm } (V, V') \texttt{ as } (\texttt{x},\texttt{y}).W & = & W[V/\texttt{x}, V'/\texttt{y}] \\
\hat{\imath}\text{`}\lambda\{\dots, i.M_i, \dots\} & = & M_{\hat{\imath}} \\
V\text{`}\lambda\texttt{x}.M & = & M[V/\texttt{x}]
\end{array}
$$

η-laws

$$
\begin{array}{rcl}
M & = & M \texttt{ to x. produce x} \\
V & = & \texttt{thunk force } V \\
M[V/\texttt{z}] & = & \texttt{pm } V \texttt{ as } \{\dots, (i,\texttt{x}).M[(i,\texttt{x})/\texttt{z}], \dots\} \\
W[V/\texttt{z}] & = & \texttt{pm } V \texttt{ as } \{\dots, (i,\texttt{x}).W[(i,\texttt{x})/\texttt{z}], \dots\} \\
M[V/\texttt{z}] & = & \texttt{pm } V \texttt{ as } (\texttt{x},\texttt{y}).M[(\texttt{x},\texttt{y})/\texttt{z}] \\
W[V/\texttt{z}] & = & \texttt{pm } V \texttt{ as } (\texttt{x},\texttt{y}).W[(\texttt{x},\texttt{y})/\texttt{z}] \\
M & = & \lambda\{\dots, i.i\text{`}M, \dots\} \\
M & = & \lambda\texttt{x}.(\texttt{x}\text{`}M)
\end{array}
$$

sequencing laws

$$
\begin{array}{rcl}
(M \texttt{ to x. } N) \texttt{ to y. } P & = & M \texttt{ to x. } (N \texttt{ to y. } P) \\
M \texttt{ to x. } \lambda\{\dots, i.N_i, \dots\} & = & \lambda\{\dots, i.(M \texttt{ to x. } N_i), \dots\} \\
M \texttt{ to x. } \lambda\texttt{y}.N & = & \lambda\texttt{y}.(M \texttt{ to x. } N)
\end{array}
$$

`print` laws

$$
\begin{array}{rcl}
(\texttt{print } c;\ M) \texttt{ to x. } N & = & \texttt{print } c;\ (M \texttt{ to x. } N) \\
\texttt{print } c;\ \lambda\{\dots, i.M_i, \dots\} & = & \lambda\{\dots, i.(\texttt{print } c;\ M), \dots\} \\
\texttt{print } c;\ \lambda\texttt{x}.M & = & \lambda\texttt{x}.(\texttt{print } c;\ M)
\end{array}
$$

`diverge` laws

$$
\begin{array}{rcl}
\texttt{diverge to x. } N & = & \texttt{diverge} \\
\texttt{diverge} & = & \lambda\{\dots, i.\texttt{diverge}, \dots\} \\
\texttt{diverge} & = & \lambda\texttt{x}.\texttt{diverge}
\end{array}
$$

Figure 4.1: CBPV equations, using conventions of Sect. 1.4.2

4.4 CK-Machine Illuminates → Equations

We recall from Sect. 1.5.2 and Sect. 3.3.2 that $\lambda\mathtt{x}$ and V‘ can be read as commands:

- $\lambda\mathtt{x}$ means “pop $\mathtt{x}$”;
- V‘ means “push V”.

This reading illuminates many equations involving $\to$ (as well as the analogous equations involving $\prod$). Here are some examples.

1. Consider the equation

$$\mathtt{print\ "hello";}\ \lambda\mathtt{x}.M = \lambda\mathtt{x}.(\mathtt{print\ "hello";}\ M) \tag{4.1}$$

   In the CK-machine reading, the LHS means

   print ”hello”;
   pop x;
   M

   while the RHS means

   pop x;
   print ”hello”;
   M

   Provided the stack is non-empty (as it must be if the initial configuration was ground), these two behaviours are the same, because popping and printing do not interfere.

2. In the β-law

$$V\text{‘}\lambda\mathtt{x}.M \simeq M[V/\mathtt{x}]$$

   the LHS means

   push V;
   pop x;
   M

   The first two lines have the effect of binding $\mathtt{x}$ to V and leaving the stack unchanged, so the overall effect is to obey M with $\mathtt{x}$ bound to V.

3. In the η-law

$$M \simeq \lambda\mathtt{x}.(\mathtt{x}\text{‘}M)$$

   the RHS means

   pop x;
   push x;
   M

   Assuming again that the stack is non-empty, the first two lines have the effect of leaving the stack unchanged and binding $\mathtt{x}$ to the top entry in the stack. But we are assuming that $\mathtt{x}$ is not used in M, so the first two lines can be removed.

4. The analogue of (4.1) for divergence is

$$\texttt{diverge} \simeq \lambda \texttt{x}.\texttt{diverge}$$

The RHS means

$$\begin{array}{l}\texttt{pop x;}\\ \texttt{diverge}\end{array}$$

Assuming again that the stack is non-empty, this diverges.

As an aside, we notice that all these 4 equations, and the push/pop reading that explains them, hold in CBN as well as in CBPV.

4.5 Reversible Derivations

Proposition 23 The CBPV equational theory possesses the following reversible derivations, all of which preserve substitution in Γ:

$$\frac{\cdots\ \Gamma, A_i \vdash^{\mathsf{v}} B\ \cdots}{\Gamma, \sum_{i\in I} A_i \vdash^{\mathsf{v}} B} \qquad \frac{\cdots\ \Gamma, A_i \vdash^{\mathsf{c}} \underline{B}\ \cdots}{\Gamma, \sum_{i\in I} A_i \vdash^{\mathsf{c}} \underline{B}}$$

$$\frac{\Gamma, A, A' \vdash^{\mathsf{v}} B}{\Gamma, A \times A' \vdash^{\mathsf{v}} B} \qquad \frac{\Gamma, A, A' \vdash^{\mathsf{c}} \underline{B}}{\Gamma, A \times A' \vdash^{\mathsf{c}} \underline{B}}$$

$$\frac{\Gamma \vdash^{\mathsf{v}} A \quad \Gamma \vdash^{\mathsf{v}} A'}{\Gamma \vdash^{\mathsf{v}} A \times A'} \qquad \frac{\Gamma \vdash^{\mathsf{c}} \underline{B}}{\Gamma \vdash^{\mathsf{v}} U\underline{B}}$$

$$\frac{\cdots\ \Gamma \vdash^{\mathsf{c}} \underline{B}_i\ \cdots}{\Gamma \vdash^{\mathsf{c}} \prod_{i\in I} \underline{B}_i} \qquad \frac{\Gamma, A \vdash^{\mathsf{c}} \underline{B}}{\Gamma \vdash^{\mathsf{c}} A \to \underline{B}}$$

□

For example, the reversible derivation for $\to$ is given by

- the operation $\theta : M \mapsto \lambda \texttt{x}.M$, which turns a computation $\Gamma, \texttt{x} : A \vdash^{\mathsf{c}} M : \underline{B}$ into a computation $\Gamma \vdash^{\mathsf{c}} N : A \to \underline{B}$
- the operation $\theta^{-1} : N \mapsto \texttt{x`}N$, which turns a computation $\Gamma \vdash^{\mathsf{c}} N : A \to \underline{B}$ into a computation $\Gamma, \texttt{x} : A \vdash^{\mathsf{c}} M : \underline{B}$

and it is clear that these operations are inverse up to provable equality, using the β- and η-laws for $\to$, and that they commute with substitution in Γ, as we explained in Sect. 2.3.4.

Notice that the only type constructor that does not possess a reversible derivation is F. All the badness present in the CBN sum type and in the CBV function type is, in CBPV, concentrated in F. For this reason, it is F that makes the categorical semantics of CBPV complicated.

Definition 21 A reversible derivation θ between computations is said to *preserve printing*, to *preserve divergence* or to *preserve sequencing in* Γ when the respective equation

$$\begin{array}{rcll} \theta(\texttt{print}\ c;\ M) & = & \texttt{print}\ c;\ \theta(M) & \\ \theta(\texttt{diverge}) & = & \texttt{diverge} & \\ \theta(P\ \texttt{to}\ \texttt{z}.\ M) & = & P\ \texttt{to}\ \texttt{z}.\ \theta(M) & (\texttt{z} \text{ an identifier in } \Gamma) \end{array}$$

is provable. □

We can make similar definitions for other effects. As with preservation of substitution, if θ preserves printing/divergence/sequencing then so does θ^{-1}.

Proposition 24 Each reversible derivation between computations listed in Prop. 23 preserves printing, divergence and sequencing in Γ. □

For the above example of $\theta : M \mapsto \lambda \mathtt{x}.M$ this result is given by the equations

$$\begin{array}{rcl} \lambda \mathtt{x}.(\mathtt{print}\ c;\ M) & = & \mathtt{print}\ c;\ \lambda \mathtt{x}.\ M \\ \lambda \mathtt{x}.\mathtt{diverge} & = & \mathtt{diverge} \\ \lambda \mathtt{x}.(P\ \mathtt{to}\ \mathtt{z}.\ M) & = & P\ \mathtt{to}\ \mathtt{z}.\ \lambda \mathtt{x}.N \end{array}$$

all of which are included in Fig. 4.1.

In fact, if θ preserves substitution and sequencing in Γ, then it automatically preserves printing and divergence, because of the provable equations

$$\begin{array}{rcl} \mathtt{print}\ c;\ M & = & (\mathtt{print}\ c; \mathtt{produce}\ ())\ \mathtt{to}\ ().\ M \\ \mathtt{diverge} & = & \mathtt{diverge}_{F0}\ \mathtt{to}\ \{\} \end{array}$$

As we stated in Sect. 2.3.4, these reversible derivations provide important information about the categorical structure of the equational theory. We shall see this in Part III.

4.6 Complex Values are Eliminable

Lemma 25 If $M = N$ is provable and M and N do not contain complex values then $M \simeq N$. □

Proof This follows from Prop. 22 and Cor. 18. □

We are now in a position to see that complex values can be eliminated from computations and from closed values.

Proposition 26

1. There is an effective procedure that, given a computation $\Gamma \vdash^{\mathsf{c}} M : \underline{B}$, possibly containing complex values, returns a computation $\Gamma \vdash^{\mathsf{c}} \tilde{M} : \underline{B}$ without complex values, such that $M = \tilde{M}$ is provable.

2. There is an effective procedure that, given a closed value $\vdash^{\mathsf{v}} V : A$, possibly containing complex values, returns a closed value $\vdash^{\mathsf{v}} \tilde{V} : A$ without complex values, such that $V = \tilde{V}$ is provable.

□

Proof

1. We will define one such procedure, and simultaneously, for each value $\Gamma \vdash^{\mathsf{v}} V : A$, possibly containing complex values, and each complex-value-free computation $\Gamma, \mathtt{v} : A \vdash^{\mathsf{c}} N : \underline{B}$, we will define a complex-value-free computation $\Gamma \vdash^{\mathsf{c}} N[V/\!\!/\mathtt{v}] : \underline{B}$ such that $N[V/\!\!/\mathtt{v}] = N[V/\mathtt{v}]$ is provable.

   By mutual induction on M and V, we define $\tilde{M}$ and $N[V/\!\!/\mathtt{v}]$ in Fig. 4.2 and we prove the equations

$$\begin{array}{rcl} \tilde{M} = M & & \\ N[V/\!\!/\mathtt{v}] & = & N[V/\mathtt{v}] \end{array}$$

2. For a value $A_0, \ldots, A_{n-1} \vdash^{\mathtt{v}} V : A$ possibly containing complex values, we will define $\tilde{V}$ to be a function that, when applied to a sequence $W_0, \ldots, W_{n-1}$ of complex-value-free closed values $\vdash^{\mathtt{v}} W_i : A_i$, returns a complex-value-free closed value $\tilde{V}(W_0, \ldots, W_{n-1})$ such that

$$\tilde{V}(W_0, \ldots, W_{n-1}) = V[\overrightarrow{W_i/\mathtt{x}_i}]$$

is provable. This is defined by induction on V in Fig. 4.2. The special case $n = 0$ gives the desired result.

□

V	$N[V/\!\!/\mathtt{v}]$
$\mathtt{x}$	$N[\mathtt{x}/\mathtt{v}]$
$\mathtt{let}\ \mathtt{x}\ \mathtt{be}\ W.\ U$	$(\mathtt{let}\ \mathtt{x}\ \mathtt{be}\ \mathtt{w}.\ N[U/\!\!/\mathtt{v}])[W/\!\!/\mathtt{w}]$
$(\hat{\imath}, W)$	$(\mathtt{let}\ \mathtt{v}\ \mathtt{be}\ (\hat{\imath}, \mathtt{w}).\ N)[W/\!\!/\mathtt{w}]$
$\mathtt{pm}\ W\ \mathtt{as}\ \{\ldots, (i, \mathtt{x}).U_i, \ldots\}$	$(\mathtt{pm}\ \mathtt{w}\ \mathtt{as}\ \{\ldots, (i, \mathtt{x}).N[U_i/\!\!/\mathtt{v}], \ldots\})[W/\!\!/\mathtt{w}]$
(W, W')	$(\mathtt{let}\ \mathtt{v}\ \mathtt{be}\ (\mathtt{w}, \mathtt{x}).\ N)[W/\!\!/\mathtt{w}][W'/\!\!/\mathtt{x}]$
$\mathtt{pm}\ W\ \mathtt{as}\ (\mathtt{x}, \mathtt{y}).U$	$(\mathtt{pm}\ \mathtt{w}\ \mathtt{as}\ (\mathtt{x}, \mathtt{y}).N[U/\!\!/\mathtt{v}])[W/\!\!/\mathtt{w}]$
$\mathtt{thunk}\ M$	$N[\mathtt{thunk}\ \tilde{M}/\mathtt{v}]$

M	$\tilde{M}$
$\mathtt{let}\ \mathtt{x}\ \mathtt{be}\ W.\ N$	$(\mathtt{let}\ \mathtt{x}\ \mathtt{be}\ \mathtt{w}.\ \tilde{N})[W/\!\!/\mathtt{w}]$
$\mathtt{pm}\ W\ \mathtt{as}\ \{\ldots, (i, \mathtt{x}).N_i, \ldots\}$	$(\mathtt{pm}\ \mathtt{w}\ \mathtt{as}\ \{\ldots, (i, \mathtt{x}).\tilde{N}_i, \ldots\})[W/\!\!/\mathtt{w}]$
$\mathtt{pm}\ W\ \mathtt{as}\ (\mathtt{x}, \mathtt{y}).N$	$(\mathtt{pm}\ \mathtt{w}\ \mathtt{as}\ (\mathtt{x}, \mathtt{y}).\tilde{N})[W/\!\!/\mathtt{w}]$
$\lambda\{\ldots, (i, \mathtt{x}).N_i, \ldots\}$	$\lambda\{\ldots, (i, \mathtt{x}).\tilde{N}_i, \ldots\}$
$\hat{\imath}\text{`}N$	$\hat{\imath}\text{`}\tilde{N}$
$\lambda\mathtt{x}.N$	$\lambda\mathtt{x}.\tilde{N}$
$V\text{`}N$	$(\mathtt{v}\text{`}\tilde{N})[V/\!\!/\mathtt{v}]$
$\mathtt{produce}\ V$	$(\mathtt{produce}\ \mathtt{v})[V/\!\!/\mathtt{v}]$
$N\ \mathtt{to}\ \mathtt{x}.\ P$	$\tilde{N}\ \mathtt{to}\ \mathtt{x}.\ \tilde{P}$
$\mathtt{force}\ V$	$(\mathtt{force}\ \mathtt{v})[V/\!\!/\mathtt{v}]$
$\mathtt{print}\ c;\ N$	$\mathtt{print}\ c;\ \tilde{N}$

V	$\tilde{V}(\overrightarrow{W_i})$
$\mathtt{x}_i$	W_i
$\mathtt{let}\ \mathtt{x}\ \mathtt{be}\ W.\ U$	$\tilde{U}(\overrightarrow{W_i}, \tilde{W}(\overrightarrow{W_i}))$
$(\hat{\imath}, W)$	$(\hat{\imath}, \tilde{W}(\overrightarrow{W_i}))$
$\mathtt{pm}\ W\ \mathtt{as}\ \{\ldots, (i, \mathtt{x}).U_i, \ldots\}$	$\tilde{U}_{\hat{\imath}}(\overrightarrow{W_i}, V')$ where $\tilde{W}(\overrightarrow{W_i}) = (\hat{\imath}, V')$
(W, W')	$(\tilde{W}(\overrightarrow{W_i}), \tilde{W}'(\overrightarrow{W_i}))$
$\mathtt{pm}\ W\ \mathtt{as}\ (\mathtt{x}, \mathtt{y}).U$	$\tilde{U}(\overrightarrow{W_i}, V', V'')$ where $\tilde{W}(\overrightarrow{W_i}) = (V', V'')$
$\mathtt{thunk}\ M$	$\mathtt{thunk}\ \tilde{M}[\overrightarrow{W_i/\mathtt{x}_i}]$

Figure 4.2: Definitions used in proof of Prop. 26

By choosing some such procedure, we can extend the operational semantics to include ground producers with complex values: if M is such a producer, we say that $M \Downarrow m, i$ iff $\tilde{M} \Downarrow m, i$. Because of Lemma 25, this relation does not depend on the choice of procedure $\tilde{-}$.

We can adapt the proof of Prop. 26 to show that complex values can be removed from a context. So it is immaterial whether, in our definition of observational equivalence, we allow contexts to contain complex values.

We can extend Prop, 22 and Cor. 18 as follows, for the printing language:

Proposition 27 Provable equality implies denotational equality, which implies observational equivalence. □

This adapts to the various effects and denotational models we will look at.

4.7 Syntactic Isomorphisms

4.7.1 Desired Examples

We frequently want to say that two CBPV types are "isomorphic". Here are some examples:

$$A \to (B \to \underline{C}) \quad \cong \quad (A \times B) \to \underline{C} \tag{4.2}$$

$$U\underline{A} \times U\underline{A}' \quad \cong \quad U(\underline{A} \sqcap \underline{A}') \tag{4.3}$$

(4.2) is the currying isomorphism. Operationally, this says that popping two operands successively is essentially the same as popping a pair of operands.

(4.3) says that a pair of thunks can be coalesced into a single thunk, of a computation that first pops a binary tag and proceeds accordingly.

These isomorphisms are obviously valid in both the printing model and the Scott model. For example, there is a canonical isomorphism (4.2) of $\mathcal{A}$-sets (in the sense of Def. 10(1)) and a canonical isomorphism (4.3) of sets. But we require a *syntactic* notion of "isomorphism" between two types. This is straightforward for value types but less so for computation types.

4.7.2 A Suitable Definition

Definition 22 1. An *isomorphism* between value types A and B is a reversible derivation θ

$$\frac{\Gamma \vdash^{\mathsf{v}} A}{\Gamma \vdash^{\mathsf{v}} B}$$

that preserves substitution in Γ. (By the Yoneda embedding, we could also characterize an isomorphism as a pair of terms $A \vdash^{\mathsf{v}} V : B$ and $B \vdash^{\mathsf{v}} W : A$ inverse up to provable equality.)

2. An *isomorphism* between computation types $\underline{A}$ and $\underline{B}$ is a reversible derivation θ

$$\frac{\Gamma \vdash^{\mathsf{c}} \underline{A}}{\Gamma \vdash^{\mathsf{c}} \underline{B}}$$

that preserves substitution and sequencing in Γ.

□

In Chap. 15.1 we will explain why we define (2) in this way. For the moment we can provide a rough intuition by saying that preservation of sequencing—which, as we said in Sect. 4.5, implies preservation of printing and divergence–is analogous to the structure preservation in Def. 10 and Def. 12. A reversible derivation that preserves substitution but not sequencing in Γ gives us only a syntactic isomorphism between $U\underline{A}$ and $U\underline{B}$. It is therefore analogous to a bijection between the *carriers* of $[\![\underline{A}]\!]$ and $[\![\underline{B}]\!]$, rather than an $\mathcal{A}$-set isomorphism between $[\![\underline{A}]\!]$ and $[\![\underline{B}]\!]$.

We can now show that the examples above are instances of Def. 22.

(4.2) can be given as the composite reversible derivation

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\Gamma \vdash^{\mathsf{c}} A \to (B \to \underline{C})}{\Gamma, A \vdash^{\mathsf{c}} B \to \underline{C}}}{\Gamma, A, B \vdash^{\mathsf{c}} \underline{C}}}{\Gamma, A \times B \vdash^{\mathsf{c}} \underline{C}}}{\Gamma \vdash^{\mathsf{c}} (A \times B) \to \underline{C}}$$

which preserves substitution and sequencing in Γ because each of its factors does.

(4.3) can be given as the composite reversible derivation

$$\dfrac{\dfrac{\dfrac{\dfrac{\Gamma \vdash^{\mathsf{v}} U\underline{A} \times U\underline{A}'}{\Gamma \vdash^{\mathsf{v}} U\underline{A} \quad \Gamma \vdash^{\mathsf{v}} U\underline{A}'}}{\Gamma \vdash^{\mathsf{c}} \underline{A} \quad \Gamma \vdash^{\mathsf{c}} \underline{A}'}}{\Gamma \vdash^{\mathsf{c}} \underline{A} \sqcap \underline{A}'}}{\Gamma \vdash^{\mathsf{v}} U(\underline{A} \sqcap \underline{A}')}$$

which preserves substitution in Γ because each of its factors does.

4.7.3 Type Canonical Forms

The reader may be familiar with the fact that every PCF type is isomorphic to one of the form $A_0 \to \cdots \to A_{n-1} \to B$, where B is a ground type. There is an analogous result for CBPV. We say that CBPV types in the following inductively defined classes are called *type canonical forms*:

$$\begin{array}{rl} A ::= & \sum_{i \in I} U\underline{B}_i \\ \underline{B} ::= & \prod_{i \in I} (U\underline{B}_i \to FA_i) \end{array}$$

Proposition 28 Every CBPV type is isomorphic to a type canonical form. □

This is easily proved by induction over types. We can understand this result as follows.

- Every closed value V is a tuple (more accurately, a hereditary tuple) of tags and thunks. All the tags can be coalesced into a single tag, and (as we explained in Sect. 4.7.1) all the thunks can be coalesced into a single thunk. So V corresponds to a pair $(\hat{\imath}, \mathtt{thunk}\, M)$.
- Every η-expanded closed computation M pops several operands and then behaves as a producer. Each of the operands is either a tag or a value, and each operand which is a value is a tuple (more accurately, a hereditary tuple) of tags and thunks. So we can coalesce all the operands into a single tag and a single thunk. Thus M corresponds to $\lambda\{\ldots, i.\lambda \mathtt{x}.N_i, \ldots\}$ where each N_i is a producer and each $\mathtt{x}$ has thunk type.
- Every closed outside K consists of several operands followed by a consumer. Again, we can coalesce all the operands into a single tag and a single thunk. Thus K corresponds to an outside $\hat{\imath} :: (\mathtt{thunk}\, M) :: L$, where L is a consumer.

4.8 Relationship With Effect-Free Languages

The effect-free analogue of CBPV is the $\times\sum\prod\to$-calculus shown in Fig. 4.3; essentially this is the $\lambda\mathtt{bool}+$-calculus of Sect. 2.3 extended with both pattern-match products and projection

Types

$A ::= \quad 1 \mid A \times A \mid \sum_{i \in I} A_i \mid \prod_{i \in I} A_i \mid A \to A$
where each set I is finite (or countable, for *infinitely wide* $\times\sum\prod\to$-calculus)

Terms

$$\frac{}{\Gamma, \mathtt{x} : A, \Gamma' \vdash \mathtt{x} : A} \qquad \frac{\Gamma \vdash M : A \quad \Gamma, \mathtt{x} : A \vdash N : B}{\Gamma \vdash \mathtt{let\ x\ be}\ M.\ N : B}$$

$$\frac{\Gamma \vdash M : A \quad \Gamma \vdash M' : A'}{\Gamma \vdash (M, M') : A \times A'} \qquad \frac{\Gamma \vdash M : A \times A' \quad \Gamma, \mathtt{x} : A, \mathtt{y} : A' \vdash N : B}{\Gamma \vdash \mathtt{pm}\ M\ \mathtt{as}\ (\mathtt{x}, \mathtt{y}).N : B}$$

$$\frac{\Gamma \vdash M : A_{\hat{\imath}}}{\Gamma \vdash (\hat{\imath}, M) : \sum_{i \in I} A_i} \qquad \frac{\Gamma \vdash M : \sum_{i \in I} A_i \quad \cdots \quad \Gamma, \mathtt{x} : A_i \vdash N_i : B \quad \cdots}{\Gamma \vdash \mathtt{pm}\ M\ \mathtt{as}\ \{\ldots, (i, \mathtt{x}).N_i, \ldots\} : B}$$

$$\frac{\cdots \quad \Gamma \vdash M_i : B_i \quad \cdots}{\Gamma \vdash \lambda\{\ldots, i.M_i, \ldots\} : \prod_{i \in I} B_i} \qquad \frac{\Gamma \vdash M : \prod_{i \in I} B_i}{\Gamma \vdash \hat{\imath}\text{`}M : B_{\hat{\imath}}}$$

$$\frac{\Gamma, \mathtt{x} : A \vdash M : B}{\Gamma \vdash \lambda \mathtt{x}.M : A \to B} \qquad \frac{\Gamma \vdash M : A \quad \Gamma \vdash N : A \to B}{\Gamma \vdash M\text{`}N : B}$$

Equations, using conventions of Sect. 1.4.2

β**-laws**

$$\begin{array}{rcl}
\mathtt{let\ x\ be}\ M.\ N & = & N[M/\mathtt{x}] \\
\mathtt{pm}\ (M, M')\ \mathtt{as}\ (\mathtt{x}, \mathtt{y}).N & = & N[M/\mathtt{x}, M'/\mathtt{y}] \\
\mathtt{pm}\ (\hat{\imath}, M)\ \mathtt{as}\ \{\ldots, (i, \mathtt{x}).N_i, \ldots\} & = & N_{\hat{\imath}}[M/\mathtt{x}] \\
\hat{\imath}\text{`}\lambda\{\ldots, i.M_i, \ldots\} & = & M_{\hat{\imath}} \\
M\text{`}\lambda \mathtt{x}.N & = & N[M/\mathtt{x}]
\end{array}$$

η**-laws**

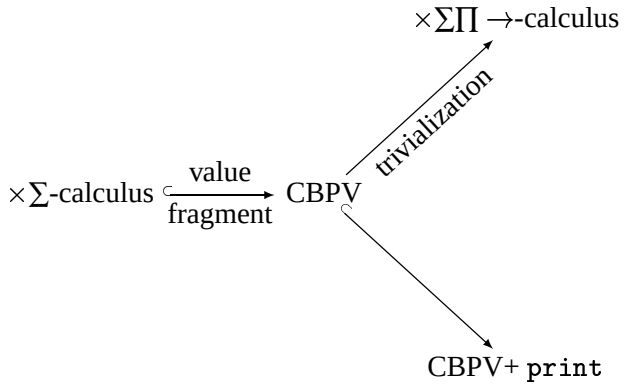
$$\begin{array}{rcl}
N[M/\mathtt{z}] & = & \mathtt{pm}\ M\ \mathtt{as}\ (\mathtt{x}, \mathtt{y}).N[(\mathtt{x}, \mathtt{y})/\mathtt{z}] \\
N[M/\mathtt{z}] & = & \mathtt{pm}\ M\ \mathtt{as}\ \{\ldots, (i, \mathtt{x}).N[(i, \mathtt{x})/\mathtt{z}], \ldots\} \\
M & = & \lambda\{\ldots, i.i\text{`}M, \ldots\} \\
M & = & \lambda \mathtt{x}.(\mathtt{x}\text{`}M)
\end{array}$$

Figure 4.3: The $\times\sum\prod\to$-Calculus

products. As this calculus is effect-free, it is not necessary to have both kinds of product, just convenient.

We use the term "$\times\sum$-calculus" for the fragment of this calculus using just $\times$ and $\sum$; similarly "$\times$-calculus" and so forth.

The important translations are shown thus:



We see two relationships here.

- $\times\sum$-calculus is a fragment of CBPV (called the *value fragment*). The embedding leaves types unchanged and transforms a term $\Gamma \vdash M : A$ into a value $\Gamma \vdash^{\mathsf{v}} V : A$.
- We can collapse effect-free CBPV into $\times\sum\prod \to$-calculus using the *trivialization transform*, written $-^{\mathsf{tr}}$. This transform discards U and F and leaves all other type constructors unchanged. It translates a value $\Gamma \vdash^{\mathsf{v}} V : A$ to a term $\Gamma^{\mathsf{tr}} \vdash V^{\mathsf{tr}} : A^{\mathsf{tr}}$, and translates a computation $\Gamma \vdash^{\mathsf{c}} M : \underline{B}$ to a term $\Gamma^{\mathsf{tr}} \vdash M^{\mathsf{tr}} : \underline{B}^{\mathsf{tr}}$.

Consequently:

- every CBPV model must include a model for $\times\sum$-calculus (the *value category*), such as **Set**, **Cpo** or **SEAM** (introduced in Sect. 5.2);
- every model for $\times\sum\prod \to$-calculus, such as **Set**, gives a model for CBPV (a *trivial model*).

Chapter 5

Recursion and Infinitely Deep CBPV

5.1 Introduction

This chapter is about the computational effect of divergence. There is little original work here; we are simply recalling and adapting well-known material, which is not specific to CBPV, for use in subsequent chapters.

First, we add recursion to CBPV—both recursive terms and recursive types—and look at the denotational semantics. Rather than use arbitrary cpos and cppos as we did in Chap. 3, we restrict the model to Scott-Ershov (SE) domains and Scott-Ershov-Abramsky-McCusker (SEAM) predomains. We do this to make the point that the category in which CBPV values are interpreted does not need to be cartesian closed. (Another benefit, as we explain below following the treatment in e.g. [SHLG94], is that the notion of (e,p)-pair used in the semantics of type recursion has a simple representation.) The material on denotational semantics of type recursion will be used further in Sect. 7.10, when we look at the denotational semantics of thunk storage.

Secondly, we look at a language feature that is, in a sense, equivalent to recursion—infinitely deep syntax. This means that the parse tree of a term or type can be infinitely deep, i.e. branches can be infinitely long. Böhm trees are a well-known example of such terms. Our only subsequent use of infinitely deep CBPV will be to state definability results for game semantics in Chap. 9.

It is important not to confuse this infinitely deep syntax with the weaker feature of *infinitely wide* syntax , which is used throughout the thesis. The latter simply means that we allow types $\sum_{i\in I} A_i$ and $\prod_{i\in I} \underline{B}_i$, where I is countably infinite. (Therefore, parse trees of types and terms are infinitely wide.) Infinitely wide syntax does not introduce divergence into the language, and all the CBPV semantics in the thesis can interpret it. It provides, for example, an easy way of giving a type of natural numbers: as $\sum_{i\in\mathbb{N}} 1$. We use it also as a metalanguage while describing the possible world semantics in Chap. 7.

It is easy to see that once we allow infinitely deep syntax, we can encode infinitely wide syntax—for example, the infinitely wide type $\sum_{i\in\mathbb{N}} 1$ can be written as the infinitely deep type $1+(1+(1+\cdots))$. In summary:

$$\begin{array}{c}\text{finitary}\\ \text{CBPV}\end{array} \subset \begin{array}{c}\text{infinitely wide}\\ \text{CBPV}\end{array} \subset \begin{array}{c}\text{infinitely deep}\\ \text{CBPV}\end{array}$$

We stress that these relationships apply equally to CBV or to CBN. However, the design philosophy of CBPV is maximalist ("How much can we add to the language without losing the semantics?"), and this perhaps makes it more natural to consider these extensions from a CBPV perspective than from a CBV or CBN perspective.

There is a serious problem that arises from both infinitely wide and infinitely deep syntax—non-computability. For example, when we define `nat` to be $\sum_{i\in\mathbb{N}} 1$, *every* function f from $\mathbb{N}$ to

$\mathbb{N}$ is definable, even if f is not computable:

$$\vdash^{\mathtt{c}} \lambda \mathtt{x}.\mathtt{pm}\ \mathtt{x}\ \mathtt{as}\ \{\ldots i.\mathtt{produce}\ f(i), \ldots\} : \mathtt{nat} \to F\mathtt{nat}$$

Consequently realizability models of CBPV (which we do not treat in this thesis) are not models of either infinitely wide CBPV or infinitely deep CBPV. This can probably be remedied by restricting infinitely deep CBPV to "computable terms" and "effectively presented types", as is done for strategies and arenas in game semantics (e.g. [HO94]). However, we have not investigated this.

5.2 SE Domains And SEAM Predomains

We can cut down the cpo model of CBPV in the following way.

Definition 23 1. A (Scott-Ershov) *Scott-Ershov (SE) domain* is a countably based, consistently complete, algebraic cppo.

2. A *Scott-Ershov-Abramsky-McCusker (SEAM) predomain* is a cpo isomorphic to one of the form $\sum_{i \in I} U \underline{X_i}$, where I is countable and each $\underline{X_i}$ is a SE domain.

3. **SEAM** is the category of SEAM predomains and continuous functions.

□

The Scott semantics for CBPV (even with recursive types, as we shall see) takes place entirely within SE domains and SEAM predomains. (Def. 23(2) is motivated by Prop. 28 for value types—cf. [AM98a].)

Notice that

- a SE domain is precisely a SEAM predomain with a least element;
- **SEAM** is not a cartesian closed category, because of the countability condition. For example, $\mathbb{N}$ is a flat SEAM predomain, but $\mathbb{N} \to \mathbb{N}$ is an uncountable flat cpo and hence not a SEAM predomain.

We recall the standard representation theory for SE domains [Plo83, SHLG94], and adapt it for SEAM predomains:

Definition 24 1. A *conditional upper semilattice* (cusl) is a poset $(A, \leqslant)$ with a least element $\perp$, in which every consistent (i.e. upper-bounded) finite subset has a least upper bound.

2. A *multi-cusl*[1] is a poset which is a disjoint union of a family of cusls.

3. An *ideal* of a poset $(A, \leqslant)$ is a subset $I \subseteq A$ such that
   - I is down-closed i.e. if $x \in I$ and $y \leqslant x$ then $y \in I$;
   - every finite subset of I has an upper bound in I.

□

Countable multi-cusls are equivalent to SEAM predomains, in the following sense.

- Given a SEAM predomain D, the poset of compact elements of D forms a countable multi-cusl.
- Given a countable multi-cusl A, the poset of ideals of A, ordered by inclusion, forms a SEAM predomain.
- These operations are inverse up to poset isomorphism.

Similarly, SE domains are equivalent to countable cusls.

[1] The term "pre-cusl" has already been used.

5.3 Divergence and Recursion

Although we have already described the Scott model of CBPV in Sect. 3.6, the model is rather pointless if we do not add recursion to the language. We now look at this addition.

5.3.1 Divergent and Recursive Terms

We add to the basic CBPV language the constructs

$$\frac{}{\Gamma \vdash^{\mathsf{c}} \mathtt{diverge} : \underline{B}} \qquad \frac{\Gamma, \mathtt{x} : U\underline{B} \vdash^{\mathsf{c}} M : \underline{B}}{\Gamma \vdash^{\mathsf{c}} \mu \mathtt{x}.M : \underline{B}}$$

While `diverge` is not strictly necessary (e.g. it can be desugared as $\mu\mathtt{x}.\mathtt{force}\,\mathtt{x}$) it is convenient to include it as a primitive.

We add the big-step rules

$$\frac{\mathtt{diverge} \Downarrow T}{\mathtt{diverge} \Downarrow T} \qquad \frac{M[\mathtt{thunk}\,\mu\mathtt{x}.M/\mathtt{x}] \Downarrow T}{\mu\mathtt{x}.M \Downarrow T}$$

The rule for `diverge` can of course never be applied, so it is technically dispensable. We have included it to reflect the operational idea of divergence: to evaluate `diverge`, one evaluates `diverge`, and so forth. We say that M *diverges* iff there does not exist T such that $M \Downarrow T$. (This definition is acceptable only in a deterministic setting.)

To the CK-machine we add transitions:

$$\begin{array}{lll} & \mathtt{diverge} & K \\ \rightsquigarrow & \mathtt{diverge} & K \\ \\ & \mu\mathtt{x}.M & K \\ \rightsquigarrow & M[\mathtt{thunk}\,\mu\mathtt{x}.M/\mathtt{x}] & K \end{array}$$

Proposition 29 Let M be a closed computation.

Soundness If $M \Downarrow T$ then $[\![M]\!] = [\![T]\!]$.

Adequacy If M diverges then $[\![M]\!] = \bot$.

□

Proof (in the style of [Tai67]) Soundness is straightforward. For adequacy, we define, by mutual induction over types, three families of relations:

for each A, $\leqslant^{\mathsf{v}}_A$ between $[\![A]\!]$ and $\mathbb{V}_A$;
for each $\underline{B}$, $\leqslant^{\mathsf{t}}_{\underline{B}}$ between $[\![\underline{B}]\!]$ and $\mathbb{T}_{\underline{B}}$;
for each $\underline{B}$, $\leqslant^{\mathsf{c}}_{\underline{B}}$ between $[\![\underline{B}]\!]$ and $\mathbb{C}_{\underline{B}}$;

The definition of these relations proceeds as follows:

$$\begin{array}{lll} a \leqslant^{\mathsf{v}}_{U\underline{B}} \mathtt{thunk}\, M & \text{iff} & \mathsf{force}\, a \leqslant^{\mathsf{c}}_{\underline{B}} M \\ a \leqslant^{\mathsf{v}}_{\sum_{i\in I} A_i} (\hat{\imath}, V) & \text{iff} & a = (\hat{\imath}, b) \text{ for some } b \leqslant^{\mathsf{v}}_{A_{\hat{\imath}}} V \\ a \leqslant^{\mathsf{v}}_{A\times A'} (V, V') & \text{iff} & a = (b, b') \text{ for some } b \leqslant^{\mathsf{v}}_{A} V \text{ and } b' \leqslant^{\mathsf{v}}_{A'} V' \\ \\ b \leqslant^{\mathsf{t}}_{FA} \mathtt{produce}\, V & \text{iff} & b = \bot \text{ or } b = \mathsf{produce}\, a \text{ for some } a \leqslant^{\mathsf{v}}_{A} V \\ f \leqslant^{\mathsf{t}}_{\prod_{i\in I}\underline{B}_i} \lambda\{\ldots, i.M_i, \ldots\} & \text{iff} & \hat{\imath} \in I \text{ implies } \hat{\imath}\text{`}f \leqslant^{\mathsf{c}}_{\underline{B}_{\hat{\imath}}} M_{\hat{\imath}} \\ f \leqslant^{\mathsf{t}}_{A\to\underline{B}} \lambda\mathtt{x}.M & \text{iff} & a \leqslant^{\mathsf{v}}_{A} V \text{ implies } a\text{`}f \leqslant^{\mathsf{c}}_{\underline{B}} M[V/\mathtt{x}] \\ \\ b \leqslant^{\mathsf{c}}_{\underline{B}} M & \text{iff} & b = \bot \text{ or, for some } T,\ M \Downarrow T \text{ and } b \leqslant^{\mathsf{t}}_{\underline{B}} T \end{array}$$

Notice that for terminal T, $b \leqslant^{\mathsf{c}}_{\underline{B}} T$ iff $b \leqslant^{\mathsf{t}}_{\underline{B}} T$. We prove by mutual induction over types the following:

- For each value $V \in \mathbb{V}_A$ the set $\{a \in [\![A]\!] : a \leqslant^{\mathsf{v}}_A V\}$ is admissible (closed under directed joins).
- [2] If $a \leqslant^{\mathsf{v}}_A V$ and $a' \leqslant^{\mathsf{v}}_A V$ and $\mathsf{produce}\, a \leqslant \mathsf{produce}\, a'$ then $a \leqslant a'$.
- For each terminal computation $T \in \mathbb{T}_{\underline{B}}$ the set $\{b \in [\![\underline{B}]\!] : b \leqslant^{\mathsf{t}}_{\underline{B}} T\}$ is admissible and contains $\bot$.
- For each computation $M \in \mathbb{C}_{\underline{B}}$ the set $\{b \in [\![\underline{B}]\!] : b \leqslant^{\mathsf{c}}_{\underline{B}} M\}$ is admissible and contains $\bot$.

Finally we show that for any computation $A_0, \ldots, A_{n-1} \vdash^{\mathsf{c}} M : \underline{B}$, if $a_i \leqslant^{\mathsf{v}}_{A_i} W_i$ for $i = 0, \ldots, n-1$ then $[\![M]\!]\overrightarrow{\mathtt{x}_i \mapsto a_i} \leqslant^{\mathsf{c}}_{\underline{B}} M[\overrightarrow{W_i/\mathtt{x}_i}]$; and similarly for any value $A_0, \ldots, A_{n-1} \vdash^{\mathsf{v}} V : A$. This is shown by mutual induction on M and V, and gives the required result. □

Corollary 30 For any closed ground producer M, we have $M \Downarrow \mathtt{produce}\, n$ iff $[\![M]\!] = \mathsf{produce}\, n$, and M diverges iff $[\![M]\!] = \bot$. Hence terms with the same denotation are observationally equivalent. □

5.3.2 Type Recursion

Both value types and computation types can be defined recursively, so we extend the type expressions as follows:

$$\begin{array}{rcl} A ::= & \cdots \mid \mathtt{X} \mid \mu\mathtt{X}.A \\ \underline{B} ::= & \cdots \mid \underline{\mathtt{X}} \mid \mu\underline{\mathtt{X}}.\underline{B} \end{array} \tag{5.1}$$

Here are some examples of CBPV recursive types:

$\mu\mathtt{X}.(1 + \mathtt{bool} \times \mathtt{X})$	the type of finite lists of booleans
$\mu\underline{\mathtt{X}}.F(1 + \mathtt{bool} \times U\underline{\mathtt{X}})$	the type of (finite or infinite) lazy lists of booleans

Notice again the flexibility of CBPV; it can describe both eager and lazy recursive types.

There is a syntactic difference between the two kinds of recursive type:

$$\frac{\Gamma \vdash^{\mathsf{v}} V : A[\mu\mathtt{X}.A/\mathtt{X}]}{\Gamma \vdash^{\mathsf{v}} \mathtt{fold}\, V : \mu\mathtt{X}.A} \qquad \frac{\Gamma \vdash^{\mathsf{v}} V : \mu\mathtt{X}\, A \quad \Gamma, \mathtt{x} : A[\mu\mathtt{X}.A/\mathtt{X}] \vdash^{\mathsf{c}} M : \underline{B}}{\Gamma \vdash^{\mathsf{c}} \mathtt{pm}\, V\ \mathtt{as}\ \mathtt{fold}\, \mathtt{x}.M : \underline{B}}$$

$$\frac{\Gamma \vdash^{\mathsf{c}} M : \underline{B}[\mu\underline{\mathtt{X}}.\underline{B}/\underline{\mathtt{X}}]}{\Gamma \vdash^{\mathsf{c}} \mathtt{fold}\, M : \mu\underline{\mathtt{X}}.\underline{B}} \qquad \frac{\Gamma \vdash^{\mathsf{c}} M : \mu\underline{\mathtt{X}}.\underline{B}}{\Gamma \vdash^{\mathsf{c}} \mathtt{unfold}\, M : \underline{B}[\mu\underline{\mathtt{X}}.\underline{B}/\underline{\mathtt{X}}]}$$

The reason we do not have a construct $\mathtt{unfold}\, V$,where V has type $\mu\mathtt{X}.A$, is that this is a *complex value*, in the sense of Sect. 4.2, and (as we explained there) we exclude complex values in order to keep the operational semantics canonical. For all non-operational purposes, however, we add the complex value rule

$$\frac{\Gamma \vdash^{\mathsf{v}} V : \mu\mathtt{X}\, A \quad \Gamma, \mathtt{x} : A[\mu\mathtt{X}.A/\mathtt{X}] \vdash^{\mathsf{v}} W : B}{\Gamma \vdash^{\mathsf{v}} \mathtt{pm}\, V\ \mathtt{as}\ \mathtt{fold}\, \mathtt{x}.W : B}$$

[2] This clause is used in proving the admissibility property for FA, but it is not really necessary because in the Scott model we immediately have a stronger result: $\mathsf{produce}\, a \leqslant \mathsf{produce}\, a'$ implies $a \leqslant a'$. However, we include this clause here so that the adequacy proof generalizes to other models where the stronger result is not valid.

—enabling us to define $\mathtt{unfold}\,V$ as $\mathtt{pm}\,V\ \mathtt{as}\ \mathtt{fold}\,\mathtt{x}.\mathtt{x}$—and the equations (using the conventions of Sect. 1.4.2)

$$\begin{array}{lrcl}
(\beta) & \mathtt{unfold}\,\mathtt{fold}\,M & = & M \\
(\beta) & \mathtt{pm}\,\mathtt{fold}\,V\ \mathtt{as}\ \mathtt{fold}\,\mathtt{x}.W & = & W[V/\mathtt{x}] \\
(\beta) & \mathtt{pm}\,\mathtt{fold}\,V\ \mathtt{as}\ \mathtt{fold}\,\mathtt{x}.M & = & M[V/\mathtt{x}] \\
(\eta) & M & = & \mathtt{fold}\,\mathtt{unfold}\,M \\
(\eta) & W[V/\mathtt{z}] & = & \mathtt{pm}\,V\ \mathtt{as}\ \mathtt{fold}\,\mathtt{x}.W[\mathtt{fold}\,\mathtt{x}/\mathtt{z}] \\
(\eta) & M[V/\mathtt{z}] & = & \mathtt{pm}\,V\ \mathtt{as}\ \mathtt{fold}\,\mathtt{x}.M[\mathtt{fold}\,\mathtt{x}/\mathtt{z}] \\
 & M\ \mathtt{to}\,\mathtt{x}.\,\mathtt{fold}\,N & = & \mathtt{fold}\,(M\ \mathtt{to}\,\mathtt{x}.\,N) \\
 & \mathtt{diverge} & = & \mathtt{fold}\,\mathtt{diverge} \\
 & \mathtt{print}\,c;\,\mathtt{fold}\,N & = & \mathtt{fold}\,(\mathtt{print}\,c;\,N)
\end{array}$$

(The last two are of course effect-specific—although divergence is necessarily possible in the presence of type recursion—but there are analogous equations for each effect.) We can then remove complex values from computations and closed values as in Prop. 26, and we have syntactic isomorphisms

$$\begin{array}{rcl}
\mu\mathtt{X}.A & \cong & A[\mu\mathtt{X}.A/\mathtt{X}] \\
\mu\underline{\mathtt{X}}.\underline{B} & \cong & \underline{B}[\mu\underline{\mathtt{X}}.\underline{B}/\underline{\mathtt{X}}]
\end{array}$$

For big-step operational semantics, we first extend the class of terminal computations

$$T ::= \quad \cdots \mid \mathtt{fold}\,M$$

and then add the following rules:

$$\frac{M[V/\mathtt{x}] \Downarrow T}{\mathtt{pm}\,\mathtt{fold}\,V\ \mathtt{as}\ \mathtt{fold}\,\mathtt{x}.M \Downarrow T}$$

$$\frac{}{\mathtt{fold}\,M \Downarrow \mathtt{fold}\,M} \qquad \frac{M \Downarrow T}{\mathtt{unfold}\,\mathtt{fold}\,M \Downarrow T}$$

To the CK-machine semantics we add

$$\begin{array}{rll}
 & \mathtt{unfold}\,\mathtt{fold}\,M & K \\
\rightsquigarrow & M & K \\
 & & \\
 & \mathtt{pm}\,\mathtt{fold}\,V\ \mathtt{as}\ \mathtt{fold}\,\mathtt{x}.M & K \\
\rightsquigarrow & M[V/\mathtt{x}] & K
\end{array}$$

5.3.3 Denotational Semantics for Type Recursion

The denotational semantics for recursive types are obtained using a general theory described in [SP82].

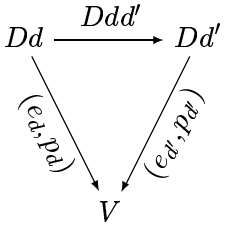
Definition 25 Let $\mathcal{C}$ be a poset-enriched category.

An (e,p)-*pair* from A to B consists of morphisms

$$A \underset{p}{\overset{e}{\rightleftarrows}} B$$

such that $e;p = \mathsf{id}_A$ and $p;e \leqslant \mathsf{id}_B$. e is called an *embedding* and p is called a *projection*—they determine each other. We write $\mathcal{C}^{\mathsf{ep}}$ for the category with the same objects as $\mathcal{C}$, and (e,p)-pairs as morphisms.

2. Let D be a directed diagram in $\mathcal{C}^{\mathsf{ep}}$ i.e. a functor from a directed poset $\mathbb{D}$, regarded as a small category, to $\mathcal{C}^{\mathsf{ep}}$. A *cocone* from D consists of an object V (the *vertex*) together with, for each $d \in \mathbb{D}$, an (e,p)-pair (e_d, p_d) from Dd to V, such that

for $d \leqslant d'$  commutes.

Such a cocone is an *O-colimit* when

$$\bigvee_{d \in \mathbb{D}} (p_d ; e_d) = \mathsf{id}_V \tag{5.2}$$

□

Many properties of O-colimits are given in [SP82].

Definition 26 An *enriched-compact category* $\mathcal{C}$ is a **Cpo**-enriched category with the following properties.

- Each hom-cpo $\mathcal{C}(A,B)$ has a least element $\bot$.
- Composition is bi-strict.

$$\begin{aligned} \bot ; g &= \bot \\ f ; \bot &= \bot \end{aligned}$$

- $\mathcal{C}$ has a zero object i.e. an object which is both initial and terminal. (Because of bi-strictness, just one of these properties is sufficient.)
- Every countable directed diagram of (e,p)-pairs has an O-colimit.

□

(The first 2 conditions could be expressed by saying that $\mathcal{C}$ is enriched in the monoidal category $(\mathbf{Cppo}_{\mathsf{strict}}, \otimes)$ of cppos and strict continuous functions with smash product.) Every enriched-compact category $\mathcal{C}$ is *algebraically compact* [Fre91] and so every locally continuous functor from $\mathcal{C}^{\mathsf{op}} \times \mathcal{C}$ to $\mathcal{C}$ has a canonical fixpoint (up to isomorphism).

To give examples of enriched-compact categories, we use the following.

Definition 27 (based on [AM98a]) Let A and B be SEAM predomains. A *partial-on-minimals* continuous function from A to B is a partial continuous function $A \xrightarrow{\quad f \quad} B$ such that f is defined on x iff f is defined on the least element below x. □

Notice that if $A \cong \sum_{i \in I} U\underline{X}_i$ to $B \cong \sum_{j \in J} U\underline{Y}_j$ then f can be specified by

- a partial function $I \xrightarrow{\quad g \quad} J$
- for each $i \in \mathsf{dom}\ g$, a continuous function from $U\underline{X}_i$ to $U\underline{Y}_{g(i)}$.

We will write $\mathbf{SEAM}_{\mathsf{partmin}}$ for the category of SEAM predomains and partial-on-minimals continuous functions and we will write $\mathbf{SE}_{\mathsf{strict}}$ for the category of SE domains and strict continuous functions. These are the categories in which we will interpret recursive value types and recursive computation types, respectively. Notice that an isomorphism in $\mathbf{SEAM}_{\mathsf{partmin}}$ is precisely an isomorphism of SEAM predomains, while an isomorphism in $\mathbf{SE}_{\mathsf{strict}}$ is precisely an isomorphism of SE domains.

Using the representation theory presented in Sect. 5.2, we can give a simple characterization of (e,p)-pairs in these two categories.

Proposition 31 [SHLG94]

1. Let A and B be the multi-cusls of compact elements of SEAM predomains X and Y. An (e,p) pair from X to Y in $\mathbf{SEAM}_{\mathsf{partmin}}$ is given by an injection i from A to B that preserves and reflects $\leqslant$, minimality, inconsistency and joins. In particular, it provides an injection from the minimal elements of A (equivalently, of X) to the minimal elements of B (equivalently, of Y).

2. Let A and B be the cusls of compact elements of SE domains $\underline{X}$ and $\underline{Y}$. An (e,p) pair from $\underline{X}$ to $\underline{Y}$ in $\mathbf{SE}_{\mathsf{strict}}$ is given by an injection from A to B that preserves and reflects $\leqslant$, $\perp$, inconsistency and joins.

□

Proof

(1) Given (e,p) we notice that e is total and injective and preserves compactness, so define i to be the restriction of e to the compact elements. Conversely, given i we define e to take $x \in X$ to $\bigvee_{a\in A, a\leqslant x} i(a)$ and p to take $y \in Y$ to $\bigvee_{a\in A, i(a)\leqslant y} a$ if $\{a \in A | i(a) \leqslant y\}$ is non-empty—otherwise p is undefined at y. It is easy to check that these constructions have the correct properties and are inverse.
Note If (e,p) is constructed from i in this way then the composite $p;e$ takes $y \in Y$ to $\bigvee_{a\in A, i(a)\leqslant y} i(a)$ if $\{a \in A | i(a) \leqslant y\}$ is non-empty—otherwise $p;e$ is undefined at y.

(2) Similar.

□

Proposition 32 1. $\mathbf{SEAM}_{\mathsf{partmin}}$ is an enriched-compact category.

2. $\mathbf{SE}_{\mathsf{strict}}$ is an enriched-compact category.

3. A product of enriched-compact categories is an enriched-compact category.

4. If $\mathcal{I}$ is a small **Cpo**-enriched category and $\mathcal{C}$ is an enriched-compact category, then the category $[\mathcal{I}, \mathcal{C}]$ of locally continuous functors is an enriched-compact category.

□

Proof

(1) The empty SEAM predomain 0 is initial and hence a zero object. Given a countable directed diagram D of (e,p)-pairs, we use Prop. 31(1) to obtain a diagram of countable multi-cusls Ad and injections $Ad \xrightarrow{i_{dd'}} Ad'$ and construct the colimit of this as a countable multi-cusl C with injections $Ad \xrightarrow{i_d} C$. (C is constructed as a quotient of the disjoint union of the posets $\{A_d\}_{d\in\mathbb{D}}$, just like colimits in **Set**.) We let V be the SEAM predomain corresponding to C. To prove (5.2), it is sufficient to prove it for compact elements of V.

If c is such a compact element (i.e. $c \in C$), then, by the construction of C, c is in the range of i_d for sufficiently large d. For such d, the note in the proof of Prop. 31(1) shows that $p_d; e_d$ takes a to a. Hence $\bigvee_{d\in\mathbb{D}}(p_d; e_d)$ takes a to a.

(2) The 1-element SE domain $\underline{1}$ is terminal and hence a zero object. The O-colimits are constructed as for (1).

(3) Trivial.

(4) Suppose D is a countable directed diagram in $[\mathfrak{I}, \mathcal{C}]^{\mathsf{ep}}$. For each $i \in \mathfrak{I}$, we obtain a diagram D_i in $\mathcal{C}^{\mathsf{ep}}$. We take the O-colimit of this; it has vertex Vi and, for each $d \in \mathbb{D}$, a morphism (e_{di}, p_{di}) from D_{di} to Vi. For any $\mathfrak{I}$-morphism $i \xrightarrow{\quad f \quad} j$ we define the $\mathcal{C}$-morphism $Vi \xrightarrow{\quad Vf \quad} Vj$ to be $\bigvee_{d\in\mathbb{D}}(p_{di}; D_{df}; e_{dj})$. It is easy to show that this makes (e_{di}, p_{di}) natural in i and that V is a locally continuous functor. We thus have a cocone; the O-colimit property is immediate from the O-colimit property at each i.

□

To interpret recursive types, we need to extend the type constructors to locally continuous functors between these categories.

Proposition 33

- U extends canonically to a locally continuous functor from $\mathbf{SE}_{\mathsf{strict}}$ to $\mathbf{SEAM}_{\mathsf{partmin}}$.
- $\sum_{i\in I}$ extends canonically to a locally continuous functor from $\mathbf{SEAM}^I_{\mathsf{partmin}}$ to $\mathbf{SEAM}_{\mathsf{partmin}}$.
- $\times$ extends canonically to a locally continuous functor from $\mathbf{SEAM}_{\mathsf{partmin}} \times \mathbf{SEAM}_{\mathsf{partmin}}$ to $\mathbf{SEAM}_{\mathsf{partmin}}$.
- F extends canonically to a locally continuous functor from $\mathbf{SEAM}_{\mathsf{partmin}}$ to $\mathbf{SE}_{\mathsf{strict}}$.
- $\prod_{i\in I}$ extends canonically to a locally continuous functor from $\mathbf{SE}^I_{\mathsf{strict}}$ to $\mathbf{SE}_{\mathsf{strict}}$.
- $\to$ extends canonically to a locally continuous functor from $\mathbf{SEAM}^{\mathsf{op}}_{\mathsf{partmin}} \times \mathbf{SE}_{\mathsf{strict}}$ to $\mathbf{SE}_{\mathsf{strict}}$.

□

Using Prop. 33 together with Prop. 32 we can interpret

- a value type A with m free value type identifiers and n free value type identifiers by a locally continuous functor

$$(\mathbf{SEAM}^{\mathsf{op}}_{\mathsf{partmin}} \times \mathbf{SEAM}_{\mathsf{partmin}})^m \times (\mathbf{SE}_{\mathsf{strict}}{}^{\mathsf{op}} \times \mathbf{SE}_{\mathsf{strict}})^n \longrightarrow \mathbf{SEAM}_{\mathsf{partmin}}$$

- a computation type $\underline{B}$ with m free value type identifiers and n free value type identifiers by a locally continuous functor

$$(\mathbf{SEAM}^{\mathsf{op}}_{\mathsf{partmin}} \times \mathbf{SEAM}_{\mathsf{partmin}})^m \times (\mathbf{SE}_{\mathsf{strict}}{}^{\mathsf{op}} \times \mathbf{SE}_{\mathsf{strict}})^n \longrightarrow \mathbf{SE}_{\mathsf{strict}}$$

In particular, a closed value type denotes a SEAM predomain and a closed computation type denotes a SE domain. We have isomorphisms

$$\begin{aligned} [\![\mu \mathtt{X}.A]\!] &\cong [\![A[\mu \mathtt{X}.A/\mathtt{X}]]\!] \\ [\![\mu \underline{\mathtt{X}}.\underline{B}]\!] &\cong [\![\underline{B}[\mu \underline{\mathtt{X}}.\underline{B}/\underline{\mathtt{X}}]]\!] \end{aligned}$$

We thus obtain a semantics for CBPV with recursive types. It is obviously sound. To prove adequacy, we use Pitts' methods [Pit96], which work for any enriched-compact category.

5.4 Infinitely Deep Terms

5.4.1 Syntax

We now want to allow the branches of a term's parse tree to be infinite branches, as in a Böhm tree. However, this is not always acceptable. To see this, consider the following infinitely deep terms:

1.
$$\dfrac{\dfrac{\dfrac{\dfrac{\vdots}{\vdash^{\mathsf{v}} \texttt{thunk force thunk} \ldots : U\underline{B}}}{\vdash^{\mathsf{c}} \texttt{force thunk force thunk} \ldots : \underline{B}}}{\vdash^{\mathsf{v}} \texttt{thunk force thunk force thunk} \ldots : U\underline{B}}}{\vdash^{\mathsf{c}} \texttt{force thunk force thunk force thunk} \ldots : \underline{B}}$$

   This is acceptable. It is a divergent computation and hence should denote $\bot$.

2.
$$\dfrac{\vdash^{\mathsf{v}} \texttt{true} : \texttt{bool} \qquad \dfrac{\dfrac{\vdots}{\texttt{x}_0 : \texttt{bool}, \texttt{x}_1 : \texttt{bool} \vdash^{\mathsf{c}} \texttt{let x}_2 \texttt{ be true}. \ldots : \underline{B}}}{\texttt{x}_0 : \texttt{bool} \vdash^{\mathsf{c}} \texttt{let x}_1 \texttt{ be true. let x}_2 \texttt{ be true}. \ldots : \underline{B}}}{\vdash^{\mathsf{c}} \texttt{let x}_0 \texttt{ be true. let x}_1 \texttt{ be true. let x}_2 \texttt{ be true}. \ldots : \underline{B}}$$

   This is acceptable. It is a divergent computation and hence should denote $\bot$.

3. This example uses the rules for complex values.
$$\dfrac{\vdash^{\mathsf{v}} \texttt{true} : \texttt{bool} \qquad \dfrac{\dfrac{\vdots}{\texttt{x}_0 : \texttt{bool}, \texttt{x}_1 : \texttt{bool} \vdash^{\mathsf{v}} \texttt{let x}_2 \texttt{ be true}. \ldots : \texttt{bool}}}{\texttt{x}_0 : \texttt{bool} \vdash^{\mathsf{v}} \texttt{let x}_1 \texttt{ be true. let x}_2 \texttt{ be true}. \ldots : \texttt{bool}}}{\vdash^{\mathsf{v}} \texttt{let x}_0 \texttt{ be true. let x}_1 \texttt{ be true. let x}_2 \texttt{ be true}. \ldots : \texttt{bool}}$$

   This is not acceptable. As a closed boolean value, it should denote either true or false—but it diverges.

4.
$$\dfrac{\vdash^{\mathsf{v}} \texttt{true} : \texttt{bool} \qquad \dfrac{\dfrac{\vdots}{\vdash^{\mathsf{v}} \texttt{fold}\,(\texttt{true}, \ldots) : \mu X.(\texttt{bool} \times X)}}{\vdash^{\mathsf{v}} \texttt{fold}\,(\texttt{true}, \texttt{fold}\,(\texttt{true}, \ldots)) : \mu X.(\texttt{bool} \times X)}}{\vdash^{\mathsf{v}} \texttt{fold}\,(\texttt{true}, \texttt{fold}\,(\texttt{true}, \texttt{fold}\,(\texttt{true}, \ldots))) : \mu X.(\texttt{bool} \times X)}$$

   This is not acceptable. It should denote an element of $[\![\mu \texttt{X}.(\texttt{bool} \times \texttt{X})]\!]$—but this is the empty cpo.

Each of these trees has a single infinite branch. We need a condition on branches that is satisfied by (1)–(2) but not by (3)–(4). The correct requirement is as follows.

Definition 28 An infinitely deep parse-tree for a term is *acceptable* when it has no branch that is, from some point upwards, an infinite branch consisting only of value-forming rules. □

We stress that it is quite acceptable for a branch to contain infinitely many value-forming rules—this happens in (1).

The branch condition in Def. 28 appears inelegant. A more abstract characterization using induction and coinduction is given in Sect. 5.6. Another approach using cpos is to define an infinitely deep term to be an ideal of finitely deep terms, using the partial order $\lhd$ which is the least compatible (i.e. preserved by all term constructors) relation on finitely deep terms such that $\mathtt{diverge} \lhd M$ for every M.

5.4.2 Scott Semantics

The cpo model for CBPV with recursion extends to a semantics for CBPV with acceptable infinitely deep terms. We need the following notion:

Definition 29 We say that $M \lhd_{\mathsf{fin}} N$ ("M is a finite approximant of N") when M is finitely deep and N is obtained by replacing every occurrence of $\mathtt{diverge}$ in M by a computation. More abstractly, we can define $\lhd_{\mathsf{fin}}$ to be the least binary relation between terms such that $\lhd_{\mathsf{fin}}$ is compatible (i.e. closed under all term constructors) and $\mathtt{diverge} \lhd_{\mathsf{fin}} N$ for all N. □

For every finitely deep term M we define its interpretation $[\![M]\!]^{\mathsf{fin}}$ in the usual way. Then, for every term M, we define its interpretation $[\![M]\!]^{\mathsf{inf}}$ to be $\bigsqcup_{M \lhd_{\mathsf{fin}} N} [\![M]\!]$. It is clear that the set $\{[\![M]\!] : M \lhd_{\mathsf{fin}} N\}$ is directed—the condition on branches ensures its nonemptiness—so the join must exist. Furthermore, if M is finitely deep then $[\![M]\!]^{\mathsf{inf}} = [\![M]\!]^{\mathsf{fin}}$, so $[\![-]\!]^{\mathsf{inf}}$ is an extension of $[\![-]\!]^{\mathsf{fin}}$.

Finally, we need to adapt the adequacy proof in Sect. 3.8 to include infinitely deep terms. We define the various relations and prove their admissibility just as we did there. We then prove that for any computation $A_0, \dots, A_{n-1} \vdash^{\mathsf{c}} M \lhd_{\mathsf{fin}} N : \underline{B}$, if $a_i \leqslant^{\mathsf{v}}_{A_i} W_i$ for $i = 0, \dots, n-1$ then $[\![M]\!]\overrightarrow{\mathtt{x}_i \mapsto a_i} \leqslant^{\mathsf{c}}_{\underline{B}} N[\overrightarrow{W_i/\mathtt{x}_i}]$; and similarly for any values $A_0, \dots, A_{n-1} \vdash^{\mathsf{v}} U \lhd_{\mathsf{fin}} V : A$. This is shown by mutual induction on M and U. By admissibility, we see that $[\![M]\!] \leqslant^{\mathsf{c}}_{\underline{B}} M$ for any closed computation M, and this gives the desired result.

5.5 Infinitely Deep Types

5.5.1 Syntax

Having extended CBPV with infinitely deep terms, we proceed to allow a type's parse-tree to have infinite branches. Fortunately here there is no need for restrictions on branches: any parse tree with finite or infinite branches is acceptable.

For example, we have a value type

$$\mathtt{bool} \times (\mathtt{bool} \times (\mathtt{bool} \times \cdots))$$

This is the unwinding of $\mu \mathtt{X}.(\mathtt{bool} \times \mathtt{X})$. There is no closed value of this type because a putative term such as $(\mathtt{true}, (\mathtt{true}, (\mathtt{true}, \dots)))$ would violate the branch restriction on terms.

5.5.2 Scott Semantics

Just as in Sect. 5.4 we extended the semantics for recursion to a semantics for infinitely deep terms, we can extend the cpo semantics for type recursion to a semantics for infinitely deep types. We say that $A \lhd_{\mathsf{fin}} B$ when A is a finitely deep type and B is obtained from A by replacing some occurrences of 0 with a value type and some occurrences of 1_Π (the computation type which is the product of the empty family) with a computation type. We first define $[\![-]\!]^{\mathsf{fin}}$, the

semantics of finitely deep types. Then we define $[\![B]\!]^{\mathsf{inf}}$ is defined to be the O-colimit of $[\![A]\!]^{\mathsf{fin}}$ over all $A \lhd_{\mathsf{fin}} B$, using the fact that the set of finite approximants of B is directed and countable. Soundness is trivial and adequacy is proved as for recursive types.

We obtain isomorphisms such as

$$\begin{array}{rcll} [\![A]\!]^{\mathsf{inf}} & \cong & [\![A]\!]^{\mathsf{fin}} & (A \text{ finitely deep}) \\ [\![\underline{B}]\!]^{\mathsf{inf}} & \cong & [\![\underline{B}]\!]^{\mathsf{fin}} & (\underline{B} \text{ finitely deep}) \\ [\![A \times A']\!]^{\mathsf{inf}} & \cong & [\![A]\!]^{\mathsf{inf}} \times [\![A']\!]^{\mathsf{inf}} & \\ [\![U\underline{B}]\!]^{\mathsf{inf}} & \cong & U[\![\underline{B}]\!]^{\mathsf{inf}} & \end{array}$$

It would be desirable for these isomorphisms to be identities, but in the cpo model they are not. If it were possible to make them into identities, then it would also be possible to make the isomorphisms

$$\begin{array}{rcl} [\![\mu \mathtt{X}.A]\!]^{\mathsf{inf}} & \cong & [\![A[\mu \mathtt{X}.A/\mathtt{X}]]\!]^{\mathsf{inf}} \\ [\![\mu \underline{X}.\underline{B}]\!]^{\mathsf{fin}} & \cong & [\![\underline{B}[\mu \underline{X}.\underline{B}/\mathtt{x}]]\!]^{\mathsf{inf}} \end{array}$$

into identities, and this is clearly not possible in the cpo model because of the Axiom of Foundation. By contrast, in models such as information systems [Sco82], precusls [SHLG94], games [McC96] and non-well-founded cpos (replacing the Axiom of Foundation by the Anti-Foundation Axiom [Acz88]) all these isomorphisms can be made into identities—we omit details.

5.6 The Inductive/Coinductive Formulation of Infinitely Deep Syntax

The branch condition on infinitely deep terms appears inelegant. In this section (which the reader may omit as it is not used in the remainder of the thesis) we give a more abstract characterization of infinitely deep syntax. We write

- $\mathsf{valtypes}$ for the set of value types;
- $\mathsf{comptypes}$ for the set of computation types;
- $\mathsf{contexts}$ for the set of contexts;
- $\mathsf{valterms}$ for the object in the category $\mathbf{Set}^{\mathsf{contexts} \times \mathsf{valtypes}}$ such that $\mathsf{valterms}_{\Gamma,A}$ is the set of values $\Gamma \vdash^{\mathsf{v}} V : A$;
- $\mathsf{compterms}$ for the object in the category $\mathbf{Set}^{\mathsf{contexts} \times \mathsf{comptypes}}$ such that $\mathsf{compterms}_{\Gamma,\underline{B}}$ is the set of computations $\Gamma \vdash^{\mathsf{c}} M : \underline{B}$.

We discuss only the case of countable tag sets, but an analogous discussion applies to finite tag sets.

To see the issues, consider first the definition of (infinitely wide) CBPV types:

$$\begin{array}{rcl} A & ::= & U\underline{B} \mid \sum_{i \in I} A_i \mid 1 \mid A \times A \\ \underline{B} & ::= & FA \mid \prod_{i \in I} \underline{B}_i \mid A \to \underline{B} \end{array}$$

where I must be countable. This defines an endofunctor on $\mathbf{Set} \times \mathbf{Set}$ given by:

$$(X,Y) \mapsto (Y + \textstyle\sum_{I \text{ countable}} X^I + 1 + X^2, X + \sum_{I \text{ countable}} Y^I + XY)$$

The reader may worry that taking a sum over all countable sets is problematic. But there are two straightforward solutions.

- We can take the sum over all countable *small* sets, relative to a Grothendieck universe.

- Less drastically, we can take the sum over all initial segments of $\mathbb{N}$. This requires us to fix bijections that allow us to regard the set of such segments as closed under finite product and countable sum (indexed over an initial segment of $\mathbb{N}$).

Thus size is not a serious problem, and we will not address it further.

For finitely deep syntax, the definition of types is inductive i.e. the pair $(\mathsf{valtypes}, \mathsf{comptypes})$ is the carrier of an initial algebra for this endofunctor. When we allow infinitely deep types, the definition of types becomes *coinductive* i.e. $(\mathsf{valtypes}, \mathsf{comptypes})$ is the carrier of a terminal coalgebra for this endofunctor.

The definition of CBPV terms given in Fig. 3.1 similarly describes an endofunctor[3] on $\mathbf{Set}^{\mathsf{contexts}\times\mathsf{valtypes}} \times \mathbf{Set}^{\mathsf{contexts}\times\mathsf{comptypes}}$. It must be of the form (F, G) where

- $F : \mathbf{Set}^{\mathsf{contexts}\times\mathsf{valtypes}} \times \mathbf{Set}^{\mathsf{contexts}\times\mathsf{comptypes}} \longrightarrow \mathbf{Set}^{\mathsf{contexts}\times\mathsf{valtypes}}$
- $G : \mathbf{Set}^{\mathsf{contexts}\times\mathsf{valtypes}} \times \mathbf{Set}^{\mathsf{contexts}\times\mathsf{comptypes}} \longrightarrow \mathbf{Set}^{\mathsf{contexts}\times\mathsf{comptypes}}$

but we will not write F and G explicitly.

We see that $(\mathsf{valterms}, \mathsf{compterms})$ is the carrier of an initial algebra for this endofunctor. But when we allow infinitely deep terms, it is not the case that $(\mathsf{valterms}, \mathsf{compterms})$ is the carrier of a terminal coalgebra for this endofunctor, because of the branch condition. However, $(\mathsf{valterms}, \mathsf{compterms})$ is certainly a fixpoint of the endofunctor in the sense that

$$\begin{aligned} \mathsf{valterms} &\cong F(\mathsf{valterms}, \mathsf{compterms}) \\ \mathsf{compterms} &\cong G(\mathsf{valterms}, \mathsf{compterms}) \end{aligned}$$

How can we characterize this fixpoint? Clearly we require some kind of mixed inductive/coinductive definition: values are inductively defined, computations are coinductively defined. But the coinductive part must be (loosely speaking) on the outside, because otherwise we will exclude infinite branches that contain infinitely many value-forming rules, and we have already seen that this is too restrictive.

To express this, we write $\mu\mathtt{X}$ for initial algebra and $\nu\mathtt{X}$ for terminal coalgebra.

Proposition 34 We have the following description of infinitely deep terms:

$$\begin{aligned} \mathsf{compterms} &\cong \nu\mathtt{Y}.G(\mu\mathtt{X}.F(\mathtt{X},\mathtt{Y}),\mathtt{Y}) \\ \mathsf{valterms} &\cong \mu\mathtt{X}.F(\mathtt{X}, \mathsf{compterms}) \end{aligned}$$

□

This is proved by giving an explicit representation of the function

$$\mathtt{Y} \mapsto \mu\mathtt{X}F(\mathtt{X},\mathtt{Y})$$

We omit details.

5.7 Relationship Between Recursion and Infinitely Deep Syntax

We said in Sect. 5.1 that recursion and infinitely deep syntax are, in a sense, equivalent. We now explain this sense, in a very informal way, using examples.

Any recursion can be unwound infinitely often. For example, if we take the recursive type $\mu\mathtt{X}.(1+\mathtt{X})$, this can be unwound to give $1+\mu\mathtt{X}.(1+\mathtt{X})$, then unwound again to give $1+(1+\mu\mathtt{x}.(1+\mathtt{X}))$, and so forth. Ultimately, we obtain the infinitely deep type $1+(1+(1+\cdots))$. Thus if we have a semantics for infinitely deep syntax, it provides a semantics for recursion too.

[3] We gloss over issues of identifier binding. See [FPD99] for a fuller discussion.

On the other hand, if we have an infinitely deep term or type, it can be expressed using a countable collection of simultaneously recursive definitions. For example, take the infinitely deep type $A = 0 + (1 + (2 + (\cdots)))$. We name the subexpressions of this as follows:

$$\begin{aligned} A_0 &= 0 + (1 + (2 + \cdots)) \\ A_1 &= 1 + (2 + (3 + \cdots)) \\ A_2 &= 2 + (3 + (4 + \cdots)) \end{aligned}$$

Now we can define these types by mutual recursion:

$$\begin{aligned} A_0 &= 0 + A_1 \\ A_1 &= 1 + A_2 \\ A_2 &= 2 + A_3 \end{aligned}$$

Thus any semantics for recursion that can interpret countable simultaneous recursions gives us a semantics for infinitely deep syntax.

Sometimes it is easier to work with recursion, sometimes with infinitely deep syntax. One advantage of the former is that, by allowing only finitely deep syntax, we have a clear notion of *compositional semantics*. Finitely deep syntax is an initial algebra for an endofunctor, and a compositional semantics is one specified by another algebra for this endofunctor—the interpretation is then given by the unique algebra homomorphism from the syntax to this algebra [GTW79]. By contrast, it is not clear in what sense the semantics for infinitely deep CBPV that we have considered can be said to be compositional.

Part II

Concrete Semantics

Chapter 6

Simple Models Of CBPV

6.1 Introduction

The goal of Part II is to advance the following claim.

> Whenever we are studying effectful higher-order languages (and remember that mere divergence makes a language "effectful"), the semantic primitives are given by CBPV. It is therefore a good choice for our language of study.

While this is certainly a radical claim, we assemble, throughout Part II, extensive evidence for it. In a wide range of fields, we see firstly that CBPV semantics is simpler than the traditional CBV and CBN semantics, and secondly that these traditional semantics can be recovered from the CBPV semantics—so we do not lose out by shifting our focus to CBPV.

Admittedly, we encounter exceptions, where a CBV model does not decompose naturally into CBPV: These exceptions are

- the model for erratic choice with the constraint that choice must be finite (Sect. 6.5.3);
- the possible worlds model for cell generation with the constraint of "parametricity in initializations" (Sect. 7.9).

Each of these is a constrained variant of a simpler CBV model that *does* decompose into a CBPV model, so we do not consider them to be a major objection to our claim. A rather different case is the CBV model for input based on Moggi's input monad [Mog91].

We have already seen the decomposition into CBPV for printing and divergence, as well as for operational semantics, in Chap. 3. In this chapter we look at global store, control effects, erratic choice and errors, and at various combinations of these effects. This range of "simple models" is based on [Mog91], although we do not treat Moggi's example of "interactive input". We omit also the extremely subtle combination of erratic choice and divergence.

We first look at semantics of values, in Sect. 6.2, as this is common to all the models in the chapter. Then we devote one section to each effect; these sections can be read independently of each other.

As the semantics of values is straightforward, the difficulty lies in the semantics of computations. To invent it, there are two useful heuristics that can be applied. One is to take a known CBV (or CBN) model and look for a decomposition.

- For example, consider the traditional global store interpretation of $A \to_{\mathbf{CBV}} B$ viz. $S \to (\llbracket A \rrbracket \to (S \times \llbracket B \rrbracket))$. We immediately see the decomposition into $U(A \to FB)$; it appears that U will denote $S \to -$, that $\to$ will denote $\to$, and that F will denote $S \times -$. Thus the CBPV type constructors have simpler interpretations than $\to_{\mathbf{CBV}}$.

- As another example, consider the traditional continuation semantics for $A \to_{\mathbf{CBV}} B$ viz. $(\llbracket A \rrbracket \times (\llbracket B \rrbracket \to \mathsf{Ans})) \to \mathsf{Ans}$. We immediately see the decomposition into $U(A \to FB)$; it appears that U and F will both denote $- \to \mathsf{Ans}$, and $\to$ will denote $\times$. Again the CBPV type constructors have simpler interpretations than $\to_{\mathbf{CBV}}$.
- For both of these examples, there are corresponding CBN semantics presented in the literature: $\underline{A} \to_{\mathbf{CBN}} \underline{B}$ is interpreted in [O'H93] as $(S \to \llbracket \underline{A} \rrbracket) \to \llbracket \underline{B} \rrbracket$ (for global store) and in [SR98] as $(\llbracket \underline{A} \rrbracket \to \mathsf{Ans}) \times \llbracket \underline{B} \rrbracket$. Each of these makes apparent the decomposition of $\underline{A} \to_{\mathbf{CBN}} \underline{B}$ into $(U\,\underline{A}) \to \underline{B}$. Again, the semantics of the CBPV type constructors are simpler. In fact the CBN semantics looks quite strange; CBPV thus provides a rational reconstruction for it.

The other heuristic, which we shall use in this chapter, is to guess in advance the form of the soundness theorem and proceed from there, using the reversible derivations of Sect. 4.5.

6.2 Semantics of Values

We describe the semantics of values at the outset, because it is straightforward and it is the same across the different models in the chapter. We will consider models using sets and models using cpos. In all of the set models,

- a value type denotes a set;
- in particular, $\sum_{i \in I} A_i$ denotes the set $\sum_{i \in I} \llbracket A_i \rrbracket$ and $A \times A'$ denotes the set $\llbracket A \rrbracket \times \llbracket A' \rrbracket$;
- a context $A_0, \ldots, A_{n-1}$ denotes the set $\llbracket A_0 \rrbracket \times \cdots \times \llbracket A_{n-1} \rrbracket$;
- a value $\Gamma \vdash^{\mathsf{v}} V : A$ denotes a function from $\llbracket \Gamma \rrbracket$ to $\llbracket A \rrbracket$.

Similarly, in the cpo models,

- a value type denotes a cpo;
- in particular, $\sum_{i \in I} A_i$ denotes the cpo $\sum_{i \in I} \llbracket A_i \rrbracket$ and $A \times A'$ denotes the cpo $\llbracket A \rrbracket \times \llbracket A' \rrbracket$;
- a context $A_0, \ldots, A_{n-1}$ denotes the cpo $\llbracket A_0 \rrbracket \times \cdots \times \llbracket A_{n-1} \rrbracket$;
- a value $\Gamma \vdash^{\mathsf{v}} V : A$ denotes a continuous function from $\llbracket \Gamma \rrbracket$ to $\llbracket A \rrbracket$.

6.3 Global Store

6.3.1 The Language

We take the simplest possible case of global store: we suppose there is just one cell `cell` that stores a value of ground type $\sum_{i \in S} 1$, written S for short.

We thus add to the basic CBPV language constructs for assignment and reading:

$$\frac{\Gamma \vdash^{\mathsf{c}} M : \underline{B}}{\Gamma \vdash^{\mathsf{c}} \mathtt{cell} := s;\ M : \underline{B}}\ (s \in S) \qquad \frac{\Gamma, \mathtt{x} : S \vdash^{\mathsf{c}} M : \underline{B}}{\Gamma \vdash^{\mathsf{c}} \mathtt{read\ cell\ as\ x}.\ M : \underline{B}}$$

(We continue our practice of treating commands as prefixes.)

For big-step semantics, we define a relation of the form $s, M \Downarrow s', T$, where $s, s' \in S$. We often write s as $\mathsf{cell} \mapsto V$, meaning "cell contains the value V". To define $\Downarrow$, we replace each rule in Fig. 3.2 of the form (3.1) by

$$\frac{s_0, M_0 \Downarrow s_1, T_0 \quad \cdots \quad s_{r-1}, M_{r-1} \Downarrow s_r, T_{r-1}}{s_0, M \Downarrow s_r, T}$$

and add the rules

$$\frac{(\mathsf{cell} \mapsto W), M \Downarrow s, T}{(\mathsf{cell} \mapsto V), (\mathtt{cell} := W;\ M) \Downarrow s, T} \qquad \frac{(\mathsf{cell} \mapsto V), M[V/\mathtt{x}] \Downarrow s, T}{(\mathsf{cell} \mapsto V), (\mathtt{read\ cell\ as\ x}.\ M) \Downarrow s, T}$$

Proposition 35 For every $s \in S$ and closed computation M, there exists unique s', T such that $s, M \Downarrow s', T$. □

This is proved similarly to Prop. 9. We can also adapt the CK-machine to this computational effect, but we omit this.

We adapt the definition of observational equivalence.

Definition 30 Given two computations $\Gamma \vdash^{\mathsf{c}} M, N : \underline{B}$, we say $M \simeq N$ when for any ground context $\mathcal{C}[]$ and any $s \in S$, we have

$$s, \mathcal{C}[M] \Downarrow s', \mathtt{produce}\ n \text{ iff } s, \mathcal{C}[N] \Downarrow s', \mathtt{produce}\ n$$

Similarly for two values $\Gamma \vdash^{\mathsf{v}} V, W : A$. □

6.3.2 Denotational Semantics

We seek a denotational semantics for the language described in Sect. 6.3.1. Since there is no divergence, we use a set model. The semantics of values is given in Sect. 6.2—the difficulty lies in the interpretation of computations.

While logically we should present the semantics first, and then state the soundness theorem, this makes the interpretation of type constructors appear unintuitive. So we will proceed in reverse order. We will state first the soundness theorem that we are aiming to achieve, even though it is not yet meaningful, and use this to motivate the semantics.

We expect the soundness result to look like this:

Proposition 36 (soundness) For every closed computation M, if $s, M \Downarrow s', T$ then $[\![M]\!]s = [\![T]\!]s'$. □

If we are using non-closed computations, as in Sect. 3.3.5, then the soundness result will look like this:

Proposition 37 For every computation $\Gamma \vdash^{\mathsf{c}} M : \underline{B}$ and every environment $\rho \in [\![\Gamma]\!]$, if $s, M \Downarrow s', T$ then $[\![M]\!](s, \rho) = [\![T]\!](s', \rho)$ □

In Prop. 37, the meaning of a computation $\Gamma \vdash^{\mathsf{c}} M : \underline{B}$ takes both store $s \in S$ and environment $\rho \in [\![\Gamma]\!]$ as arguments. For this to be meaningful, M must denote a function from $S \times [\![\Gamma]\!]$ to some set—we call this set $[\![\underline{B}]\!]$.

Thus a computation type will denote a set. We must next decide how to interpret type constructors.

To find the interpretations for $U, \prod, \to$, we use the reversible derivations of Prop. 23:

U Since we have the reversible derivation

$$\frac{\Gamma \vdash^{\mathsf{c}} \underline{B}}{\Gamma \vdash^{\mathsf{v}} U\underline{B}}$$

we see that a function from $[\![\Gamma]\!]$ to $[\![U\underline{B}]\!]$ should correspond to a function from $S \times [\![\Gamma]\!]$ to $[\![\underline{B}]\!]$. This suggests that we set $[\![U\underline{B}]\!]$ to be $S \to [\![\underline{B}]\!]$.

$\prod$ Since we have the reversible derivation

$$\frac{\cdots \quad \Gamma \vdash^{\mathsf{c}} \underline{B}_i \quad \cdots}{\Gamma \vdash^{\mathsf{c}} \prod_{i \in I} \underline{B}_i}$$

we see that a function from $S \times [\![\Gamma]\!]$ to $[\![\prod_{i \in I} \underline{B}_i]\!]$ should correspond to a family of functions in which the ith function is from $S \times [\![\Gamma]\!]$ to $[\![\underline{B}_i]\!]$. This suggests that we set $[\![\prod_{i \in I} \underline{B}_i]\!]$ to be $\prod_{i \in I} [\![\underline{B}_i]\!]$.

$\to$ Since we have the reversible derivation

$$\frac{\Gamma, A \vdash^{\mathsf{c}} \underline{B}}{\Gamma \vdash^{\mathsf{c}} A \to \underline{B}}$$

we see that a function from $S \times [\![\Gamma]\!]$ to $[\![A \to \underline{B}]\!]$ should correspond to a function from $S \times ([\![\Gamma]\!] \times [\![A]\!])$ to $[\![\underline{B}]\!]$. This suggests that we set $[\![A \to \underline{B}]\!]$ to be $[\![A]\!] \to [\![\underline{B}]\!]$.

For F, there is no reversible derivation. But it is clear that a producer $\Gamma \vdash^{\mathsf{c}} M : FA$ should denote a function from $S \times [\![\Gamma]\!]$ to $S \times [\![A]\!]$, because if we run M in a particular state s and environment ρ we obtain a store s'—which is observable—and a value of type A. Thus a function from $S \times [\![\Gamma]\!]$ to $[\![FA]\!]$ corresponds to a function from $S \times [\![\Gamma]\!]$ to $S \times [\![A]\!]$. This suggests that we set $[\![FA]\!]$ to be $S \times [\![A]\!]$.

The semantics of terms is straightforward. Here are some example clauses:

$$\begin{aligned}
[\![\mathtt{produce}\ V]\!](s,\rho) &= (s, [\![V]\!]\rho) \\
[\![M\ \mathtt{to}\ \mathtt{x}.\ N]\!](s,\rho) &= [\![N]\!](s', (\rho, \mathtt{x} \mapsto a)) \text{ where } [\![M]\!](s,\rho) = (s', a) \\
[\![\mathtt{thunk}\ M]\!]\rho &= \lambda s.([\![M]\!](s,\rho)) \\
[\![\mathtt{force}\ V]\!](s,\rho) &= s`([\![V]\!]\rho) \\
[\![\lambda \mathtt{x}.M]\!](s,\rho) &= \lambda x.([\![M]\!](s, (\rho, \mathtt{x} \mapsto x))) \\
[\![\mathtt{cell} := V; M]\!](\mathsf{cell} \mapsto i, \rho) &= [\![M]\!](\mathsf{cell} \mapsto [\![V]\!]\rho, \rho) \\
[\![\mathtt{read}\ \mathtt{cell}\ \mathtt{as}\ \mathtt{x}.\ M]\!](\mathsf{cell} \mapsto i, \rho) &= [\![M]\!](\mathsf{cell} \mapsto i, (\rho, \mathtt{x} \mapsto i))
\end{aligned}$$

It is easy to prove Prop. 36.

Corollary 38 (by Prop. 35) If M is a ground producer, then $s, M \Downarrow s', \mathtt{produce}\ n$ iff $[\![M]\!]s = (s', n)$. Hence terms with the same denotation are observationally equivalent. □

It is also easy to show that provably equal terms have the same denotation i.e. the model validates the CBPV equations.

Notice the semantics of CBV functions:

$$[\![A \to_{\mathbf{CBV}} B]\!] = [\![U(A \to FB)]\!] = S \to ([\![A]\!] \to (S \times [\![B]\!]))$$

As we said in Sect. 6.1, this is the traditional CBV semantics for global store.

6.3.3 Combining Global Store With Other Effects

The model for global store in Sect. 6.3.2 generalizes. If we have any CBPV model, we can obtain from it a model for global store.

As an example, consider the printing model for CBPV. We seek a model for global store together with `print`. The big-step semantics will have the form $s, M \Downarrow m, s', T$—we omit the details, which are straightforward.

The denotational semantics for global store with `print` is organized as follows:

- a value type (and hence a context) denotes a set;
- a computation type denotes an $\mathcal{A}$-set;
- a value $\Gamma \vdash^{\mathsf{v}} V : A$ denotes a function from $[\![\Gamma]\!]$ to $[\![A]\!]$;
- a computation $\Gamma \vdash^{\mathsf{c}} M : \underline{B}$ denotes a function from $S \times [\![\Gamma]\!]$ to $[\![\underline{B}]\!]$.

We use CBPV as a metalanguage describing the printing model, as explained in Sect. 3.8. For example,we write FA for the free $\mathcal{A}$-set on the set A, and we write $U\underline{B}$ for the carrier of the $\mathcal{A}$-set $\underline{B}$. With this notation, the semantics of types is given by

$$\begin{array}{rclcrcl} [\![U\underline{B}]\!] & = & U(S \to [\![\underline{B}]\!]) & & [\![FA]\!] & = & F(S \times [\![A]\!]) \\ [\![\Sigma_{i\in I} A_i]\!] & = & \Sigma_{i\in I}[\![A_i]\!] & & [\![\Pi_{i\in I}\underline{B}_i]\!] & = & \Pi_{i\in I}[\![\underline{B}_i]\!] \\ [\![A \times A']\!] & = & [\![A]\!] \times [\![A']\!] & & [\![A \to B]\!] & = & [\![A]\!] \to [\![B]\!] \end{array}$$

Semantics of terms: some example clauses

$$\begin{array}{rcl} [\![\mathtt{produce}\, V]\!](s,\rho) & = & \mathsf{produce}\,(s, [\![V]\!]\rho) \\ [\![M \,\mathtt{to}\, \mathtt{x}.\, N]\!](s,\rho) & = & [\![M]\!](s,\rho)\,\mathsf{to}\,(s',a).\, [\![N]\!](s', (\rho, \mathtt{x} \mapsto a)) \\ [\![\mathtt{thunk}\, M]\!]\rho & = & \mathsf{thunk}\,\lambda s.([\![M]\!](s,\rho)) \\ [\![\mathtt{force}\, V]\!](s,\rho) & = & s\text{`}\mathsf{force}\,([\![V]\!]\rho) \\ [\![\lambda \mathtt{x}.M]\!](s,\rho) & = & \lambda x.([\![M]\!](s, (\rho, \mathtt{x} \mapsto x))) \\ [\![\mathtt{cell} := V; M]\!](\mathsf{cell} \mapsto i, \rho) & = & [\![M]\!](\mathtt{cell} \mapsto [\![V]\!]\rho, \rho) \\ [\![\mathtt{read}\, \mathtt{cell}\, \mathtt{as}\, \mathtt{x}.\, M]\!](\mathsf{cell} \mapsto i, \rho) & = & [\![M]\!](\mathsf{cell} \mapsto i, \rho, \mathtt{x} \mapsto i) \\ [\![\mathtt{print}\, c;\, M]\!](s,\rho) & = & c * ([\![M]\!](s,\rho)) \end{array}$$

Proposition 39 (soundness) If $s, M \Downarrow m, s', T$ then $[\![M]\!]s = m * ([\![T]\!]s)$. □

Corollary 40 (by the analogue of Prop. 35) If M is a ground producer, then $s, M \Downarrow m, s', \mathtt{produce}\, n$ iff $[\![M]\!]s = (m, s', n)$. Hence terms with the same denotation are observationally equivalent. □

It is also easy to show that provably equal terms have the same denotation i.e. the model validates the CBPV equations.

In a similar way, we can obtain a model for global store with recursion. The semantics of types and terms are as above except that we now understand the metalinguistic U,F etc. as referring to the Scott model rather than the printing model.

6.4 Control Effects

6.4.1 `letcos` and `changecos`

For our explanation of control effects, we will use the CK-machine only, not big-step semantics. The way that big-step semantics is structured makes it unsuitable for a language with control effects.

We explain the basic idea of control effects before discussing typing rules. We add to CBPV[1] two commands `letcos x` and `changecos` K.

[1]The CBV control operators (as in ML) are translated into CBPV as follows:

$$\begin{array}{rcl} \mathtt{cont}\, A & \text{as} & \mathtt{os}\, FA \\ \mathtt{letcc}\, \mathtt{x}.\, M & \text{as} & \mathtt{letcos}\, \mathtt{x}.\, M \\ \mathtt{throw}\, M\, N & \text{as} & M\, \mathtt{to}\, \mathtt{x}.\, N\, \mathtt{to}\, \mathtt{y}.\, (\mathtt{changecos}\, \mathtt{x};\, \mathtt{produce}\, \mathtt{y}) \end{array}$$

We have to use the terminology "current outside" rather than "current continuation" (the phrase used in CBV) because in CBPV not every outside is a continuation.

- `letcos x` means "let `x` be the current outside".
- `changecos` K means "change the current outside to K".

Thus we have two additional machine transitions

$$\begin{array}{lll} & \texttt{letcos x}.\ M & K \\ \rightsquigarrow & M[K/\texttt{x}] & K \\ \\ & \texttt{changecos}\ K;\ M & L \\ \rightsquigarrow & M & K \end{array}$$

We illustrate these constructs with an example program:

```
let y be thunk(
           λx.
           changecos x;
           λz.
           produce 3+z
         ).
(7‘
 print "hello";
 letcos a.
 ((λ u.
    true‘
    a‘
    force y
  ) to v in
 produce v+2
 )
) to w.
produce w+5
```

- By the time we reach the line `letcos a`, we have printed `hello` and the current outside consists of an operand 7 together with the `to w` consumer (i.e. the consumer that begins on the line `to w in`), so `a` is bound to this outside.
- By the time we force `y` the current consists of operands `a` and `true` and the `to v` consumer.
- When we force `y`, we pop the top operand, which is `a`, and bind `x` to it. We now change the current outside to `a`.
- We pop the top operand, which is 7 (again) and bind `z` to it.
- We produce 10 to the current consumer which is the `to w` consumer. Hence we produce 15.

Notice how the stack discipline has been completely lost; in the absence of the control effects we would expect `force y` to produce a value to the `to v` consumer, but we have used `changecos` to override this.

When working with control effects, we will

1. use the CK-machine on non-closed configurations
2. regard `nil` as a free identifier (whose type will be given in the next section).

The advantage of (2) is that it makes for a smoother treatment of types and denotational semantics. The advantage of (1) is that it makes (2) possible.

The list of terminal configurations is then as follows:

$\mathtt{produce}\, V$	$\mathtt{z}$
$\lambda\{\ldots, i.M_i, \ldots\}$	$\mathtt{z}$
$\lambda \mathtt{x}.M$	$\mathtt{z}$
$\mathtt{force}\, \mathtt{z}$	K
$\mathtt{pm}\, \mathtt{z}\, \mathtt{as}\, \{\ldots, i.M_i, \ldots\}$	K
$\mathtt{pm}\, \mathtt{z}\, \mathtt{as}\, (\mathtt{x},\mathtt{y}).M$	K

Here $\mathtt{z}$ can be any free identifier, including $\mathtt{nil}$.

6.4.2 Typing

In Chap. 3 outsides were regarded as a separate syntactic category, but now they will be values. For every computation type $\underline{B}$, we introduce a value type $\mathtt{os}\ \underline{B}$; an outside that accompanies insides of type $\underline{B}$ is deemed to be a value of type $\mathtt{os}\ \underline{B}$.

Thus the two classes of types are now given by

$$\begin{array}{rcl} A & ::= & U\underline{B} \mid \sum_{i\in I} A_i \mid 1 \mid A \times A \mid \mathtt{os}\ \underline{B} \\ \underline{B} & ::= & FA \mid \prod_{i\in I} \underline{B}_i \mid A \to \underline{B} \end{array}$$

and we add the rules for $\mathtt{letcos}$ and $\mathtt{changecos}$:

$$\frac{\Gamma, \mathtt{x} : \mathtt{os}\ \underline{B} \vdash^{\mathsf{c}} M : \underline{B}}{\Gamma \vdash^{\mathsf{c}} \mathtt{letcos}\ \mathtt{x}.\ M : \underline{B}} \qquad \frac{\Gamma \vdash^{\mathsf{v}} K : \mathtt{os}\ \underline{B} \quad \Gamma \vdash^{\mathsf{c}} M : \underline{B}}{\Gamma \vdash^{\mathsf{c}} \mathtt{changecos}\ K;\ M : \underline{B}'}$$

Notice that $\mathtt{changecos}\ K;\ M$ can be given any type, like $\mathtt{diverge}$. It is a *non-returning command* in the sense of Sect. 3.9.2.

It should be clear that we no longer need the judgement $\Gamma \vdash^{\mathsf{k}}_{\underline{C}} K : \underline{B}$, because we can now write this as $\Gamma, \mathtt{nil} : \mathtt{os}\ \underline{C} \vdash^{\mathsf{v}} K : \mathtt{os}\ \underline{B}$. We replace the typing rules in Fig. 3.4—and the rule in Sect. 3.9.2 for the dummy outside $\mathtt{neverused}$—by the following:

$$\frac{\Gamma, \mathtt{x} : A \vdash^{\mathsf{c}} M : \underline{B} \quad \Gamma \vdash^{\mathsf{v}} K : \mathtt{os}\ \underline{B}}{\Gamma \vdash^{\mathsf{v}} [\,]\ \mathtt{to}\ \mathtt{x}.\ M :: K : \mathtt{os}\ FA} \qquad \frac{}{\Gamma \vdash^{\mathsf{v}} \mathtt{neverused} : \mathtt{os}\ \underline{\mathtt{nrcomm}}}$$

$$\frac{\Gamma \vdash^{\mathsf{v}} K : \mathtt{os}\ \underline{B}_{\hat{\imath}}}{\Gamma \vdash^{\mathsf{v}} \hat{\imath} :: K : \mathtt{os} \prod_{i\in I} \underline{B}_i} \qquad \frac{\Gamma \vdash^{\mathsf{v}} V : A \quad \Gamma \vdash^{\mathsf{v}} K : \mathtt{os}\ \underline{B}}{\Gamma \vdash^{\mathsf{v}} V :: K : \mathtt{os}\ (A \to \underline{B})}$$

Def. 15 is replaced by

Definition 31 A Γ-*configuration* consists of

- a computation type $\underline{B}$;
- a pair M, K where $\Gamma \vdash^{\mathsf{c}} M : \underline{B}$ and $\Gamma \vdash^{\mathsf{v}} K : \mathtt{os}\ \underline{B}$.

□

Thus a Γ-configuration of type $\underline{B}$ in the sense of Def. 15 is a $(\Gamma, \mathtt{nil} : \mathtt{os}\ \underline{B})$-configuration in the sense of Def. 31.

All the results of Sect. 3.3.3 adapt. In particular we have

Proposition 41 (deterministic subject reduction) (cf. Prop. 10) For every Γ-configuration M, K, precisely one of the following holds.

1. M, K is not terminal, and $M, K \rightsquigarrow N, L$ for unique N, L. N, L is a Γ-configuration.
2. M, K is terminal, and there does not exist N, L such that $M, K \rightsquigarrow N, L$.

□

Proposition 42 (cf. Prop. 11) For every Γ configuration M, K there is a unique terminal Γ-configuration N, L such that $M, K \rightsquigarrow^* N, L$, and there is no infinite sequence of transitions from M, K. □

We defer the proof of this to Sect. 8.6.

6.4.3 Observational Equivalence

In the presence of control effects, we replace Def. 20(2) by

Definition 32 Given two computations $\Gamma \vdash^{\mathsf{c}} M, N : \underline{B}$, we say $M \simeq N$ when for any ground context $\vdash^{\mathsf{c}} \mathcal{C}[] : \sum_{i \in I} 1$ we have

$$\mathcal{C}[M], \mathtt{nil} \rightsquigarrow^* \mathtt{produce}\ n, \mathtt{nil} \text{ iff } \mathcal{C}[N], \mathtt{nil} \rightsquigarrow^* \mathtt{produce}\ n, \mathtt{nil}$$

Similarly for values $\Gamma \vdash^{\mathsf{v}} V, W : A$. □

Since $\mathcal{C}[M]$ and $\mathcal{C}[N]$ are required to be closed (by the definition of ground context), they cannot contain `nil`.

6.4.4 Denotational Semantics

We seek a denotational semantics for the language described in Sect. 6.4.1. For reasons we shall see below, this semantics is called a *continuation semantics*. Since there is no divergence, we use a set model. The semantics of values is given in Sect. 6.2—the difficulty lies in the interpretation of computations.

As in Sect. 6.3.2, we will state first the soundness theorem that we are aiming to achieve, even though it is not yet meaningful, and use this to motivate the semantics.

Proposition 43 (soundness) Suppose that $M, K \rightsquigarrow N, L$ where

$$\begin{array}{ll} \Gamma \vdash^{\mathsf{c}} M : \underline{B} & \Gamma \vdash^{\mathsf{v}} K : \mathtt{os}\ \underline{B} \\ \Gamma \vdash^{\mathsf{c}} N : \underline{C} & \Gamma \vdash^{\mathsf{v}} L : \mathtt{os}\ \underline{C} \end{array}$$

Then, for any environment $\rho \in [\![\Gamma]\!]$,

$$[\![M]\!](\rho, [\![K]\!]\rho) = [\![N]\!](\rho, [\![L]\!]\rho) \tag{6.1}$$

□

Notice the similarity between this statement and Prop. 37. There, a computation M can change the store, so $[\![M]\!]$ takes store as an argument. Here, a computation M can change its outside, so $[\![M]\!]$ takes its outside as an argument.

In Prop. 43, we know that $[\![K]\!]\rho \in [\![\mathtt{os}\ \underline{B}]\!]$. So $[\![M]\!]$ must be a function from $[\![\Gamma]\!] \times [\![\mathtt{os}\ \underline{B}]\!]$ to some set; similarly, $[\![N]\!]$ must be a function from $[\![\Gamma]\!] \times [\![\mathtt{os}\ \underline{C}]\!]$ to the same set. This set, which we call Ans (the "set of answers") cannot depend on the type of M, because M and N have different types. It is an arbitrary set which remains fixed throughout the denotational semantics.

We proceed to the semantics of types, which is given in Fig. 6.1(1). At first sight, these equations looks strange, but they make sense once we know that whenever we see $[\![\underline{B}]\!]$, for a computation type $\underline{B}$, we should mentally replace it with $[\![\mathtt{os}\ \underline{B}]\!]$. To put it another way, the semantic brackets $[\![-]\!]$ for computation types contain a "hidden" `os`. To explain the semantics of types, we first look at some equations which do not require any mental replacement (because they do not mention denotations of computation types) and so make sense immediately.

1. Official presentation—compositional

Values types			Computation types		
$[\![U\underline{B}]\!]$	$=$	$[\![\underline{B}]\!] \to \mathsf{Ans}$	$[\![FA]\!]$	$=$	$[\![A]\!] \to \mathsf{Ans}$
$[\![\sum_{i\in I} A_i]\!]$	$=$	$\sum_{i\in I}[\![A_i]\!]$	$[\![\prod_{i\in I}\underline{B}_i]\!]$	$=$	$\sum_{i\in I}[\![\underline{B}_i]\!]$
$[\![A \times A']\!]$	$=$	$[\![A]\!] \times [\![A']\!]$	$[\![A \to \underline{B}]\!]$	$=$	$[\![A]\!] \times [\![\underline{B}]\!]$
$[\![\mathtt{os}\ \underline{B}]\!]$	$=$	$[\![\underline{B}]\!]$			

2. Intuitive presentation—not compositional

Value types					
$[\![U\underline{B}]\!]$	$=$	$[\![\mathtt{os}\ \underline{B}]\!] \to \mathsf{Ans}$	$[\![\mathtt{os}\ FA]\!]$	$=$	$[\![A]\!] \to \mathsf{Ans}$
$[\![\sum_{i\in I} A_i]\!]$	$=$	$\sum_{i\in I}[\![A_i]\!]$	$[\![\mathtt{os}\ \prod_{i\in I}\underline{B}_i]\!]$	$=$	$\sum_{i\in I}[\![\mathtt{os}\ \underline{B}_i]\!]$
$[\![A \times A']\!]$	$=$	$[\![A]\!] \times [\![A']\!]$	$[\![\mathtt{os}\ (A \to \underline{B})]\!]$	$=$	$[\![A]\!] \times [\![\mathtt{os}\ \underline{B}]\!]$
Computation types					
$[\![\underline{B}]\!]$	$=$	$[\![\mathtt{os}\ \underline{B}]\!]$			

Figure 6.1: Semantics of types—two equivalent presentations

- $[\![\mathtt{os}\ (A \to \underline{B})]\!] = [\![A]\!] \times [\![\mathtt{os}\ \underline{B}]\!]$ follows from the fact that a closed outside for $A \to \underline{B}$ consists of a closed value of type A together with a closed outside for $\underline{B}$.

- $[\![\prod_{i\in I}\underline{B}_i]\!] = \sum_{i\in I}[\![\underline{B}_i]\!]$ follows from the reversible derivation for $\prod$, but more obviously from the fact that a closed outside for $\prod_{i\in I}\underline{B}_i$ consists of a tag $\hat{\imath} \in I$ together with a closed outside for $\underline{B}_{\hat{\imath}}$.

- $[\![\mathtt{os}\ FA]\!] = [\![A]\!] \to \mathsf{Ans}$ is plausible, because a consumer of A-values takes an A-value to an answer.

- $[\![U\underline{B}]\!] = [\![\mathtt{os}\ \underline{B}]\!] \to \mathsf{Ans}$ follows from the reversible derivation for U.

Now these equations, together with the standard equations of $\sum$ and $\times$, completely determine the semantics of value types. In other words, there is a unique function $[\![-]\!]$ from value types to sets that satisfies them. But this function is not given compositionally. Therefore, in Fig. 6.1, we write $[\![\underline{B}]\!]$ as shorthand for $[\![\mathtt{os}\ \underline{B}]\!]$, for the sole reason that this enables us to rearrange these equations into a compositional semantics.

We can now say that a computation $\Gamma \vdash^{\mathsf{c}} M : \underline{B}$ denotes a function from $[\![\Gamma]\!] \times [\![\underline{B}]\!]$ to Ans. Here are some example clauses for semantics of terms:

$$
\begin{array}{rcl}
[\![\mathtt{produce}\ V]\!](\rho,k) & = & ([\![V]\!]\rho)\text{`}k \\
[\![M\ \mathtt{to}\ \mathtt{x}.\ N]\!](\rho,k) & = & [\![M]\!](\rho, \lambda a.([\![N]\!]((\rho, \mathtt{x} \mapsto a), k))) \\
[\![\mathtt{thunk}\ M]\!]\rho & = & \lambda k.([\![M]\!](\rho,k)) \\
[\![\mathtt{force}\ V]\!](\rho,k) & = & k\text{`}([\![V]\!]\rho) \\
[\![\lambda \mathtt{x}.M]\!](\rho,(a,k)) & = & [\![M]\!]((\rho, \mathtt{x} \mapsto a), k) \\
[\![V\text{`}M]\!](\rho,k) & = & [\![M]\!](\rho, ([\![V]\!]\rho, k)) \\
[\![\mathtt{letcos}\ \mathtt{x}.\ M]\!](\rho,k) & = & [\![M]\!]((\rho, \mathtt{x} \mapsto k), k) \\
[\![\mathtt{changecos}\ K;\ M]\!](\rho,k) & = & [\![M]\!](\rho, [\![K]\!]\rho) \\
[\![[]\ \mathtt{to}\ \mathtt{x}.\ N :: K]\!]\rho & = & \lambda a.([\![N]\!]((\rho, \mathtt{x} \mapsto a), [\![K]\!]\rho)) \\
[\![V :: K]\!]\rho & = & ([\![V]\!]\rho, [\![K]\!]\rho)
\end{array}
$$

Notice the similarity between these semantic equations and the machine transitions.

We can now prove Prop. 43 straightforwardly.

Corollary 44 (by Prop. 42) Suppose Ans has 2 elements $a \neq b$. For a set N and element $n \in N$, write $I_{N,n}$ for the function from N to Ans that takes n to a and everything else to b. Then for any closed ground producer M of type $F\sum_{i\in N} 1$, we have $M,\mathtt{nil} \rightsquigarrow^* \mathtt{produce}\ n,\mathtt{nil}$ iff $[\![M]\!]I_{N,n} = a$. Hence denotational equality implies observational equivalence. □

It is also easy to show that provably equal terms have the same denotation i.e. the model validates the CBPV equations.

Definition 33 A *continuation* is a value that (for a given environment) denotes a function to Ans. □

We note that there are two kinds of continuation: thunks and consumers.

We emphasize[2] that outsides and continuations are quite distinct concepts. Only consumers fall into both categories. An outside such as $V :: K$ is not a continuation because it denotes a pair. The situation is summarized in Fig. 6.2. The reader should beware: other authors sometimes use

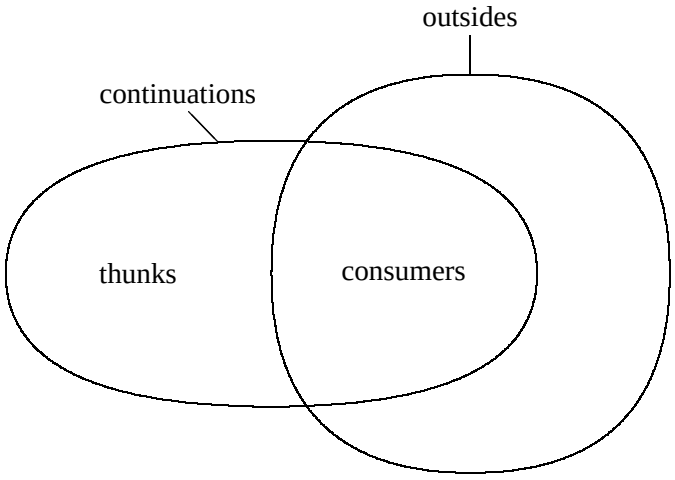


Figure 6.2: Continuations and Outsides

the word "continuation" to mean "consumer" or "outside". For example:

- in the CBV literature, the "current continuation" would be more precisely described as the current consumer;
- the "continuation type" $\mathtt{cont}\ A$ in ML would be more accurately described as a type of consumers (of values of type A);
- [HS97, Lai98, SR98], which treat CBN as well as CBV, use "continuation" to mean what we call an outside.

Notice the semantics of CBV functions:

$$[\![A \to_{\mathbf{CBV}} B]\!] = [\![U(A \to FB)]\!] = ([\![A]\!] \times ([\![B]\!] \to \mathsf{Ans})) \to \mathsf{Ans}$$

As we said in Sect. 6.1, this is precisely the traditional continuation semantics for CBV.

[2] especially to readers familiar with continuations in the CBV setting, where all outsides are consumers, and so the current outside is usually called the "current continuation"

6.4.5 Combining Control Effects With Other Effects

The continuation model in Sect. 6.4.4 generalizes. If we have any CBPV model, we can obtain from it a continuation model for control effects.

As an example, consider the printing model for CBPV. We seek a model for control effects together with `print`. The CK-machine semantics will have the form $M, K \rightsquigarrow m, N, L$ as in Sect. 3.4.2.

We fix an $\mathcal{A}$-set $\underline{\mathsf{Ans}}$. This plays the same role as the set Ans in Sect. 6.4.4. The denotational semantics for control effects with `print` is then organized as follows:

- a value type (and hence a context) denotes a set;
- a computation type denotes[3] a set;
- a value $\Gamma \vdash^{\mathsf{v}} V : A$ denotes a function from $[\![\Gamma]\!]$ to $[\![A]\!]$;
- a computation $\Gamma \vdash^{\mathsf{c}} M : \underline{B}$ denotes a function from $[\![\Gamma]\!] \times [\![\underline{B}]\!]$ to $\underline{\mathsf{Ans}}$.

As explained in Sect. 3.8, we use CBPV as a metalanguage describing the printing model. For example, we write $U\underline{B}$ for the carrier of an $\mathcal{A}$-set $\underline{B}$ and FA for the free $\mathcal{A}$-set on a set A. With this notation, the semantics of types is given by

$$\begin{array}{rclcrcl}
[\![U\underline{B}]\!] & = & U([\![\underline{B}]\!] \to \underline{\mathsf{Ans}}) & \qquad & [\![FA]\!] & = & U([\![A]\!] \to \underline{\mathsf{Ans}}) \\
[\![\textstyle\sum_{i\in I} A_i]\!] & = & \textstyle\sum_{i\in I} [\![A_i]\!] & & [\![\textstyle\prod_{i\in I} \underline{B}_i]\!] & = & \textstyle\sum_{i\in I} [\![\underline{B}_i]\!] \\
[\![A \times A']\!] & = & [\![A]\!] \times [\![A']\!] & & [\![A \to \underline{B}]\!] & = & [\![A]\!] \times [\![\underline{B}]\!] \\
[\![\mathtt{os}\ \underline{B}]\!] & = & [\![\underline{B}]\!] & &
\end{array}$$

Semantics of terms: some example clauses.

$$\begin{array}{rcl}
[\![\mathtt{produce}\ V]\!](\rho,k) & = & ([\![V]\!]\rho)\text{`}(\mathsf{force}\ k) \\
[\![M\ \mathtt{to}\ \mathtt{x}.\ N]\!](\rho,k) & = & [\![M]\!](\rho, \mathsf{thunk}\ \lambda a.([\![N]\!]((\rho, \mathtt{x} \mapsto a), k))) \\
[\![\mathtt{thunk}\ M]\!]\rho & = & \mathsf{thunk}\ \lambda k.([\![M]\!](\rho,k)) \\
[\![\mathtt{force}\ V]\!](\rho,k) & = & k\text{`}\mathsf{force}\ ([\![V]\!]\rho) \\
[\![\lambda \mathtt{x}.M]\!](\rho,(a,k)) & = & [\![M]\!]((\rho, \mathtt{x} \mapsto a), k) \\
[\![V\text{`}M]\!](\rho,k) & = & [\![M]\!](\rho, ([\![V]\!]\rho, k)) \\
[\![\mathtt{letcos}\ \mathtt{x}.\ M]\!](\rho,k) & = & [\![M]\!]((\rho, \mathtt{x} \mapsto k), k) \\
[\![\mathtt{changecos}\ K;\ M]\!](\rho,k) & = & [\![M]\!](\rho, [\![K]\!]\rho) \\
[\![\mathtt{print}\ c;\ M]\!](\rho,k) & = & c * ([\![M]\!](\rho,k)) \\
[\![[]\ \mathtt{to}\ \mathtt{x}.\ N :: K]\!]\rho & = & \mathsf{thunk}\ \lambda a.([\![N]\!]((\rho, \mathtt{x} \mapsto a), [\![K]\!]\rho)) \\
[\![V :: K]\!]\rho & = & ([\![V]\!]\rho, [\![K]\!]\rho)
\end{array}$$

Proposition 45 (soundness) Suppose that $M, K \rightsquigarrow m, N, L$ where

$$\begin{array}{ll}
\Gamma \vdash^{\mathsf{c}} M : \underline{B} & \Gamma \vdash^{\mathsf{v}} K : \mathtt{os}\ \underline{B} \\
\Gamma \vdash^{\mathsf{c}} N : \underline{C} & \Gamma \vdash^{\mathsf{v}} L : \mathtt{os}\ \underline{C}
\end{array}$$

Then, for any environment $\rho \in [\![\Gamma]\!]$,

$$[\![M]\!](\rho, [\![K]\!]\rho) = m * [\![N]\!](\rho, [\![L]\!]\rho) \tag{6.2}$$

□

[3] As in Sect. 6.4.4, $[\![\underline{B}]\!]$ should be thought of as shorthand for $[\![\mathtt{os}\ \underline{B}]\!]$.

Corollary 46 (by the analogue of Prop. 42) Suppose $\underline{\mathtt{Ans}}$ has two elements a, b with the property that

$$\begin{array}{lcll} m * a & \neq & m' * a & \text{for } m \neq m' \in \mathcal{M} \\ m * a & \neq & m' * b & \text{for } m, m' \in \mathcal{M} \end{array}$$

(This property is satisfied by the free $\mathcal{A}$-set on a set of size $\geqslant 2$.) For a set N and element $n \in N$, write $I_{N,n}$ for the function from N to $\underline{\mathtt{Ans}}$ that takes n to a and everything else to b.

Then for any closed ground producer M of type $F\sum_{i \in N} 1$, we have $M, \mathtt{nil} \rightsquigarrow^* m, \mathtt{produce}\, n, \mathtt{nil}$ iff $[\![M]\!] I_{N,n} = m * a$. Hence terms with the same denotation are observationally equivalent. □

It is also easy to show that provably equal terms have the same denotation i.e. the model validates the CBPV equations.

In a similar way, we can obtain a model for control effects with recursion. The semantics of types and terms are as above except that we now understand the metalinguistic U etc. as referring to the Scott model rather than the printing model. We need only replace the semantic equation for `print` by an equation for recursion.

6.5 Erratic Choice

Contrary to the false claim made in [Lev99], it is only in languages without divergence that the following approach accurately models erratic choice.

6.5.1 The Language

We add to the language the following erratic choice construct

$$\frac{\cdots \quad \Gamma \vdash^{\mathsf{c}} M_i : \underline{B} \quad \cdots}{\Gamma \vdash^{\mathsf{c}} \mathtt{choose}\, \{\ldots, i.M_i, \ldots\} : \underline{B}}$$

where i ranges over I. According to taste, we can allow I to be finite, countable, nonempty or arbitrary. The meaning of $\mathtt{choose}\, \{\ldots, i.M_i, \ldots\}$ is "choose some $i \in I$, then execute M_i". Thus to the big-step semantics we add

$$\frac{M_{\hat{\imath}} \Downarrow T}{\mathtt{choose}\, \{\ldots, i.M_i, \ldots\} \Downarrow T}$$

and to the CK-machine we add the transition

$$\begin{array}{lll} & \mathtt{choose}\, \{\ldots, i.M_i, \ldots\} & K \\ \rightsquigarrow & M_{\hat{\imath}} & K \end{array}$$

It can be proved that computations cannot diverge. Our formulation of the big-step semantics does not allow us to express this fact, but the CK-machine does allow us to express it:

Proposition 47 There is no infinite sequence of transitions from any configuration M, K. □

6.5.2 Denotational Semantics

We seek a denotational semantics for the language described in Sect. 6.5.1. Since there is no divergence, we use a set model. The semantics of values is given in Sect. 6.2—the difficulty lies in the interpretation of computations.

This can be motivated in a similar way to Sect. 6.3.2, by starting with the soundness statement and using reversible derivations. We summarize the semantics here.

- A computation type denotes a set.

- A computation $\Gamma \vdash^{\mathsf{c}} M : \underline{B}$ denotes a relation from $[\![\Gamma]\!]$ to $[\![\underline{B}]\!]$.

Writing $\mathcal{P}A$ for the powerset of A, the semantics of types is given by

$$\begin{array}{rcl} [\![U\underline{B}]\!] & = & \mathcal{P}[\![\underline{B}]\!] \\ [\![FA]\!] & = & [\![A]\!] \\ [\![\prod_{i\in I}\underline{B}_i]\!] & = & \sum_{i\in I}[\![\underline{B}_i]\!] \\ [\![A \to \underline{B}]\!] & = & [\![A]\!] \times [\![\underline{B}]\!] \end{array}$$

Semantics of terms—some example clauses:

$$\begin{array}{rcl} (\rho,a) \in [\![\mathtt{produce}\, V]\!] & \text{iff} & [\![V]\!]\rho = a \\ (\rho,b) \in [\![M \ \mathtt{to}\ \mathtt{x}.\ N]\!] & \text{iff} & \text{for some } a,\ (\rho,a) \in [\![M]\!] \text{ and } ((\rho,\mathtt{x}\mapsto a),b) \in [\![N]\!] \\ [\![\mathtt{thunk}\, M]\!]\rho & = & \{b \in [\![\underline{B}]\!] : (\rho,b) \in [\![M]\!]\} \\ (\rho,b) \in [\![\mathtt{force}\, V]\!] & \text{iff} & b \in [\![V]\!]\rho \\ (\rho,(a,b)) \in [\![\lambda \mathtt{x}.M]\!] & \text{iff} & ((\rho,\mathtt{x}\mapsto a),b) \in [\![M]\!] \\ (\rho,b) \in [\![V\,`M]\!] & \text{iff} & (\rho,([\![V]\!]\rho,b)) \in [\![M]\!] \\ (\rho,b) \in [\![\mathtt{choose}\, \{\ldots,i.M_i,\ldots\}]\!] & \text{iff} & \text{for some } i,\ (\rho,b) \in [\![M_i]\!] \end{array}$$

Proposition 48 (soundness and adequacy) For any closed computation M, we have

$$[\![M]\!] = \bigcup_{M \Downarrow T} [\![T]\!]$$

□

Proof For ($\supseteq$) we induct on $M \Downarrow T$. For ($\subseteq$) we define subsets $\mathsf{red}^{\mathsf{v}}_A$, $\mathsf{red}^{\mathsf{t}}_{\underline{B}}$ and $\mathsf{red}^{\mathsf{c}}_{\underline{B}}$ exactly as in Prop. 9, except that we replace the clause for $M \in \mathsf{red}^{\mathsf{c}}_{\underline{B}}$ by the following:

$M \in \mathsf{red}^{\mathsf{c}}_{\underline{B}}$ iff for all $b \in [\![M]\!]$ there exists $T \in \mathsf{red}^{\mathsf{t}}_{\underline{B}}$ such that $M \Downarrow T$ and $b \in [\![T]\!]$.

We note that if $T \in \mathsf{red}^{\mathsf{t}}_{\underline{B}}$ then $T \in \mathsf{red}^{\mathsf{c}}_{\underline{B}}$ (unlike in the proof of 9, the converse is not apparent at this stage of the proof). The rest of the proof follows that of Prop. 9. □

Corollary 49 For any closed ground producer M, we have $M \Downarrow \mathtt{produce}\, n$ iff $n \in [\![M]\!]$. □

It is also easy to show that provably equal terms have the same denotation i.e. the model validates the CBPV equations.

Notice the semantics of CBV functions:

$$[\![A \to_{\mathbf{CBV}} B]\!] = \mathcal{P}([\![A]\!] \times [\![B]\!])$$

This is the set of relations from $[\![A]\!]$ to $[\![B]\!]$, a traditional nondeterministic semantics for CBV.

6.5.3 Finite Choice

We said in Sect. 6.5.1 that in the rule

$$\frac{\cdots \ \Gamma \vdash^{\mathsf{c}} M_i : \underline{B} \ \cdots}{\Gamma \vdash^{\mathsf{c}} \mathtt{choose}\, \{\ldots,i.M_i,\ldots\} : \underline{B}}$$

we can, if we like, restrict the set I that i ranges over to be finite (or countable or nonempty). This effect of *finite erratic choice* provides an interesting example of a CBV model that does not decompose naturally into CBPV.

We modify the CBV model for erratic choice given above so that $A \to_{\mathbf{CBV}} B$ denotes not $[\![A]\!] \to \mathcal{P}[\![B]\!]$, as it previously did (up to isomorphism), but $[\![A]\!] \to \mathcal{P}_{\mathsf{fin}}[\![B]\!]$, where $\mathcal{P}_{\mathsf{fin}}X$ is the set of finite subsets of X. Our decomposition of CBV erratic choice semantics into CBPV was based on the isomorphism

$$A \to \mathcal{P}B \cong \mathcal{P}(A \times B)$$

but there is no analogous isomorphism for $A \to \mathcal{P}_{\mathsf{fin}}B$. So we do not have a CBPV semantics for finite erratic choice. (Similarly if we restrict to countable subsets or to nonempty subsets.)

This is a situation we shall see again in Sect. 7.9: the "basic" CBV model (in this example, general erratic choice) decomposes into CBPV, but a constrained model (finite erratic choice) does not.

6.6 Errors

The *errors* feature we consider here is much weaker than the *exceptions* feature of ML and Java. In particular, we do not provide a handling facility.

6.6.1 The Language

We fix a set E of *errors*. We add to CBPV a command $\mathtt{error}\ e$ for each $e \in E$. The effect of this command is to halt execution, reporting e as an "error message". Thus we add to the basic CBPV language the rule

$$\frac{}{\Gamma \vdash^{\mathsf{c}} \mathtt{error}\ e : \underline{B}}$$

Notice that $\mathtt{error}\ e$ can have any type, like $\mathtt{diverge}$.

The big-step semantics now has two judgements $M \Downarrow T$ and $M \Downarrow e$. For each rule in Fig. 3.2 of the form (3.1) and each $0 \leqslant s < r$ we add a rule

$$\frac{M_0 \Downarrow T_0 \quad \cdots \quad M_{s-1} \Downarrow T_{s-1} \quad M_s \Downarrow e}{M \Downarrow e}$$

and we add the rule

$$\frac{}{\mathtt{error}\ e \Downarrow e}$$

Proposition 50 For every closed computation M, precisely one of the following holds

- there exists unique T such that $M \Downarrow T$ and there does not exist e such that $M \Downarrow e$
- there does not exist T such that $M \Downarrow T$ and there exists unique e such that $M \Downarrow e$.

□

For the CK-machine, we first extend the class of configurations: a configuration may be either of the form M, K as in Sect. 3.3.2, or of the form e for $e \in E$. We extend the class of

terminal configurations to be the following:

	$\mathtt{produce}\, V$	$\mathtt{nil}$
	$\lambda\{\ldots, i.M_i, \ldots\}$	$\mathtt{nil}$
	$\lambda \mathtt{x} M$	$\mathtt{nil}$
	e	

and we add the transition

	$\mathtt{error}\, e$	K
$\rightsquigarrow$	e	

6.6.2 Denotational Semantics

We seek a denotational semantics for the language described in Sect. 6.6.1. Since there is no divergence, we use a set model. The semantics of values is given in Sect. 6.2—the difficulty lies in the interpretation of computations.

The semantics is very similar to our $\mathcal{A}$-set semantics for printing. In Chap. 12 we shall see that they are instances of a general construction.

Definition 34 (cf. Def. 8)

- An *E-set* (X, error) consists of a set X together with a function error from E to X. We call X the carrier and error the structure.
- An *element* of (X, error) is an element of X.
- A *function* from a set W to (X, error) is a function from W to X.

□

Here are some ways of constructing E-sets (cf. Def. 9).

1. For any set X, the *free* E-set on X has carrier $X + E$ and we set $\mathsf{error}\ e$ to be $\mathsf{inr}\ e$.
2. For an $i \in I$-indexed family of E-sets (X_i, error), we set $\prod_{i\in I}(X_i, \mathsf{error})$ to have carrier $\prod_{i\in I} X_i$ and structure given pointwise: $\hat{i}\text{`}(\mathsf{error}\ e) = \mathsf{error}\ e$.
3. For any set X and E-set (Y, error), we define the E-set $X \to (Y, \mathsf{error})$ to have carrier $X \to Y$ and structure given pointwise: $x\text{`}(\mathsf{error}\ e) = \mathsf{error}\ e$.

Computation types denote E-sets and value types denote sets in the evident way: in particular FA denotes the free E-set on $[\![A]\!]$ and $U\underline{B}$ denotes the carrier of $[\![\underline{B}]\!]$. A computation $\Gamma \vdash^{\mathsf{c}} M : \underline{B}$ denotes a function from $[\![\Gamma]\!]$ to $[\![\underline{B}]\!]$. We omit semantics of terms; the key clause is

$$[\![\mathtt{error}\ e]\!]\rho = \mathsf{error}\ e$$

Proposition 51 (soundness)

- If $M \Downarrow T$ then $[\![M]\!] = [\![T]\!]$.
- If $M \Downarrow e$ then $[\![M]\!] = \mathsf{error}\ e$.

□

Corollary 52 (by Prop. 50) For a closed ground producer M, we have $M \Downarrow \mathtt{produce}\ n$ iff $[\![M]\!] = \mathsf{inl}\ n$, and $M \Downarrow e$ iff $[\![M]\!] = \mathsf{inr}\ e$. □

It is also easy to show that provably equal terms have the same denotation i.e. the model validates the CBPV equations.

Notice that we recover the traditional semantics of CBV functions:

$$[\![A \to B]\!] = [\![A]\!] \to ([\![B]\!] + E)$$

6.7 Combining Printing and Divergence

6.7.1 The Language

When we combine printing and divergence, the form of the big-step semantics changes. Given a closed computation M there are two possibilities:

1. M could print a finite string m and then terminate as T. As before, we write this $M \Downarrow m, T$.

2. M could print a finite or infinite string m and never terminate. We write this as $M \Uparrow m$.

We write $\mathcal{A}^{*\infty}$ for the set of finite or infinite strings of characters in $\mathcal{A}$.

We would like to provide a big-step definition for these behaviours, but we do not know how to do this. Instead, we will define them using the CK-machine.

Definition 35 1. We say that $M \Downarrow m, T$ when there exists a finite sequence of transitions

$$\begin{array}{cc} M_i, K_i \rightsquigarrow m_i, M_{i+1}, K_{i+1} & \text{for } 0 \leqslant i < r \\ \text{where} & M_0, K_0 = M, \mathtt{nil} \\ & M_r, K_r = T, \mathtt{nil} \\ & m_0 * \cdots * m_{r-1} = m \end{array}$$

2. We say that $M \Uparrow m$ when there exists an infinite sequence of transitions

$$\begin{array}{cc} M_i, K_i \rightsquigarrow m_i, M_{i+1}, K_{i+1} & \text{for } 0 \leqslant i < \infty \\ \text{where} & M_0, K_0 = M, \mathtt{nil} \\ & m_0 * \cdots = m \end{array}$$

□

We would strongly prefer this to be a proposition (analogous to Prop. 12) rather than a definition, but this requires a big-step characterization[4] of $\Uparrow$, which we have not found. We leave this to future work.

6.7.2 Denotational Semantics

The denotational semantics is very similar to our $\mathcal{A}$-set semantics for printing and to our E-set semantics for errors. In Chap. 12 we shall see that they are instances of a general construction.

Because the denotational semantics is a cpo model, the semantics of value types is as given in Sect. 6.2 and we will use CBPV as a metalanguage for the Scott model as explained in Sect. 3.8. For example we write $U\underline{B}$ for the cppo $\underline{B}$ regarded as a cpo, and we write FA for the lift of the cpo A. Using this notation we have the following; it is analogous to Def. 8.

Definition 36 1. An *$\mathcal{A}$-cppo* $(\underline{B}, *)$ consists of a cppo $\underline{B}$ together with a function $*$ from $\mathcal{A} \times U\underline{B}$ to $\underline{B}$. We call $\underline{B}$ the *carrier* and $*$ the *structure*.

2. An *element* of $(\underline{B}, *)$ is an element of $\underline{B}$.

3. A *continuous function* from a cpo W to $(\underline{B}, *)$ is a continuous function from W to $\underline{B}$.

□

[4] Another approach is to use *small-step semantics*, but we have avoided this style of operational semantics throughout the thesis, because it is inaccurate for infinitely deep terms. For example, the computation $M = 3\text{`}3\text{`}3\text{`}\cdots$ diverges, but small-step semantics would suggest that it is terminal, as there is no rewrite $M \rightsquigarrow N$. We consider this a symptom of a deeper problem: that small-step semantics, by contrast with big-step and CK-machine semantics, does not give a correct description of flow of control, because only conversion of redexes is represented.

Each computation type will denote an $\mathcal{A}$-set, and each computation $\Gamma \vdash^{\mathsf{c}} M : \underline{B}$ will denotes a continuous function from $[\![\Gamma]\!]$ to $[\![\underline{B}]\!]$. As for $\mathcal{A}$-sets, given an $\mathcal{A}$-cppo, we can extend $*$ to a function from $\mathcal{A}^* \times U\underline{B}$ to $\underline{B}$. We write $c * b$ instead of $*(c, \mathsf{thunk}\ b)$. Furthermore, we define a function stream from $\mathcal{A}^{*\infty}$ to $\underline{B}$ where $\mathsf{stream}\ m$ is defined to be the lub of $n * \bot$ for all finite prefixes n of m.

By analogy with Def. 9, we can define the free $\mathcal{A}$-cppo on a cpo A, whose carrier is the cppo $\mu\underline{X}.F(A + \mathcal{A} \times U\underline{X})$, and we can define product and function $\mathcal{A}$-cpos. We omit the details of the semantics and the proof of the following:

Proposition 53 (soundness and adequacy) 1. If $M \Downarrow m, T$ then $[\![M]\!] = m * [\![T]\!]$.

2. If $M \Uparrow m$ then $[\![M]\!] = \mathsf{stream}\ m$.

□

Corollary 54 For a closed ground producer M, we have $M \Downarrow m, \mathtt{produce}\ n$ iff $[\![M]\!] = m * \mathsf{produce}\ n$, and $M \Uparrow m$ iff $[\![M]\!] = \mathsf{stream}\ M$. Therefore, terms with the same denotation are observationally equivalent. □

It is also easy to show that provably equal terms have the same denotation i.e. the model validates the CBPV equations.

6.8 Summary

The easy part of all these models—the semantics of values— was given in Sect. 6.2. In Fig. 6.3 we summarize the more difficult part—the semantics of computations.

Remember that, when giving semantics for global store + printing and the semantics for control + printing, we use CBPV as a metalanguage for the printing model: U means "carrier", F means "free $\mathcal{A}$-set". But we can combine global store or control with any effect that we have a model for, by understanding U and F as referring to this model.

For each effect, we see that the semantics of UF gives a monad in the style of Moggi [Mog91]. It is remarkable how each monad decomposes into U and F in a specific way that fits the operational semantics. In some ways, we have covered the same ground of Moggi; but the key improvements are that

- we have included CBN as well as CBV;
- our language has operational semantics.

Looking at a few examples, we see in Fig. 6.4 that we recover traditional semantics for CBV, and we obtain strange-looking semantics for CBN. As stated in Sect. 6.1, the CBN semantics for global store was presented in [O'H93], while the CBN semantics for control appeared in [SR98]. We can see that CBPV provides an explanation of these apparently mysterious models.

The models are arranged into two groups. In the terminology of Sect. 6.2 (where the semantics of values is described), the first group are set models and the second group are cpo models.

effect	a comp. type denotes	a computation $\Gamma \vdash^{\mathsf{c}} M : \underline{B}$ denotes
printing	an $\mathcal{A}$-set	a function from $[\![\Gamma]\!]$ to $[\![\underline{B}]\!]$
global store	a set	a function from $S \times [\![\Gamma]\!]$ to $[\![\underline{B}]\!]$
global store + printing	an $\mathcal{A}$-set	a function from $S \times [\![\Gamma]\!]$ to $[\![\underline{B}]\!]$
control	a set	a function from $[\![\Gamma]\!] \times [\![\underline{B}]\!]$ to Ans
control + printing	a set	a function from $[\![\Gamma]\!] \times [\![\underline{B}]\!]$ to $\underline{\mathsf{Ans}}$
erratic choice	a set	a relation from $[\![\Gamma]\!]$ to $[\![\underline{B}]\!]$
errors	an E-set	a function from $[\![\Gamma]\!]$ to $[\![\underline{B}]\!]$
divergence	a cppo	a continuous function from $[\![\Gamma]\!]$ to $[\![\underline{B}]\!]$
divergence + printing	an $\mathcal{A}$-cppo	a continuous function from $[\![\Gamma]\!]$ to $[\![\underline{B}]\!]$

effect	U	F	$UF = T$
printing	carrier	free $\mathcal{A}$-set	$\mathcal{A}^* \times -$
global store	$S \to -$	$S \times -$	$S \to (S \times -)$
global store + printing	$U(S \to -)$	$F(S \times -)$	$U(S \to F(S \times -))$
control	$- \to \mathsf{Ans}$	$- \to \mathsf{Ans}$	$(- \to \mathsf{Ans}) \to \mathsf{Ans}$
control + printing	$U(- \to \underline{\mathsf{Ans}})$	$U(- \to \underline{\mathsf{Ans}})$	$U(U(- \to \underline{\mathsf{Ans}}) \to \underline{\mathsf{Ans}})$
erratic choice	$\mathcal{P}$	$-$	$\mathcal{P}$
errors	carrier	free E-set	$- + E$
divergence	$-$	lift	lift
divergence + printing	carrier	free $\mathcal{A}$-cppo	$\mu X.(- + \mathcal{A} \times X)_\perp$

effect	$\to$	$\prod_{i \in I}$
printing	$\to$	$\prod_{i \in I}$
global store	$\to$	$\prod_{i \in I}$
global store + printing	$\to$	$\prod_{i \in I}$
control	$\times$	$\sum_{i \in I}$
control + printing	$\times$	$\sum_{i \in I}$
erratic choice	$\times$	$\sum_{i \in I}$
errors	$\to$	$\prod_{i \in I}$
divergence	$\to$	$\prod_{i \in I}$
divergence + printing	$\to$	$\prod_{i \in I}$

Figure 6.3: Summary of simple CBPV models

effect	$[\![A \to_{\mathbf{CBV}} B]\!] = [\![U(A \to FB)]\!]$	$[\![\underline{A} \to_{\mathbf{CBN}} \underline{B}]\!] = [\![U\underline{A} \to \underline{B}]\!]$
global store	$S \to ([\![A]\!] \to (S \times [\![B]\!]))$	$(S \to [\![\underline{A}]\!]) \to [\![\underline{B}]\!]$
control	$([\![A]\!] \times ([\![B]\!] \to \mathsf{Ans})) \to \mathsf{Ans}$	$([\![\underline{A}]\!] \to \mathsf{Ans}) \times [\![\underline{B}]\!]$
erratic choice	$\mathcal{P}([\![A]\!] \times [\![B]\!])$	$(\mathcal{P}[\![A]\!]) \times [\![B]\!]$

Figure 6.4: Induced semantics for CBV and CBN function types

Chapter 7

Possible World Model for Cell Generation

7.1 Introduction (Part 1)

In Sect. 6.3, we looked at semantics of *global* storage cells. But most programming languages provide facilities for generating new cells (i.e. memory locations) during execution. In such a language there may be, at one time, 3 cells storing a boolean and 1 cell storing a number, and at a later time, 5 boolean-storing cells and 6 number-storing cells, because 2 new boolean-storing cells and 5 new number-storing cells have been generated. Consequently, to describe the state of the memory at a given time, we require two pieces of information:

- the *world*, which tells us how many boolean-storing cells, how many number-storing cells, etc., are in existence;
- the *store* , which tells us what values these cells are storing.

These two pieces of information together are called a *world-store*. We stress that the world can only increase: new cells are generated but (at least in principle) none are ever destroyed. Writing w for the earlier world and w' for the later world, in the above example, we say that $w \leqslant w'$. We write $\mathcal{W}$ for the poset of worlds, regarded as a small category.

How can we provide a denotational semantics for such a language? One plausible approach is to use the global store semantics of Sect. 6.3, replacing the set of stores by the set of world-stores. However, by contrast with the store in Sect. 6.3, which can change in any way at all, the world-store is constrained in the way it can change by the fact that the world can only increase. Our goal is to provide a model that, unlike the global store model, embodies this important constraint.

The key idea which leads to such a model is that of *possible worlds*: a value type, instead of denoting just one set (or cpo), has a different denotation in each world. In the above example, the type `ref bool` (the type of cells storing booleans) denotes a 3-element set in the world w but a 5-element set in the world w'. We write $[\![A]\!]w$ for the denotation of A in the world w.

The various sets denoted by a type A are related. In the above example, $[\![\texttt{ref bool}]\!]w$ is clearly a subset of $[\![\texttt{ref bool}]\!]w'$. In general, whenever $w \leqslant w'$, there will be a function $[\![A]\!]^w_{w'}$ from $[\![A]\!]w$ to $[\![A]\!]w'$. These functions satisfy the equations

$$\begin{aligned} [\![A]\!]^w_w a &= a \\ [\![A]\!]^w_{w''} a &= [\![A]\!]^{w'}_{w''}([\![A]\!]^w_{w'} a) \quad \text{for } w \leqslant w' \leqslant w'' \end{aligned}$$

In summary, every value type A denotes a functor from $\mathcal{W}$ to **Set**.

A number of possible world models have been described. [Mog90, Ole82, PS93, Rey81, Sta94]. The model we will present differs from these in several respects. For example, in these

models a type denotes a functor not from $\mathcal{W}$ but from the larger category of *world injections* or the even larger category of *store shapes* [Ole82]. So this chapter is not the usual story of "traditional CBN and CBV semantics decompose into CBPV semantics", at least not in a straightforward way—the relationships between the various models are rather subtle. For that reason, it is probably wise for readers familiar with older models to regard the model in this chapter as essentially different; we discuss this in Sect. 7.9.

Still, the chapter does provide a good example of what can be achieved with CBPV. For whereas previous models (with the exception of [Ghi97]) interpret only storage of ground values, our approach can interpret storage of anything: ground values, cell locations and thunks. Such a facility is, after all, commonplace in practical CBV languages such as ML and Scheme. The disadvantage of our approach is that, precisely because of this generality, it does not easily accommodate parametricity constraints and the stack allocation present in Idealized Algol.

We will first describe the operational semantics and the denotational semantics of values. This is just laying the foundations and is not new or surprising. Then we will continue our discussion in Sect. 7.6 to look at how we can use a semantics for CBV to suggest a semantics for CBPV. In particular, we will see how the possible world semantics follows the slogan "U denotes $\prod$, F denotes $\sum$".

7.2 The Language

We add `ref` types to CBPV, giving the following type system:

$$\begin{array}{rcl} A & ::= & U\underline{B} \mid \sum_{i\in I} A_i \mid 1 \mid A \times A \mid \texttt{ref}\ A \\ \underline{B} & ::= & FA \mid \prod_{i\in I} \underline{B}_i \mid A \to \underline{B} \end{array}$$

As in ML, a value of type $\texttt{ref}\ A$ is a cell (i.e. a memory location) that stores a value of type A. For example, a value of type $\texttt{ref}\ \texttt{ref}\ \texttt{bool}$ is a cell storing a cell storing a boolean.

Maintaining our convention that commands are prefixes, we add the following terms.

Assigning to a cell

$$\frac{\Gamma \vdash^{\mathsf{v}} V : \texttt{ref}\ A \quad \Gamma \vdash^{\mathsf{v}} W : A \quad \Gamma \vdash^{\mathsf{c}} M : \underline{B}}{\Gamma \vdash^{\mathsf{c}} V := W;\ M : \underline{B}}$$

This computation replaces the current contents of the cell V with W, and then executes M.

Reading a cell

$$\frac{\Gamma \vdash^{\mathsf{v}} V : \texttt{ref}\ A \quad \Gamma, \mathtt{x} : A \vdash^{\mathsf{c}} M : \underline{B}}{\Gamma \vdash^{\mathsf{c}} \texttt{read}\ V\ \texttt{as}\ \mathtt{x}.\ M : \underline{B}}$$

This computation binds `x` to the current contents of the cell V (in other words, substitutes the current contents of V for `x`), and then executes M.

Generating a new cell

$$\frac{\Gamma \vdash^{\mathsf{v}} V : A \quad \Gamma, \mathtt{x} : \texttt{ref}\ A \vdash^{\mathsf{c}} M : \underline{B}}{\Gamma \vdash^{\mathsf{c}} \texttt{new}\ \mathtt{x} := V;\ M}$$

This computation generates a new cell `x`, initially storing V, and then executes M. There are recursive variants of this construct, but we will omit them.

Equality of cells The basic construct is

$$\frac{\Gamma \vdash^{\mathsf{v}} V : \mathtt{ref}\ A \quad \Gamma \vdash^{\mathsf{v}} V' : \mathtt{ref}\ A}{\Gamma \vdash^{\mathsf{v}} V = V' : \mathtt{bool}}$$

This is a complex value (in the sense of Sect. 4.2); so, for the sake of operational semantics, we regard as primitive the construct

$$\frac{\Gamma \vdash^{\mathsf{v}} V : \mathtt{ref}\ A \quad \Gamma \vdash^{\mathsf{v}} V' : \mathtt{ref}\ A \quad \Gamma \vdash^{\mathsf{c}} M : \underline{B} \quad \Gamma \vdash^{\mathsf{c}} M' : \underline{B}}{\Gamma \vdash^{\mathsf{c}} \mathtt{if}\ V = V'\ \mathtt{then}\ M\ \mathtt{else}\ M' : \underline{B}}$$

7.3 Operational Semantics

7.3.1 Worlds

As computation proceeds, new cells are generated. The number of cells presently in existence, together with their type, is described by a *world*.

Definition 37 1. A *world* w is a function from the set $\mathsf{valtypes}$ of value types to $\mathbb{N}$, such that

$$\sum_{A \in \mathsf{valtypes}} w_A < \infty$$

Informally, w_A is the number of A-storing cells in the world w. The condition ensures that the total collection of cells in a world is finite.

2. Using the notation of Sect. 1.4.1, we write $\mathsf{cells}\ w$ for the finite set $\sum_{A \in \mathsf{valtypes}} \$w_A$, the *set of cells in* w.

3. The *empty world* 0 is given by

$$0_A = 0 \text{ for all } A.$$

4. Let w be a world and let A be a value type. We use the phrase *w extended with an A-storing cell l* to mean the world w' defined by

$$w'_B = \begin{cases} w_B + 1 & \text{if } B = A \\ w_B & \text{otherwise} \end{cases}$$

The new cell l is then defined to be w_A.

5. Let w and w' be worlds. We say that $w \leqslant w'$ when

$$w_A \leqslant w'_A \text{ for all } A.$$

6. We write $\mathcal{W}$ for the poset of worlds regarded as a category. If $w \leqslant x$, we write $\frac{w}{x}$ for the unique morphism from w to x.

□

For the sake of the operational semantics, we need to define extra judgements of *w-values* and *w-computations*. These are terms that can explicitly refer to the cells in w. We write these judgements

$$w|\Gamma \vdash^{\mathsf{v}} V : A \qquad\qquad w|\Gamma \vdash^{\mathsf{c}} M : \underline{B}$$

where w is a world. They are defined by the same typing rules as ordinary values and computations, together with the additional rule

$$\frac{}{w|\Gamma \vdash^{\mathsf{v}} \mathtt{cell}_A i : \mathtt{ref}\ A} \qquad \text{where } i \in \$w_A$$

Notice that if $w \leqslant w'$ then every w-value is also a w'-value and every w-computation is also a w'-computation.

7.3.2 Stores

The world tells us only how many cells there are of each type, not what they contain. This latter information is provided by the *store*.

Definition 38 1. Let w be a world. A *w-store* is a function associating to each pair (A,i), where A is a value type and $i \in \$w_A$, a closed w-value $w| \vdash^{\mathsf{v}} V_{Ai} : A$. We write it

$$(\ldots, \mathtt{cell}_A i \mapsto V_{Ai}, \ldots)$$

2. A *world-store* is a world w together with a w-store s.

□

The operations on stores are similar to the store-handling constructs in the language:

Definition 39 Let (w,s) be a world-store and let A be a value type.

1. If l is an A-storing cell in w, we use the phrase *the contents of cell l in s* to mean the function s applied to (A,l).

2. If l is an A-storing cell in w and $w| \vdash V : A$, we use the phrase *s with cell l assigned V* for the w-store s' which is the same as s except that

$$s'(A,l) = V$$

3. If $w| \vdash V : A$, we use the phrase *(w,s) extended with a cell l storing V* to mean the world-store (w',s') where

   - w' is w extended with an A-storing cell l;
   - s' is the w'-store in which the contents of each cell of w is the same as in s (except that in s' it is regarded as a w'-value rather than a w-value) and the contents of the new cell l is V (regarded as a w'-value).

□

7.3.3 Operational Rules

The operational semantics of our dynamically generated store is similar to the operational semantics of global store in Sect. 6.3.1. We define a relation of the form $w,s,M \Downarrow w',s',T$ where

- w,s is a world-store;
- M is a closed w-computation;
- w',s' is a world-store such that $w' \geqslant w$;
- T is a closed terminal w'-computation.

We replace each rule in Fig. 3.2 of the form (3.1) by

$$\frac{w_0,s_0,M_0 \Downarrow w_1,s_1,T_0 \quad \cdots \quad w_{r-1},s_{r-1},M_{r-1} \Downarrow w_r,s_r,T_{r-1}}{w_0,s_0,M \Downarrow w_r,s_r,T}$$

and we add the following rules:

$$\frac{w, s, M[V/\mathtt{x}] \Downarrow w', s', T}{w, s, \mathtt{read}\ \mathtt{cell}_A l\ \mathtt{as}\ \mathtt{x}.\ M \Downarrow w', s', T} \qquad V \text{ is the contents of } A\text{-storing cell } l \text{ in } s$$

$$\frac{w, s', M \Downarrow w', s', T}{w, s, \mathtt{cell}_A l := V;\ M \Downarrow w', s', T} \qquad s' \text{ is } s \text{ with } A\text{-storing cell } l \text{ assigned } V$$

$$\frac{w', s', M[\mathtt{cell}_A l/\mathtt{x}] \Downarrow w', s', T}{w, s, \mathtt{new}\ \mathtt{x} := V;\ M \Downarrow w', s', T} \qquad (w', s') \text{ is } (w, s) \text{ extended with a cell } l \text{ storing } V$$

$$\frac{w, s, M \Downarrow w', s', T}{w, s, \mathtt{if}\ \mathtt{cell}_A l = \mathtt{cell}_A l\ \mathtt{then}\ M\ \mathtt{else}\ M' \Downarrow w', s', T}$$

$$\frac{w, s, M' \Downarrow w', s', T}{w, s, \mathtt{if}\ \mathtt{cell}_A l = \mathtt{cell}_A l'\ \mathtt{then}\ M\ \mathtt{else}\ M' \Downarrow w', s', T}\ (l \neq l')$$

Similarly, we can adapt the CK-machine to include these constructs.

7.3.4 Observational Equivalence

Definition 40 Given two computations $\Gamma \vdash^{\mathsf{c}} M, N : \underline{B}$, we say that $M \simeq N$ when for every ground context $\mathcal{C}[]$ and every world-store w, s and every n we have

$$\exists w', s'(w, s, \mathcal{C}[M] \Downarrow w', s', \mathtt{produce}\ n) \text{ iff } \exists w', s'(w, s, \mathcal{C}[N] \Downarrow w', s', \mathtt{produce}\ n)$$

We similarly define $\simeq$ for values. □

Some important observational equivalences are

$$\mathtt{new}\ \mathtt{x} := V;\ M \simeq M \tag{7.1}$$

$$\mathtt{new}\ \mathtt{x} := V;\ \mathtt{new}\ \mathtt{y} := W;\ M \simeq \mathtt{new}\ \mathtt{y} := W;\ \mathtt{new}\ \mathtt{x} := V;\ M \tag{7.2}$$

$$\mathtt{new}\ \mathtt{x} := V;\ (M\ \mathtt{to}\ \mathtt{y}.\ N) \simeq M\ \mathtt{to}\ \mathtt{y}.\ (\mathtt{new}\ \mathtt{x} := V; N) \tag{7.3}$$

using the conventions of Sect. 1.4.2. The equivalence (7.3) fails in the presence of control effects.

7.4 Thunk-Storage Free Fragment

In Chap. 6, we studied various effects in the absence of divergence. This enabled us to present simple set-based semantics and to avoid a difficult adequacy proof. But this convenient style of exposition is impossible for a language that stores thunks, because, as Landin showed, recursion can be encoded in terms of thunk storage.

We will therefore proceed in two stages.

1. For most of the chapter, we will look at the *thunk-storage free fragment* obtained by restricting the language in Sect. 7.2 to the following types:

$$\begin{array}{lll} D & ::= & \sum_{i \in I} D_i \mid 1 \mid D \times D \mid \mathtt{ref}\ D \\ A & ::= & D \mid U\underline{B} \mid \sum_{i \in I} A_i \mid 1 \mid A \times A \\ \underline{B} & ::= & FA \mid \prod_{i \in I} \underline{B}_i \mid A \to \underline{B} \end{array}$$

   The first line is the class of *data types*; these are the types whose values can be stored. We amend Def. 37–39 accordingly, replacing "value type" by "data type". This restriction of types eliminates divergence:

Proposition 55 For any world-store w, s and any closed w-computation M, there is a unique w', s', T such that $w, s, M \Downarrow w', s', T$. □

2. In Sect. 7.10 we will explain how to adapt our approach to model the whole language, including thunk storage.

Even the thunk-storage free fragment allows storage of cells and is therefore more liberal than languages allowing storage of ground values only.

7.5 Denotation of Values and Stores

7.5.1 Value Types

As we said in Sect. 7.1, a value type A (and likewise a data type and a context) will denote a covariant functor from $\mathcal{W}$ to **Set**. $[\![A]\!]w$ should be thought of as the set of denotations of closed w-values of type A. $\sum$ and $\times$ are interpreted pointwise, while $\texttt{ref}\ D$ is interpreted as in the example of Sect. 7.1.

$$\begin{aligned} [\![\textstyle\sum_{i\in I} A_i]\!]w &= \textstyle\sum_{i\in I} [\![A_i]\!]w \\ [\![A \times A']\!]w &= [\![A]\!]w \times [\![A']\!]w \\ [\![\texttt{ref}\ D]\!]w &= \$w_D \end{aligned}$$

These equations complete the semantics of data types, but for value types we still require the interpretation of U.

Notice that $[\![\texttt{ref}\ D]\!]$ is not defined in terms of $[\![D]\!]$. Therefore, the semantics of types is not compositional. Indeed, even the definition of "world" (Def. 37) involves the set of value types, so it is syntax dependent. This seems to be unavoidable: after all, `ref` does not even preserve isomorphisms i.e. $A \cong A'$ does not imply $\texttt{ref}\ A \cong \texttt{ref}\ A'$.

7.5.2 Values

Suppose we have a value $\Gamma \vdash^{\mathsf{v}} V : A$. For each world w and each environment ρ of closed w-values, we obtain, by substitution, a closed w-value of type A. Thus V will denote, for each world w, a function $[\![V]\!]w$ from $[\![\Gamma]\!]w$ to $[\![A]\!]w$. These functions are related: if $w \leqslant w'$ then (7.4) must commute.

$$\begin{array}{ccc} [\![\Gamma]\!]w & \xrightarrow{[\![V]\!]w} & [\![A]\!]w \\ \Big\downarrow{\scriptstyle [\![\Gamma]\!]^w_{w'}} & & \Big\downarrow{\scriptstyle [\![A]\!]^w_{w'}} \\ [\![\Gamma]\!]w' & \xrightarrow[[\![V]\!]w']{} & [\![A]\!]w' \end{array} \tag{7.4}$$

In summary, V denotes a natural transformation from $[\![\Gamma]\!]$ to $[\![A]\!]$.

Informally, (7.4) says that if we have an environment ρ of closed w-values, substitute into V and then regard the result as a closed w'-value, we obtain the same as if we regard ρ as an environment of closed w'-values and substitute it into V.

More generally, a w_0-value $w_0|\Gamma \vdash^{\mathsf{v}} V : A$ denotes, for each $w \geqslant w_0$ a function from $[\![\Gamma]\!]w$ to $[\![A]\!]w$, satisfying (7.4) for each $w_0 \leqslant w \leqslant w'$. If Γ is empty, it is clear that this is equivalent to giving an element of $[\![A]\!]w_0$, as we stated in Sect. 7.5.1.

7.5.3 Stores

Suppose that w is a world. Then each w-store denotes an element of the set Sw, defined by the finite product

$$Sw = \bigtimes_{(D,l) \in \mathsf{cells}\ w} [\![D]\!]w$$

We use the term *w-store* in two senses: syntactically (a w-store is a tuple of closed w-values of the appropriate types) and semantically (a w-store is an element of Sw). Thus a syntactic w-store denotes a semantic w-store.

The 3 operations defined on syntactic world-stores in Sect. 7.3.2 (reading, assignment and cell generation) have evident analogues for semantic world-stores.

In the thunk-storage free fragment, this syntactic/semantic distinction is of no significance, because each semantic w-store is the denotation of precisely one syntactic w-store. This is a consequence of

Proposition 56 For every data type D and world w, the function $[\![-]\!]$ from the set of closed w-values to to $[\![D]\!]$w is a bijection. □

It is important to see that Sw is not a covariant or contravariant in w. To see this, suppose $w \leqslant w'$.

- On the one hand, a w-store cannot, in general, be extended to a w'-store. For example, suppose w is the empty world and w' has a single cell c storing type 0. Then there is one w-store but no w'-store, because there is no value for c to store.
- On the other hand, a w'-store cannot, in general, be restricted to a w-store. For suppose w has a single cell c storing type `ref` 1 and w' has one more cell d storing type 1. Then there is one w'-store—where c stores d and d stores $()$—but no w-store, because there is no w-value for c to store.

By contrast, when we allow storage of ground values only, S is indeed contravariant: if $w \leqslant w'$, then a w'-store can be restricted to a w-store. This is why our possible world model is so different from the older possible world models, which treat ground store only and exploit the contravariance of S.

We write $\mathsf{disc}\ \mathcal{W}$ for the discrete category (i.e. no non-identity morphisms) whose objects are worlds We say that S is a *discrete functor* (as opposed to a covariant or contravariant functor) from $\mathcal{W}$ to **Set**; this means that it is a functor from $\mathsf{disc}\ \mathcal{W}$ to **Set**.

7.6 Introduction (Part 2)

One of our heuristics for designing a CBPV semantics is to find a CBV semantics and look for the CBPV decomposition of $\to_{\mathbf{CBV}}$. Such a CBV semantics for cell generation was considered by O'Hearn, independently of our work. His suggested interpretation was

$$[\![A \to_{\mathbf{CBV}} B]\!]w = \textstyle\prod_{w' \geqslant w}(Sw' \to [\![A]\!]w' \to \sum_{w'' \geqslant w'}(Sw'' \times [\![B]\!]w'')) \tag{7.5}$$

The intuition behind (7.5) is as follows. Suppose V is a CBV function in world w; it should denote an element of the LHS of (7.5). Now V can be applied in any future world-store (w', s'), where $w' \geqslant w$ and s' is a w'-store, to an operand which is a w'-value of type A. It will then change the world-store to (w'', s''), where $w'' \geqslant w'$ and s'' is a w''-store, and finally produce a w''-value of type B. The RHS of (7.5) precisely describes this narrative.

The CBPV decomposition of $A \to_{\mathbf{CBV}} B$ as $U(A \to FB)$ is immediately apparent in (7.5):

$$\begin{array}{ccccccc} \prod_{w' \geqslant w}(Sw' \to & [\![A]\!]w' & \to & \sum_{w'' \geqslant w'}(Sw'' \times & [\![B]\!]w'')) \\ U & (A & \to & F & B) \end{array}$$

This suggests that a computation type will, like a value type, denote a different set in each world—we shall see in Sect. 7.7.2 that it denotes *contravariant* functor from $\mathcal{W}$ to **Set**—and that these denotations are given by

$$[\![U\underline{B}]\!]w = \prod_{w' \geqslant w}(Sw' \to [\![\underline{B}]\!]w') \tag{7.6}$$

$$[\![A \to \underline{B}]\!]w = [\![A]\!]w \to [\![B]\!]w \tag{7.7}$$

$$[\![FA]\!]w = \sum_{w' \geqslant w}(Sw' \times [\![A]\!]w') \tag{7.8}$$

This can be summarized by the slogan "U denotes $\prod$, F denotes $\sum$". As $U\underline{B}$ is a value type, it must denote a covariant functor from $\mathcal{W}$ to **Set**, and it is easy to see that (7.6) describes one. For if $w \leqslant x$ then any element $\{f_{w'}\}_{w' \geqslant w} \in [\![U\underline{B}]\!]w$ can be restricted to $\{f_{w'}\}_{w' \geqslant x} \in [\![U\underline{B}]\!]x$.

To understand these equations more clearly, let w contain a single number cell l, represented as `cell`$\,0$, and consider the following w-value of type $U(\texttt{ref nat} \to F\texttt{ref nat})$:

```
thunk ( λx.
        read cell 0 as y.
        read x as z.
        cell 0 := z;
        x := y;
        new w := y + (3 × z);
        produce z   )
```

This can be forced in any world-store (w', s'), where $w' \geqslant w$. It first pops a cell l' in w. Then it changes the world-store to (w'', s'') where

- w'' is w' extended with a `nat`-storing cell l'';
- writing y and z for the contents of cells l and l' in s' respectively, s'' is the w''-store in

  which
  l stores z
  l' stores y
  l'' stores $y + 3 \times z$
  every other cell stores what it stored in in s', regarded as a w''-value

We can see from this example

- a w-value of $U\underline{B}$ type (i.e. a thunk in world w) can be forced at any future world (w', s'), where $w' \geqslant w$—this roughly explains (7.6);
- a w-computation of type $A \to \underline{B}$ pops a w-value of type A from the stack and then proceeds as a w-value of type $\underline{B}$—this roughly explains (7.7), but see the discussion in Sect. 7.7.1;
- a w-computation of type FA changes the world-store to (w', s'), where $w' \geqslant w$, and then produces a w'-value of type A—this roughly explains (7.8).

7.7 Denotation of Computations

7.7.1 Semantics of Judgements

The most surprising of our semantic equations is (7.7). The reader (especially if familiar with other possible world models) may wonder as follows.

> Suppose M is a w-computation of type $A \to \underline{B}$ (e.g. the example program in Sect. 7.6, with `thunk` removed). Then, for any $w' \geqslant w$, M can be regarded as a w'-computation and so its operand can be a w'-value. How, then, can M denote just a function from $[\![A]\!]w$, as (7.7) tells us?

The answer is that M does not denote just a function from $[\![A]\!]w$. In this respect, the semantics of computation types is deceptive: whereas $[\![A]\!]w$ is the set of denotations of closed w-values of type A (as we said in Sect. 7.5.1), $[\![\underline{B}]\!]w$ is not the set of denotations of closed w-computations of type $\underline{B}$. The essential information that makes sense of the situation is the semantics of judgements, which we now describe.

Recall from Sect. 6.3.2 that, in the global store model, a computation $\Gamma \vdash^{\mathtt{c}} M : \underline{B}$ denotes a function from $S \times [\![\Gamma]\!]$ to $[\![\underline{B}]\!]$. By contrast, in our possible world model, $[\![M]\!]$ depends not just on the store and environment but also on the world. Therefore, for each w, M denotes a function $[\![M]\!]w$ from $Sw \times [\![\Gamma]\!]w$ to $[\![\underline{B}]\!]w$. Unlike the semantics of values, these functions are not required to be related by any kind of naturality constraint.

More generally, given a w_0-computation $w_0|\Gamma \vdash^{\mathtt{c}} M : \underline{B}$, its denotation provides for each $w \geqslant w_0$ a function $[\![M]\!]w$ from $Sw \times [\![\Gamma]\!]w$ to $[\![\underline{B}]\!]w$. Again, there is no naturality constraint relating these functions for varying w.

We can therefore see that a closed w-computation of type $A \to \underline{B}$ will denote, for each world-store (w', s') where $w' \geqslant w$, a function from $[\![A]\!]w'$ to $[\![\underline{B}]\!]w'$. This answers the reader's question above.

7.7.2 A Computation Type Denotes A Contravariant Functor

If we now attempt to write semantic equations for the various term constructors, all are straightforward except for two: `new` and sequencing. We look at the former. For simplicity, we will suppose that Γ is empty.

Suppose that $\vdash^{\mathtt{v}} V : A$ and $\mathtt{x} : \mathtt{ref}\ A \vdash^{\mathtt{c}} M : \underline{B}$. We wish to describe the denotation of $\mathtt{new}\ \mathtt{x} := V;\ M$ in the world-store (w, s)—this denotation should be an element of $[\![\underline{B}]\!]w$. First, we extend (w, s) with an A-storing cell l initialized to $[\![V]\!]w$, giving a world-store (w', s'). Then, we look at the denotation of M (with $\mathtt{x}$ bound to the new cell l) in the world-store (w', s')—this denotation is an element of $[\![\underline{B}]\!]w'$. How can we obtain an element of $[\![\underline{B}]\!]w$, as required?

The answer is that $[\![\underline{B}]\!]$ must provide extra information. If $w \leqslant w'$, then $[\![\underline{B}]\!]$ must provide a function $[\![\underline{B}]\!]^w_{w'}$ from $[\![\underline{B}]\!]w'$ to $[\![\underline{B}]\!]w$. These functions should respect identity and compositions. In summary, a computation type $\underline{B}$ denotes a *contravariant* functor from $\mathcal{W}$ to **Set**.

This is reminiscent of the structure $*$ in the semantics of printing. There, the role of $*$ in $[\![\underline{B}]\!] = (X, *)$ was to "absorb" printing into computations of type $\underline{B}$. Here, the role of $[\![\underline{B}]\!]^w_{w'}$ is to "absorb" cell generation into computations of type $\underline{B}$.

The contravariant functors denoted by computation types are given as follows.

- We have already said that $[\![FA]\!]w$ is $\sum_{w' \geqslant w}(Sw' \times [\![A]\!]w')$. Consequently, if $w \leqslant x$ then $[\![FA]\!]x \subseteq [\![FA]\!]w$.
- The denotation of $\prod_{i \in I} \underline{B}_i$ is given pointwise. Since each $[\![\underline{B}_i]\!]w$ is contravariant in w it is clear that $\prod_{i \in I} [\![\underline{B}_i]\!]w$ is contravariant in w.
- The denotation of $A \to \underline{B}$ is given pointwise. Since $[\![A]\!]w$ is covariant in w and $[\![\underline{B}]\!]w$ is contravariant in w, it is clear that $[\![A]\!]w \to [\![\underline{B}]\!]w$ is contravariant in w.

7.7.3 Summary

We now summarize the denotational semantics of the thunk-storage free fragment of CBPV with cell generation. The semantics is organized as follows.

- A value type (and likewise a data type and a context) denotes a covariant functor from $\mathcal{W}$ to **Set**.
- A value $\Gamma \vdash^{\mathtt{v}} V : A$ denotes, for each w, a function $[\![V]\!]w$ from $[\![\Gamma]\!]w$ to $[\![A]\!]w$ such that, for $w \leqslant w'$, diagram (7.4) commutes. In other words, $[\![V]\!]$ is a natural transformation from $[\![\Gamma]\!]$ to $[\![A]\!]$.

- A w_0-value $w_0|\Gamma \vdash^{\mathsf{v}} V : A$ denotes, for each $w \geqslant w_0$, a function $[\![V]\!]w$ from $[\![\Gamma]\!]w$ to $[\![A]\!]w$ such that, for $w_0 \leqslant w \leqslant w'$, diagram (7.4) commutes.
- A computation type denotes a contravariant functor from $\mathcal{W}$ to **Set**.
- A computation $\Gamma \vdash^{\mathsf{c}} M : \underline{B}$ denotes, for each w, a function from $Sw \times [\![\Gamma]\!]w$ to $[\![\underline{B}]\!]w$.
- A w_0-computation $w_0|\Gamma \vdash^{\mathsf{c}} M : \underline{B}$ denotes, for each $w \geqslant w_0$ a function from $Sw \times [\![\Gamma]\!]w$ to $[\![\underline{B}]\!]w$.

The semantics of types is given by

$$\begin{array}{rcl}
Sw & = & \bigtimes_{(D,l)\,\in\,\mathsf{cells}\ w} [\![D]\!]w \\
[\![U\underline{B}]\!]w & = & \prod_{w' \geqslant w}(Sw' \to [\![\underline{B}]\!]w') \\
[\![\Sigma_{i\in I} A_i]\!]w & = & \Sigma_{i\in I}[\![A_i]\!]w \\
[\![A \times A']\!]w & = & [\![A]\!]w \times [\![A']\!]w \\
[\![\mathtt{ref}\ D]\!]w & = & \$w_D \\
[\![FA]\!]w & = & \Sigma_{w' \geqslant w}(Sw' \times [\![A]\!]w) \\
[\![\prod_{i\in I}\underline{B}_i]\!]w & = & \prod_{i\in I}[\![\underline{B}_i]\!]w \\
[\![A \to \underline{B}]\!]w & = & [\![A]\!]w \to [\![\underline{B}]\!]w
\end{array}$$

Some examples of semantics of terms:

$$\begin{array}{rcl}
[\![\mathtt{produce}\ V]\!]ws\rho & = & (w, s, [\![V]\!]w\rho) \\
[\![M\ \mathtt{to}\ \mathtt{x}.\ N]\!]ws\rho & = & \\
\multicolumn{3}{c}{\mathsf{pm}\ [\![M]\!]ws\rho\ \mathsf{as}\ (w', s', a).\ [\![\underline{B}]\!]^w_{w'}([\![N]\!]w's'([\![\Gamma]\!]^w_{w'}\rho, \mathtt{x} \mapsto a))} \\
[\![\mathtt{thunk}\ M]\!]w\rho & = & \lambda w'.\lambda s'.([\![M]\!]w's'[\![\Gamma]\!]^w_{w'}\rho) \\
[\![\mathtt{force}\ V]\!]ws\rho & = & s\text{`}w\text{`}\ [\![V]\!]w\rho \\
[\![\lambda \mathtt{x} M]\!]ws\rho & = & \lambda a.([\![M]\!]ws(\rho, \mathtt{x} \mapsto a)) \\
[\![V\text{`}M]\!]ws\rho & = & ([\![V]\!]w\rho)\text{`}([\![M]\!]ws\rho) \\
[\![V := W;\ M]\!]ws\rho & = & [\![M]\!]ws'\rho \\
\multicolumn{3}{c}{\text{where } s' \text{ is } s \text{ with } A\text{-storing cell } [\![V]\!]w\rho \text{ assigned } [\![W]\!]w\rho} \\
[\![\mathtt{read}\ V\ \mathtt{as}\ \mathtt{x}.\ M]\!]ws\rho & = & [\![M]\!]ws(\rho, \mathtt{x} \mapsto a) \\
\multicolumn{3}{c}{\text{where } a \text{ is the contents of } A\text{-storing cell } [\![V]\!]w\rho \text{ in } s} \\
[\![\mathtt{new}\ \mathtt{x} := V;\ M]\!]ws\rho & = & [\![\underline{B}]\!]^w_{w'}[\![M]\!]w's'([\![\Gamma]\!]^w_{w'}\rho, \mathtt{x} \mapsto l) \\
\multicolumn{3}{c}{\text{where } (w', s') \text{ is } (w, s) \text{ extended with a cell } l \text{ storing } [\![V]\!]w'\rho}
\end{array}$$

Proposition 57 (Soundness) Suppose $w, s, M \Downarrow w', s', T$, where M and T have type $\underline{B}$. Then $[\![M]\!]ws = ([\![\underline{B}]\!]^w_{w'})([\![T]\!]w's')$. □

The contravariance of $[\![\underline{B}]\!]$ is essential in formulating this statement, just as, for the printing semantics, the structure map of $[\![\underline{B}]\!]$ is essential in formulating Prop. 17.

Corollary 58 (by Prop. 55) If M is a closed ground w-producer then $w, s, M \Downarrow w', s', \mathtt{produce}\, n$ iff $[\![M]\!]ws = (w', s', n)$. Hence terms with the same denotation are observationally equivalent. □

It is also easy to show that provably equal terms have the same denotation i.e. the model validates the CBPV equations.

7.8 Combining Cell Generation With Other Effects

7.8.1 Introduction

The model for cell generation in Sect. 7.7 generalizes. If we have any CBPV model (more accurately, any adjunction model, as explained in Sect. 14.4.4), we can obtain from it a model for cell generation. We show two examples of this construction: starting with the $\mathcal{A}$-set model for printing, and starting with the Scott model for divergence.

7.8.2 Cell Generation + Printing

When we add both cell generation and printing to CBPV, the big-step semantics will have the form $w, s, M \Downarrow m, w', s', T$. The principal difference between the model in Sect. 7.7 and the model for cell generation + printing is that in the latter a computation type denotes a contravariant functor from $\mathcal{W}$ to $\mathcal{A}$**Set**, the category of $\mathcal{A}$-sets and homomorphisms (rather than to **Set**).

We use infinitely wide CBPV as a metalanguage describing the printing model, as explained in Sect. 3.8. For example,we write FA for the free $\mathcal{A}$-set on the set A, and we write $U\underline{B}$ for the carrier of the $\mathcal{A}$-set $\underline{B}$. With this notation, the semantics of types is given by

$$\begin{array}{rcl}
[\![U\underline{B}]\!]w & = & U\prod_{w' \geqslant w}(Sw' \to [\![\underline{B}]\!]w') \\
[\![FA]\!]w & = & F\sum_{w' \geqslant w}(Sw' \times [\![A]\!]w') \\
[\![\prod_{i\in I}\underline{B}_i]\!]w & = & \prod_{i\in I}[\![\underline{B}_i]\!]w \\
[\![A \to \underline{B}]\!]w & = & [\![A]\!]w \to [\![\underline{B}]\!]w
\end{array}$$

Some examples of semantics of terms:

$$\begin{array}{rcl}
[\![\mathtt{produce}\, V]\!]ws\rho & = & \mathsf{produce}\,(w, s, [\![V]\!]w\rho) \\
[\![M \,\mathtt{to}\, \mathtt{x}.\, N]\!]ws\rho & = & \\
\multicolumn{3}{r}{[\![M]\!]ws\rho\,\mathsf{to}\,(w', s', a).\ [\![\underline{B}]\!]^w_{w'}([\![N]\!]w's'([\![\Gamma]\!]^w_{w'}\rho, \mathtt{x} \mapsto a))} \\
[\![\mathtt{thunk}\, M]\!]w\rho & = & \mathsf{thunk}\,\lambda w'.\lambda s'.([\![M]\!]w's'[\![\Gamma]\!]^w_{w'}\rho) \\
[\![\mathtt{force}\, V]\!]ws\rho & = & s\text{`}w\text{`}\,\mathsf{force}\,[\![V]\!]w\rho \\
[\![\lambda \mathtt{x} M]\!]ws\rho & = & \lambda a.([\![M]\!]ws(\rho, \mathtt{x} \mapsto a)) \\
[\![V\text{`}M]\!]ws\rho & = & ([\![V]\!]w\rho)\text{`}([\![M]\!]ws\rho) \\
[\![V := W;\ M]\!]ws\rho & = & [\![M]\!]ws'\rho \\
\multicolumn{3}{r}{\text{where } s' \text{ is } s \text{ with } A\text{-storing cell } [\![V]\!]w\rho \text{ assigned } [\![W]\!]w\rho} \\
[\![\mathtt{read}\, V \,\mathtt{as}\, \mathtt{x}.\, M]\!]ws\rho & = & [\![M]\!]ws(\rho, \mathtt{x} \mapsto a) \\
\multicolumn{3}{r}{\text{where } a \text{ is the contents of } A\text{-storing cell } [\![V]\!]w\rho \text{ in } s} \\
[\![\mathtt{new}\, \mathtt{x} := V;\ M]\!]ws\rho & = & [\![\underline{B}]\!]^w_{w'}[\![M]\!]w's'([\![\Gamma]\!]^w_{w'}\rho, \mathtt{x} \mapsto l) \\
\multicolumn{3}{r}{\text{where } (w', s') \text{ is } (w, s) \text{ extended with a cell } l \text{ storing } [\![V]\!]w'\rho} \\
[\![\mathtt{print}\, c;\ M]\!]ws\rho & = & c * ([\![M]\!]ws\rho)
\end{array}$$

Proposition 59 (Soundness) Suppose $w, s, M \Downarrow m, w', s', T$, where M and T have type $\underline{B}$. Then $[\![M]\!]ws = m * ([\![\underline{B}]\!]^w_{w'})([\![T]\!]w's')$. □

Corollary 60 (by the analogue of Prop. 55 for printing) If M is a closed ground w-producer then $w, s, M \Downarrow m, w', s', \mathtt{produce}\ n$ iff $[\![M]\!]ws = (m, w', s', n)$. Hence terms with the same denotation are observationally equivalent. □

7.8.3 Cell Generation + Divergence

We add divergence and recursion to the thunk-storage free fragment of CBPV with cell generation. The semantics is organized as follows:

- A data type denotes a covariant functor from $\mathcal{W}$ to **Set**, so Sw is a set for each world w.

- A value type (and likewise a context) denotes a covariant functor from $\mathcal{W}$ to **Cpo**.
- A value $\Gamma \vdash^{\mathsf{v}} V : A$ denotes, for each w, a continuous function $[\![V]\!]w$ from $[\![\Gamma]\!]w$ to $[\![A]\!]w$ such that, for $w \leqslant w'$, diagram (7.4) commutes. In other words, $[\![V]\!]$ is a natural transformation from $[\![\Gamma]\!]$ to $[\![A]\!]$ in the functor category $[\mathcal{W}, \mathbf{Cpo}]$.
- A w_0-value $w_0|\Gamma \vdash^{\mathsf{v}} V : A$ denotes, for each $w \geqslant w_0$, a function $[\![V]\!]w$ from $[\![\Gamma]\!]w$ to $[\![A]\!]w$ such that, for $w_0 \leqslant w \leqslant w'$, diagram (7.4) commutes.
- A computation type denotes a contravariant functor from $\mathcal{W}$ to $\mathbf{Cppo}_{\mathsf{strict}}$ (the category of pointed cpos and strict functions).
- A computation $\Gamma \vdash^{\mathsf{c}} M : \underline{B}$ denotes, for each w, a continuous function from $Sw \times [\![\Gamma]\!]w$ to $[\![\underline{B}]\!]w$.
- A w_0-computation $w_0|\Gamma \vdash^{\mathsf{c}} M : \underline{B}$ denotes, for each $w \geqslant w_0$ a continuous function from $Sw \times [\![\Gamma]\!]w$ to $[\![\underline{B}]\!]w$.

The equations giving the semantics of types and terms are exactly the same as in Sect. 7.8.2, except that we now interpret the metalinguistic CBPV constructs as referring to the Scott model rather than the printing model.

Proposition 61 (Soundness/Adequacy) 1. Suppose $w, s, M \Downarrow w', s', T$, where M and T have type $\underline{B}$. Then $[\![M]\!]ws = ([\![\underline{B}]\!]^w_{w'})([\![T]\!]w's')$.

2. Suppose w, s, M diverges. Then $[\![M]\!]ws = \bot$.

□

Proof (in the style of [Tai67]) (1) is straightforward. For (2), define, by mutual induction over types, three families of relations:
for each A and w, $\leqslant^{\mathsf{v}}_{Aw}$ between $[\![A]\!]w$ and $\mathbb{V}^w_A$;
for each $\underline{B}$ and w, $\leqslant^{\mathsf{t}}_{\underline{B}w}$ between $[\![\underline{B}]\!]w$ and triples w', s, T where $w' \geqslant w$, $s \in Sw'$ and $T \in \mathbb{T}^{w'}_{\underline{B}}$
for each $\underline{B}$ and w, $\leqslant^{\mathsf{c}}_{\underline{B}w}$ between $[\![\underline{B}]\!]w$ and triples w', s, M where $w' \geqslant w$, $s \in Sw'$ and $T \in \mathbb{C}^{w'}_{\underline{B}}$.
The definition of these relations proceeds as follows:

$$
\begin{array}{lll}
a \leqslant^{\mathsf{v}}_{U\underline{B}w} \mathtt{thunk}\, M & \text{iff} & \text{for all } x \geqslant w, s \in Sx, s\text{`}x\text{`}\mathsf{force}\, a \leqslant^{\mathsf{c}}_{\underline{B}x} x, s, M \\
a \leqslant^{\mathsf{v}}_{\sum_{i\in I} A_i w} (\hat{\imath}, V) & \text{iff} & a = (\hat{\imath}, b) \text{ for some } \hat{\imath} \in I \text{ and } b \leqslant^{\mathsf{v}}_{A_{\hat{\imath}} w} V \\
a \leqslant^{\mathsf{v}}_{A\times A' w} (V, V') & \text{iff} & a = (b, b') \text{ for some } b \leqslant^{\mathsf{v}}_{Aw} V \text{ and } b' \leqslant^{\mathsf{v}}_{A' w} V' \\
 & & \\
b \leqslant^{\mathsf{t}}_{FAw} x, s, \mathtt{produce}\, V & \text{iff} & b = \bot \text{ or } b = \mathsf{produce}\,(x, s, a) \text{ for some } a \leqslant^{\mathsf{v}}_{Ax} V \\
f \leqslant^{\mathsf{t}}_{\prod_{i\in I} \underline{B}_i w} x, s, \lambda\{\ldots, i.M_i, \ldots\} & \text{iff,} & \text{for each } \hat{\imath} \in I,\ \hat{\imath}\text{`}f \leqslant^{\mathsf{c}}_{\underline{B}_{\hat{\imath}} w} x, s, M_{\hat{\imath}} \\
f \leqslant^{\mathsf{t}}_{A\to \underline{B}w} x, s, \lambda \mathtt{x}.M & \text{iff,} & \text{for all } a \in [\![A]\!]w, ([\![A]\!]^w_x)a \leqslant^{\mathsf{v}}_{Ax} V \text{ implies} \\
 & & \qquad a\text{`}f \leqslant^{\mathsf{c}}_{\underline{B}w} x, s, M[V/\mathtt{x}] \\
 & & \\
b \leqslant^{\mathsf{c}}_{\underline{B}w} x, s, M & \text{iff} & b = \bot \text{ or } x, s, M \Downarrow x', s', T \text{ and } b \leqslant^{\mathsf{t}}_{\underline{B}w} x', s', T
\end{array}
$$

Note that for terminal T, $b \leqslant^{\mathsf{c}}_{\underline{B}w} x, s, T$ iff $b \leqslant^{\mathsf{t}}_{\underline{B}w} x, s, T$. We prove by mutual induction over types the following:

- For each value $V \in \mathbb{V}^w_A$ the set $\{a \in [\![A]\!] : a \leqslant^{\mathsf{v}}_{Aw} V\}$ is admissible (closed under directed joins).
- [1] If $a \leqslant^{\mathsf{v}}_{Aw} V$ and $a' \leqslant^{\mathsf{v}}_{Aw} V$ and $\mathsf{produce}\, a \leqslant \mathsf{produce}\, a'$ then $a \leqslant a'$.

[1] This clause is used in proving the admissibility property for FA, but it is not really necessary because in the Scott model we immediately have a stronger result: $\mathsf{produce}\, a \leqslant \mathsf{produce}\, a'$ implies $a \leqslant a'$. However, we include this clause here so that the adequacy proof generalizes to other models where the stronger result is not valid.

- For each terminal computation x, s, T the set $\{b \in [\![\underline{B}]\!]w : b \leqslant^{\mathsf{t}}_{\underline{B}} T\}$ is admissible and contains $\bot$.
- For each computation $M \in \mathbb{C}_{\underline{B}}$ the set $\{b \in [\![\underline{B}]\!]w : b \leqslant^{\mathsf{c}}_{\underline{B}} M\}$ is admissible and contains $\bot$.
- If $a \leqslant^{\mathsf{v}}_{Aw} V$ and $w \leqslant x$ then $([\![A]\!]^w_x)a \leqslant^{\mathsf{v}}_{Ax} V$.
- If $b \leqslant^{\mathsf{c}}_{\underline{B}w'} x, s, M$ and $w \leqslant w'$ then $([\![\underline{B}]\!]^w_{w'})b \leqslant^{\mathsf{c}}_{\underline{B}w} x, s, M$.
- If $b \leqslant^{\mathsf{t}}_{\underline{B}w'} x, s, T$ and $w \leqslant w'$ then $([\![\underline{B}]\!]^w_{w'})b \leqslant^{\mathsf{t}}_{\underline{B}w} x, s, T$.

We show that, for any datatype D, $a \vdash^{\mathsf{v}}_{Dw} V$ iff $a = [\![V]\!]w$, by induction on D.

Finally, we show that for any computation $w|A_0, \ldots, A_{n-1} \vdash^{\mathsf{c}} M : \underline{B}$, if $w \leqslant x$, $s \in Sx$ and $a_i \leqslant^{\mathsf{v}}_{A_i x} W_i$ for $i = 0, \ldots, n-1$ then $[\![M]\!]xs\overrightarrow{a_i} \leqslant^{\mathsf{c}}_{\underline{B}x} x, s, M[\overrightarrow{W_i/\mathtt{x}_i}]$; and that for any value $wA_0, \ldots, A_{n-1} \vdash^{\mathsf{v}} V : A$, if $w \leqslant x$ and $a_i \leqslant^{\mathsf{v}}_{A_i x} W_i$ for $i = 0, \ldots, n-1$ then $[\![W]\!]x\overrightarrow{a_i} \leqslant^{\mathsf{v}}_{Ax} V[\overrightarrow{W_i/\mathtt{x}_i}]$. □

Corollary 62 If M is a closed ground w-producer, then $w, s, M \Downarrow w', s', \mathtt{produce}\, n$ iff $[\![M]\!]ws = \mathsf{produce}\,(w', s', n)$, and w, s, M diverges iff $[\![M]\!]ws = \bot$. Hence terms with the same denotation are observationally equivalent. □

It is also easy to show that provably equal terms have the same denotation i.e. the model validates the CBPV equations.

7.9 Related Models and Parametricity

In the models of this chapter, the observational equivalences (7.1) and (7.3) are not validated. (7.2) is validated when V and W have distinct types, but not when V and W have the same type. By contrast, the older models do validate these equivalences, so it is immediately clear that there are substantial differences.

For example, the Idealized Algol model of [Ole82] interprets `comm`—corresponding to[2] the CBPV type $U\underline{\mathtt{comm}}$ of thunks of commands—at w by $Sw \to Sw$. Contrast this with our model which interprets $U\underline{\mathtt{comm}}$ at w by $\prod_{w' \geqslant w}(Sw' \to \sum_{w'' \geqslant w} Sw'')$. There are two differences between these semantics, both a consequence of the fact that in Idealized Algol only ground values can be stored. (Indeed the cell storage model of [Ghi97] provides a semantics of `comm` similar to ours.) To understand these differences, suppose that V is a closed w-value of type $U\underline{\mathtt{comm}}$.

- Our model specifies the behaviour of V when forced in any future world-store (hence the $\prod$), and we have imposed no relationship between the behaviour in different worlds. In the Idealized Algol model, by contrast, if V is forced at (w', s'), its behaviour is determined by the restriction of s' to a w-store. The extra cells will be unaffected, because they cannot be stored in the cells of w. Thus the contravariance of S (mentioned in Sect. 7.5.3) is essential to this model.
- Our model says that, when V is forced, new cells can be generated (hence the $\sum$). In the Idealized Algol model, any new cells can be garbage-collected when the command is completed, because they will not be stored anywhere. This feature is known as the *stack discipline* of Idealized Algol.

[2] Although Idealized Algol is a CBN language, the possible world model of [Ole82] is essentially a model of thunks—this is why this model does not allow direct interpretation of conditionals at all types. Hence a type of this language is interpreted as a U type, and denotes a covariant functor.

It is because of these differences that, in the Idealized Algol model, the naturality condition is required across all *world-injections*—this is explained in [OT95].

Moggi's model [Mog90] for CBV with `ref` 1 uses contravariance of S in a similar way to the Idealized Algol models. But as the language (called ν-calculus in [Sta94]) includes producers other than ground producers (unlike Idealized Algol, where the only producers are commands), there is no stack discipline. So the interpretation of $T1$ (again corresponding to $U\underline{\texttt{comm}}$ in CBPV) at w is $Sw \to \sum_{w' \geqslant w} Sw'$. However, this summation is quotiented by an equivalence relation, and this is how the observational equivalences of Sect. 7.3.4 are validated. Such quotienting is easy when working with sets (as Moggi does), but problematic when working with cpos. For although it is possible to quotient a cpo by an equivalence relation [Jun90], we do not have the simple characterization of elements that we have in the case of sets.

We can recover all of these models of storage by taking the CBV model for storage given in Sect. 7.6 and imposing a weak form of relational parametricity [OT95] called "parametricity in initializations". We will not describe this restricted model here; it does not exhibit a simple CBPV decomposition in the way that the basic model does. (This situation is analogous to that of the CBV model for finite nondeterminism, discussed in Sect. 6.5.3.) Furthermore, as with Moggi's model, it is not simple (although possible) to generalize from sets to cpos, because of the quotienting required.

A rather different possible world model, whose relationship to ours we have not investigated, is Odersky's model for `ref` 1 in a CBN language [Ode94].

7.10 Modelling Thunk Storage And Infinitely Deep Types

We now wish to adapt the model for cell generation + divergence (Sect. 7.8.3) the full language of storage in Sect. 7.2, rather than just the fragment in Sect. 7.4. The difficult part is the semantics of types, which should be organized as follows.

- S is a discrete functor from $\mathcal{W}$ to **SEAM**.
- $[\![A]\!]$ is a covariant functor from $\mathcal{W}$ to **SEAM**, for each value type A.
- $[\![\underline{B}]\!]$ is a contravariant functor from $\mathcal{W}$ to $\mathbf{SE}_{\mathsf{strict}}$, for each computation type $\underline{B}$.

These functors should satisfy the isomorphisms

$$\begin{aligned}
S &\cong \bigtimes_{(A,l) \in \mathsf{cells}} [\![A]\!] \\
[\![U\underline{B}]\!] &\cong U_S[\![\underline{B}]\!] \\
[\![\textstyle\sum_{i\in I} A_i]\!] &\cong \textstyle\sum_{i\in I}[\![A_i]\!] \\
[\![A \times A']\!] &\cong [\![A]\!] \times [\![A']\!] \\
[\![\texttt{ref}\ A]\!] &\cong \mathsf{ref}\ A \\
[\![FA]\!] &\cong F_S[\![A]\!] \\
[\![\textstyle\prod_{i\in I} \underline{B}_i]\!] &\cong \textstyle\prod_{i\in I}[\![\underline{B}_i]\!] \\
[\![A \to \underline{B}]\!] &\cong [\![A]\!] \to [\![\underline{B}]\!]
\end{aligned}$$

where we use the following terminology.

Definition 41 1. For $\underline{X}$ an object of $[\mathcal{W}^{\mathsf{op}}, \mathbf{SE}_{\mathsf{strict}}]$ and S an object of $[\mathsf{disc}\ \mathcal{W}, \mathbf{SEAM}]$, we write $U_S\underline{X}$ for the object of $[\mathcal{W}, \mathbf{SEAM}]$ given at w by $U\prod_{w' \geqslant w}(Sw' \to \underline{X}w')$ and at $\frac{w}{x}$ by the evident restriction.

2. For X an object of $[\mathcal{W}, \mathbf{SEAM}]$ and S an object of $[\mathsf{disc}\ \mathcal{W}, \mathbf{SEAM}]$, we write $F_S X$ for the object of $[\mathcal{W}, \mathbf{SE}_{\mathsf{strict}}]$ given at w by $F\sum_{w' \geqslant w}(Sw' \times Xw')$ and at $\frac{w}{x}$ by the evident inclusion.

3. For a value type A, we write $\mathsf{ref}\ A$ for the object of $[\mathcal{W}, \mathbf{SEAM}]$ given by $\$w_A$ at w and by inclusions at $\overset{w}{w'}$.
4. For $\{X_A\}_{A \in \mathsf{valtypes}}$ a family of objects in $[\mathcal{W}, \mathbf{SEAM}]$ we write $\bigtimes_{(A,l) \in \mathsf{cells}} X_A$ for the object of $[\mathsf{disc}\ \mathcal{W}, \mathbf{SEAM}]$ given at w by $\bigtimes_{(A,l) \in \mathsf{cells}\ w} X_A w$.
5. We write $\sum, \times, \prod, \to$ for the evident pointwise operations on covariant and contravariant functors.

□

Constructing the functors is not simply an application of Prop. 32(4), because $[\mathcal{W}, \mathbf{SEAM}]$ is not enriched-compact. Instead, we use the following.

Proposition 63 Write $[\mathcal{W}, \mathbf{SEAM} \subset \mathbf{SEAM}_{\mathsf{partmin}}]$ for the full subcategory of $[\mathcal{W}, \mathbf{SEAM}_{\mathsf{partmin}}]$ whose objects are functors F from $\mathcal{W}$ to **SEAM** (i.e. $F^w_{w'}$ is total, for $w \leqslant w'$). Then we have the following.

1. $[\mathsf{disc}\ \mathcal{W}, \mathbf{SEAM}_{\mathsf{partmin}}]$ is enriched-compact.
2. $[\mathcal{W}, \mathbf{SEAM} \subset \mathbf{SEAM}_{\mathsf{partmin}}]$ is enriched-compact.
3. $[\mathcal{W}^{\mathsf{op}}, \mathbf{SE}_{\mathsf{strict}}]$ is enriched-compact.

□

Proof (1) and (3) follow from Prop. 32(4). For (2), suppose $D : \mathbb{D} \longrightarrow [\mathcal{W}, \mathbf{SEAM} \subset \mathbf{SEAM}_{\mathsf{partmin}}]^{\mathsf{ep}}$ is a countable directed diagram of (e,p)-pairs. We construct the colimit V as in the proof of Prop. 32(4). Thus for $w \in \mathcal{W}$, the SEAM predomain Vw is defined as a colimit in $\mathbf{SEAM}_{\mathsf{partmin}}$, and for $w \leqslant w'$, the morphism $V^w_{w'}$ is defined to be $\bigvee_{d \in \mathbb{D}} (p_{dw}; D_{d^w_{w'}}; e_{dw'})$. The only extra fact we have to show, by comparison with the proof of Prop. 32(4), is that $V^w_{w'}$ is total, so that V is an object of $[\mathcal{W}, \mathbf{SEAM} \subset \mathbf{SEAM}_{\mathsf{partmin}}]$. We argue as follows. Given $x \in Vw$, we know by (5.2) that

$$\bigvee_{d \in \mathbb{D}} (p_{dw}; e_{dw}) x = x$$

Therefore, for sufficiently large d, p_{dw} is defined at x. Hence, for such d, $p_{dw}; D_{d^w_{w'}}; e_{dw'}$ is defined at x, because $D_{d^w_{w'}}$ is total. So $V^w_{w'}$ is defined at x, as required. □

We are seeking a semantics of types in which

- S is an object of $[\mathsf{disc}\ \mathcal{W}, \mathbf{SEAM}_{\mathsf{partmin}}]$;
- $[\![A]\!]$ is an object of $[\mathcal{W}, \mathbf{SEAM} \subset \mathbf{SEAM}_{\mathsf{partmin}}]$, for each value type A;
- $[\![\underline{B}]\!]$ is an object of $[\mathcal{W}^{\mathsf{op}}, \mathbf{SE}_{\mathsf{strict}}]$, for each computation type $\underline{B}$.

In summary, the semantics of types is given by an object of the enriched-compact category

$$\mathcal{B} = [\mathsf{disc}\ \mathcal{W}, \mathbf{SEAM}_{\mathsf{partmin}}] \times [\mathcal{W}, \mathbf{SEAM} \subset \mathbf{SEAM}_{\mathsf{partmin}}]^{\mathsf{valtypes}} \times [\mathcal{W}^{\mathsf{op}}, \mathbf{SE}_{\mathsf{strict}}]^{\mathsf{comptypes}}$$

We must specify a locally continuous functor from $\mathcal{B}^{\mathsf{op}} \times \mathcal{B}$ to $\mathcal{B}$; the semantics of types will then be given as the canonical fixpoint of this functor. The functor is constructed straightforwardly using the following analogue of Prop. 33.

Proposition 64 • $\{X_A\}_{A \in \mathsf{valtypes}} \mapsto \bigtimes_{(A,l) \in \mathsf{cells}}$ extends canonically to a locally continuous functor from $[\mathcal{W}, \mathbf{SEAM} \subset \mathbf{SEAM}_{\mathsf{partmin}}]^{\mathsf{valtypes}}$ to $[\mathsf{disc}\ \mathcal{W}, \mathbf{SEAM}_{\mathsf{partmin}}]$.

- $(S, \underline{X}) \mapsto U_S\underline{X}$ extends canonically to a locally continuous functor U from $[\mathsf{disc}\ \mathcal{W}, \mathbf{SEAM}_{\mathsf{partmin}}] \times [\mathcal{W}^{\mathsf{op}}, \mathbf{SE}_{\mathsf{strict}}]$ to $[\mathcal{W}, \mathbf{SEAM} \subset \mathbf{SEAM}_{\mathsf{partmin}}]$.
- $\sum_{i \in I}$ extends canonically to a locally continuous functor from $[\mathcal{W}, \mathbf{SEAM} \subset \mathbf{SEAM}_{\mathsf{partmin}}]^I$ to $[\mathcal{W}, \mathbf{SEAM} \subset \mathbf{SEAM}_{\mathsf{partmin}}]$.
- $\times$ extends canonically to a locally continuous functor from $[\mathcal{W}, \mathbf{SEAM} \subset \mathbf{SEAM}_{\mathsf{partmin}}] \times [\mathcal{W}, \mathbf{SEAM} \subset \mathbf{SEAM}_{\mathsf{partmin}}]$ to $[\mathcal{W}, \mathbf{SEAM} \subset \mathbf{SEAM}_{\mathsf{partmin}}]$.
- $(S, X) \mapsto F_S X$ extends canonically to a locally continuous functor F from $[\mathsf{disc}\ \mathcal{W}, \mathbf{SEAM}_{\mathsf{partmin}}] \times [\mathcal{W}, \mathbf{SEAM} \subset \mathbf{SEAM}_{\mathsf{partmin}}]$ to $[\mathcal{W}^{\mathsf{op}}, \mathbf{SE}_{\mathsf{strict}}]$.
- $\prod_{i \in I}$ extends canonically to a locally continuous functor from $[\mathcal{W}^{\mathsf{op}}, \mathbf{SE}_{\mathsf{strict}}]^I$ to $[\mathcal{W}^{\mathsf{op}}, \mathbf{SE}_{\mathsf{strict}}]$.
- $\to$ extends canonically to a locally continuous functor from $[\mathcal{W}, \mathbf{SEAM} \subset \mathbf{SEAM}_{\mathsf{partmin}}]^{\mathsf{op}} \times [\mathcal{W}^{\mathsf{op}}, \mathbf{SE}_{\mathsf{strict}}]$ to $[\mathcal{W}^{\mathsf{op}}, \mathbf{SE}_{\mathsf{strict}}]$.

□

We omit the construction of all these functors, which is tedious. It is clear from Prop. 64 that the isomorphisms above indeed define a functor from $\mathcal{B}^{\mathsf{op}} \times \mathcal{B}$ to $\mathcal{B}$, and we thus obtain our semantics of types.

Having constructed functors satisfying these isomorphisms, we define operations on semantic stores and then define semantics of terms just as in Sect. 7.8.3 (where these isomorphisms are all identities).

Proposition 65 (Soundness/Adequacy) 1. Suppose $w, s, M \Downarrow w', s', T$, where M and T have type $\underline{B}$. Then $[\![M]\!]w[\![s]\!] = ([\![\underline{B}]\!]^w_{w'})([\![T]\!]w'[\![s']\!])$.

2. Suppose w, s, M diverges. Then $[\![M]\!]w[\![s]\!] = \bot$.

□

(1) is straightforward. To prove (2), we adapt the proof of Prop. 61(2). To show the existence of the required logical relation, we apply Pitts' methods [Pit96] (which work for any enriched-compact category) to the category $\mathcal{B}$.

Corollary 66 For a closed ground w-producer M, there exists s' such that $w, s, M \Downarrow w', s', \mathtt{produce}\ n$ iff there exists s' such that $[\![M]\!]w[\![s]\!] = \mathsf{produce}\ (w', s', n)$, and w, s, M diverges iff $[\![M]\!]w[\![s]\!] = \bot$. Hence terms with the same denotation are observationally equivalent. □

Finally, we observe that everything we have done in this section works if, in addition to general storage, we allow infinitely deep types.

Chapter 8

Jump-With-Argument

8.1 Introduction

In Sect. 6.4.5, we presented a continuation semantics for CBPV+control, together with other effects such as printing. But a continuation semantics is far more than just a denotational model. As Steele explained in the CBV setting [Ste78], it provides a *jumping implementation*.

Our continuation semantics used CBPV as a metalanguage and was parametrized in an $\mathcal{A}$-set called $\underline{\texttt{Ans}}$. Thus Sect. 6.4.5 can be seen as defining a syntactic transform from CBPV+control into CBPV + $\underline{\texttt{Ans}}$, where $\underline{\texttt{Ans}}$ is a free computation-type identifier. This transform is called the *outside-passing style* (OPS) transform, because every computation is regarded as taking its outside as a parameter. (In the CBV setting, it is called the *continuation-passing style* (CPS) transform because all outsides are consumers and hence continuations.) Now the range of the OPS transform is not the whole of CBPV + $\underline{\texttt{Ans}}$, but a special fragment which we call *Jump-With-Argument* (JWA). This fragment, which is very similar to Steele's intermediate language and various like calculi [App91, AJ89, Dan92, SF93, Thi97a], is worth separating out from CBPV+$\underline{\texttt{Ans}}$ and studying independently, because it can be regarded as a language of jump instructions. In Steele's words:

> Continuation-passing style, while apparently applicative in nature, admits a peculiarly imperative interpretation[...] As a result, it is easily converted to an imperative machine language. [Ste78]

To summarize: the OPS transform is a transform from CBPV+control to a jumping language called JWA, which is a fragment of CBPV+$\underline{\texttt{Ans}}$.

$$\text{CBPV + control} \xrightarrow[\text{transform}]{\text{OPS}} \text{JWA} \hookrightarrow \text{CBPV + } \underline{\texttt{Ans}}$$

We can add (non-control) effects to this picture, e.g. printing

$$\begin{array}{c}\text{CBPV + control}\\ +\texttt{print}\end{array} \xrightarrow[\text{transform}]{\text{OPS}} \begin{array}{c}\text{JWA}\\ +\texttt{print}\end{array} \hookrightarrow \begin{array}{c}\text{CBPV + } \underline{\texttt{Ans}}\\ +\texttt{print}\end{array}$$

and we will use printing as our example non-control effect throughout this chapter. While some treatments in the literature use divergence, the issues are clearer with printing (as in Sect. 2.2).

The OPS transform translates $\texttt{force}\, V$ and $\texttt{produce}\, V$ into jump commands. It therefore makes apparent the intuitive point that we made in Sect. 1.5.3: that `force` and `produce` are the only two instructions that cause execution to move to elsewhere in the program. This shows once

again the advantage of working with CBPV rather than CBV and CBN, which do not make the flow of control so explicit.

In addition to the jumping operational semantics for JWA, we also describe a more conventional operational semantics based on rewriting and explain their agreement.

We then form an equational theory for JWA and describe its categorical semantics (fortunately much simpler than the categorical semantics of CBPV). Using this equational theory, we see that CBPV+control is equivalent to JWA, along the OPS transform. (This is very similar to the "equational correspondence" result of [SF93].) Consequently models for CBPV+control are essentially the same as models for JWA—a useful fact, because JWA is so much simpler.

Finally, in Sect. 8.10, we explain how JWA can be regarded as a type theory for classical logic. This is based on the classic treatments of [Fri78, Gri90, LRS93]. But we urge the reader, except during Sect. 8.10, to ignore logical issues—in particular, ignore the fact that the symbol $\neg$ for continuation types is the same as the traditional symbol for logical negation.

8.2 The Language

JWA is an extension of Thielecke's "CPS calculus" [Thi97a]. Unlike Thielecke's language, but like the untyped languages of [Dan92, SF93, Ste78], it includes values and abstraction.

The types of JWA are given by

$$A ::= \quad \neg A \mid \textstyle\sum_{i \in I} A_i \mid 1 \mid A \times A \tag{8.1}$$

where each set I is finite (or countable, for *infinitely wide* JWA).

A value of type $\neg A$ is an "A-accepting continuation"—intuitively this means a point that we jump to taking an argument of type A. For example, a BASIC line number would have type $\neg 1$, because the argument is trivial. Although we have followed the established usage of $\neg$ for continuation types, we consider it to be a confusing usage because $\neg$ is not exactly logical negation, as we explain in Sect. 8.10. There are two kinds of judgement

$$\Gamma \vdash^{\mathsf{v}} V : A \qquad\qquad \Gamma \vdash^{\mathsf{n}} M$$

called respectively *values* and *non-returning commands* (in the sense of Sect. 3.9.2—we will sometimes omit the word "non-returning"). The terms of JWA are given in Fig. 8.1. It is convenient to use the syntax $\mathtt{let}\ V\ \mathtt{be}\ \mathtt{x}.\ M$ rather than $\mathtt{let}\ \mathtt{x}\ \mathtt{be}\ V.\ M$.

The embedding of JWA in CBPV+$\underline{\mathtt{Ans}}$ (with printing, if desired) is given in Fig. 8.1. We thus obtain, for each $\mathcal{A}$-set $\underline{\mathtt{Ans}}$, a denotational semantics for JWA + `print`. By using the empty $\mathcal{A}$-set for $\underline{\mathtt{Ans}}$, we can prove

Proposition 67 There is no closed non-returning command in JWA + `print`. □

Because of this situation, we always consider operational semantics for *non-closed* non-returning commands; execution terminates when we attempt to jump to or pattern-match a free identifier. Of course, Prop. 67 becomes false if we add divergence or errors, because `diverge` and $\mathtt{error}\ e$ are both non-returning commands.

8.3 Jumping

8.3.1 Intuitive Reading of Jump-With-Argument

We said in Sect. 8.2 that an A-accepting continuation (a value of type $\neg A$) is a point that we jump to taking an argument of type A. We explain the constructs for $\neg$ in a similar way:

- $V \nearrow W$ is the command "jump to the point W taking argument V".

$$\frac{}{\Gamma, \mathtt{x} : A, \Gamma' \vdash^{\mathsf{v}} \mathtt{x} : A} \qquad \frac{\Gamma \vdash^{\mathsf{v}} V : A \quad \Gamma, \mathtt{x} : A \vdash^{\mathsf{n}} M}{\Gamma \vdash^{\mathsf{n}} \mathtt{let}\ V\ \mathtt{be}\ \mathtt{x}.\ M}$$

$$\frac{\Gamma \vdash^{\mathsf{v}} V : A_{\hat{\imath}}}{\Gamma \vdash^{\mathsf{v}} (\hat{\imath}, V) : \sum_{i \in I} A_i} \qquad \frac{\Gamma \vdash^{\mathsf{v}} V : \sum_{i \in I} A_i \quad \cdots \quad \Gamma, \mathtt{x} : A_i \vdash^{\mathsf{n}} M_i \ \cdots}{\Gamma \vdash^{\mathsf{n}} \mathtt{pm}\ V\ \mathtt{as}\ \{\ldots, (i, \mathtt{x}).M_i, \ldots\}}$$

$$\frac{\Gamma \vdash^{\mathsf{v}} V : A \quad \Gamma \vdash^{\mathsf{v}} V' : A'}{\Gamma \vdash^{\mathsf{v}} (V, V') : A \times A'} \qquad \frac{\Gamma \vdash^{\mathsf{v}} V : A \times A' \quad \Gamma, \mathtt{x} : A, \mathtt{y} : A' \vdash^{\mathsf{n}} M}{\Gamma \vdash^{\mathsf{n}} \mathtt{pm}\ V\ \mathtt{as}\ (\mathtt{x}, \mathtt{y}).M}$$

$$\frac{\Gamma, \mathtt{x} : A \vdash^{\mathsf{n}} M}{\Gamma \vdash^{\mathsf{v}} \gamma \mathtt{x}.M : \neg A} \qquad \frac{\Gamma \vdash^{\mathsf{v}} V : A \quad \Gamma \vdash^{\mathsf{v}} W : \neg A}{\Gamma \vdash^{\mathsf{n}} V \nearrow W}$$

For printing, we add the construct $\dfrac{\Gamma \vdash^{\mathsf{n}} M}{\Gamma \vdash^{\mathsf{n}} \mathtt{print}\ c;\ M}$

To see Jump-With-Argument as a fragment of CBPV+$\underline{\mathtt{Ans}}$, regard

$\neg A$	as	$U(A \to \underline{\mathtt{Ans}})$
a non-returning command	as	a computation of type $\underline{\mathtt{Ans}}$
$\gamma \mathtt{x}.M$	as	$\mathtt{thunk}\ \lambda \mathtt{x}.M$
$V \nearrow W$	as	V‘$\mathtt{force}\ W$

Figure 8.1: Terms of Jump-With-Argument, and its embedding into CBPV+$\underline{\mathtt{Ans}}$

- $\gamma\mathtt{x}.M$ is a point. When we jump to it taking argument V, we bind $\mathtt{x}$ to V and then obey the command M.

The easiest way to grasp this will be to see the execution of an example program. Our explanation of jumping is informal; its aim is to convey the jumping intuitions of JWA.

8.3.2 Graphical Syntax For JWA

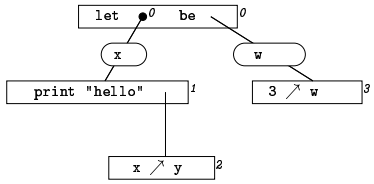
The term syntax (more accurately, the abstract syntax) for JWA is rather inconvenient for jumping around. We therefore use a graphical flowchart-like syntax, illustrated in Fig. 8.2, where

- since γ represents a point, it is written $\bullet$
- each instruction is enclosed in a rectangle;
- binding occurrences of identifiers and patterns are placed on edges (enclosed in a rounded rectangle).

We give the name *jumpabout* to this kind of tree of rectangles, edges and points. (We will not give a precise definition, as our explanation of jumping is informal.)

8.3.3 Execution

We will illustrate execution using the command $\mathtt{y} : \mathtt{nat} \vdash^{\mathtt{n}} M$ where M is the last example in Fig. 8.2. Here is the graphical syntax; we have numbered each point and each rectangle for ease of reference.

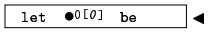


This jumpabout is called the *code*. As we execute it, we form another jumpabout called the *trace*. We call a point in the code jumpabout a *code point*, and a point in the trace a *trace point*; similarly for rectangles. During execution, the code stays fixed but the trace grows. The basic cycle of execution is

- copy the current instruction from the code to the trace;
- obey the current instruction in the trace.

We begin at the top; then we follow the sequence of instructions down the code, except when we jump.

cycle 0 copy instruction We copy the instruction `let • be` from code rectangle 0, giving



Notice that we number each trace point, and indicate in square brackets the code point that it was copied from—the latter is called the *teacher*. Thus code point 0 is the teacher of trace point 0. To reduce clutter, we omit the numbers and teachers of trace rectangles.

We write ◀ for "where we are now".

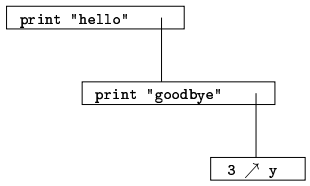
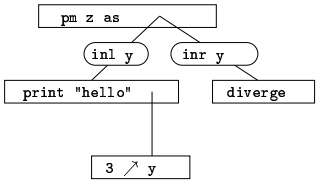
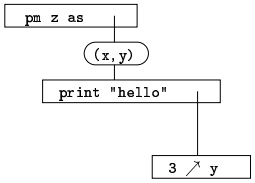
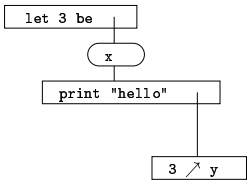
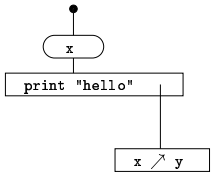
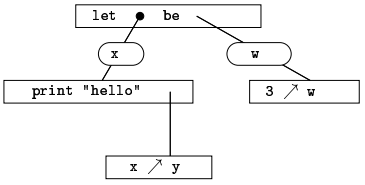
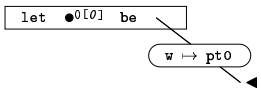
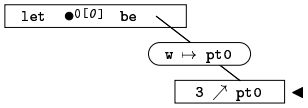
Term Syntax	Graphical Syntax
`print "hello"; print "goodbye";` $3 \nearrow$ `y`	
`pm z as` $\left\{ \begin{array}{l} \texttt{inl y.}\ \ \texttt{print "hello"; } 3 \nearrow \texttt{y} \\ \texttt{inr y.}\ \ \texttt{diverge} \end{array} \right.$	
`pm z as (x,y). print "hello";` $3 \nearrow$ `y`	
`let` 3 `be x. print "hello";` $\texttt{x} \nearrow \texttt{y}$	
$\gamma\texttt{x}.(\texttt{print "hello"; x} \nearrow \texttt{y})$	
`let` $\gamma\texttt{x}.(\texttt{print "hello"; x} \nearrow \texttt{y})$ `be w.` $(3 \nearrow \texttt{w})$	

Figure 8.2: Examples of Graphical Syntax for Jump-With-Argument

obey instruction Now we must obey this instruction, by binding the value `pt0` to `x` giving

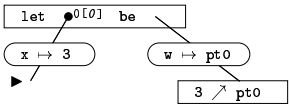


cycle 1 **copy instruction** We copy the instruction 3 ↗ `w` from code rectangle 3 to the trace, giving

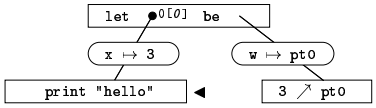


Notice that we have replaced `w` with its binding, point 0. We find this binding by looking up the branch of the trace.

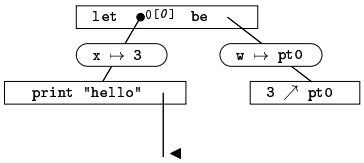
obey instruction The instruction tells us to jump to point 0 (i.e. trace point 0) taking 3 as an argument. We obey it by jumping to trace point 0 and binding `x` to 3, giving



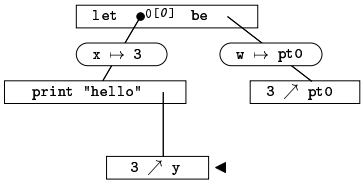
cycle 2 **copy instruction** We copy the next instruction `print "hello"` from code rectangle 1 to the trace, giving



obey instruction We obey this by just printing "`hello`". No binding is made, so we are ready for the next instruction:



cycle 3 **copy instruction** We copy the next instruction `x` ↗ `y` from code rectangle 2 to the trace, giving



Notice that we have replaced `x` with its binding 3. We find this binding by looking up the branch of the trace.

terminate This instruction tells us to jump to `y` taking 3. But because `y` is free, we cannot obey this instruction, so execution terminates.

Notice that the function taking trace points and trace rectangles to their teachers describes a jumpabout homomorphism called the *teacher homomorphism* from the trace to the code. As the trace grows during execution, the teacher homomorphism grows with it.

This example program is very simplistic. More interesting situations arise when we jump several times to the same trace point. Each time, a new branch of the trace comes into existence, and so (unlike in our example) there can be many trace points with the same teacher. This is explained more easily with a live demonstration on a board than on paper.

8.4 The OPS Transform

8.4.1 OPS Transform As Jumping Implementation

The OPS transform from CBPV+control to JWA is given in Fig. 8.3. By composing it with the embedding in Fig. 8.1, we recover the continuation semantics given in Sect. 6.4.5.

As we have described (if only informally) a jumping operational semantics for JWA, we obtain from the OPS transform a jumping implementation of CBPV+control. Notice the following.

- As depicted in Fig. 6.2, there are 2 kinds of values that are translated as continuations: thunks and consumers. This means that thunks and consumers can be regarded as points.
- There are 2 computations that are translated as jumps $\texttt{force}\, V$ (which is a jump to the thunk V) and $\texttt{produce}\, V$ (which is a jump to the current consumer). This makes precise the intuition of Sect. 1.5.3.
- The transform of a computation describes its behaviour in the CK-machine. For instance, the computation $\lambda \mathtt{x}.M$ first pattern-matches its outside as $V :: K$, then binds $\mathtt{x}$ to V, then performs M with outside K. This is exactly described by the transform of $\lambda \mathtt{x}.M$.

8.4.2 Related Transforms

The various CPS transforms that appear in the literature, listed and discussed in [Thi97a], can all be recovered from the OPS transform, because in each case the source language is a fragment of CBPV+control.

- Primary among these transforms is the CBV CPS transform [DHM91, Plo76]. This is recovered from the OPS transform along the embedding of CBV into CBPV. The CBV control operators (as in ML) are translated into CBPV as follows:

$$\begin{array}{rcl} \texttt{cont}\, A & \text{as} & \texttt{os}\, FA \\ \texttt{letcc}\, \mathtt{x}.\, M & \text{as} & \texttt{letcos}\, \mathtt{x}.\, M \\ \texttt{throw}\, M\, N & \text{as} & M\ \texttt{to}\ \mathtt{x}.\, N\ \texttt{to}\ \mathtt{y}.\, (\texttt{changecos}\, \mathtt{x};\ \texttt{produce}\, \mathtt{y}) \end{array}$$

- The "CBN CPS transform" [Plo76] is, in our terminology of Sect. 2.7.3, not CBN but lazy. Thus, as explained in [HD97], it is recovered from the CBV CPS transform via the thunking transform which we present in Sect. A.5. Many of the other CPS transforms listed in [Thi97a] are likewise fragments of CBV.
- The "CBN CPS transform" of [HS97] is in our terminology OPS rather than CPS, but it is genuinely CBN and is recovered from our OPS transform via the embedding of CBN in CBPV.

This provides yet another illustration of the unifying power of working with CBPV—it provides the *source language* for the OPS transform from which all the others are obtained.

A	$\overline{A}$
$U\underline{B}$	$\neg\overline{\underline{B}}$
$\sum_{i\in I}A_i$	$\sum_{i\in I}\overline{A_i}$
$A\times A'$	$\overline{A}\times\overline{A'}$
$\mathtt{os}\ \underline{B}$	$\overline{\underline{B}}$

$\underline{B}$	$\overline{\underline{B}}$
FA	$\neg\overline{A}$
$\prod_{i\in I}\underline{B_i}$	$\sum_{i\in I}\overline{\underline{B_i}}$
$A\to\underline{B}$	$\overline{A}\times\overline{\underline{B}}$
$\underline{\mathtt{nrcomm}}$	1

$A_0,\ldots,A_{n-1}\vdash^{\mathsf{v}} V:B$	$\overline{A_0},\ldots,\overline{A_{n-1}}\vdash^{\mathsf{v}}\overline{V}:\overline{B}$
$\mathtt{x}$	$\mathtt{x}$
$(\hat{\imath},V)$	$(\hat{\imath},\overline{V})$
(V,V')	$(\overline{V},\overline{V'})$
$\mathtt{thunk}\ M$	$\gamma\mathtt{k}.\overline{M}$
$[]\ \mathtt{to}\ \mathtt{x}.\ M :: K$	$\gamma\mathtt{x}.(\overline{M}[\overline{K}/\mathtt{k}])$
$\mathtt{neverused}$	$()$
$\hat{\imath} :: K$	$(\hat{\imath},\overline{K})$
$V :: K$	$(\overline{V},\overline{K})$
Complex Values	
$\mathtt{let}\ \mathtt{x}\ \mathtt{be}\ V.\ W$	$\mathtt{let}\ \overline{V}\ \mathtt{be}\ \mathtt{x}.W$
$\mathtt{pm}\ V\ \mathtt{as}\ \{\ldots,(i,\mathtt{x}).W_i,\ldots\}$	$\mathtt{pm}\ \overline{V}\ \mathtt{as}\ \{\ldots,(i,\mathtt{x}).\overline{W_i},\ldots\}$
$\mathtt{pm}\ V\ \mathtt{as}\ (\mathtt{x},\mathtt{y}).W$	$\mathtt{pm}\ \overline{V}\ \mathtt{as}\ (\mathtt{x},\mathtt{y}).\overline{W}$

$A_0,\ldots,A_{n-1}\vdash^{\mathsf{c}} M:\underline{B}$	$\overline{A_0},\ldots,\overline{A_{n-1}},\mathtt{k}:\overline{\underline{B}}\vdash^{\mathsf{n}}\overline{M}$
$\mathtt{let}\ \mathtt{x}\ \mathtt{be}\ V.\ M$	$\mathtt{let}\ \overline{V}\ \mathtt{be}\ \mathtt{x}.\ \overline{M}$
$\mathtt{pm}\ V\ \mathtt{as}\ \{\ldots,(i,\mathtt{x}).M_i,\ldots\}$	$\mathtt{pm}\ \overline{V}\ \mathtt{as}\ \{\ldots,(i,\mathtt{x}).\overline{M_i},\ldots\}$
$\mathtt{pm}\ V\ \mathtt{as}\ (\mathtt{x},\mathtt{y}).M$	$\mathtt{pm}\ \overline{V}\ \mathtt{as}\ (\mathtt{x},\mathtt{y}).\overline{M}$
$\mathtt{produce}\ V$	$\overline{V}\nearrow\mathtt{k}$
$M\ \mathtt{to}\ \mathtt{x}.\ N$	$\mathtt{let}\ \mathtt{k}\ \mathtt{be}\ \gamma\mathtt{x}.\overline{N}.\ \overline{M}$
$\lambda\{\ldots,i.M_i,\ldots\}$	$\mathtt{pm}\ \mathtt{k}\ \mathtt{as}\ \{\ldots,(i,\mathtt{k}).\overline{M_i},\ldots\}$
$\hat{\imath}\text{`}M$	$\mathtt{let}\ \mathtt{k}\ \mathtt{be}\ (\hat{\imath},\mathtt{k}).\ \overline{M}$
$\lambda\mathtt{x}.M$	$\mathtt{pm}\ \mathtt{k}\ \mathtt{as}\ (\mathtt{x},\mathtt{k}).\overline{M}$
$V\text{`}M$	$\mathtt{let}\ \mathtt{k}\ \mathtt{be}\ (\overline{V},\mathtt{k}).\ \overline{M}$
$\mathtt{force}\ V$	$\mathtt{k}\nearrow\overline{V}$
$\mathtt{coerce}\ M$	$\mathtt{let}\ \mathtt{k}\ \mathtt{be}\ ().\ \overline{M}$
$\mathtt{letcos}\ \mathtt{x}.\ M$	$\mathtt{let}\ \mathtt{x}\ \mathtt{be}\ \mathtt{k}.\ \overline{M}$
$\mathtt{changecos}\ K;\ M$	$\mathtt{let}\ \mathtt{k}\ \mathtt{be}\ \overline{K}.\ \overline{M}$
$\mathtt{print}\ c;\ M$	$\mathtt{print}\ c;\ \overline{M}$

Figure 8.3: The OPS transform from CBPV + control to JWA, with printing

Transitions

$$\begin{array}{ll} & \mathtt{let}\ V\ \mathtt{be}\ \mathtt{x}.M \\ \rightsquigarrow & M[V/\mathtt{x}] \end{array}$$

$$\begin{array}{ll} & \mathtt{pm}\ (\hat{i}, V)\ \mathtt{as}\ \{\ldots, (i,\mathtt{x}).M_i, \ldots\} \\ \rightsquigarrow & M_{\hat{i}}[V/\mathtt{x}] \end{array}$$

$$\begin{array}{ll} & \mathtt{pm}\ (V, V')\ \mathtt{as}\ (\mathtt{x},\mathtt{y}).M \\ \rightsquigarrow & M[V/\mathtt{x}, V'/\mathtt{y}] \end{array}$$

$$\begin{array}{ll} & V \nearrow \gamma\mathtt{x}.M \\ \rightsquigarrow & M[V/\mathtt{x}] \end{array}$$

Terminal Configurations

$$V \nearrow \mathtt{z}$$
$$\mathtt{pm}\ \mathtt{z}\ \mathtt{as}\ \{\ldots, (i,\mathtt{x}).M_i, \ldots\}$$
$$\mathtt{pm}\ \mathtt{z}\ \mathtt{as}\ (\mathtt{x},\mathtt{y}).M$$

Figure 8.4: Rewrite Machine for Jump-With-Argument

Furthermore, the *target language* JWA of the OPS transform is also CBPV, in the sense that it is a fragment of CBPV+$\underline{\mathtt{Ans}}$ (as we explained in Fig. 8.1). Admittedly, the JWA rewrites of Fig. 8.4 can be seen as CBN rewrites or as CBV rewrites—this is essentially the *indifference* result of [Plo76]. But for denotational/equational purposes it is inappropriate to regard JWA as embedded in CBN, because this embedding does not preserve the η-law for sum types. Regarding JWA as embedded in CBV is also somewhat problematic: $\neg A$ is then regarded as

$$A \to_{\mathbf{CBV}} \mathtt{Ans} = U(A \to F\mathtt{Ans})$$

Thus $\underline{\mathtt{Ans}}$ in Fig. 8.1 has been replaced by $F\mathtt{Ans}$, and we have lost generality. In the case of printing, for example, the embedding in Fig. 8.1 provides us with a JWA semantics for *any* $\mathcal{A}$-set, whereas the embedding in CBV allows only a free $\mathcal{A}$-set.

In summary: we have benefited from ensuring that both the source language and the target language of the OPS transform are CBPV.

8.5 Rewrite Machine

8.5.1 Rewrite Machine for Effect-Free JWA

Whilst the jumping operational semantics for JWA is intuitive, it is useful to have also a more conventional operational semantics based on rewriting. The *rewrite machine* is given in Fig. 8.4. It is similar to the CK-machine, except that there is no need for a stack. Recalling from Sect. 8.2 that we work with non-closed non-returning commands, we fix a context Γ, and define a Γ-*configuration* to be a non-returning command $\Gamma \vdash^{\mathtt{n}} M$.

Proposition 68 (deterministic subject reduction) (cf. Prop. 41) For every Γ-configuration M, precisely one of the following holds.

1. M is not terminal, and $M \rightsquigarrow N$ for unique N. N is a Γ-configuration.

2. M is terminal, and there does not exist N such that $M \rightsquigarrow N$.

□

Definition 42 (cf. Prop. 16) We define the relation $\rightsquigarrow^*$ on configurations inductively:

$$\frac{}{M \rightsquigarrow^* M} \qquad \frac{M' \rightsquigarrow^* N}{M \rightsquigarrow^* N}\,(M \rightsquigarrow M')$$

□

Proposition 69 (cf. 42) For every Γ configuration M there is a unique terminal Γ-configuration N such that $M \rightsquigarrow^* N$, and there is no infinite sequence of transitions from M. □

Proof (in the style of [Tai67]) We fix Γ. We say that a Γ-configuration M with the properties described in Prop. 42 is *reducible*. Thus our aim is to prove that every Γ-configuration is reducible.

For each type A we define a set $\mathsf{red}^{\mathsf{v}}_A$ of *reducible* values $\Gamma \vdash^{\mathsf{v}} V : A$ by induction on types.

- $\Gamma \vdash^{\mathsf{v}} V : \neg A$ is reducible iff V is either a free identifier or $\gamma\mathtt{x}.M$ where for all $V \in \mathsf{red}^{\mathsf{v}}_A$, $M[V/\mathtt{x}]$ is reducible.
- $\Gamma \vdash^{\mathsf{v}} V : \sum_{i\in I} A_i$ is reducible iff V is either a free identifier or $(\hat{\imath}, W)$ where $W \in \mathsf{red}^{\mathsf{v}}_{A_i}$.
- $\Gamma \vdash^{\mathsf{v}} V : A \times A'$ is reducible iff V is either a free identifier or (W, W') where $W \in \mathsf{red}^{\mathsf{v}}_A$ and $W' \in \mathsf{red}^{\mathsf{v}}_{A'}$.

By mutual induction on M and V, we prove the following.

- For any non-returning command $\Gamma, A_0, \ldots, A_{n-1} \vdash^{\mathsf{n}} M$, and any $W_0 \in \mathsf{red}^{\mathsf{v}}_{A_0}, \ldots, W_{n-1} \in \mathsf{red}^{\mathsf{v}}_{A_{n-1}}$, the Γ-configuration $M[\overrightarrow{W_i/\mathtt{x}_i}]$ is reducible.
- For any value $\Gamma, A_0, \ldots, A_{n-1} \vdash^{\mathsf{v}} V : B$, and any $W_0 \in \mathsf{red}^{\mathsf{v}}_{A_0}, \ldots, W_{n-1} \in \mathsf{red}^{\mathsf{v}}_{A_{n-1}}$, the value $V[\overrightarrow{W_i/\mathtt{x}_i}]$ is reducible.

□

8.5.2 Adapting the Rewrite Machine for `print`

In Fig. 8.4 a transition has the form

$$M \qquad \rightsquigarrow \qquad M' \tag{8.2}$$

When we add `print` to the language, we want a transition to have the form

$$M \qquad \rightsquigarrow \qquad m \qquad M'$$

for some $m \in \mathcal{A}^*$. We therefore replace each transition (8.2) in Fig. 8.4 by

$$M \qquad \rightsquigarrow \qquad 1 \qquad M'$$

and we add the transition

$$\mathtt{print}\ c;\ M \qquad \rightsquigarrow \qquad c \qquad M$$

We replace Def. 42 by the following.

Definition 43 We define the relation $\rightsquigarrow^*$, whose form is $M \rightsquigarrow^* m, M'$ inductively:

$$\frac{}{M \rightsquigarrow^* 1, M} \qquad \frac{M' \rightsquigarrow^* n, N}{M \rightsquigarrow^* m * n, N}\ (M \rightsquigarrow m, M')$$

□

It is easy to prove analogues of Prop. 68–69.

Proposition 70 (soundness) If $M \rightsquigarrow^* m, T$ then, for any $\rho \in [\![\Gamma]\!]$,

$$[\![M]\!]\rho = m * ([\![T]\!]\rho)$$

□

Corollary 71 Suppose $\underline{\mathsf{Ans}}$ has two elements a, b with the property thatat

$$\begin{array}{lcll} m * a & \neq & m' * a & \text{for } m \neq m' \in \mathcal{M} \\ m * a & \neq & m' * b & \text{for } m, m' \in \mathcal{M} \end{array}$$

(This property is satisfied by the free $\mathcal{A}$-set on a set of size $\geqslant 2$.) For a set N and element n, write $I_{N,n}$ for the function from N to $\underline{\mathsf{Ans}}$ that takes n to a and everything else to b.

Then for any non-returning command $\mathtt{k} : \neg\sum_{i \in N} 1 \vdash^{\mathsf{n}} M$, we have $M \rightsquigarrow^* m, n \nearrow \mathtt{k}$ iff $[\![M]\!](\mathtt{k} \mapsto I_{N,n}) = m * a$. Hence terms with the same denotation are observationally equivalent. □

8.6 Relating Operational Semantics

We now have 3 operational semantics, each of which can be extended for `print`:

1. the CK-machine for CBPV+control,
2. the rewrite machine for JWA,
3. the jumping machine for JWA (which we have described only informally).

The OPS transform exactly preserves machine progress, from (1) to (2):

Proposition 72 Let M, K be a Γ-configuration of the CK-machine for CBPV + control. Then $\overline{M}[\overline{K}/\mathtt{k}]$ is a $\overline{\Gamma}$-configuration of the rewrite machine. Furthermore:

- M, K is terminal iff $\overline{M}[\overline{K}/\mathtt{k}]$ is terminal.
- If $M, K \rightsquigarrow N, L$, then $\overline{M}[\overline{K}/\mathtt{k}] \rightsquigarrow \overline{N}[\overline{L}/\mathtt{k}]$.

□

This enables us to deduce the termination of the CK-machine (Prop. 42 and the weaker Prop. 11) from the termination of the JWA rewrite machine (Prop. 69).

Likewise, the relationship between the (2) and (3) is exact: one transition of the rewrite machine corresponds to one cycle of the jumping machine. (We do not prove this here.) For example, the computation in Sect. 8.3.3 rewrites in 3 transitions to $3 \nearrow \mathtt{y}$, printing `hello` in the last transition. These transitions correspond to cycle 0, cycle 1 and cycle 2 in the jumping execution.

8.7 Equations

8.7.1 Observational Equivalence

Definition 44 A *ground context* is a non-returning command $\mathtt{k} : \neg\sum_{i\in I} 1 \vdash^{\mathsf{n}} \mathcal{C}$ with one or more occurrences of a hole which may be a non-returning command or a value. □

Definition 45 Let $\Gamma \vdash^{\mathsf{n}} M$ and $\Gamma \vdash^{\mathsf{n}} N$ be non-returning commands. We say that $M \simeq N$ when for any ground context $\mathcal{C}$

$$\mathcal{C}[M] \rightsquigarrow^* T \text{ iff } \mathcal{C}[N] \rightsquigarrow^* T$$

Similarly for values. □

We can easily adapt Def. 45 to different computational effects, e.g. printing.

8.7.2 Equational Theory

Before formulating the equational theory, we add the following complex values to JWA:

$$\frac{\Gamma \vdash^{\mathsf{v}} V : A \quad \Gamma, \mathtt{x} : A \vdash^{\mathsf{v}} W : B}{\Gamma \vdash^{\mathsf{v}} \mathtt{let}\ V\ \mathtt{be}\ \mathtt{x}.\ W : B}$$

$$\frac{\Gamma \vdash^{\mathsf{v}} V : \sum_{i\in I} A_i \quad \cdots \quad \Gamma, \mathtt{x} : A_i \vdash^{\mathsf{v}} W_i : B \quad \cdots}{\Gamma \vdash^{\mathsf{v}} \mathtt{pm}\ V\ \mathtt{as}\ \{\ldots, (i,\mathtt{x}).W_i, \ldots\} : B}$$

$$\frac{\Gamma \vdash^{\mathsf{v}} V : A \times A' \quad \Gamma, \mathtt{x} : A, \mathtt{y} : A' \vdash^{\mathsf{v}} W : B}{\Gamma \vdash^{\mathsf{v}} \mathtt{pm}\ V\ \mathtt{as}\ (\mathtt{x},\mathtt{y}).W : B}$$

As in Sect. 4.2, we exclude these values when considering operational semantics, but include them otherwise. As with CBPV, it can be shown that the notion of observational equivalence for complex-value-free terms is unaffected by the presence of complex values in ground contexts.

β**-laws**

$$\begin{array}{rcl}
\mathtt{let}\ V\ \mathtt{be}\ \mathtt{x}.\ M & = & M[V/\mathtt{x}] \\
\mathtt{let}\ V\ \mathtt{be}\ \mathtt{x}.\ W & = & W[V/\mathtt{x}] \\
\mathtt{pm}\ (\hat{\imath}, V)\ \mathtt{as}\ \{\ldots, (i,\mathtt{x}).M_i, \ldots\} & = & M_{\hat{\imath}}[V/\mathtt{x}] \\
\mathtt{pm}\ (\hat{\imath}, V)\ \mathtt{as}\ \{\ldots, (i,\mathtt{x}).W_i, \ldots\} & = & W_{\hat{\imath}}[V/\mathtt{x}] \\
\mathtt{pm}\ (V, V')\ \mathtt{as}\ (\mathtt{x},\mathtt{y}).M & = & M[V/\mathtt{x}, V'/\mathtt{y}] \\
\mathtt{pm}\ (V, V')\ \mathtt{as}\ (\mathtt{x},\mathtt{y}).W & = & W[V/\mathtt{x}, V'/\mathtt{y}] \\
V \nearrow \gamma\mathtt{x}.M & = & M[V/\mathtt{x}]
\end{array}$$

η**-laws**

$$\begin{array}{rcl}
M[V/\mathtt{z}] & = & \mathtt{pm}\ V\ \mathtt{as}\ \{\ldots, (i,\mathtt{x}).M[(i,\mathtt{x})/\mathtt{z}], \ldots\} \\
W[V/\mathtt{z}] & = & \mathtt{pm}\ V\ \mathtt{as}\ \{\ldots, (i,\mathtt{x}).W[(i,\mathtt{x})/\mathtt{z}], \ldots\} \\
M[V/\mathtt{z}] & = & \mathtt{pm}\ V\ \mathtt{as}\ (\mathtt{x},\mathtt{y}).M[(\mathtt{x},\mathtt{y})/\mathtt{z}] \\
W[V/\mathtt{z}] & = & \mathtt{pm}\ V\ \mathtt{as}\ (\mathtt{x},\mathtt{y}).W[(\mathtt{x},\mathtt{y})/\mathtt{z}] \\
V & = & \gamma\mathtt{x}.(\mathtt{x} \nearrow V)
\end{array}$$

Figure 8.5: JWA equations, using conventions of Sect. 1.4.2

In addition to the usual reversible derivations for $\times$ and $\sum$, the JWA equational theory satisfies the following reversible derivation for $\neg$:

$$\frac{\Gamma, A \vdash^{\mathsf{n}}}{\Gamma \vdash^{\mathsf{v}} \neg A}$$

As with CBPV (Prop. 26), complex values can be eliminated from non-returning commands and from closed values.

8.7.3 Type Canonical Forms

We saw in Sect. 4.7.1 that two thunks can be coalesced into a single thunk. Similarly, two continuations can be coalesced into a single continuation:

$$\neg A \times \neg A' \cong \neg(A + A') \tag{8.3}$$

To make this precise, we need to define syntactic isomorphism.

Definition 46 An *isomorphism* between value types A and B is a reversible derivation θ

$$\frac{\Gamma \vdash^{\mathsf{v}} A}{\Gamma \vdash^{\mathsf{v}} B}$$

that preserves substitution in Γ. (By the Yoneda embedding, this definition could be replaced by a characterization using terms $A \vdash^{\mathsf{v}} V : B$ and $B \vdash^{\mathsf{v}} W : A$ inverse up to provable equality.) □

Now (8.3) can be given as the composite reversible derivation

$$\dfrac{\dfrac{\dfrac{\dfrac{\Gamma \vdash^{\mathsf{v}} \neg A \times \neg A'}{\Gamma \vdash^{\mathsf{v}} \neg A \quad \Gamma \vdash^{\mathsf{v}} \neg A'}}{\Gamma, A \vdash^{\mathsf{n}} \quad \Gamma, A' \vdash^{\mathsf{n}}}}{\Gamma, A + A' \vdash^{\mathsf{n}}}}{\Gamma \vdash^{\mathsf{v}} \neg(A + A')}$$

which commmutes with substitution in Γ because each of its factors does.

As in Sect. 4.7.3, we say that JWA types in the following inductively defined class are called *type canonical forms*

$$A ::= \quad \Sigma_{i \in I} \neg A_i$$

Explicitly, a type canonical form can be written

$$\Sigma_{i \in I} \neg \Sigma_{j \in J_i} \neg \Sigma_{k \in K_{ij}} \neg \Sigma_{l \in L_{ijk}} \neg \cdots \tag{8.4}$$

Proposition 73 Every JWA type is isomorphic to a type canonical form. □

This is easily proved, by induction over types. We explain it as follows. Every closed value V is a tuple (more accurately, a hereditary tuple) of tags and continuations. All the tags can be coalesced into a single tag, and we have just seen that all the continuations can be coalesced into a single continuation. Consequently, V corresponds to a pair $(i, \gamma \mathtt{x}.M)$.

8.8 Categorical Semantics[1]

8.8.1 Non-Return Models

There are two approaches to categorical semantics for JWA:

1. axiomatizing $\neg$ directly, following Thielecke [Thi97b, Thi97a];
2. the traditional approach using an exponentiating object [Hof95, SR98].

[1] This section relies on Chap. 10.

Approach (1), which we prefer, is organized as follows.

Definition 47 A *non-return model* for JWA consists of

- a countably distributive category $\mathcal{C}$ (the "value category")
- a functor $\mathcal{G}$ from $\mathcal{C}^{\mathsf{op}}$ to **Set**
- for each $A \in \mathsf{ob}\ \mathcal{C}$, a representation for the functor $\lambda\Gamma.\mathcal{G}(\Gamma \times A)$ from $\mathcal{C}^{\mathsf{op}}$ to **Set**, whose vertex we call $\neg A$—explicitly, this is an isomorphism

$$\mathcal{G}(\Gamma \times A) \cong \mathcal{C}(\Gamma, \neg A) \qquad \text{natural in } \Gamma \tag{8.5}$$

□

We call an element g of $\mathcal{G}\Gamma$ a *non-returning morphism* from Γ, and we write $\Gamma \xrightarrow{\ g\ }$. The functoriality of $\mathcal{G}$ gives us composition for non-returning morphisms: for each $\mathcal{C}$-morphism $\Gamma' \xrightarrow{\ f\ } \Gamma$ and a non-returning morphism $\Gamma \xrightarrow{\ g\ }$, we define the *composite* non-returning morphism $\Gamma' \xrightarrow{\ f;g\ }$ to be $(\mathcal{G}f)g$. This operation satisfies identity and associativity laws

$$\begin{aligned} \mathsf{id};g &= g \\ (f;f');g &= f;(f';g) \end{aligned}$$

where g is a non-returning morphism. Conversely, this operation and these equations give us a functor $\mathcal{G}$ from $\mathcal{C}^{\mathsf{op}}$ to **Set**.

It is easy to give semantics of JWA in a non-return model: values are interpreted in $\mathcal{C}$, and a non-returning command $\Gamma \vdash^{\mathsf{n}} M$ denotes a non-returning morphism from $[\![\Gamma]\!]$. The isomorphism (8.5) corrresponds to the reversible derivation for $\neg$. Indeed we can say

Proposition 74 Models of the JWA equational theory and non-return models are equivalent. □

We can make this precise and prove it in the same way that we make Prop. 97 precise and prove it (in outline) in Sect. 10.4.4.

8.8.2 Examples of Non-Return Models

There are two significant constructions that yield non-return models: the $\underline{\mathsf{Ans}}$ construction and the families construction.

The $\underline{\mathsf{Ans}}$ Construction

We can construct a non-return model from a CBPV model with a chosen computation object $\underline{\mathsf{Ans}}$. To explain this, we will use the value/producer models of Chap. 13, but any of the other categorical semantics for CBPV would serve as well. Suppose we have a CBPV value/producer model $(\mathcal{C}, \mathcal{E}, \iota, \dots)$ with a chosen computation object $\underline{\mathsf{Ans}}$. Then we construct a non-return model $(\mathcal{C}, \mathcal{G}, \dots)$ by setting $\mathcal{G}$ to be $\mathcal{E}(-, \underline{\mathsf{Ans}})$. We set $\neg A$ to be $U(A \to \underline{\mathsf{Ans}})$, with the isomorphism

$$\mathcal{E}(\Gamma \times A, \underline{\mathsf{Ans}}) \cong \mathcal{C}(\Gamma, U(A \to \underline{\mathsf{Ans}}))$$

The semantics for JWA+`print` in Sect. 8.2 is obtained by applying this construction to the printing model for CBPV.

Pointer Game Model: The Families Construction

The pointer game model for JWA given in Sect. 9.3 is a non-return model obtained using a construction called *families*, based on [AM98a]. We start from the following structure.

Definition 48 A *pre-families non-return model* consists of

- a cartesian category $\acute{\mathcal{C}}$
- a functor $\acute{\mathcal{G}}$ from $\acute{\mathcal{C}}^{\mathsf{op}}$ to **Set**
- for each countable family of $\acute{\mathcal{C}}$-objects $\{R_i\}_{i\in I}$ a representation for the functor $\lambda\Gamma.\prod_{i\in I}\acute{\mathcal{G}}(\Gamma\times R_i)$, whose vertex we call $\neg_{i\in I}R_i$—explicitly, this is an isomorphism

$$\prod_{i\in I}\acute{\mathcal{G}}(\Gamma\times R_i)\cong\acute{\mathcal{C}}(\Gamma,\neg_{i\in I}R_i)\text{ natural in }\Gamma$$

□

Definition 49 (families construction) Let $(\acute{\mathcal{C}},\acute{\mathcal{G}},\dots)$ be a pre-families non-return model. We obtain a non-return model $(\mathcal{C},\mathcal{G},\dots)$ as follows.

- A $\mathcal{C}$-object is a countable family of $\acute{\mathcal{C}}$-objects.
- The $\mathcal{C}$ homset from $\{R_i\}_{i\in I}$ to $\{S_j\}_{j\in J}$ is the set $\prod_{i\in I}\Sigma_{j\in J}\acute{\mathcal{C}}(R_i,S_j)$ with the evident identities and composition.
- Thus $\mathcal{C}$ is countably distributive, with

$$\begin{array}{rl}\{R_i\}_{i\in I}\times\{S_j\}_{j\in J} & \text{given by}\quad \{R_i\times S_j\}_{(i,j)\in I\times J}\\ \Sigma_{i\in I}\{R_{ij}\}_{j\in J_i} & \text{given by}\quad \{R_{ij}\}_{(i,j)\in\,\sum_{i\in I}J_i}\end{array}$$

- The $\mathcal{G}$ homset from $\{R_i\}_{i\in I}$ is $\prod_{i\in I}\acute{\mathcal{G}}R_i$ with the evident composition.
- This gives a non-return model, with

$$\neg\{R_i\}_{i\in I}\quad\text{given by the singleton family}\quad\{\neg_{i\in I}R_i\}$$

□

The pointer game model of Sect. 9.3 is obtained by applying the families construction to the following pre-families non-return model. We describe the homsets here; the operations on strategies (e.g. composition) are described in Sect. B.5.4.

- An object of $\acute{\mathcal{C}}$ is an unlabelled arena.
- A $\acute{\mathcal{C}}$-morphism from R to S is an O-first strategy from R^{P} to S^{O}.
- $R\times R'$ is given as $R\uplus R'$.
- A $\acute{\mathcal{G}}$-morphism from R (i.e. an element of $\acute{\mathcal{G}}A$) is a P-first strategy on R^{P}.
- $\neg_{i\in I}R_i$ is given as $\mathsf{pt}_{i\in I}R_i$.

Similar pre-families models are obtained by imposing constraints of visibility, innocence etc.

8.8.3 Exponentiating Object Models

The traditional approach (2) to categorical semantics for JWA is formulated as follows.

Definition 50 1. An *exponentiating object* in a cartesian category $\mathcal{C}$ is an object R together with an exponent from X to R for every $\mathcal{C}$-object X.

2. An *exponentiating object model* for JWA is a countably distributive category $\mathcal{C}$ together with an exponentiating object. For each object X, we write $\neg X$ for the vertex of the given exponent from X to R.

□

It is easy to see that exponentiating object models and non-return models are equivalent:

- if we have an exponentiating object model, we obtain a non-return model by setting $\mathcal{G}\Gamma$ to be $\mathcal{C}(\Gamma, R)$;
- if we have a non-return model, we obtain an exponentiating object model by setting R to be $\neg 1$;
- these constructions are inverse, up to a suitable notion of isomorphism.

Warning If we obtain a JWA model from a CBPV model with computation object $\underline{\mathsf{Ans}}$, the exponentiating object R is the value object $U\underline{\mathsf{Ans}}$, so it should not be confused with $\underline{\mathsf{Ans}}$. In the example of printing, R is the carrier of the $\mathcal{A}$-set $\underline{\mathsf{Ans}}$, and so it does not provide sufficient information to interpret `print`, which requires the structure $*$ of $\underline{\mathsf{Ans}}$.

8.9 The OPS Transform Is An Equivalence

8.9.1 The Main Result

In Fig. 8.3 we presented the OPS transform from CBPV+control to JWA. In this section, we show that it is an equivalence, so models of CBPV+control and models of JWA are essentially the same thing. However, JWA is much simpler and more elegant.

The sense in which the OPS transform is an equivalence is the following—a typed variant of the "equational correspondence" approach of [SF93]:

Proposition 75 1. Every JWA type A is isomorphic to $\overline{B}$ for some value type B.

2. Every JWA type A is isomorphic to $\overline{\underline{B}}$ for some computation type $\underline{B}$.

3. For every context Γ and value type A in CBPV+control, the OPS transform defines a bijection from the provable-equality classes of values $\Gamma \vdash^{\mathsf{v}} V : A$ to the provable-equality classes of values $\overline{\Gamma} \vdash^{\mathsf{v}} W : \overline{A}$.

4. For every context Γ and computation type $\underline{B}$ in CBPV+control, the OPS transform defines a bijection from the provable-equality classes of computations $\Gamma \vdash^{\mathsf{c}} M : \underline{B}$ to the provable-equality classes of non-returning commands $\overline{\Gamma}, \mathtt{k} : \overline{\underline{B}} \vdash^{\mathsf{n}} N$.

(3)–(4) can be extended to values and computations with holes i.e. contexts. □

For Prop. 75 to make sense, we will have to give an equational theory for CBPV+control. We do this in Sect. 8.9.2. We then prove Prop. 75 in Sect. 8.9.3.

Using Prop. 72, we deduce

Corollary 76 The OPS transform is fully abstract. □

8.9.2 Equational Theory For CBPV + Control

We want to contrive an equational theory for CBPV+control that will make Prop. 75 true. Before we do this, we add some useful syntax:

$$\frac{\Gamma \vdash^{\mathtt{v}} V : \mathtt{os} \prod_{i\in I} \underline{B}_i \quad \cdots \quad \Gamma, \mathtt{y} : \mathtt{os}\ \underline{B}_i \vdash^{\mathtt{c}} M_i : \underline{C} \quad \cdots}{\Gamma \vdash^{\mathtt{c}} \mathtt{pm}\ V\ \mathtt{as}\ \{\ldots, i :: \mathtt{y}.M_i, \ldots\} : \underline{C}}$$

$$\frac{\Gamma \vdash^{\mathtt{v}} V : \mathtt{os}\ (A \to \underline{B}) \quad \Gamma, \mathtt{x} : A, \mathtt{y} : \mathtt{os}\ \underline{B} \vdash^{\mathtt{c}} M : \underline{C}}{\Gamma \vdash^{\mathtt{c}} \mathtt{pm}\ V\ \mathtt{as}\ \mathtt{x} :: \mathtt{y}.M : \underline{C}}$$

These constructs can be desugared as follows:

sugar	unsugared
$\mathtt{pm}\ V\ \mathtt{as}\ \{\ldots, i :: \mathtt{y}.M_i, \ldots\}$	$\mathtt{letcos\ z.\ changecos}\ V;\ \lambda\{\ldots, i.\ \mathtt{letcos\ y.\ changecos\ z};\ M_i, \ldots\}$
$\mathtt{pm}\ V\ \mathtt{as}\ \mathtt{x} :: \mathtt{y}.M$	$\mathtt{letcos\ z.\ changecos}\ V;\ \lambda \mathtt{x}.\ \mathtt{letcos\ y.\ changecos\ z}.\ M$

We then add complex values corresponding to these constructs:

$$\frac{\Gamma \vdash^{\mathtt{v}} V : \mathtt{os} \prod_{i\in I} \underline{B}_i \quad \cdots \quad \Gamma, \mathtt{y} : \mathtt{os}\ \underline{B}_i \vdash^{\mathtt{v}} W_i : C \quad \cdots}{\Gamma \vdash^{\mathtt{v}} \mathtt{pm}\ V\ \mathtt{as}\ \{\ldots, i :: \mathtt{y}.W_i, \ldots\} : C}$$

$$\frac{\Gamma \vdash^{\mathtt{v}} V : \mathtt{os}\ (A \to \underline{B}) \quad \Gamma, \mathtt{x} : A, \mathtt{y} : \mathtt{os}\ \underline{B} \vdash^{\mathtt{v}} W : C}{\Gamma \vdash^{\mathtt{v}} \mathtt{pm}\ V\ \mathtt{as}\ \mathtt{x} :: \mathtt{y}.W : C}$$

As usual, these can be eliminated from any computation or closed values, and do not affect observational equivalence for complex-value-free terms.

The equational theory for CBPV+control consists of the usual CBPV equations, together with the following (using the conventions of Sect. 1.4.2).

$$\begin{aligned}
\mathtt{pm}\ \hat{\imath} :: K\ \mathtt{as}\ \{\ldots, i :: \mathtt{y}.M_i, \ldots\} &= M_{\hat{\imath}}[K/\mathtt{y}] \\
\mathtt{pm}\ V :: K\ \mathtt{as}\ \mathtt{x} :: \mathtt{y}.M &= M[V/\mathtt{x}, K/\mathtt{y}] \\
M[V/\mathtt{z}] &= \mathtt{pm}\ V\ \mathtt{as}\ \{\ldots, i :: \mathtt{y}.M[i :: \mathtt{y}/\mathtt{z}], \ldots\} \\
M[V/\mathtt{z}] &= \mathtt{pm}\ V\ \mathtt{as}\ \mathtt{x} :: \mathtt{y}.M[\mathtt{x} :: \mathtt{y}/\mathtt{z}] \\
\mathtt{letcos\ x.\ changecos\ x};\ M &= M \\
\mathtt{changecos}\ V;\ \mathtt{letcos\ x}.\ M &= \mathtt{changecos}\ V;\ M[V/\mathtt{x}] \\
\mathtt{changecos}\ K;\ \mathtt{changecos}\ L;\ M &= \mathtt{changecos}\ L;\ M \\
V &= []\ \mathtt{to\ x}.\ (\mathtt{changecos}\ V; \mathtt{produce\ x}) :: \mathtt{neverused} \\
\mathtt{changecos}\ K;\ M\ \mathtt{to\ x}.\ N &= \mathtt{changecos}\ ([]\ \mathtt{to\ x}.\ N :: K);\ M \\
\mathtt{changecos}\ K; \mathtt{coerce}_{\underline{B}} M &= M \\
\mathtt{changecos\ neverused}; M &= M \\
K &= \mathtt{neverused} \\
\mathtt{print}\ c;\ \mathtt{changecos}\ K;\ M &= \mathtt{changecos}\ K;\ \mathtt{print}\ c;\ M
\end{aligned}$$

Notice that these equations are extremely *ad hoc* and inelegant, by contrast with the simplicity of the JWA equational theory.

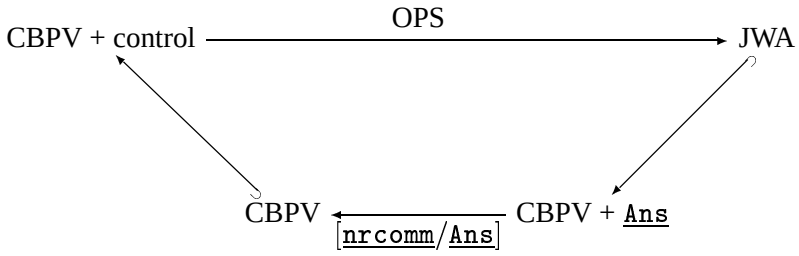
Proposition 77 The OPS transform preserves provable equality. □

The equation for `print` is of course effect-specific, but we would have a similar equation for other effects. For example, for divergence we would have the equation

$$\mathtt{diverge} = \mathtt{changecos}\ K;\ \mathtt{diverge}$$

8.9.3 Proving The Main Result

The aim of this section is to prove Prop. 75. Our approach (there are others) makes use of the following:



We write $-^{\mathsf{f}}$ for the composite transform from JWA to CBPV + control indicated in this diagram. We will show that this composite transform is inverse to the OPS transform up to isomorphism. The heart of the proof is these isomorphisms, which, far from being trivial, are full of control effects. In outline, when given a value $\overline{\Gamma} \vdash^{\mathsf{v}} W : \overline{A}$, we first apply $-^{\mathsf{f}}$, giving a term $\overline{\Gamma}^{\mathsf{f}} \vdash^{\mathsf{v}} W^{\mathsf{f}} : \overline{A}^{\mathsf{f}}$ without control effects, and then apply the isomorphisms to obtain a value $\Gamma \vdash^{\mathsf{v}} \hat{W} : A$ with control effects that transforms back to W (up to provable equality).

We first construct these isomorphisms.

1. For each CBPV value type A, we define a function α_A from CBPV+control values $\Gamma \vdash^{\mathsf{v}} V : A$ to CBPV+control values $\Gamma \vdash^{\mathsf{v}} V : \overline{A}^{\mathsf{f}}$, and a function α_A^{-1} in the opposite direction.

2. For each CBPV computation type $\underline{B}$, we define a function $\alpha_{\underline{B}}$ from CBPV+control values $\Gamma \vdash^{\mathsf{v}} V : \mathtt{os}\ \underline{B}$ to CBPV+control values $\Gamma \vdash^{\mathsf{v}} V : \overline{\mathtt{os}\ \underline{B}}^{\mathsf{f}}$, and a function $\alpha_{\underline{B}}^{-1}$ in the opposite direction.

3. For each JWA type A, we define a function β_A from JWA values $\Gamma \vdash^{\mathsf{v}} V : A$ to JWA values $\Gamma \vdash^{\mathsf{v}} V : \overline{A^{\mathsf{f}}}$, and a function β_A^{-1} in the opposite direction.

The definitions are given in Fig. 8.6. Notice the usage of `neverused`; this is indispensable. It is straightforward to check that α_A and α_A^{-1} are inverse up to provable equality and commute with substitution in Γ; similarly for $\alpha_{\underline{B}}$ and for β_A.

We have thus proved Prop. 75(1). It is then straightforward to prove Prop. 75(2) by induction on A.

Lemma 78 1. For any CBPV+control+`print` value $\Gamma \vdash^{\mathsf{v}} V : B$, we can prove

$$V = \alpha_B^{-1}\overline{V}^{\mathsf{f}}[\overrightarrow{\alpha_{A_i}\mathtt{x}_i/\mathtt{x}_i}]$$

2. For any CBPV+control+`print` computation $\Gamma \vdash^{\mathsf{c}} M : \underline{B}$, we can prove

$$M = \mathtt{letcos\ k.coerce}\ \overline{M}^{\mathsf{f}}[\overrightarrow{\alpha_{A_i}\mathtt{x}_i/\mathtt{x}_i}, \alpha_{\underline{B}}\mathtt{k}/\mathtt{k}]$$

3. For any JWA+`print` value $\Gamma \vdash^{\mathsf{v}} V : B$, we can prove

$$V = \beta_B^{-1}\overline{V^{\mathsf{f}}}[\overrightarrow{\beta_{A_i}\mathtt{x}_i/\mathtt{x}_i}]$$

4. For any JWA+`print` non-returning command $\Gamma \vdash^{\mathsf{n}} M$, we can prove

$$M = \overline{M^{\mathsf{f}}}[\overrightarrow{\beta_{A_i}/\mathtt{x}_i}, ()/\mathtt{k}]$$

These results can be extended to computations and values with holes i.e. contexts. □

A	$\alpha_A V$	$\alpha_A^{-1} W$
$\sum_{i\in I} A_i$	$\mathtt{pm}\ V\ \mathtt{as}\ \{\ldots,(i,\mathtt{x}).(i,\alpha_{A_i}\mathtt{x}),\ldots\}$	$\mathtt{pm}\ W\ \mathtt{as}\ \{\ldots,(i,\mathtt{x}).(i,\alpha_{A_i}^{-1}\mathtt{x}),\ldots\}$
$A\times A'$	$\mathtt{pm}\ V\ \mathtt{as}\ (\mathtt{x},\mathtt{y}).(\alpha_A\mathtt{x},\alpha_{A'}\mathtt{y})$	$\mathtt{pm}\ W\ \mathtt{as}\ (\mathtt{x},\mathtt{y}).(\alpha_A^{-1}\mathtt{x},\alpha_{A'}^{-1}\mathtt{y})$
$U\underline{B}$	$\mathtt{thunk}\ \lambda\mathtt{k}.(\mathtt{changecos}\ \alpha_{\underline{B}}^{-1}\mathtt{k};\mathtt{force}\ V)$	$\mathtt{thunk}\ (\mathtt{letcos}\ \mathtt{k}.\mathtt{coerce}\ \underline{B}((\alpha_{\underline{B}}\mathtt{k})\text{`}\mathtt{force}\ W))$
$\mathtt{os}\ \underline{B}$	$\alpha_{\underline{B}} V$	$\alpha_{\underline{B}}^{-1} W$

$\underline{B}$	$\alpha_{\underline{B}} V$	$\alpha_{\underline{B}}^{-1} W$
FA	$\mathtt{thunk}\ \lambda\mathtt{x}.(\mathtt{changecos}\ V;\mathtt{produce}\ \alpha_A^{-1}\mathtt{x})$	$[]\ \mathtt{to}\ \mathtt{x}.\ (\alpha_A\mathtt{x}\text{`}\mathtt{force}\ W) :: \mathtt{neverused}$
$\prod_{i\in I}\underline{B}_i$	$\mathtt{pm}\ V\ \mathtt{as}\ \{\ldots,(i :: \mathtt{x}).(i,\alpha_{\underline{B}_i}\mathtt{x}),\ldots\}$	$\mathtt{pm}\ W\ \mathtt{as}\ \{\ldots,(i,\mathtt{x}).(i :: \alpha_{\underline{B}_i}^{-1}\mathtt{x}),\ldots\}$
$A\to\underline{B}$	$\mathtt{pm}\ V\ \mathtt{as}\ (\mathtt{x} :: \mathtt{y}).(\alpha_A\mathtt{x},\alpha_{\underline{B}}\mathtt{y})$	$\mathtt{pm}\ W\ \mathtt{as}\ (\mathtt{x},\mathtt{y}).(\alpha_A^{-1}\mathtt{x} :: \alpha_{\underline{B}}^{-1}\mathtt{y})$
$\underline{\mathtt{nrcomm}}$	$()$	$\mathtt{neverused}$

A	$\beta_A V$	$\beta_A^{-1} W$
$\sum_{i\in I} A_i$	$\mathtt{pm}\ V\ \mathtt{as}\ \{\ldots,(i,\mathtt{x}).(i,\beta_{A_i}\mathtt{x}),\ldots\}$	$\mathtt{pm}\ W\ \mathtt{as}\ \{\ldots,(i,\mathtt{x}).(i,\beta_{A_i}^{-1}\mathtt{x}),\ldots\}$
$A\times A'$	$\mathtt{pm}\ V\ \mathtt{as}\ (\mathtt{x},\mathtt{y}).(\beta_A\mathtt{x},\beta_{A'}\mathtt{y})$	$\mathtt{pm}\ W\ \mathtt{as}\ (\mathtt{x},\mathtt{y}).(\beta_A^{-1}\mathtt{x},\beta_{A'}^{-1}\mathtt{y})$
$\neg A$	$\gamma(\mathtt{x},()).(\beta_A^{-1}\mathtt{x}\nearrow V)$	$\gamma\mathtt{x}.((\beta_A\mathtt{x},())\nearrow W)$

Figure 8.6: Syntactic Isomorphisms α and β used in proof of Prop. 75

These are proved by induction over terms.

Lemma 79 1. For any CBPV value type A and value $\Gamma\vdash^{\mathsf{v}} V : A$, we can prove

$$\overline{\alpha_A V} = \beta_{\overline{A}}\overline{V}$$

Similarly for computation types.

□

This is proved by induction over types.

To prove Prop. 75(3), suppose we are given a JWA value $\overline{\Gamma}\vdash^{\mathsf{v}} W : \overline{A}$, where Γ is the context $A_0,\ldots,A_{m-1}$. We set $\Gamma\vdash^{\mathsf{v}}\hat{W} : A$ to be $\alpha_A^{-1} W^{\mathsf{f}}[\overrightarrow{\alpha_{A_i}\mathtt{x}_i/\mathtt{x}_i}]$. Then $\hat{-}$ is inverse to the OPS transform up to provable equality, and so the latter defines a bijection.

To prove Prop. 75(4), suppose we are given a JWA non-returning command $\overline{\Gamma},\mathtt{k} : \overline{\underline{B}}\vdash^{\mathsf{n}} N$, where Γ is the context $A_0,\ldots,A_{m-1}$. We set $\Gamma\vdash^{\mathsf{c}}\hat{N} : \underline{B}$ to be $\mathtt{letcos}\ \mathtt{k}.\mathtt{changecos}\ \alpha_{\underline{B}}\mathtt{k}; N^{\mathsf{f}}[\overrightarrow{\alpha_{A_i}\mathtt{x}_i/\mathtt{x}_i}]$. Then $\hat{-}$ is inverse to the OPS transform up to provable equality, and so the latter defines a bijection.

These arguments can be adapted to values and computations with holes i.e. contexts.

8.10 JWA And Classical Logic

We said in Sect. 8.2 that the type constructor $\neg$ in JWA is not exactly the same as logical negation. This is because (for example) there does not exist a value $\neg\neg 0\vdash^{\mathsf{v}} V : 0$, even though $\neg\neg 0\vdash 0$ is provable in both intuitionistic and classical logic.

But although the JWA *value* judgement $\vdash^{\mathsf{v}}$ is not related to logic, the JWA *non-returning command* judgement $\vdash^{\mathsf{n}}$ is related to logic.

- By *intuitionistic propositional logic* we mean the type theory $\times\sum\prod\to$-calculus defined in Fig. 4.3, omitting the terms.
- By *classical propositional logic* we mean Gentzen's system LK (see e.g. [GLT88]), where we write $\sum$ for disjunction and $\times$ for conjunction.

Thus $\neg$ is a primitive of classical logic but not of intuitionistic logic.

The key result, which lies at the heart of much work relating continuations to logic [Fri78, Gri90, HS97, LRS93, Mur90], is the following:

Proposition 80 Let Γ be a context in Jump-With-Argument. Then the following are equivalent.

1. $\Gamma \vdash$ is provable in classical propositional logic.
2. There exists a non-returning command $\Gamma \vdash^{\mathtt{n}} M$ in effect-free JWA.
3. $\Gamma[(- \to \mathtt{Ans})/\neg] \vdash \mathtt{Ans}$ is provable in intuitionistic propositional logic extended with a propositional identifier $\mathtt{Ans}$.
4. $\Gamma[(- \to 0)/\neg] \vdash 0$ is provable in intuitionistic propositional logic.

□

Proof

(2)⇒(3) This is given by the composite transform

$$\text{JWA} \hookrightarrow \text{CBPV} + \underline{\mathtt{Ans}} \xrightarrow{\text{trivialization}} \times\textstyle\sum\prod \to\text{-calculus} + \mathtt{Ans}$$

which takes $\neg$ to $- \to \mathtt{Ans}$ and takes a non-returning command to a term of type $\mathtt{Ans}$.

(3)⇒(4) We substitute 0 for $\mathtt{Ans}$.

(4)⇒(1) This is because

- intuitionistic provability implies classical provability;
- $\neg A$ and $A \to 0$ are equivalent in classical logic.

(1)⇒(2) There are many ways of proving this; here is just one. We define a transform from propositions to JWA types as follows:

$$\begin{array}{ll} A & \overline{A} \\ \neg A & \neg\overline{A} \\ \sum_{i\in I} A_i & \sum_{i\in I}\overline{A_i} \\ A \times A' & \overline{A} \times \overline{A'} \\ \prod_{i\in I} A_i & \neg\sum_{i\in I}\neg\overline{A_i} \\ A \to B & \neg(\overline{A} \times \neg\overline{B}) \end{array}$$

We then show that if $A_0, \ldots, A_{m-1} \vdash B_0, \ldots, B_{n-1}$ is provable, then there is a non-returning command $\overline{A_0}, \ldots, \overline{A_{m-1}}, \neg\overline{B_0}, \ldots, \neg\overline{B_{n-1}} \vdash^{\mathtt{n}} M$ in effect-free JWA. The required result is an immediate consequence, because the transform $\overline{-}$ leaves JWA types unchanged.

□

We emphasize that Prop. 80 is a result about provability rather than about proofs, for the following reasons.

- LK is not equipped with any canonical notion of equality between proofs.
- Intuitionistic logic *is* equipped with a canonical notion of equality between proofs—the equational theory defined in Fig. 4.3—but all proofs of $\Gamma \vdash 0$ are equal. (We will prove this later—Prop. 106(3).)

It is clear that any LK sequent can be converted (although not in any canonical way) into an equivalent sequent of the form $\Gamma \vdash$ where Γ uses only $\sum \times \neg$. Thus, not only can the non-returning command judgement of effect-free JWA be seen as classical (in the light of Prop. 80), but it is as expressive as LK. Consequently, JWA can be viewed as a type theory for classical logic, provided we regard values (which are not logically significant) as merely auxiliary.

The reader may wonder how JWA compares to another proposed type theory for classical logic: Parigot's $\lambda\mu$-calculus [Ong96, Par92]. The answer is that, whereas JWA is the target language of the OPS transform, $\lambda\mu$-calculus resembles its the source language in the sense that, unlike JWA, it is a language with control operators [Bie98]. As such, it needs to be arbitrarily declared CBV or CBN in order to have a semantics. Surely it would be more canonical to use CBPV+control in place of either CBV or CBN $\lambda\mu$-calculus. But we have seen that CBPV + control is equivalent to JWA, and that the latter is much simpler.

Chapter 9

Pointer Games

9.1 Introduction

9.1.1 Pointer Games And Their Problems

In this chapter we look at the game semantics of Hyland and Ong [HO94], discovered also by Nickau [Nic96]. It is based on a certain kind of two-player game, where (generally speaking) a player moves by

1. pointing to a previous move of the other player;
2. passing a token.

We call such a game a "pointer game". This is to distinguish it from many other kinds of game, such as the purely token-passing games of [AJM94], which are quite different and which we shall not be looking at. Pointer games are extremely powerful: in a series of striking results, they have provided universal[1] models for PCF+`case` [HO94], recursive types [McC97], control effects [Lai97], ground store [AM97], general store [AHM98], erratic choice [HM99] and more.

However, despite its remarkable successes, the account of pointer game models in the literature suffers from a number of problems. The collective effect of these problems is that the reaction of many readers is negative. They perceive game semantics as complicated and technical and they do not see its elegance. In this chapter, we aim to rectify *some* of these problems, but not all. We need to describe the various problems with some care, in order to say which are the ones we are aiming to rectify.

1. **lack of intuition** The rules and constraints of play are not clearly motivated by operational intuitions.
2. **difficulty of expression** Even when a strategy is intuitively clear, it is difficult to express it in a clean, rigorous way, as we lack a convenient language for strategies. This is even more the case for operations on strategies (e.g. composition): it is usually obvious how to apply these operations in any particular example, but giving a precise description is messy.
3. **difficulty of reasoning** Partly because of 2, it is difficult to reason about game semantics, in particular to prove that strategies are equal.

These problems apply to different parts of the account:

[1] A model is *universal* [LP97] when every morphism from $[\![\Gamma]\!]$ to $[\![A]\!]$ is the denotation of some term $\Gamma \vdash M : A$. In the logical setting, this property is called *full completeness* [Abr92].

- (1) is a problem for the semantics of *types* and *judgements*.
- (2) is a problem for the semantics of *term constructors*.
- (3) is a problem throughout.

Why do we not consider the semantics of term constructors to be affected by (1)? The reason is that the semantics of term constructors tends to be determined by the semantics of types and judgements—not in a precise technical sense, but rather in the sense that, for any given term, there is only one "reasonable" denotation of the form specified by the semantics of types and judgements. So it is the semantics of types and judgements where the need for intuitive explanation of definitions is most pressing.

In this chapter we address problem (1), so that the semantics of types and judgements is clearer and more intuitive. There are 3 ways in which we do this:

- by using CBPV instead of CBN, as we explain in Sect. 9.1.2
- by incorporating store and control effects into the language, as we explain in Sect. 9.1.4 (this, of course, is not original)
- in Sect. 9.3–9.4, by reducing the CBPV model to a pointer game model for JWA, in which we see that an arena family is a representation of a *type canonical form*.

We do not address problems (2)–(3) at all. Thus, in our presentation, the semantics of terms, although easy to see in particular examples, is hard to express cleanly and hard to reason about, just as in previous work. We therefore relegate it to Appendix B. Improving this situation remains a challenge for game semantics.

9.1.2 CBPV Makes Pointer Games More Intuitive

In Sect. 9.2, we provide pointer game semantics for CBPV, and we will see that it is more intuitive than CBN semantics. It is, in our view, unfortunate that—despite the CBV models of [AM98a, HY97]—the bulk of the work on pointer games, especially operational analysis such as [DHR96], has focussed on CBN. The reason for this focus was discussed and criticized in Sect. 2.8: the widespread belief that CBN (or cartesian closed categories) is the canonical, mathematically well behaved choice. We hope that Sect. 9.2 will persuade game semanticists of the advantages of CBPV.

To support our claim, here are 3 pointer game notions which are clearer from a CBPV viewpoint than from a CBN viewpoint.

question/answer distinction Tokens are divided into two classes, which have been given the names "questions" and "answers". Passing a question-token is called "asking a question"; passing an answer-token is called "answering". What is the operational intuition behind this division? No clear explanation is given in the literature (beyond the vague assertion that "a question is a demand for data"). But from a CBPV perspective, we can explain these terms exactly:

- "asking a question" means forcing a thunk
- "answering" means producing

We saw in Sect. 1.5.3 that forcing a thunk and producing are precisely the two instructions that cause execution to move to another part of the program. CBPV differs from CBN in making these explicit.

pointers The significance of the pointers between moves is sometimes downplayed in the literature (they are frequently described as "auxiliary", while the tokens are confusingly called "moves"). However, their role is not merely technical but also conceptual: they indicate where execution moves to. Answering a question means forcing a thunk, so the pointer from a question-move indicates the thunk being forced. Answering means producing, so the pointer from an answer-move indicates the consumer being produced to.

For example, suppose one player pushes some values, including a thunk V and then forces a thunk W—this forcing is represented as question-move m. If later the other player forces V, then the relevant question-move will point back to m. If the execution of W ends with a `produce` instruction, then, as in the example program of Sect. 1.5.2, control returns to just after `force` W (strictly speaking, this point is the consumer that was on the stack at the time W was forced) and so this answer-move points back to m.

As another example, suppose one player produces some data, including a thunk V—this producing is represented as an answer-move m. If the other player later forces V, the relevant question-move will point back to m.

bracketing condition Pointer game models usually impose on Player (or on both Player and Opponent) the constraint that the pointer from an answer-move must be to the *pending* question-move. Thus, in the representation of play, the pointers from Player's answer-moves (or the pointers from both Player's and Opponent's answer-moves) can be omitted because they can be inferred. This corresponds to the observation we made in Sect. 1.5.3 that a `produce` instruction does not need to specify the point (i.e. the consumer) that execution returns to, as it is simply the top of the stack.

Using control effects, however, the programmer can choose any available consumer and install it as the current outside before producing. Hence, as Laird explained [Lai97], strategies for control effects do not obey the bracketing condition.

We will not be looking further at the bracketing condition, but we will see examples of the question/answer distinction and the pointers between moves.

9.1.3 The Language That We Model

It is a surprising feature of pointer game semantics that the universal model for a language with divergence, general store (excluding equality testing on cells) and control effects is much simpler than the universal model for a language without all these features (such as PCF). By including these features from the outset, we can dispense with the machinery of views, visibility, innocence and bracketing, which obscure the more important aspects of pointer games. And we can simplify the definition of "arena", because it becomes possible for an answer-token to succeed an answer-token (as a consequence of the `os` type constructor).

A further advantage of this approach is that the pointer game model we obtain is not just universal but also fully abstract, with no need for the further quotienting which is frequently found in the literature. All the details of a term's denotation can be worked out by applying contexts involving store and control. This result is folklore.

The language that the semantics of Sect. 9.2 models is infinitary CBPV with divergence, general store (as in Chap. 7, but excluding equality testing on cells) and control effects. The main reason we use *infinitary* CBPV is that we thereby obtain definability results for types as well as for terms. Furthermore, we thereby obviate the need for a computability restriction on strategies.

We maintain the constraint of determinism on strategies, because the language is deterministic.

9.1.4 Changes In Presentation

Readers accustomed to other presentations of game semantics should be warned of some ways that our presentation differs.

- The traditional organization of semantics of types is that a CBN type (a computation type) denotes an arena whereas a CBV type (a value type) denotes a family of arenas. In our exposition, a computation type and a value type will both denote a family of arenas.
- Like [HO94] but unlike [McC96], we require an arena to be a forest, and we do not label tokens as P/O from the outset, but infer this later from the depth, as we explain below.

9.2 Pointer Game Model For CBPV

9.2.1 Arenas

The tokens that are passed during pointer games are arranged in structures called *arenas*, which are defined as follows.

Definition 51 An *arena* R is a structure $(\mathsf{tok}\ R, \mathsf{rt}\ R, \vdash_R, \lambda_R^{\mathsf{QA}})$.

- $\mathsf{tok}\ R$ is a countable set of *tokens*.
- $\mathsf{rt}\ R$ is a subset of $\mathsf{tok}\ R$, whose elements are called *roots* and $\vdash_R$ is a binary relation on $\mathsf{tok}\ R$. We read $t \vdash_R u$ as "t is the *predecessor* of u" or as "u is a *successor* of t". This must give a "forest", i.e.

  unique predecessor each root has no predecessor and each non-root has a unique predecessor

  well-foundedness there is no infinite chain of predecessors $\cdots \vdash_R t_1 \vdash_R t_0$.
- $\lambda_R^{\mathsf{QA}} : \mathsf{tok}\ R \longrightarrow \{\mathsf{Q}, \mathsf{A}\}$ is a *labelling* function which indicates whether a move using a given token is a question (Q) or answer (A).

□

Arenas are built up using the following constructions.

Definition 52 1. Let $\{R_i\}_{i \in I}$ be a countable family of arenas. We define $\mathsf{pt}^{\mathsf{Q}}_{i \in I} R_i$ to be the arena with I roots, all labelled Q, and a copy of R_i grafted underneath the ith root. More formally, it has tokens

   - $\mathsf{root}\ i$ for each $i \in I$, labelled Q
   - $\mathsf{under}(i, r)$ for each $i \in I$ and $r \in \mathsf{tok}\ R_i$, labelled the same as r

   where

   - the roots are the tokens $\mathsf{root}\ i$
   - $\mathsf{root}\ i \vdash \mathsf{under}(i, r)$, whenever r is a root of R_i
   - $\mathsf{under}(i, r) \vdash \mathsf{under}(i, s)$ for $r \vdash s$ in R_i

2. Let $\{R_i\}_{i \in I}$ be a countable family of arenas. We define $\mathsf{pt}^{\mathsf{A}}_{i \in I} R_i$ to be the arena with I roots, all labelled A, and a copy of R_i grafted underneath the ith root. Thus it is the same as $\mathsf{pt}^{\mathsf{Q}}_{i \in I} R_i$ except that $\mathsf{root}\ i$ is labelled A.

3. We define the arena $R \uplus R'$ to be the disjoint union of the arenas R and R'. Formally, it has tokens

- $\mathsf{inl}\ r$ for each $r \in \mathsf{tok}\ R$, labelled the same as r
- $\mathsf{inr}\ r$ for each $r \in \mathsf{tok}\ R'$, labelled the same as r

where

- the roots are the tokens $\mathsf{inl}\ r$, where $r \in \mathsf{rt}\ R$, and $\mathsf{inr}\ r$, where $r \in \mathsf{rt}\ R'$
- $\mathsf{inl}\ r \vdash \mathsf{inl}\ s$ when $r \vdash_R s$
- $\mathsf{inr}\ r \vdash \mathsf{inr}\ s$ when $r \vdash_{R'} s$

□

We can classify the tokens in an arena as *even depth* or *odd depth*:

- a root is even depth;
- a successor of an even depth token is odd depth;
- a successor of an odd depth token is even depth.

It aids readability, although it is not technically necessary, to add to an arena another labelling function $\lambda^{\mathsf{PO}} : \mathsf{tok}\ R \longrightarrow \{\mathsf{P}, \mathsf{O}\}$ designating tokens as P-tokens or O-tokens, as in [McC96]. Sometimes we want even depth tokens to be P-tokens and odd depth tokens to be O-tokens, and sometimes vice versa.

Definition 53 Let R be an arena.

- We write R^{P} for R together with a labelling function λ^{PO} designating even-depth tokens (including roots) as P and odd-depth tokens as O.
- We write R^{O} for R together with a labelling function λ^{PO} designating even-depth tokens (including roots) as O and odd-depth tokens as P.

□

Some useful terminology:

Definition 54

1. An *isomorphism* from arena R to arena S is a bijection from $\mathsf{tok}\ R$ to $\mathsf{tok}\ S$ preserving and reflecting $\vdash$ and Q/A labelling.
2. We say that R is a *sub-arena* of S, written $R \subseteq S$, when
   - $\mathsf{tok}\ R \subseteq \mathsf{tok}\ S$
   - $\vdash_R$ is $\vdash_S$ restricted to $\mathsf{tok}\ R$ and likewise for Q/A labelling
   - If $r \in \mathsf{tok}\ R$ and $s \vdash_S r$ then $s \in \mathsf{tok}\ R$.
3. We write $\mathsf{Qrt}\ R$ for the set of roots of R that are Q-tokens, and we write $\mathsf{Art}\ R$ for the set of roots of R that are A-tokens.
4. Let a be a token in an arena R. Then we write $R \restriction_a$ for the arena consisting of those tokens of R hereditarily preceded by a (excluding a itself), with the Q/A labelling and $\vdash$ relation inherited from R.

□

9.2.2 Semantics of Types

Each type, whether a value type or a computation type, denotes a countable family of arenas. This semantics of types is somewhat out-of-a-hat, because we have not yet seen the semantics of judgements, and so the reader may want to read this section in conjunction with the next section (Sect. 9.2.3), which provides examples.

In the following clauses, notice that Q-tokens are introduced by U, while A-tokens are introduced by F. This accords with what we said in Sect. 9.1.2: passing a Q-token means forcing a thunk, which has U type, while passing an A-token means producing, and a producer has F type.

The type constructors other than `ref` (which we deal with below) are interpreted as follows.

- If $\underline{B}$ denotes $\{R_i\}_{i\in I}$, then $U\underline{B}$ denotes the singleton family $\{\mathsf{pt}^{\mathsf{Q}}_{i\in I} R_i\}$.
- If, for each $i \in I$, A_i denotes $\{R_{ij}\}_{j\in J_i}$, then $\sum_{i\in I} A_i$ denotes $\{R_{ij}\}_{(i,j)\in \sum_{i\in I} J_i}$.
- 1 denotes the singleton family containing the empty arena.
- If A denotes $\{R_i\}_{i\in I}$ and A' denotes $\{S_j\}_{j\in J}$ then $A \times A'$ denotes $\{R_i \uplus S_j\}_{(i,j)\in I\times J}$.
- If A denotes $\{R_i\}_{i\in I}$, then FA denotes the singleton family $\{\mathsf{pt}^{\mathsf{A}}_{i\in I} R_i\}$.
- If, for each $i \in I$, $\underline{B}_i$ denotes $\{R_{ij}\}_{j\in J_i}$, then $\prod_{i\in I}\underline{B}_i$ denotes $\{R_{ij}\}_{(i,j)\in \sum_{i\in I} J_i}$.
- If A denotes $\{R_i\}_{i\in I}$ and $\underline{B}$ denotes $\{S_j\}_{j\in J}$ then $A \to \underline{B}$ denotes $\{R_i \uplus S_j\}_{(i,j)\in I\times J}$.
- If $\underline{B}$ denotes $\{R_i\}_{i\in I}$ then `os` $\underline{B}$ also denotes $\{R_i\}_{i\in I}$.

There is a remarkable pattern here.

- U and F are interpreted the same way (except for the Q/A labelling).
- $\sum$ and $\prod$ are interpreted the same way.
- $\times$ and $\to$ are interpreted the same way.

For infinitely deep types, we say that $\{R_i\}_{i\in I} \subseteq \{S_j\}_{j\in J}$ when $I \subseteq J$ and R_i is a sub-arena of S_i for each $i \in I$. The class of arena families has all countable directed joins, and so, using the notation of Sect. 5.4.2, we set the denotation of a type A to be $\bigcup_{B\lhd_{\mathsf{fin}} A} [\![B]\!]$. This can also be used to interpret recursive types, as in [McC96].

We can now give a definability result for types.

Proposition 81 The *type canonical forms* for infinitary CBPV + `os` are given coinductively by

$$\begin{aligned} A &::= \sum_{i\in I}(U\underline{B}_i \times \mathtt{os}\ FA_i) \\ \underline{B} &::= \prod_{i\in I}(U\underline{B}_i \to FA_i) \end{aligned}$$

1. Every countable family of arenas A is isomorphic to the denotation of some value type canonical form $\theta_{\mathsf{val}} A$.
2. Every countable family of arenas $\underline{B}$ is isomorphic to the denotation of some computation type canonical form $\theta_{\mathsf{comp}}\underline{B}$.

□

Proof We define θ_{val} and θ_{comp} by mutual guarded induction:

$$\begin{aligned}\theta_{\mathsf{val}}\{R_i\}_{i\in I} &= \Sigma_{i\in I}(U\theta_{\mathsf{comp}}\{R_i \upharpoonright_j\}_{j\in \mathsf{Qrt}\ R_i} \times \mathsf{os}\ F\theta_{\mathsf{val}}\{R_i \upharpoonright_j\}_{j\in \mathsf{Art}\ R_i})\\ \theta_{\mathsf{comp}}\{R_i\}_{i\in I} &= \Pi_{i\in I}(U\theta_{\mathsf{comp}}\{R_i \upharpoonright_j\}_{j\in \mathsf{Qrt}\ R_i} \to F\theta_{\mathsf{val}}\{R_i \upharpoonright_j\}_{j\in \mathsf{Art}\ R_i})\end{aligned}$$

By coinductive reasoning, these equations have a unique solution. We want to construct isomorphisms

$$\{R_i\}_{i\in I} \cong [\![\theta_{\mathsf{val}}\{R_i\}_{i\in I}]\!] \tag{9.1}$$
$$\{R_i\}_{i\in I} \cong [\![\theta_{\mathsf{comp}}\{R_i\}_{i\in I}]\!] \tag{9.2}$$

This is complicated, so we relegate it to Sect. B.6. □

These type canonical forms are distinguished by the fact that every type is isomorphic to a type canonical form, using an isomorphism that does not involve `letcos` or `changecos`. Prop. 81 makes it clear that we can view an arena family as being a representation of a type canonical form.

The two types of commands introduced in Sect. 3.9.2 are interpreted as follows:

- The type $\underline{\texttt{comm}}$, because it is isomorphic to $F1$, denotes a singleton family containing an arena with a single A-token. We call this token done.
- The type $\underline{\texttt{nrcomm}}$, because it is isomorphic to $F0$, denotes a singleton family containing the empty arena.

It remains to interpret `ref`: the denotation of $\texttt{ref}\ A$ is the same as the denotation of $U((FA)\,\Pi\,(A \to \underline{\texttt{comm}}))$. As explained in [AM97], this is based on Reynolds' conception of a cell[2] as an "object with two methods": reading and assignment [Rey81]. Explicitly, we can convert a value V of type $\texttt{ref}\ A$ into a value $\tilde{V}$ of type $U((FA)\,\Pi\,(A \to \underline{\texttt{comm}}))$, defined by

$$\tilde{V} = \texttt{thunk}\,\lambda\begin{cases} 0. & \texttt{read}\ V\ \texttt{as x. produce x}\\ 1. & \lambda\texttt{x}.\ V := \texttt{x}\end{cases}$$

Reading and assignment can then be recovered from $\tilde{V}$ as follows:

$$\begin{aligned}\texttt{read}\ V\ \texttt{as x}.\ M &= 0\text{`}\texttt{force}\ \tilde{V}\ \texttt{to x}.\ M\\ V := W;\ M &= W\text{`}1\text{`}\texttt{force}\ \tilde{V};\ M\end{aligned}$$

9.2.3 Closed Terms—Rules and Examples

We are going to look at the semantics of judgements in stages. We first look at closed terms and then, in Sect. 9.2.5, at non-closed terms. In this section, we describe the games for closed terms in a way which is informal, although completely precise; a formal description is given in Sect. 9.2.4. We also provide examples, to illustrate the correspondence between CBPV terminology and pointer game terminology that we explained in Sect. 9.1.2.

Suppose S is an arena. The O-*first game* in S^{O} is the game whose rules are as follows:

- Play alternates between Player (P) and Opponent (O). Opponent moves first.
- In each move a token of S is passed.
- In the initial move, Opponent passes a root of S.

[2]Recall from Sect. 9.1.3 that we excluded equality testing of cells from the language. If we had not done so, this conception of cells would not be valid.

- Player moves by pointing to a previous O-move m and passing a successor of the token passed in move m.
- Opponent moves (except in the initial move) by pointing to a previous P-move m and passing a successor of the token passed in move m.

It is also permitted for either player to diverge instead of moving, except for the initial move, which Opponent must play. A consequence of these rules is that Player can pass only a P-token of S^{O} and Opponent can pass only an O-token. We write $\mathsf{Ostrat}\ S^{\mathsf{O}}$ for the set of strategies for the O-first game in S^{O}. This will be defined formally in Sect. 9.2.4.

Suppose S is an arena. The P-*first game* in S^{P} is the game whose rules are as follows:

- Play alternates between Player and Opponent. Player moves first.
- In each move a token of S is passed.
- Player moves by either
  - passing a root of S, or
  - pointing to a previous O-move m and passing a successor of the token passed in move m.
- Opponent moves by pointing to a previous P-move m and passing a successor of the token passed in move m.

It is also permitted for either player to diverge instead of moving. A consequence of these rules is that Player can pass only a P-token of S^{P} and Opponent can pass only an O-token. We write $\mathsf{Pstrat}\ S^{\mathsf{P}}$ for the set of strategies for the P-first game in S^{P}. This will be defined formally in Sect. 9.2.4.

Notice that in these games (as in the more general games below) there is an asymmetry between the players: Player may pass a root in any move, whereas Opponent may pass a root in the initial move only. Notice also that the Q/A labelling of tokens is ignored by the rules of these game. It is only in the presence of the bracketing condition, which we have not imposed, that the Q/A labelling has any technical—as opposed to conceptual—significance.

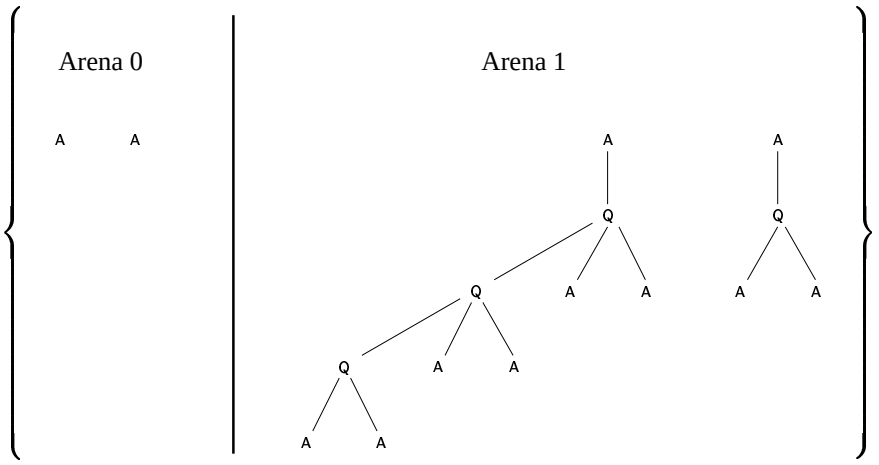
We use these games in the semantics of judgements for closed terms.

- If $[\![A]\!] = \{S_j\}_{j\in J}$ then a closed value $\vdash^{\mathsf{v}} V : A$ denotes an element of $\sum_{j\in J}\mathsf{Ostrat}\ S_j^{\mathsf{O}}$.
- If $[\![\underline{B}]\!] = \{S_j\}_{j\in J}$ then a closed computation $\vdash^{\mathsf{c}} M : \underline{B}$ denotes an element of $\prod_{j\in J}\mathsf{Pstrat}\ S_j^{\mathsf{P}}$.

We give some examples. Consider the type

$$\underline{B} = (F\mathtt{bool}) \sqcap F(U(U(UF\mathtt{bool} \to F\mathtt{bool}) \to F\mathtt{bool}) + UF\mathtt{bool})$$

It denotes an arena family of size 2, depicted thus:



Let M be the following closed computation of type $\underline{B}$.

$$\lambda \begin{cases} 0. & \texttt{produce false}, \\ 1. & \texttt{produce inl thunk } \lambda \texttt{x}. \left((\texttt{thunk produce true}) \texttt{`force x to} \right. \\ & \quad \begin{cases} \texttt{true}. & \texttt{diverge}, \\ \texttt{false}. & \left. (\texttt{thunk produce false}) \texttt{`force x to y. produce true} \right) \end{cases} \end{cases}$$

$[\![M]\!]0$ is a strategy for the P-first game on Arena 0^{P}. It describes the behaviour of M when 0 is on the stack:

P-**move 0** **CBPV terminology** M pops the 0 and then produces `false`.

game terminology Player answers false.

$[\![M]\!]1$ is a strategy for the P-first game on Arena 1^{P}. It describes the behaviour of M when 1 is on the stack:

P-**move 0** **CBPV terminology** M pops the 1 and produces `inl` of a thunk.

game terminology So Player passes the answer-token corresponding to inl.

O-**move 1** **game terminology** Now suppose that Opponent points to move 0 and passes the question-token.

CBPV terminology This means that the context pushes a thunk V and forces the thunk it has just received from M.

Whenever Player forces V in future, this will be represented by a question-move pointing to move 1, because it is in move 1 that V is passed to M's stack.

P-**move 2** **CBPV terminology** M now pops the thunk V pushed in move 1, pushes the operand `thunk produce true` and forces V.

game terminology So Player points to move 1 and pass the question-token.

O-**move 3** **game terminology** Now suppose that Opponent points to move 2 and passes the question-token.

CBPV terminology This means that the context pops the thunk pushed in moved 2 (actually `thunk produce true`) and forces it.

P-**move 4** **CBPV terminology** Then M produces `true`, to the consumer that was on the stack in move 3.

game terminology So Player points to move 3 and passes the answer-token corresponding to `true`.

O-**move 5** **game terminology** Suppose that Opponent points to move 1 and passes the question-token.

CBPV terminology This means that the context pushes a thunk W and forces the thunk it has just received from M.

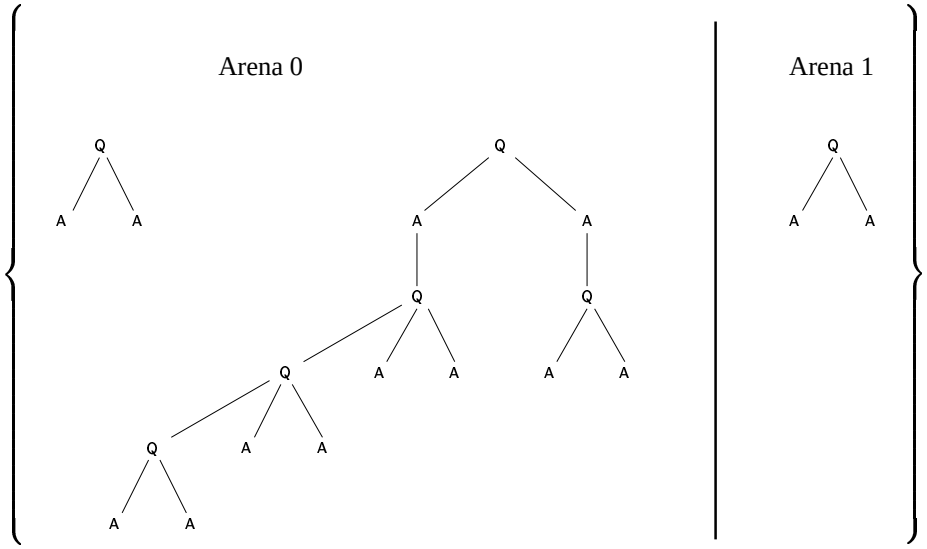
P-**move 6** **CBPV terminology** Then M pops the thunk W pushed in move 5, pushes the operand `thunk produce true` and forces W.

game terminology So Player points to move 5 and passes the question-token.

And so forth. Using store, we could modify M so that in move 6 it forces V instead of forcing W. In that case Player will point to move 1 instead of pointing to move 5, even though the token passed will be the same.

This example illustrates how "asking a question" in game terminology corresponds to "forcing a thunk" in CBPV terminology, and how "answering" in game terminology corresponds to "producing" in CBPV terminology. Notice, however, that the pushing and popping that features in the CBPV description of events is not reflected in the pointer game narrative. This is because they are determined by the type structure. For example, *every* computation of type $A \to \underline{B}$, when η-expanded, begins with popping an operand of type A. So there is no need for the denotational semantics to give us this information.

As another example, the value type $U\underline{B} + UF\texttt{bool}$ denotes an arena family of size 2, depicted thus



Now `inl thunk` M is a closed value of this type. It denotes $(0,\sigma)$ where σ is a strategy for the O-first game on Arena 0^{O}. The 0 represents the tag inl while σ represents the behaviour of `thunk` M:

O-**move 0** **game terminology** Suppose Opponent passes the left question-root.

CBPV terminology This means that the context pushes 0 and forces `thunk` M.

P-**move 1** **CBPV terminology** Then the forced thunk produces `false`, to the consumer on the stack at the time of move 0.

game terminology So Player points to move 0 and passes the answer-token corresponding to false.

O-**move 0** **game terminology** Alternatively, suppose Opponent passes the right question-root.

CBPV terminology This means that the context pushes 1 and forces `thunk` M.

P-**move 1** **CBPV terminology** Then the forced thunk produces `inl` of a thunk, to the consumer on the stack at the time of move 0.

game terminology So Player points to move 0 and passes the answer-token corresponding to inl.

etc. as above

If, in an O-first game, we allowed Opponent to pass a root again, after the initial move, then we could form strategies representing a thunk that behaves differently each time it is forced. However, even with store, there is no closed value of U type that behaves in this way. That is why we allow Opponent to pass a root in the initial move only.

9.2.4 Formal Representation of Strategies

We now make precise our informal definition of the O-first and P-first games.

Definition 55 Let R be an arena. An O-*first finite play* a on R^{O} consists of

- a nonzero number $|a| \in \mathbb{N}$, called the *length of* a—we call $m \in \$|a|$ an O-*move* in a if even, a P-*move* in a if odd, and the *initial move* in a if zero;
- for each move m in a, a token $\mathsf{token}_a m$ in R called "the token passed in move m"—we say that move m is a *root move* if $\mathsf{token}_a m$ is a root token.
- for each non-root move m in a, a move $\mathsf{pointer}_a m$ in a called "the pointer from move m"

such that

- the token passed in the initial move is a root token;
- if m is a P-move then
  - move m is not a root-move
  - the pointer from move m is an O-move $n < m$
  - the token passed in move m is a successor of the token passed in move n;
- if m is a non-initial O-move then
  - move m is a non-root move
  - the pointer from move m is a P-move $n < m$
  - the token passed in move m is a successor of the token passed in move n.

We say that a finite play a is

- *awaiting*-O if it its length is even (nonzero)
- *awaiting*-P if its length is odd.

□

It is also possible to define an *infinite play*, which is either an infinite sequence of moves or a finite play followed by divergence. However, because we are working in the deterministic setting, we will not require infinite plays.

Proposition 82 Let a be an O-first finite play on R^{O}. For each move m in a

- if m is an O-move, the token passed in m is an O-token
- if m is a P-move, the token passed in m is a P-token.

□

There are various equivalent ways of representing a strategy. Here is one of them, taken from [HO94].

Definition 56 An O-*first strategy* for R^{O} is a set σ of O-first finite plays on R^{O} which is

contingent-complete if $a \in \sigma$ is awaiting-O and b is a one-move extension of a then $b \in \sigma$

prefix-closed if $a \in \sigma$ and b is a nonempty prefix of a then $b \in \sigma$

deterministic if $b, b' \in \sigma$ are both one-move extensions of a which is awaiting-P then $b = b'$.

□

We frequently describe a strategy by giving only the plays that are awaiting-O.

The definition of P-first finite play and P-first strategy is the same as O-first , except that

- a P-first finite play can be empty
- a move in a P-first finite play is a P-move when even and an O-move when odd
- a P-first finite play is awaiting-P when its length is even, and awaiting-O when its length is odd.

9.2.5 Non-Closed Terms—Rules and Examples

Like a value type, a context Γ denotes a countable family of arenas. The empty context has the same denotation as 1, and context extension is interpreted the same way as $\times$.

As in Sect. 9.2.3, we will define the games before looking at examples.

Suppose R and S are arenas. The O-*first game* from R^{P} to S^{O} is the game whose rules are as follows:

- Play alternates between Player and Opponent. Opponent moves first.
- In each move, either a token of R (a *source token*) or a token of S (a *target token*) is passed.
- In the initial move, Opponent passes a root of S.
- Player moves by either
  - passing a root of R, or
  - pointing to a previous O-move m and passing a successor of the token passed in move m.
- Opponent moves (except in the initial move) by pointing to a previous P-move m and passing a successor of the token passed in move m.

It is also permitted for either player to diverge instead of moving, except for the initial move, which Opponent must play. A consequence of these rules is that Player can pass only a P-token of R^{P} or S^{O} and Opponent can pass only an O-token. We write $\mathsf{Ostrat}\ (R^{\mathsf{P}}, S^{\mathsf{O}})$ for the set of strategies for the O-first game from R^{P} to S^{O}. This can be defined formally as in Sect. 9.2.4.

Suppose R and S are arenas. The P-*first game* from R^{P} to S^{P} is the game whose rules are as follows:

- Play alternates between Player and Opponent. Player moves first.
- In each move, either a token of R (a *source token*) or a token of S (a *target token*) is passed.
- Player moves by either
  - passing a root of R, or
  - passing a root of S, or
  - pointing to a previous O-move m and passing a successor of the token passed in move m.
- Opponent moves by pointing to a previous P-move m and passing a successor of the token passed in move m.

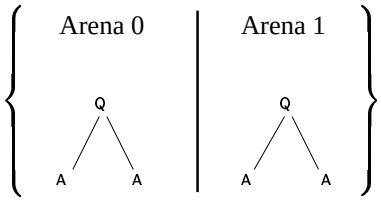
It is also permitted for either player to diverge instead of moving. A consequence of these rules is that Player can pass only a P-token of R^{P} and S^{P} and Opponent can pass only an O-token. We write $\mathsf{Pstrat}\,(R^{\mathsf{P}}, S^{\mathsf{P}})$ for the set of strategies for the P-first game from R^{P} to S^{P}. This can be defined formally as in Sect. 9.2.4.

We use these games in the semantics of judgements for general terms.

- If $[\![\Gamma]\!] = \{R_i\}_{i\in I}$ and $[\![A]\!] = \{S_j\}_{j\in J}$ then a value $\Gamma \vdash^{\mathsf{v}} V : A$ denotes an element of $\prod_{i\in I}\Sigma_{j\in J}\mathsf{Ostrat}\,(R_i^{\mathsf{P}}, S_j^{\mathsf{O}})$. Notice that these elements form a cpo.
- If $[\![\Gamma]\!] = \{R_i\}_{i\in I}$ and $[\![\underline{B}]\!] = \{S_j\}_{j\in J}$ then a computation $\Gamma \vdash^{\mathsf{c}} M : \underline{B}$ denotes an element of $\prod_{i\in I}\prod_{j\in J}\mathsf{Pstrat}\,(R_i^{\mathsf{P}}, S_j^{\mathsf{P}})$. Notice that these elements form a pointed cpo.

(The cpo/cppo structure on the model allows us to interpret recursive and infinitely deep terms.)

We give an example. Let A be the type $UF\texttt{bool} + UF\texttt{bool}$. This denotes the arena family of size 2 depicted thus:



Let V be the complex value

$$\mathtt{x} : A \vdash^{\mathsf{v}} \mathtt{pm\ x\ as} \begin{cases} \mathtt{inl\ y}. & \mathtt{inl\ thunk}\,(\mathtt{force\ y\ to\ z.\ produce\ true}), \\ \mathtt{inr\ y}. & \mathtt{inl\ thunk\ produce\ false} \end{cases} : A$$

If $\mathtt{x}$ is bound to $\mathtt{inl}\ W$, then V is $\mathtt{inl}\ W'$ (where W and W' are thunks). So $[\![V]\!]0$ is $(0,\sigma)$ where σ is the following strategy for the O-first game from Arena 0^{P} to Arena 0^{O}.

O-**move 0** **game terminology** Suppose Opponent passes the target question-root.

CBPV terminology This means that the context forces the thunk W'.

P-**move 1** **CBPV terminology** Then the term forces the thunk W.

game terminology So Player passes the source question-root.

O-**move 2** **game terminology** Now suppose Opponent points to move 1 and passes an answer token.

CBPV terminology This means that the forcing of W in move 1 produces a boolean $\mathtt{z}$.

P-**move 3** **CBPV terminology** Then the term produces `true` to the consumer on the stack in move 0.

game terminology So Player points to move 0 and passes the target answer-token corresponding to `true`.

If $\mathtt{x}$ is bound to $\mathtt{inr}\ W$, then V is $\mathtt{inl}\ W'$ (where W and W' are thunks). So $[\![V]\!]1$ is $(0,\tau)$ where τ is the following strategy for the O-first game from Arena 1^{P} to Arena 0^{O}.

O-**move 0** **game terminology** Suppose Opponent passes the target question root.

CBPV terminology This means that the context forces the thunk W'.

P-**move 1** **CBPV terminology** Then the term produces `false` to the consumer on the stack in move 0.

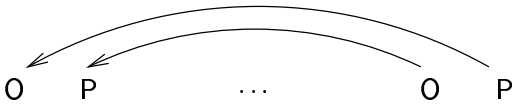
game terminology So Player points to move 0 and passes the target answer-token corresponding to false.

9.2.6 Further Examples

Copycat Strategies

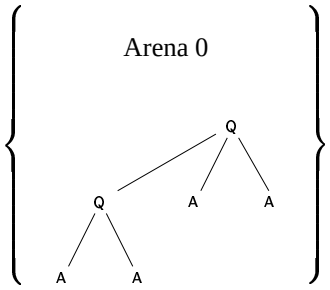
For every arena R, there is a canonical O-first strategy from R^{P} to R^{O}, called the *copycat strategy*. in which each P-move mimics the preceding O-move.

- If Opponent points to an earlier P-move (as he always must, except in the initial move), then Player mimics this by pointing to the corresponding O-move:



- If Opponent passes a source token, then Player mimics this by passing the corresponding target token.
- If Opponent passes a target token, then Player mimics this by passing the corresponding source token.
- In particular, in the initial move Opponent passes a target root token, and Player mimics this by passing the corresponding source root token.

Copycat strategies are used in the semantics of identifiers. If A denotes the arena family $\{R_i\}_{i\in I}$ then the identifier $\mathtt{x} : A \vdash^{\mathsf{v}} \mathtt{x} : A$ at i denotes i together with the copycat strategy from R_i^{P} to R_i^{O}. We can see why this should be so by η-expanding. For example, suppose A is the type $U(UF\mathtt{bool} \to F\mathtt{bool})$, which denotes the singleton arena family



The identifier $\mathtt{x} : A \vdash^{\mathsf{v}} \mathtt{x} : A$ can be η-expanded as

$$\begin{array}{l}\mathtt{thunk}\ \lambda\mathtt{y}.\\ \quad \mathtt{thunk}\,(\mathtt{force}\ \mathtt{y}\ \mathtt{to}\ \left\{\begin{array}{ll}\mathtt{true}. & \mathtt{produce}\ \mathtt{true}\\ \mathtt{false}. & \mathtt{produce}\ \mathtt{false}\end{array}\right\})`\\ \mathtt{force}\ \mathtt{x}\ \mathtt{to}\ \left\{\begin{array}{ll}\mathtt{true}. & \mathtt{produce}\ \mathtt{true}\\ \mathtt{false}. & \mathtt{produce}\ \mathtt{false}\end{array}\right\}\end{array}$$

The copycat behaviour is described in this η-expansion. Here is a possible play:

O-move 0 game terminology Suppose Opponent passes the target Q-root.

CBPV terminology This means that the context pushes an operand `y` and forces the term.

P-move 1 CBPV terminology Then the term pushes a thunk and forces `x`.

game terminology So Player passes the source Q-root.

O-move 2 game terminology Now suppose Opponent points to move 1 and passes the source Q-token.

CBPV terminology This means that the context forces the thunk pushed in move 1.

P-**move 3** **CBPV terminology** Then the term forces the thunk `y`, which was received in move 0.

game terminology So Player points to move 0 and passes the target Q-token.

O-**move 4** **game terminology** Now suppose Opponent points to move 3 and passes the target A-token for `true`.

CBPV terminology This means that the thunk `y` forced in move 3 produces `true`.

P-**move 5** **CBPV terminology** Then the thunk forced in move 2 produces `true`.

game terminology So Player points to move 2 and passes the source A-token for `true`.

O-**move 6** **game terminology** Now suppose Opponent points to move 1 and passes the source A-token for `true`.

CBPV terminology This means that the thunk `x` forced in move 1 produces `true`.

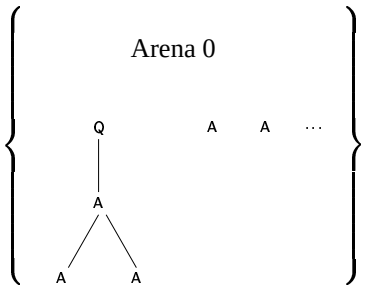
P-**move 7** **CBPV terminology** Then the whole term, forced in move 0, produces `true` to the consumer that was on the stack in move 0.

game terminology So Player points to move 0 and passes the answer-token `true`.

And so forth.

Answer-Move Pointing To Answer-Move

We give an example of a strategy in which an A-move points to an A-move, because this possibility has not been treated previously in the games literature. Let $\underline{B}$ be the computation type $UF\texttt{os}\ F\texttt{bool} \to F\texttt{nat}$. This denotes the singleton arena family



A closed computation of this type is

$$\lambda\texttt{x}.(\texttt{force x to y}.\ (\texttt{changecos y; produce true}))$$

This (at 0) denotes a P-first strategy for Arena 0^{P}, as follows.

P-**move 0** **CBPV terminology** The computation pops the thunk `x` and forces it.

game terminology So Player plays the root Q-token.

O-**move 1** **game terminology** Now suppose Opponent points to move 0 and plays the A-token.

CBPV terminology This means that the thunk `x`, forced in move 0, produces a consumer `y` to the consumer that was then (and in fact still is) on the stack.

P-**move 2** **CBPV terminology** Then the computation produces `true` to this consumer `y` produced in move 1.

game terminology So Player points to move 1 and passes the A-token for `true`.

9.2.7 Recovering CBN and CBV Models

From the CBPV model we have presented, we can recover the standard CBN and CBV models. The CBV case is trivial. The CBN case is somewhat more complicated, because, in the Hyland-Ong semantics, a CBN type denotes an arena (not a family of arenas) and a term denotes an O-first strategy. In our terminology, this is the semantics of thunks.

Proposition 83 1. Let A be a CBN type, and A^{n} its translation into CBPV. Then (up to isomorphism) we can recover the Hyland-Ong semantics for A as the single arena in the singleton family denoted by $U(A^{\mathsf{n}})$.

2. Let $\Gamma \vdash M : \underline{B}$ be a CBN term. Then (up to isomorphism) we can recover the Hyland-Ong semantics for M as the single strategy in the singleton family denoted by the closed value $\mathtt{thunk}\,\lambda\overrightarrow{\mathtt{x}}.M$.

□

As an example, notice that, the type $\underline{B}$ given in Sect. 9.2.3 is (the translation of) the CBN type

$$\mathtt{bool} \,\&\, (((\mathtt{bool} \to \mathtt{bool}) \to \mathtt{bool}) + \mathtt{bool})$$

and M is (the translation of) the CBN term

$$\lambda \begin{cases} 0. & \mathtt{false}, \\ 1. & \mathtt{inl}\ \lambda \mathtt{x}.\,(\mathtt{if}\ (\mathtt{true}\text{`}\mathtt{x}) \begin{cases} \mathtt{then} & \mathtt{diverge}, \\ \mathtt{else} & \mathtt{if}\ \mathtt{false}\text{`}\mathtt{x}\ \mathtt{then}\ \mathtt{true}\ \mathtt{else}\ \mathtt{true} \end{cases}) \end{cases}$$

However, because forcing and producing are not made explicit in CBN, it is difficult to formulate from the CBN viewpoint a narrative comparable to that in Sect. 9.2.3, explaining in detail the denotation of M. This illustrates well the advantage of working with CBPV instead of CBN.

As another example, consider the following statement, variants of which are found throughout the games literature:

> In game semantics, each number is modelled as a simple interaction: the environment starts the computation with an initial move q (a *question*: "What is the number?") and P may respond by playing a natural number (an *answer* to the question). [AM98b]

An example of this would be the denotation of the PCF term 3. But using Prop. 83, we see this strategy as the denotation of the CBPV term `thunk produce` 3 (more accurately, the single strategy in that singleton family). The initial O-move represents the context forcing the thunk, and Player's response represents the program producing 3.

A further example of the difficulties that have been caused by the bias towards CBN is *linear head reduction*, a complicated reduction strategy reflecting the interaction present in CBN game semantics [DHR96]. Rather than substituting for every occurrence of `x` simultaneously, like β-reduction, linear head reduction substitutes for each occurrence of `x` as it is used. Now, we recall that `x` in CBN decomposes into `force x` in CBPV, so we expect a question-move every time `x` is used in CBN. Linear head reduction can be seen as an attempt to provide some explicit interaction corresponding to this move.

9.2.8 Properties

The interpretation of term constructors in detail, which is long and technical, is given in Sect. B.5.1. The properties below seem clear in the light of similar results for CBN and CBV, but it remains to check all the details. Therefore we have called them "claim" rather than "proposition".

Claim 84 (equational soundness) The game semantics we have described validates all the equations of CBPV, as well as the equations for control constructs given in Sect. 8.9.2. □

Because we have in the language both control effects and general store (excluding equality testing on cells), the form of the operational soundness/adequacy theorem is somewhat complicated.

Claim 85 For a configuration w, s, M, K, write $[\![w, s, M, K]\!]$ for

$$[\![\overrightarrow{\texttt{new y}_{Aj} := s_{Aj}}.\ (\texttt{changecos}\ K;\ M)[\overrightarrow{\texttt{y}_{Aj}/\texttt{cell}_{Aj}}]]\!]$$

where s_{Aj} is the contents of A-storing cell j in s. Then we have:

soundness If $w, s, M, K \rightsquigarrow w', s', M', K'$ then $[\![w, s, M, K]\!] = [\![w', s', M', K']\!]$

adequacy If w, s, M, K diverges then $[\![w, s, M, K]\!] = \bot$.

□

We can adapt this for the fragments of the language that exclude control effects and/or store. Finally, the results that make the pointer game model distinctive:

Claim 86 Let Γ and A be `ref`-free. Let Γ denotes $\{R_i\}_{i \in I}$ and let A denote $\{S_j\}_{j \in J}$. Then

universality every element of $\prod_{i \in I} \sum_{j \in J} \mathsf{Ostrat}\,(R_i^{\mathsf{P}}, S_j^{\mathsf{O}})$ is the denotation of some value $\Gamma \vdash^{\mathsf{v}} V : A$.

full abstraction two values $\Gamma \vdash^{\mathsf{v}} V, V' : A$ have the same denotation iff they are observationally equivalent.

Similarly for computations. □

9.3 Pointer Game Model For Jump-With-Argument

9.3.1 Two Interaction Semantics For CBPV

We have now seen two semantics for CBPV based on the interaction between different parts of a program.

- In Sect. 8.4.1 we saw that the translation of CBPV into Jump-With-Argument provides a "jumping implementation" for CBPV. In particular, both `force` instructions and `produce` instructions in CBPV are implemented as jump instructions in JWA.
- In Sect. 9.2 we gave a pointer game model for CBPV, in which forcing is represented as a Q-move and producing is represented as an A-move.

There seems to be a relationship between these two models: a jump in JWA is represented as a move in the game model. To elucidate this relationship, we first give a pointer game model for JWA—this is interesting in its own right. We then see that the pointer game model for CBPV (except for the Q/A distinction) can be recovered from it.

This result is hardly surprising, because we saw in Sect. 8.9 that CBPV+control is equivalent to JWA, so any model for CBPV+control must arise from a model of JWA. Furthermore, the semantics of types in the CBPV model exactly follows the form of a continuation model (U and F have the same denotation, and so forth). But the model for JWA is more intuitive than the CBPV model, so our reduction of the CBPV model to the JWA model achieves the goal described in Sect. 9.1.1—of providing an intuitive explanation for the semantics of types and judgements in

pointer games—to an even greater extent than we achieved it in Sect. 9.2. In particular, we see in the proof of Prop. 87 that the previously unmotivated notion of arena family can now understood as a representation of the JWA type canonical form (8.4).

The pointer game model for JWA embodies the following intuitions:

- a move represents a jump
- if move m points to prior move n, then move m represents a jump to a continuation received in move n.

Game semantics is especially apposite for JWA, because the essential intuitions of the language are about jumping. The functional semantics we saw in Sect. 8.2 fails to capture these intuitions.

9.3.2 Unlabelled Arenas

The basic difference between the model for CBPV and the model for JWA is that, in the latter, there is no Q/A labelling of tokens.

Definition 57 An *unlabelled arena* R is a structure $(\mathsf{tok}\ R, \mathsf{rt}\ R, \vdash_R)$.

- $\mathsf{tok}\ R$ is a countable set of *tokens*.
- $\mathsf{rt}\ R$ is a subset of $\mathsf{tok}\ R$, whose elements are called *roots* and $\vdash_R$ is a binary relation on $\mathsf{tok}\ R$, giving a forest as in Def. 51.

We write $\mathsf{rt}\ R$ for the set of roots of R. □

Thus an unlabelled arena is that the same as an arena, but without the Q/A labelling on tokens.

Unlabelled arenas are built up using the following constructions.

Definition 58 (cf. Def. 52)

1. Let $\{R_i\}_{i \in I}$ be a countable family of unlabelled arenas. We define $\mathsf{pt}_{i \in I} R_i$ to be the unlabelled arena with I roots and a copy of R_i grafted underneath the ith root. More formally, it has tokens

   - $\mathsf{root}\ i$ for each $i \in I$
   - $\mathsf{under}(i, r)$ for each $i \in I$ and $r \in \mathsf{tok}\ R_i$

   where

   - the roots are the tokens $\mathsf{root}\ i$
   - $\mathsf{root}\ i \vdash \mathsf{under}(i, r)$, whenever r is a root of R_i
   - $\mathsf{under}(i, r) \vdash \mathsf{under}(i, s)$ for $r \vdash s$ in R_i

2. We define the unlabelled arena $R \uplus R'$ to be the disjoint union of the unlabelled arenas R and R'. Formally, it has tokens

   - $\mathsf{inl}\ r$ for each $r \in \mathsf{tok}\ R$
   - $\mathsf{inr}\ r$ for each $r \in \mathsf{tok}\ R'$

   where

   - the root tokens are $\mathsf{inl}\ r$, where $r \in \mathsf{rt}\ R$, and $\mathsf{inr}\ r$, where $r \in \mathsf{rt}\ R'$
   - $\mathsf{inl}\ r \vdash \mathsf{inl}\ s$ when $r \vdash_R s$
   - $\mathsf{inr}\ r \vdash \mathsf{inr}\ s$ when $r \vdash_{R'} s$

□

9.3.3 Semantics of Types

A JWA type denotes a countable family of unlabelled arenas. The type constructors other than `ref` (which we deal with below) are interpreted as follows.

- If A denotes $\{R_i\}_{i\in I}$, then $\neg A$ denotes the singleton family $\{\mathsf{pt}_{i\in I} R_i\}$.
- If, for each $i \in I$, A_i denotes $\{R_{ij}\}_{j\in J_i}$, then $\sum_{i\in I} A_i$ denotes $\{R_{ij}\}_{(i,j)\in \sum_{i\in I} J_i}$.
- 1 denotes the singleton family containing the empty arena.
- If A denotes $\{R_i\}_{i\in I}$ and A' denotes $\{S_j\}_{j\in J}$ then $A \times A'$ denotes $\{R_i \uplus S_j\}_{(i,j)\in I\times J}$.

For infinitely deep types, we say that $\{R_i\}_{i\in I} \subseteq \{S_j\}_{j\in J}$ when $I \subseteq J$ and R_i is a sub-arena of S_i for each $i \in I$. The class of arena families has all countable directed joins, and so we set the denotation of a type A to be $\bigcup_{B\lhd_{\mathsf{fin}} A}[\![B]\!]$. This can also be used to interpret recursive types, as in [McC97].

We can now give a definability result for types.

Proposition 87 The *type canonical forms* for infinitary JWA are given coinductively by

$$A ::= \quad \sum_{i\in I} \neg A_i$$

Every countable family of arenas A is isomorphic to the denotation of some type canonical form θA. □

Proof We define θ by guarded induction:

$$\theta\{R_i\}_{i\in I} \quad = \quad \sum_{i\in I} \neg\theta\{R_i \upharpoonright_j\}_{j\in \mathsf{rt}\ R_i}$$

By standard coinductive reasoning, this equation has a unique solution. We construct the isomorphism

$$\{R_i\}_{i\in I} \quad \cong \quad [\![\theta\{R_i\}_{i\in I}]\!]$$

This is done as in the proof of Prop. 81, which we relegated to Sect. B.6. □

The significance of this result is that we can view an unlabelled arena family as being nothing more than a representation of a type canonical form. From this perspective, the interpretation of $\sum$, of $\times$ and of $\neg$ are all clear.

It remains to interpret `ref`: the denotation of `ref` A is the same as the denotation of $\neg(\neg A + (A \times \neg 1))$. Like in Sect. 9.2.2, this is based on Reynolds' conception of a cell [3] as an "object with two methods": reading and assignment [Rey81]. Explicitly, we can convert a value V of type `ref` A into a value $\tilde{V}$ of type $\neg((\neg A) + (A \times \neg 1))$, defined by

$$\tilde{V} = \gamma \left\{ \begin{array}{ll} \texttt{inl k}. & \texttt{read}\ V\ \texttt{as x}.\ (\texttt{x} \nearrow \texttt{k}) \\ \texttt{inr (x,k)}. & V := \texttt{x};\ (() \nearrow \texttt{k}) \end{array} \right.$$

Reading and assignment can then be recovered from $\tilde{V}$ as follows:

$$\begin{array}{rcl} \texttt{read}\ V\ \texttt{as x}.\ M & = & (\texttt{inl}\ \gamma\texttt{x}.M) \nearrow \tilde{V} \\ V := W;\ M & = & \texttt{inr}\ (V, \gamma().M) \nearrow \tilde{V} \end{array}$$

[3] Recall from Sect. 9.1.3 that we excluded equality testing of cells from the language. If we had not done so, this conception of cells would not be valid.

9.3.4 Semantics of Judgements

Like a type, a context Γ denotes a countable family of unlabelled arenas. The empty context has the same denotation as 1, and context extension is interpreted the same way as $\times$. The semantics of judgements is as follows:

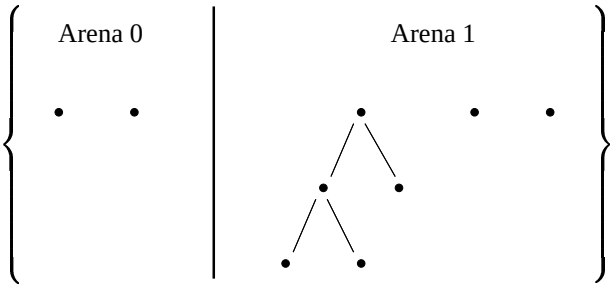
- If $[\![\Gamma]\!] = \{R_i\}_{i\in I}$ and $[\![A]\!] = \{S_j\}_{j\in J}$ then a value $\Gamma \vdash^{\mathsf{v}} V : A$ denotes an element of $\prod_{i\in I}\Sigma_{j\in J}\mathsf{Ostrat}\,(R_i^{\mathsf{P}}, S_j^{\mathsf{O}})$. Notice that these elements form a cpo.
- If $[\![\Gamma]\!] = \{R_i\}_{i\in I}$ then a non-returning command $\Gamma \vdash^{\mathsf{n}} M$ denotes an element of $\prod_{i\in I}\mathsf{Pstrat}\,R_i^{\mathsf{P}}$. Notice that these elements form a pointed cpo.

(The cpo/cppo structure on the model allows us to interpret recursive and infinitely deep terms.)

We give an example. Consider the type

$$A = \neg 2 + \neg(\neg\neg 2 \times \neg 1 + 2)$$

It denotes an arena family of size 2, depicted thus



Now consider $\mathtt{x} : A \vdash^{\mathsf{n}} M$, where M is the following non-returning command:

$$\mathtt{pm\ x\ as}\ \begin{cases} \mathtt{inl\ y}. & \mathtt{y} \nwarrow \mathtt{false} \\ \mathtt{inr\ z}. & \mathtt{z} \nwarrow \mathtt{inl}\ (\gamma\mathtt{w}.(\mathtt{w} \nwarrow \mathtt{true}), \gamma().(\mathtt{z} \nwarrow \mathtt{inr\ true})) \end{cases}$$

$[\![M]\!]0$ is a strategy for the P-first game on Arena 0^{P}. It describes the behaviour of M when $\mathtt{x}$ is bound to $\mathtt{inl}\ V$:

P-**move 0** **JWA terminology** M jumps to V taking the argument `false`.

game terminology So Player passes the root token corresponding to false.

$[\![M]\!]1$ is a strategy for the P-first game on Arena 1^{P}. It describes the behaviour of M when $\mathtt{x}$ is bound to $\mathtt{inr}\ V$:

P-**move 0** **JWA terminology** M jumps to V and produces `inl` of a pair of continuations.

game terminology So Player passes the root-token corresponding to inl.

O-**move 1** **game terminology** Now suppose that Opponent points to move 0 and passes the left token.

JWA terminology This means that the context jumps to the first continuation that it received in move 0, taking a continuation W.

P-**move 2** **JWA terminology** M jumps to W—the continuation it received in move 1—taking `true` as argument.

game terminology So Player points to move 1 and passes the token corresponding to true.

O-**move 3** **game terminology** Now suppose that Opponent points to move 0 and passes the right token.

JWA terminology This means that the context jumps to the second continuation that it received in move 1, taking $()$ as argument.

P-**move 4** **JWA terminology** M jumps to V taking `inr true` as argument.

game terminology So Player passes the root-token corresponding to inr true.

There is no pointer, because the continuation being jumped to, viz. V, was received not during play but before play began.

And so forth.

9.3.5 Properties

The interpretation of term constructors in detail, which is long and technical, is given in Sect. B.5.3. The properties below seem clear in the light of similar results for CBN and CBV, but it remains to check all the details. Therefore we have called them "claim" rather than "proposition".

Claim 88 (equational soundness) The game semantics we have described validates all the equations of Jump-With-Argument. □

Claim 89 For a configuration w, s, M, write $[\![w, s, M]\!]$ for

$$[\![\overrightarrow{\texttt{new}\ \texttt{y}_{Aj} := s_{Aj}}.\ M[\overrightarrow{\texttt{y}_{Aj}/\texttt{cell}_{A}\,j}]]\!]$$

where s_{Aj} is the contents of A-storing cell j in s. Then we have:

soundness If $w, s, M \rightsquigarrow w', s', M'$ then $[\![w, s, M]\!] = [\![w', s', M']\!]$

adequacy If w, s, M diverges, then $[\![w, s, M]\!] = \bot$.

□

We can adapt this for the fragments of the language that exclude store. Finally, the results that make the pointer game model distinctive:

Claim 90 Let Γ and A be `ref`-free. Let Γ denotes $\{R_i\}_{i\in I}$ and let A denote $\{S_j\}_{j\in J}$. Then

universality every element of $\prod_{i\in I}\sum_{j\in J}\mathsf{Ostrat}\,(R_i^{\mathsf{P}}, S_j^{\mathsf{O}})$ is the denotation of some value $\Gamma \vdash^{\mathsf{v}} V : A$.

full abstraction two values $\Gamma \vdash^{\mathsf{v}} V, V' : A$ have the same denotation iff they are observationally equivalent.

Similarly for non-returning commands. □

9.4 Obtaining The CBPV Model From The JWA Model, Which Is Simpler

Using the OPS transform, we can obtain the CBPV model—except for the Q/A distinction—from the JWA model. The advantage of this is that the type canonical forms of JWA are simpler and we can immediately see that unlabelled arenas are representations of them. Furthermore, in CBPV the consumer that we produce to is implicit (as we explained in Sect. 1.5.3), whereas in JWA the destination of a jump is always explicit. These two facts make the JWA model more intuitive than the CBPV model. Therefore, by reducing the latter to the former, the intuitive motivation for the definitions is strengthened.

If R is an arena, write $\mathsf{forgetQA}\ R$ for the unlabelled arena obtained by discarding the labelling function on R.

Proposition 91 Let A be a CBPV value type, denoting the arena family $\{R_i\}_{i\in I}$. Then its OPS transform $\overline{A}$ denotes (up to isomorphism) the unlabelled arena family $\{\mathsf{forgetQA}\ R_i\}_{i\in I}$. Similarly for computation types. □

Claim 92 1. Suppose Γ denotes the arena family $\{R_i\}_{i\in I}$ and A denotes the arena family $\{S_j\}_{j\in J}$. We have

$$\prod_{i\in I}\sum_{j\in J}\mathsf{Ostrat}\ (R_i^{\mathsf{P}}, S_j^{\mathsf{O}}) = \prod_{i\in I}\sum_{j\in J}\mathsf{Ostrat}\ (\mathsf{forgetQA}\ {R_i}^{\mathsf{P}}, \mathsf{forgetQA}\ {S_j}^{\mathsf{O}})$$

Clearly a value $\Gamma \vdash^{\mathsf{v}} V : A$ denotes an element of the LHS, while its OPS transform $\overline{\Gamma} \vdash^{\mathsf{v}} \overline{V} : \overline{A}$ denotes an element of the RHS. These denotations are the same.

2. Suppose Γ denotes the arena family $\{R_i\}_{i\in I}$ and $\underline{B}$ denotes the arena family $\{S_j\}_{j\in J}$. We have

$$\prod_{i\in I}\prod_{j\in J}\mathsf{Pstrat}\ (R_i^{\mathsf{P}}, S_j^{\mathsf{P}}) \cong \prod_{(i,j)\in I\times J}\mathsf{Pstrat}\ (\mathsf{forgetQA}\ R_i \uplus \mathsf{forgetQA}\ S_j)^{\mathsf{P}}$$

Clearly a computation $\Gamma \vdash^{\mathsf{c}} M : \underline{B}$ denotes an element of the LHS, while its OPS transform $\overline{\Gamma}, \overline{\underline{B}} \vdash^{\mathsf{n}} \overline{M}$ denotes an element of the RHS. These denotations are the same modulo the isomorphism.

□

It is also possible to obtain the JWA model from the CBPV model, by embedding JWA in CBPV + $\underline{\texttt{Ans}}$ and then replacing $\underline{\texttt{Ans}}$ with $\underline{\texttt{nrcomm}}$. But because the CBPV model is more complicated than the JWA model, this direction is not very interesting.

Part III

Categorical Semantics

Chapter 10

Background: Models Of Effect-Free Languages

10.1 The Goals Of Categorical Semantics

We consider the benefits of categorical semantics for an equational theory to be as follows.

1. It simplifies and organizes the task of constructing a concrete denotational model for the theory. Although it is possible to describe a model directly, this is often messy.
2. If the term model is an instance of the chosen categorical structure, this can give a different perspective on the structure of the equational theory.
3. It places models of the theory within a wider mathematical context (e.g. the theory of monads).

While not all categorical accounts achieve all of these goals, and some achieve other valuable goals that we have not listed, these will provide useful guidelines.

What our categorical account will *not* provide is an alternative motivation for CBPV. We do not believe that CBPV can be motivated from a purely categorical perspective—the operational perspective is essential.

10.2 Overview Of Chapter

Before we can study models of CBPV, we need to study an easier subject: models of effect-free languages. This is the subject of this chapter. All the semantics in this chapter are well known, but our organization and emphasis are somewhat novel.

Our first task is to establish the basic principles of categorical semantics. We do this by looking, in some detail, at the most familiar instance of it: the characterization of models of $\times$-calculus (a language with finite products) as cartesian categories.

We then turn to semantics of other type constructors: countable products, exponents and sum types. All of these categorical semantics are based on the theory of representable functors, which we review in Sect. 10.3. However, the semantics for sum types is based on representable functors in a locally indexed setting (this appears in [Jac99]), so after giving an *ad hoc* treatment of semantics of sum types in Sect. 10.5.3–10.5.4, we look at the theory of locally indexed categories in Sect. 10.6.

10.3 Representable Functors

We review the theory of representable functors, which is essential to categorical semantics. For the reader's convenience, we present each definition and result both for covariant and contravari-

ant functors. All of this section is standard material, e.g. [Mac71].

The basic notion is the *homset functor*:

Definition 59 Let $\mathcal{B}$ be a category. We write $\mathsf{Hom}_{\mathcal{B}}$—or just $\mathcal{B}$—for the functor

$$\mathcal{B}^{\mathsf{op}} \times \mathcal{B} \longrightarrow \mathbf{Set}$$

$$(X,Y) \longmapsto \mathcal{B}(X,Y)$$

$$(f,h) \longmapsto \lambda g.(f;g;h)$$

□

We also need *element categories*:

Definition 60 **covariant** Let $\mathcal{F}$ be a functor from $\mathcal{B}$ to **Set**. We define $\mathsf{el}\ \mathcal{F}$ (the "category of covariant elements of $\mathcal{F}$") to be the category in which

- an object is a pair (X,x) where $X \in \mathsf{ob}\ \mathcal{B}$ and $x \in \mathcal{F}X$;
- a morphism from (X,x) to (Y,y) is a $\mathcal{B}$-morphism $X \xrightarrow{f} Y$ such that $(\mathcal{F}f)x = y$.

contravariant Let $\mathcal{F}$ be a functor from $\mathcal{B}^{\mathsf{op}}$ to **Set**. We define $\mathsf{el}^{\mathsf{op}}\ \mathcal{F}$ (the "category of contravariant elements of $\mathcal{F}$") to be the category in which

- an object is a pair (X,x) where $X \in \mathsf{ob}\ \mathcal{B}$ and $x \in \mathcal{F}X$;
- a morphism from (X,x) to (Y,y) is a $\mathcal{B}$-morphism $X \xrightarrow{f} Y$ such that $(\mathcal{F}f)y = x$.

□

Proposition 93 (Yoneda lemma) Let $\mathcal{B}$ be a category.

covariant Let $\mathcal{F}$ be a functor from $\mathcal{B}$ to **Set**, and let V be an object of $\mathcal{B}$. Then we have a canonical bijection

$$\mathcal{F}V \quad \cong \quad [\mathcal{B},\mathbf{Set}](\lambda X.\mathcal{B}(V,X),\mathcal{F})$$

$$v \longmapsto \lambda X.\lambda f.(\mathcal{F}f)v$$

$$(\alpha V)\mathsf{id}_V \longmapsfrom \alpha$$

Rider Given $v \in \mathsf{LHS}$, the corresponding $\alpha \in \mathsf{RHS}$ is an isomorphism iff (V,v) is initial in $\mathsf{el}\ \mathcal{F}$.

contravariant Let $\mathcal{F}$ be a functor from $\mathcal{B}^{\mathsf{op}}$ to **Set**, and let V be an object of $\mathcal{B}$. Then we have a canonical bijection

$$\mathcal{F}V \quad \cong \quad [\mathcal{B}^{\mathsf{op}},\mathbf{Set}](\lambda X.\mathcal{B}(X,V),\mathcal{F})$$

$$v \longmapsto \lambda X.\lambda f.(\mathcal{F}f)v$$

$$(\alpha V)\mathsf{id}_V \longmapsfrom \alpha$$

Rider Given $v \in \mathsf{LHS}$, the corresponding $\alpha \in \mathsf{RHS}$ is an isomorphism iff (V,v) is terminal in $\mathsf{el}^{\mathsf{op}}\ \mathcal{F}$.

□

The rider is not usually included in the Yoneda Lemma, but it is crucial: we will use it to prove Prop. 94.

Definition 61 Let $\mathcal{B}$ be a category.

covariant Let $\mathcal{F} : \mathcal{B} \longrightarrow$ **Set** be a functor. A *representation* for $\mathcal{F}$ consists of

isomorphism style a $\mathcal{B}$-object V (the *vertex*) together with an isomorphism

$$\mathcal{B}(V, X) \cong \mathcal{F}X \quad \text{natural in } X.$$

element style an initial object (V, v) in $\mathsf{el}\ \mathcal{F}$.

contravariant Let $\mathcal{F} : \mathcal{B}^{\mathsf{op}} \longrightarrow$ **Set** be a functor. A *representation* for $\mathcal{F}$ consists of

isomorphism style a $\mathcal{B}$-object V (the *vertex*) together with an isomorphism

$$\mathcal{B}(X, V) \cong \mathcal{F}X \quad \text{natural in } X$$

element style a terminal object (V, v) in $\mathsf{el}^{\mathsf{op}}\mathcal{F}$.

□

Proposition 94 covariant Def. 61(covariant, isomorphism style) and Def. 61(covariant, element-style) are equivalent.

contravariant Def. 61(contravariant, isomorphism style) and Def. 61(contravariant, element-style) are equivalent.

□

Proof Immediate from the rider in Prop. 93 □

It commmonly happens that we have described a construction on objects as the vertex of a functor's representation. We then wish to extend the construction to morphisms, giving a functor. For example, given $\mathcal{B}$-objects A and B, the object $A \times B$ can be described as the vertex of a representation of the functor $\lambda X.(\mathcal{B}(X, A) \times \mathcal{B}(X, B))$ from $\mathcal{B}^{\mathsf{op}}$ to **Set**. If we know that such a representation exists for every A and B (as it must in a cartesian category), then we want to extend $\times$ to a functor from $\mathcal{B} \times \mathcal{B}$ to $\mathcal{B}$.

The general result that enables us to do this is the following.

Proposition 95 (Parametrized Representability) covariant Let $\mathcal{F} : \mathcal{I} \times \mathcal{B} \longrightarrow$ **Set** be a functor. Suppose that for each $I \in \mathcal{I}$ there is a representation

$$\mathcal{B}(V(I), X) \cong \mathcal{F}(I, X) \quad \text{natural in } X. \tag{10.1}$$

Then V extends uniquely to a functor from $\mathcal{I}$ to $\mathcal{B}^{\mathsf{op}}$ so as to make (10.1) natural in I.

contravariant Let $\mathcal{F} : \mathcal{I} \times \mathcal{B}^{\mathsf{op}} \longrightarrow$ **Set** be a functor. Suppose that for each $I \in \mathcal{I}$ there is a representation

$$\mathcal{B}(X, V(I)) \cong \mathcal{F}(I, X) \quad \text{natural in } X. \tag{10.2}$$

Then V extends uniquely to a functor from $\mathcal{I}$ to $\mathcal{B}$ so as to make (10.2) natural in I.

□

Finally, we review the Yoneda embedding.

Definition 62 Let $\mathcal{B}$ be a category. We write $\hat{\mathcal{B}}$ for the *presheaf category* $[\mathcal{B}^{\mathsf{op}}, \mathbf{Set}]$. The *Yoneda embedding* is the functor

$$\mathcal{B} \xrightarrow{\quad\mathcal{Y}\quad} \hat{\mathcal{B}}$$

$$B \longmapsto \lambda X.\mathcal{C}(X,B)$$

$$\mathcal{B}(B,C) \xrightarrow{\mathcal{Y}(B,C)} [\mathcal{B}^{\mathsf{op}}, \mathbf{Set}](\lambda X.\mathcal{B}(X,B), \lambda X.\mathcal{B}(X,C))$$

$$g \longmapsto \lambda X.\lambda f.(f;g)$$

□

Proposition 96 The functor $\mathcal{Y}$ is fully faithful. □

Proof Putting B for V and $\mathcal{Y}C$ for $\mathcal{F}$ in Prop. 93(contravariant), we learn that $\mathcal{Y}(B,C)$ is a bijection. □

10.4 Categorical Semantics Of $\times$-Calculus

10.4.1 The Theorem

As in Chap. 6, we will begin by stating a plausible result, and then consider what definitions are required to make it true. Here is the result:

Proposition 97 Models of $\times$-calculus and cartesian categories are equivalent. □

By $\times$-calculus, we mean the equational theory presented in Fig. 10.1. We have used a pattern-match syntax, but a projection syntax would be equally acceptable.

Types

$$A ::= \quad 1 \mid A \times A$$

Terms

$$\frac{}{\Gamma, \mathtt{x} : A, \Gamma' \vdash \mathtt{x} : A} \qquad \frac{\Gamma \vdash M : A \quad \Gamma, \mathtt{x} : A \vdash N : B}{\Gamma \vdash \mathtt{let\ x\ be}\ M.\ N : B}$$

$$\frac{\Gamma \vdash M : A \quad \Gamma \vdash M' : A'}{\Gamma \vdash (M, M') : A \times A'} \qquad \frac{\Gamma \vdash M : A \times A' \quad \Gamma, \mathtt{x} : A, \mathtt{y} : A' \vdash N : B}{\Gamma \vdash \mathtt{pm}\ M\ \mathtt{as}\ (\mathtt{x},\mathtt{y}).N : B}$$

Equations, using conventions of Sect. 1.4.2

$$\begin{array}{lrcl}
(\beta) & \mathtt{let\ x\ be}\ M.\ N & = & N[M/\mathtt{x}] \\
(\beta) & \mathtt{pm}\ (M, M')\ \mathtt{as}\ (\mathtt{x},\mathtt{y}).N & = & N[M/\mathtt{x}, M'/\mathtt{y}] \\
(\eta) & N[M/\mathtt{z}] & = & \mathtt{pm}\ M\ \mathtt{as}\ (\mathtt{x},\mathtt{y}).N[(\mathtt{x},\mathtt{y})/\mathtt{z}]
\end{array}$$

Figure 10.1: The $\times$-Calculus

Prop. 97 should perhaps be called the "Fundamental Theorem of Categorical Semantics", at least for simply typed languages. The rest of Sect. 10.4 can be omitted by the reader who is content to understand Prop. 97 informally.

Our first step is to describe the semantics of types:

Definition 63 A *semantics of types for ×-calculus* , also called a × *object structure* is a tuple $\tau = (\tau_{\mathsf{ob}}, 1, \times)$ consisting of

- a class τ_{ob}, whose elements we call *objects*;
- an object 1;
- a binary operation $\times : \tau_{\mathsf{ob}} \times \tau_{\mathsf{ob}} \longrightarrow \tau_{\mathsf{ob}}$

□

We can now replace Prop. 97 by the following stronger statement; like Prop. 97 it is not precise because we have not yet defined "model of ×-calculus".

Proposition 98 Let τ be a × object structure. Then

- models of ×-calculus with semantics of types given by τ, and
- cartesian categories with object structure τ.

are equivalent. □

This statement will be easier to make precise than Prop. 97. We will form two categories and they will be equivalent.

10.4.2 Cartesian Categories

Definition 64 1. A *cartesian category* is a category $\mathcal{C}$ with a distinguished terminal object and distinguished binary products.

2. The *object structure* of a cartesian category $\mathcal{C}$ is the × object structure $(\mathsf{ob}\ \mathcal{C}, 1, \times)$ where
   - 1 is the distinguished terminal object of $\mathcal{C}$;
   - $A \times A'$ is the vertex of the distinguished product of the $\mathcal{C}$-objects A and A'.

□

We can now make precise one half of Prop. 98.

Definition 65 Let τ be a × object structure. The category $\mathbf{CartCat}_\tau$ is defined as follows:

- an object is a cartesian category with object structure τ.
- a morphism is an identity-on-objects functor preserving all structure.

□

Notice that the fixing of τ has allowed us to sidestep the question of whether a general cartesian functor should preserve structure on the nose or up to isomorphism, because the only cartesian functors we use are identity-on-objects.

10.4.3 Direct Models Of $\times$-Calculus

The real problem in making Prop. 98 precise is to give, direct from the equational theory, an *a priori* notion of "model for $\times$-calculus". The approach that we use was developed independently in [Jef99] and in [Lev96].

Definition 66 Let τ be a $\times$ object structure (as in Sect. 10.4).

1. A *τ-multigraph* s consists of a set $s(A_0, \ldots, A_{n-1}; B)$ (whose elements are called *edges* from $A_0, \ldots, A_{n-1}$ to B) for each finite sequence of τ-objects $A_0, \ldots, A_{n-1}$ and each τ-object B.

2. We write MGraph_τ for the category of τ-multigraphs for τ, with the obvious morphisms.

□

It is clear that, given semantics of types τ, a semantics of the $\vdash$ judgement for $\times$-calculus is precisely a τ-multigraph s. For s tells us that a term $A_0, \ldots, A_{n-1} \vdash M : B$ will denote an edge from $[\![A_0]\!], \ldots, [\![A_{n-1}]\!]$ to $[\![B]\!]$, but does not tell us *which* edge—that will depend on the particular term M.

The term "multigraph" is used because of the similarity with graphs. The only difference is that in a graph, the source of an edge is a single object, whereas in a multigraph it is a sequence of objects.

We still need a way of characterizing a semantics of terms for the $\times$-calculus. The key fact is that, for each object structure τ, *the terms and equations of $\times$-calculus define a monad $\mathcal{T}$ on MGraph_τ*. To see this, suppose that s is a τ-multigraph; we will build from s another τ-multigraph $\mathcal{T}s$ as follows. We think of s as a *signature*—each edge in s is regarded as a "function-symbol". We inductively define the *terms built from the signature s*, using the rules from Fig. 10.1, together with the rule

$$\frac{\Gamma \vdash M_0 : B_0 \quad \cdots \quad \Gamma \vdash M_{n-1} : B_{n-1}}{\Gamma \vdash f(M_0, \ldots, M_{n-1}) : C}$$

for each edge f from $B_0, \ldots, B_{n-1}$ to C in s. We define $\mathcal{T}s$ to be the τ-multigraph in which an edge from Γ to B is an equivalence class (under provable equality, using the equations given in Fig. 10.1) of terms $\Gamma \vdash M : B$ built from the signature s. This new multigraph can be thought of as the "free $\times$-calculus model generated by s", keeping τ fixed throughout.

The rest of the monad structure is given as follows:

- ηs takes an edge f in s to the term $f(\mathtt{x}_0, \ldots, \mathtt{x}_{n-1})$;
- μs is the "flattening transform": it takes $t(M_0, \ldots, M_{n-1})$, where t is an edge in $\mathcal{T}s$, to $t[\overrightarrow{(\mu s)M_i/\mathtt{x}_i}]$. It preserves all other term constructors; for example, it maps (M, M') to $((\mu s)M, (\mu s)M')$.
- If $s \xrightarrow{\alpha} s'$ is a multigraph homomorphism, then $\mathcal{T}\alpha$ replaces each occurrence of a function symbol f in a term built from s by αf. Thus it takes $f(M_0, \ldots, M_{n-1})$, where f is an edge in s, to $\alpha(f)((\mathcal{T}\alpha)M_0, \ldots, (\mathcal{T}\alpha)M_{n-1})$, and it preserves all other term constructors.

We omit the proof that $\mathcal{T}$ preserves identity and composition, η and μ are natural and $(\mathcal{T}, \eta, \mu)$ satisfies all the monad laws. These are all straightforward inductions.

Now let us think what information a direct model for $\times$-calculus should provide.

semantics of types It should provide a $\times$ object structure τ.

semantics of judgement It should provide a τ-multigraph s. Then we know that a term $A_0, \ldots, A_{n-1} \vdash M : B$ is going to denote an edge from $[\![A_0]\!], \ldots, [\![A_{n-1}]\!]$ to $[\![B]\!]$.

semantics of terms Given a term $A_0, \ldots, A_{n-1} \vdash M : B$ generated from signature s, it should specify the edge from $[\![A_0]\!], \ldots, [\![A_{n-1}]\!]$ to $[\![B]\!]$ in s that M denotes. Furthermore, provably equal terms should denote the same edge. Thus the model should provide a multigraph homomorphism θ from $\mathcal{T}s$ to s.

In fact, (s, θ) should be an algebra for the $\mathcal{T}$ monad. We summarize:

Definition 67 1. A *direct model for $\times$-calculus* consists of

- a $\times$ object structure τ;
- an algebra (s, θ) for the $\mathcal{T}$ monad on MGraph_τ.

2. We write $\mathbf{Direct}_\tau$ for the category of direct models for $\times$-calculus with object structure τ. Morphisms are algebra homomorphisms.

□

10.4.4 Equivalence Of Direct Models And Cartesian Categories

We can now formulate Prop. 98 precisely.

Proposition 99 Let τ be a $\times$ object structure. Then the categories $\mathbf{Direct}_\tau$ and $\mathbf{CartCat}_\tau$ are equivalent. □

Proof (outline) We write $A_0 \times \cdots \times A_{n-1}$ for the object $(\cdots(1 \times A_0)\cdots) \times A_{n-1}$.

- Let $\mathcal{C}$ be a cartesian category based on τ. Then we construct a τ-multigraph s by setting $s(A_0, \ldots, A_{n-1}; B)$ to be $\mathcal{C}(A_0 \times \cdots \times A_{n-1}, B)$. We define a structure θ on s in the evident way, by induction over terms built from the signature s, and verify all the required equations.
- Let (s, θ) be a direct model based on τ. Then we construct a cartesian category $\mathcal{C}$ based on τ by setting $\mathcal{C}(A, B)$ to be $s(A; B)$. The structure is defined in the evident way and all required equations verified.
- These operations are inverse up to isomorphism.

□

Our fixing of object structure in Prop. 99 sidesteps some subtle coherence issues. Observe, for example, that if instead of Def. 64 we had defined "cartesian category" to be "category with distinguished n-ary products for every $n \in \mathbb{N}$"—which is clearly an acceptable definition—then our approach would not work. However, we will not consider these issues further.

10.5 Adding Type Constructors

10.5.1 Countable Products

Suppose we extend $\times$-calculus with countable products, as shown in Fig. 10.2. We call this extended calculus $\times\prod$-calculus. Although we have already included finite products, they are so closely intertwined with context formation in Prop. 97 that it is reasonable to consider countable products separately.

To give a categorical semantics, we recall the following definition:

Definition 68 Let $\{A_i\}_{i \in I}$ be a family of objects in $\mathcal{B}$. Then a *product* for $\{A_i\}_{i \in I}$ is a representation for the functor $\lambda X. \prod_{i \in I} \mathcal{B}(X, A_i)$ from $\mathcal{B}^{\mathsf{op}}$ to **Set**. □

Types

$$A ::= \quad 1 \mid A \times A \mid \prod_{i \in I} A_i$$

Extra Terms

$$\frac{\cdots \ \Gamma \vdash M_i : B_i \ \cdots}{\Gamma \vdash \lambda\{\ldots, i.M_i, \ldots\} : \prod_{i \in I} B_i} \qquad \frac{\Gamma \vdash M : \prod_{i \in I} B_i}{\Gamma \vdash \hat{\imath}\text{`}M : B_{\hat{\imath}}}$$

Extra Equations, using conventions of Sect. 1.4.2

$$\begin{array}{lccc} (\beta) & \hat{\imath}\text{`}\lambda\{\ldots, i.M_i, \ldots\} & = & M_{\hat{\imath}} \\ (\eta) & M & = & \lambda\{\ldots, i.i\text{`}M, \ldots\} \end{array}$$

Figure 10.2: Extending $\times$-calculus with countable products, to give $\times\prod$-calculus

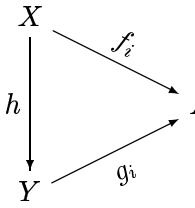
Since Def. 61 provides two equivalent definitions of "representation", we can think of a product for $\{A_i\}_{i \in I}$ in two ways:

isomorphism style as an isomorphism

$$\mathcal{B}(X, V) \cong \prod_{i \in I} \mathcal{B}(X, A_i) \quad \text{natural in } X \tag{10.3}$$

element style as a terminal object in the following category:

- an object is a *cone* i.e. a family of morphisms $X \xrightarrow{f_i} A_i$ with the same source X;
- a morphism from the cone $X \xrightarrow{f_i} A_i$ to the cone $Y \xrightarrow{g_i} A_i$ is a morphism $X \xrightarrow{h} Y$ such that



commutes for all $i \in I$.

Of these two descriptions, it is the isomorphism style which is more valuable to us, because (10.3) clearly describes the reversible derivation for $\prod$

$$\frac{\cdots \ \Gamma \vdash B_i \ \cdots}{\Gamma \vdash \prod_{i \in I} B_i}$$

Furthermore, the fact that (10.3) is natural in X corresponds to the fact that the reversible derivation preserves substitution in Γ (in the sense of Sect. 4.5).

The following result is therefore not surprising.

Proposition 100 Models of $\times\prod$-calculus and categories with countable products are equivalent. □

We can make this precise and prove it in the same way that we made Prop. 97 precise and proved it (in outline).

Notice that in a category with countable products, the isomorphism

$$\mathcal{C}(X, \prod_{i\in I} A_i) \cong \prod_{i\in I} \mathcal{C}(X, A_i)$$

can be said to be natural in each A_i as well as in X. This is just a consequence of Prop. 95. But it is only the naturality in X that needs to be checked when constructing a model, and that actually says something about the language (viz. that the reversible derivation commutes with substitution).

10.5.2 Exponents

Suppose we extend $\times$-calculus with exponents, as shown in Fig. 10.3. We call this extended calculus $\times\,\to$-calculus.

Types

$$A ::= \quad 1 \mid A \times A \mid A \to A$$

Extra Terms

$$\frac{\Gamma, \mathtt{x} : A \vdash M : B}{\Gamma \vdash \lambda \mathtt{x}.M : A \to B} \qquad \frac{\Gamma \vdash M : A \quad \Gamma \vdash N : A \to B}{\Gamma \vdash M \text{`} N : B}$$

Extra Equations, using conventions of Sect. 1.4.2

$$\begin{array}{lccc} (\beta) & M\text{`}\lambda \mathtt{x}.N & = & N[M/\mathtt{x}] \\ (\eta) & M & = & \lambda \mathtt{x}.(\mathtt{x}\text{`}M) \end{array}$$

Figure 10.3: Extending $\times$-calculus with exponents, to give $\times\,\to$-calculus

To give a categorical semantics for exponents, we make the following definition:

Definition 69 1. Let A and B be objects in a cartesian category $\mathcal{C}$. An *exponent* from A to B is a representation for the functor $\lambda X.\mathcal{C}(X \times A, B)$ from $\mathcal{C}^{\mathsf{op}}$ to **Set**.

2. A *cartesian closed category* is a cartesian category with a distinguished exponent from A to B for each $A, B \in \mathsf{ob}\ \mathcal{C}$.

□

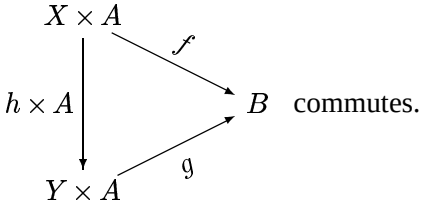
Since Def. 61 provides two equivalent definitions of "representation", we can think of an exponent from A to B in two ways:

isomorphism style as an isomorphism

$$\mathcal{C}(X, V) \cong \mathcal{C}(X \times A, B) \quad \text{natural in } X \tag{10.4}$$

element style as a terminal object in the following category:

- an object is a pair (X, f) where $X \times A \xrightarrow{f} B$;
- a morphism from (X, f) to (Y, g) is a morphism $X \xrightarrow{h} Y$ such that



Of these two descriptions, it is the isomorphism style which is more valuable to us, because (10.4) clearly describes the reversible derivation for $\to$

$$\frac{\Gamma, A \vdash B}{\Gamma \vdash A \to B}$$

Furthermore, the fact that (10.4) is natural in X corresponds to the fact that the reversible derivation preserves substitution in Γ (in the sense of Sect. 4.5).

The following result is therefore not surprising.

Proposition 101 Models of $\times \to$-calculus and cartesian closed categories are equivalent. □

We can make this precise and prove it in the same way that we made Prop. 97 precise and proved it (in outline).

Notice that in a cartesian closed category, the isomorphism

$$\mathcal{C}(X, A \to B) \cong \mathcal{C}(X \times A, B)$$

can be said to be natural in A and B as well as in X. This is just a consequence of Prop. 95. But it is only the naturality in X that needs to be checked when constructing a model, and that actually says something about the language (viz. that the reversible derivation commutes with substitution).

When we add both countable products and exponents to $\times$-calculus, we obtain the $\times\prod\to$ calculus. Its categorical semantics is straightforward:

Definition 70 A *countably cartesian closed category* is a cartesian closed category with a distinguished product for every countable family of objects. □

Proposition 102 Models of $\times\prod\to$-calculus and countably cartesian closed categories are equivalent. □

We can make this precise and prove it in the same way that we made Prop. 97 precise and proved it (in outline).

10.5.3 Element-Style Semantics for Sum Types

Suppose we extend $\times$-calculus with countable sum types, as shown in Fig. 10.4. We call this extended calculus $\times\sum$-calculus.

Types

$$A ::= \quad 1 \mid A \times A \mid \textstyle\sum_{i \in I} A_i$$

Extra Terms

$$\frac{\Gamma \vdash M : A_{\hat{\imath}}}{\Gamma \vdash (\hat{\imath}, M) : \sum_{i \in I} A_i} \qquad \frac{\Gamma \vdash M : \sum_{i \in I} A_i \quad \cdots \quad \Gamma, \mathtt{x} : A_i \vdash N_i : B \ \cdots}{\Gamma \vdash \mathtt{pm}\ M\ \mathtt{as}\ \{\ldots, (i, \mathtt{x}).N_i, \ldots\} : B}$$

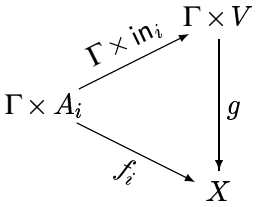
Extra Equations, using conventions of Sect. 1.4.2

$$\begin{array}{lrcl} (\beta) & \mathtt{pm}\ (\hat{\imath}, M)\ \mathtt{as}\ \{\ldots, (i, \mathtt{x}).N_i, \ldots\} & = & N_{\hat{\imath}}[M/\mathtt{x}] \\ (\eta) & N[M/\mathtt{z}] & = & \mathtt{pm}\ M\ \mathtt{as}\ \{\ldots, (i, \mathtt{x}).N[(i, \mathtt{x})/\mathtt{z}], \ldots\} \end{array}$$

Figure 10.4: Extending $\times$-calculus with sum types, to give $\times\sum$-calculus

We can give a simple categorical semantics for $\times\sum$-calculus using the following.

Definition 71 1. Let $\mathcal{C}$ be a cartesian category. A *distributive coproduct* for $\{A_i\}_{i\in I}$ consists of an object V together with morphisms $A_i \xrightarrow{\mathsf{in}_i} V$ such that for any family of morphisms $\Gamma \times A_i \xrightarrow{f_i} X$ there is a unique morphism $\Gamma \times V \xrightarrow{g} X$ such that for each $i \in I$ the diagram



commutes.

Notice that this definition is in element style only.

2. A *distributive category* is a cartesian category $\mathcal{C}$ with a distinguished distributive coproduct for every finite family of objects. (This is roughly the same as, and equivalent to, the definition in [CLW93, Coc93].)

3. A *countably distributive category* is a cartesian category $\mathcal{C}$ with a distinguished distributive coproduct for every countable family of objects.

□

We note the relationship between distributive coproducts and ordinary coproducts.

Proposition 103 Let $\mathcal{C}$ be a cartesian category.

1. Every distributive coproduct in $\mathcal{C}$ is a coproduct.

2. If $\mathcal{C}$ is cartesian closed, then every coproduct in $\mathcal{C}$ is a distributive coproduct.

□

[CLW93] gives the category of vector spaces as an example of a cartesian category with finite coproducts which is not distributive.

We can now formulate a categorical semantics for $\times\sum$-calculus.

Proposition 104 Models of $\times\sum$-calculus and countably distributive categories are equivalent. □

We can make this precise and prove it in the same way that we made Prop. 97 precise and proved it (in outline).

When we add countable products, exponents and sum types to $\times$-calculus, we obtain the $\times\sum\prod\to$-calculus.

Definition 72 1. A *bicartesian closed category* is a cartesian closed category with distinguished finite coproducts.

2. A *countably bicartesian closed category* is a countably cartesian closed category with distinguished countable coproducts.

□

Because of Prop. 103(2), we do not need to require distributive coproducts in Def. 72.

Proposition 105 Models of $\times\sum\prod\to$-calculus and countably bicartesian closed categories are equivalent. □

We can make this precise and prove it in the same way that we made Prop. 97 precise and proved it (in outline).

The category of countable sets and functions provides an example of a countably distributive category that is not cartesian closed. A more important example for our purposes is the category **SEAM** of SEAM predomains and continuous functions, defined in Def. 23. This is the category in which values are interpreted in the Scott model.

We see that distributive coproducts have more importance for us than they would have if we were primarily concerned with effect-free languages, where the presence of exponents makes it possible to use ordinary coproducts instead.

Here are some useful results, based on results in [Coc93].

Proposition 106 1. Suppose 0 is a distributive initial object (i.e. the vertex of a distributive coproduct of the empty family of objects) in a cartesian category. Then all morphisms to 0 are equal.

2. Suppose $(V, \{\mathsf{in}_i\}_{i \in I})$ is a distributive coproduct in a cartesian category. Then each in_i is monic.

3. In $\times\sum$-calculus, any two terms $\Gamma \vdash M, N : 0$ are provably equal.

4. If $\Gamma \vdash (\hat{\imath}, M) = (\hat{\imath}, N) : \sum_{i \in I} A_i$ is provable in $\times\sum$-calculus then $M = N$ is provable.

□

Proof We prove (3) by taking an identifier $\mathtt{z} : 0$ not in Γ. Then

$$M = M[M/\mathtt{z}] = \mathtt{pm}\ M\ \mathtt{as}\ \{\} = N[M/\mathtt{z}] = N$$

is provable. (1) is proved similarly.

To prove (4), we notice that

$$\begin{array}{rcl} M & = & \mathtt{pm}\ (\hat{\imath}, M)\ \mathtt{as}\ \{\ldots, (i,\mathtt{x}).M, \ldots, (\hat{\imath},\mathtt{x}).\mathtt{x}, \ldots\} \\ & = & \mathtt{pm}\ (\hat{\imath}, N)\ \mathtt{as}\ \{\ldots, (i,\mathtt{x}).M, \ldots, (\hat{\imath},\mathtt{x}).\mathtt{x}, \ldots\} \\ & = & N \end{array}$$

(2) is proved similarly. □

10.5.4 Isomorphism-Style Semantics For Sum Types

We saw in Sect. 10.3 that a representation of a functor can be described in two ways: isomorphism style and element style. We then saw in Sect. 10.5.1–10.5.2 that, for categorical semantics, it is the isomorphism style which is appropriate, because it matches a reversible derivation. By contrast, the element style appears *ad hoc*.

Unfortunately, in Def. 71, we defined "distributive coproduct" in element style only. We would like a definition in isomorphism style that matches the reversible derivation

$$\frac{\cdots\ \ \Gamma, A_i \vdash B\ \ \cdots}{\Gamma, \sum_{i \in I} A_i \vdash B}$$

A plausible attempt at this is the following

Definition 73 Let $\mathcal{C}$ be a cartesian category. A *pseudo-distributive coproduct* for $\{A_i\}_{i.\ I}$ consists of an object V together with an isomorphism

$$\mathcal{C}(\Gamma \times V, X) \cong \prod_{i \in I} \mathcal{C}(\Gamma \times A_i, X) \quad \text{natural in } \Gamma \in \mathcal{C}^{\mathsf{op}} \text{ and } X \in \mathcal{C}.$$

□

While every distributive coproduct gives a pseudo-distributive coproduct, the converse is probably false (we have not found a counterexample, but expect that one exists). The problem with Def. 73 is that, while the condition of naturality in Γ is correct (it just says that the reversible derivation commutes with substitution in Γ), the condition of naturality in $X \in \mathcal{C}$ is too weak.

Here is the correct isomorphism-style definition:

Definition 74 Let $\mathcal{C}$ be a cartesian category. A *distributive coproduct* for $\{A_i\}_{i.\ I}$ consists of an object V together with an isomorphism

$\mathcal{C}(\Gamma \times V, X) \cong \prod_{i \in I} \mathcal{C}(\Gamma \times A_i, X)$ natural in $\Gamma \in \mathcal{C}^{\mathsf{op}}$, and natural in X in the following strong sense:

for every $\mathcal{C}$-morphism $\Gamma \times X \xrightarrow{f} Y$, the diagram

$$\begin{array}{ccc} \mathcal{C}(\Gamma \times V, X) & \cong & \prod_{i \in I} \mathcal{C}(\Gamma \times A_i, X) \\ \Big\downarrow{\scriptstyle \mathcal{C}(\Gamma \times V, f)} & & \Big\downarrow{\scriptstyle \prod_{i \in I} \mathcal{C}(\Gamma \times A_i, f)} \\ \mathcal{C}(\Gamma \times V, Y) & \cong & \prod_{i \in I} \mathcal{C}(\Gamma \times A_i, Y) \end{array} \quad \text{commutes} \tag{10.5}$$

where we write $\mathcal{C}(\Gamma \times A, f)$—abusing notation—for the function that takes $\Gamma \times A \xrightarrow{g} X$ to

$$\Gamma \times A \xrightarrow{(\pi, g)} \Gamma \times X \xrightarrow{f} Y$$

□

Proposition 107 Def. 71 and Def. 74 are equivalent. □

Prop. 107 is a consequence of Prop. 113(1) as we explain in Sect. 10.6.6.

Notice that the only difference between Def. 73 and Def. 74 is that Def. 73 requires the commutativity of (10.5) only for morphisms $X \xrightarrow{f} Y$, whereas Def. 74 requires (10.5) for all morphisms $\Gamma \times X \xrightarrow{f} Y$—a stronger condition. This strong naturality condition can be seen in syntactic form as the "commuting conversion" equation

$$N[\mathtt{pm\ x\ as}\ \{\ldots,(i,\mathtt{y}).M_i,\ldots\}/\mathtt{z}] = \mathtt{pm\ x\ as}\ \{\ldots,(i,\mathtt{y}).N[M_i/\mathtt{z}],\ldots\} \tag{10.6}$$

for any term $\Gamma, \mathtt{z} : B \vdash N : C$.

We have now achieved an isomorphism-style characterization of "distributive coproduct". But a large part of the picture is missing, because Def. 74 is not just the isolated definition that it appears to be. By studying indexed categories, we will see that Def. 74 is actually an instance of the notion of "indexed coproduct". This achieves goal 3 of Sect. 10.1.

10.6 Locally Indexed Categories

10.6.1 Introduction

It is well-known that indexed categories are important for studying dependently typed languages. But they are also important for studying features of simply typed languages, especially sum types and computational effects. Although it is possible to study both these features without using indexed categories (as we did for sum types in Sect. 10.5.3–10.5.4), the treatment using indexed categories is simpler—and simplicity is a primary goal of categorical semantics.

Fortunately the indexed categories used for simply typed languages are of a special kind, where the morphisms are indexed but the objects are not. For this reason we call them *locally indexed categories*. They are easier to work with than general indexed categories: in particular, there are no coherence issues.

Our aims in this section are

- to make the reader as comfortable with locally indexed categories as with ordinary categories—in particular, looking at analogues of definitions, results and idioms from ordinary category theory;
- to show how locally indexed category theory simplifies Def. 74 (following [Jac99]).

10.6.2 Basics

Let $\mathcal{C}$ be a category.

Definition 75 A *locally $\mathcal{C}$-indexed category $\mathcal{D}$* consists of

- a class of objects $\mathsf{ob}\ \mathcal{D}$—we will underline these objects, except where they are the same as the objects of $\mathcal{C}$;
- for each object $\Gamma \in \mathsf{ob}\ \mathcal{C}$ and each pair of objects $\underline{X}, \underline{Y} \in \mathsf{ob}\ \mathcal{D}$, a set ("homset") $\mathcal{D}_\Gamma(\underline{X}, \underline{Y})$ of *morphisms* from $\underline{X}$ to $\underline{Y}$ over Γ—if f is such a morphism, we write $\underline{X} \xrightarrow[\Gamma]{f} \underline{Y}$
- for each object $\Gamma \in \mathsf{ob}\ \mathcal{C}$ and each object $\underline{X} \in \mathsf{ob}\ \mathcal{D}$, an *identity* morphism $\underline{X} \xrightarrow[\Gamma]{\mathsf{id}_{\Gamma,\underline{X}}} \underline{X}$
- for each morphism $\underline{X} \xrightarrow[\Gamma]{f} \underline{Y}$ and each morphism $\underline{Y} \xrightarrow[\Gamma]{g} \underline{z}$, a *composite* morphism $\underline{X} \xrightarrow[\Gamma]{f;g} \underline{Z}$
- for each $\mathcal{D}$-morphism $\underline{X} \xrightarrow[\Gamma]{f} \underline{Y}$ and each $\mathcal{C}$-morphism $\Gamma' \xrightarrow{k} \Gamma$, a *reindexed* $\mathcal{D}$-morphism $\underline{X} \xrightarrow[\Gamma']{k^*f} \underline{Y}$

such that

$$\begin{aligned}
\mathsf{id};f &= f \\
f;\mathsf{id} &= f \\
(f;g);h &= f;(g;h) \\
k^*\mathsf{id} &= \mathsf{id} \\
k^*(f;g) &= (k^*f);(k^*g) \\
\mathsf{id}^*f &= f \\
(l;k)^*f &= l^*(k^*f)
\end{aligned}$$

□

Definition 76 Let $\mathcal{D}$ be a locally $\mathcal{C}$-indexed category.

1. For each $\mathcal{C}$-morphism Γ, we write $\mathcal{D}_\Gamma$ (the "fibre over Γ") for the category whose objects are $\mathcal{D}$-objects and whose morphisms are $\mathcal{D}$-morphisms over Γ.

2. For each $\mathcal{C}$-morphism $\Gamma' \xrightarrow{k} \Gamma$, we write $\mathcal{D}_f$ (the "reindexing functor over f") or f^* for the identity-on-objects functor from $\mathcal{D}_\Gamma$ to $\mathcal{D}_{\Gamma'}$ given on morphisms by k^*.

$\mathcal{D}$ thus gives rise to a functor from $\mathcal{C}^{\mathsf{op}}$ to **Set**. □

A functor from $\mathcal{C}^{\mathsf{op}}$ to **Set** is often called a *strict $\mathcal{C}$-indexed category* [Cro94]. The word "strict" distinguishes it from a general $\mathcal{C}$-indexed category, which is a *pseudofunctor* from $\mathcal{C}^{\mathsf{op}}$ to **Set**, meaning that it can preserve composition up to isomorphism, rather than on the nose.

Here are some interesting characterizations of locally indexed categories, not used in the sequel.

Proposition 108 1. A locally $\mathcal{C}$-indexed category with class of objects $\mathcal{A}$ is precisely a strict $\mathcal{C}$-indexed category where all fibres have class of objects $\mathcal{A}$ and all reindexing functors are identity-on-objects.

2. A locally $\mathcal{C}$-indexed category is precisely a $[\mathcal{C}^{\mathsf{op}}, \mathbf{Set}]$-enriched category.

□

The most important examples of locally indexed categories are given by the following construction (sometimes called the *simple fibration*):

Definition 77 Let $\mathcal{C}$ be a cartesian category. We form a locally $\mathcal{C}$-indexed category $\mathsf{self}\ \mathcal{C}$ as follows:

- the objects are $\mathsf{ob}\ \mathcal{C}$;
- a morphism $X \xrightarrow[\Gamma]{f} Y$ is a $\mathcal{C}$-morphism $\Gamma \times X \xrightarrow{f} Y$;
- the identity on X over Γ is given by $\Gamma \times X \xrightarrow{\pi'} X$;
- the composite of

$$X \xrightarrow[\Gamma]{f} Y \xrightarrow[\Gamma]{g} Z$$

is given by

$$\Gamma \times X \xrightarrow{(\pi, f)} \Gamma \times Y \xrightarrow{g} Z$$

- the reindexing of $X \xrightarrow[\Gamma]{f} Y$ along $\Gamma' \xrightarrow{k} \Gamma$ is given by

$$\Gamma' \times X \xrightarrow{k \times X} \Gamma \times X \xrightarrow{f} Y$$

It is easy to verify the identity, associativity and reindexing laws. □

As with ordinary categories, we can form products and opposites of locally $\mathcal{C}$-indexed categories:

Definition 78 1. Given two locally $\mathcal{C}$-indexed categories $\mathcal{D}$ and $\mathcal{D}'$, we define the locally $\mathcal{C}$-indexed category $\mathcal{D} \times \mathcal{D}'$ by

$$\begin{aligned} \mathsf{ob}\,(\mathcal{D} \times \mathcal{D}') &= \mathsf{ob}\ \mathcal{D} \times \mathsf{ob}\ \mathcal{D}' \\ (\mathcal{D} \times \mathcal{D}')_\Gamma((A, A'), (B, B')) &= \mathcal{D}_\Gamma(A, B) \times \mathcal{D}'_\Gamma(A', B') \end{aligned}$$

with the evident identities, composition and reindexing.

2. Given a locally $\mathcal{C}$-indexed category $\mathcal{D}$, define the locally $\mathcal{C}$-indexed category $\mathcal{D}^{\mathsf{op}}$ by

$$\begin{array}{rcl} \mathsf{ob}\,\mathcal{D}^{\mathsf{op}} & = & \mathsf{ob}\,\mathcal{D} \\ \mathcal{D}^{\mathsf{op}}_{\Gamma}(A,B) & = & \mathcal{D}_{\Gamma}(B,A) \end{array}$$

with the evident identities, composition and reindexing.

□

Definition 79 Let $\mathcal{D}$ and $\mathcal{D}'$ be locally $\mathcal{C}$-indexed categories. A (locally $\mathcal{C}$-indexed) *functor* F from $\mathcal{D}$ to $\mathcal{D}'$ associates

- to each object $\underline{X} \in \mathsf{ob}\,\mathcal{D}$ an object $F\underline{X} \in \mathsf{ob}\,\mathcal{D}'$;
- to each morphism $X \xrightarrow[\Gamma]{f} \underline{Y}$ in $\mathcal{D}$ a morphism $F\underline{X} \xrightarrow[\Gamma]{Ff} F\underline{Y}$ in $\mathcal{D}'$

such that

$$\begin{array}{rcl} F\mathsf{id} & = & \mathsf{id} \\ F(f;g) & = & (Ff);(Fg) \\ F(k^*f) & = & k^*(Ff) \end{array}$$

□

Definition 80 A functor $\mathcal{D} \xrightarrow{F} \mathcal{D}'$ is *fully faithful functor* iff for every $F\underline{X} \xrightarrow[\Gamma]{f} F\underline{Y}$ there is a unique $\underline{X} \xrightarrow[\Gamma]{g} \underline{Y}$ such that $Fg = f$. □

Definition 81 Let $\mathcal{D}$ and $\mathcal{D}'$ be locally $\mathcal{C}$-indexed categories and let F and G be functors from $\mathcal{D}$ to $\mathcal{D}'$. A (locally $\mathcal{C}$-indexed) *natural transformation* α from F to G provides, for each object $\Gamma \in \mathsf{ob}\,\mathcal{C}$ and each object $\underline{X} \in \mathsf{ob}\,\mathcal{D}$ a morphism $F\underline{X} \xrightarrow[\Gamma]{\alpha_\Gamma \underline{X}} G\underline{X}$ such that

- for each $\Gamma' \xrightarrow{k} \Gamma$ in $\mathcal{C}$ and each $\underline{X} \in \mathcal{D}$ we have $k^*\alpha_\Gamma \underline{X} = \alpha_{\Gamma'}\underline{X}$;
- for each $\Gamma \in \mathsf{ob}\,\mathcal{C}$ and each $\underline{X} \xrightarrow[\Gamma]{f} \underline{Y}$ the diagram

$$\begin{array}{ccc} F\underline{X} & \xrightarrow{\alpha_\Gamma \underline{X}} & G\underline{X} \\ {\scriptstyle Ff}\downarrow & & \downarrow{\scriptstyle Gf} \\ F\underline{Y} & \xrightarrow[\alpha_\Gamma \underline{Y}]{} & G\underline{Y} \end{array} \quad \text{commutes.}$$

□

Notice that, if $\mathcal{C}$ has a terminal object 1, then $\alpha_\Gamma \underline{X}$ need be specified only for $\Gamma = 1$—this determines the rest of α.

By analogy with the 2-category **Cat** of ordinary categories, functors and natural transformations we can form a 2-category of locally $\mathcal{C}$-indexed categories, functors and natural transformations.

10.6.3 Homset Functors And The OpGrothendieck Construction

Homset functors are of central importance in the theory of representable functors (as we saw in Sect. 10.3) and also in the theory of adjunctions. So we certainly need to adapt them to the locally indexed setting. But there is a problem: for a locally indexed category $\mathcal{D}$, it is meaningless to look for a functor from $\mathcal{D}^{\mathsf{op}} \times \mathcal{D}$ to **Set**, because $\mathcal{D}^{\mathsf{op}} \times \mathcal{D}$ is a locally indexed category whereas **Set** is an ordinary category.

To surmount this problem, we need a way to obtain an ordinary category from a locally indexed category $\mathcal{D}$. We use the *opGrothendieck construction* . (The opGrothendieck construction is dual to the well-known Grothendieck construction, which we shall not use at all.)

Definition 82 Let $\mathcal{D}$ be a locally $\mathcal{C}$-indexed category. Then $\mathsf{opGroth}\ \mathcal{D}$ is the ordinary category defined as follows:

- an object of $\mathsf{opGroth}\ \mathcal{D}$ is a pair ${}_{\Gamma}\underline{X}$ where $\Gamma \in \mathsf{ob}\ \mathcal{C}$ and $\underline{X} \in \mathsf{ob}\ \mathcal{D}$;
- a morphism from ${}_{\Gamma}\underline{X}$ to ${}_{\Gamma'}\underline{Y}$ in $\mathsf{opGroth}\ \mathcal{D}$ consists of a pair ${}_{k}f$ where $\Gamma' \xrightarrow{\ k\ } \Gamma$ in $\mathcal{C}$ and $\underline{X} \xrightarrow[\Gamma']{g} \underline{Y}$ in $\mathcal{D}$;
- the identity on ${}_{\Gamma}\underline{X}$ is given by ${}_{\mathsf{id}}\mathsf{id}$;
- the composite of

$$ {}_{\Gamma}\underline{X} \xrightarrow{\ {}_{k}f\ } {}_{\Gamma'}\underline{Y} \xrightarrow{\ {}_{l}g\ } {}_{\Gamma''}\underline{Z} $$

  is given by ${}_{l;k}((l^{*}f);g)$.

It is easy to verify the identity and associativity laws. □

We can now define the homset functor, by analogy with Def. 59.

Definition 83 Let $\mathcal{D}$ be a locally $\mathcal{C}$-indexed category. We write $\mathsf{Hom}_{\mathcal{D}}$—or just $\mathcal{D}$—for the functor

$$ \mathsf{opGroth}(\mathcal{D}^{\mathsf{op}} \times \mathcal{D}) \longrightarrow \mathbf{Set} $$

$$ {}_{\Gamma}(\underline{X},\underline{Y}) \longmapsto \mathcal{D}_{\Gamma}(\underline{X},\underline{Y}) $$

$$ {}_{k}(f,h) \longmapsto \lambda g.(f;(k^{*}g);h) $$

□

The following will be used only in Sect. 14.6.4.

Lemma 109 Let $\mathcal{D}$ be a locally $\mathcal{C}$-indexed category. Then each $\mathcal{C}$-morphism $\Gamma' \xrightarrow{\ k\ } \Gamma$ induces a morphism in $\mathsf{opGroth}\ \mathcal{D}$

$$ {}_{\Gamma}\underline{Y} \xrightarrow{\ {}_{k}\underline{Y}\ } {}_{\Gamma'}k^{*}\underline{Y} \qquad \text{natural in } \underline{Y} \in \mathcal{D}_{\Gamma} $$

□

10.6.4 Naturality In Several Identifiers

One frequently uses the idiom "$\alpha(X,Y)$ is natural in X and Y", without specifying *jointly natural* or *separately natural*. As is well-known, this usage is justified because, for product categories, joint naturality and separate naturality are equivalent:

Proposition 110 Let $\mathcal{B}$ and $\mathcal{B}'$ be categories, and let $\mathcal{F}$ and $\mathcal{G}$ be functors from $\mathcal{B} \times \mathcal{B}'$ to **Set**. (Any category can be used in place of **Set**, but only the case of **Set** is relevant to us.) Suppose we are given a function $\mathcal{F}(X,Y) \xrightarrow{\alpha(X,Y)} \mathcal{G}(X,Y)$ for each $X \in \mathsf{ob}\ \mathcal{B}$ and $Y \in \mathsf{ob}\ \mathcal{B}'$. Then α is a natural transformation from $\mathcal{F}$ to $\mathcal{G}$ iff

- $\alpha(X,Y)$ is natural in $X \in \mathcal{B}$ for each $Y \in \mathsf{ob}\ \mathcal{B}'$;
- $\alpha(X,Y)$ is natural in $Y \in \mathcal{B}'$ for each $X \in \mathsf{ob}\ \mathcal{B}$.

□

In a similar way, we want to use an idiom "$\alpha_\Gamma \underline{X}$ is natural in Γ and $\underline{X}$". So we adapt Prop. 110 from product categories to ord categories.

Proposition 111 Let $\mathcal{D}$ be a locally $\mathcal{C}$-indexed category, and let $\mathcal{F}$ and $\mathcal{G}$ be functors from $\mathsf{opGroth}\ \mathcal{D}$ to **Set**. Suppose we are given a function $\mathcal{F}_\Gamma \underline{X} \xrightarrow{\alpha_\Gamma \underline{X}} \mathcal{G}_\Gamma \underline{X}$, for each $\Gamma \in \mathsf{ob}\ \mathcal{C}$ and $\underline{X} \in \mathsf{ob}\ \mathcal{D}$. Then α is a natural transformation from $\mathcal{F}$ to $\mathcal{G}$ iff

- $\alpha_\Gamma \underline{X}$ is natural in $\Gamma \in \mathcal{C}^{\mathsf{op}}$ for each $\underline{X} \in \mathsf{ob}\ \mathcal{D}$;
- $\alpha_\Gamma X$ is natural in $X \in \mathcal{D}_\Gamma$ for each $\Gamma \in \mathsf{ob}\ \mathcal{C}$.

□

Finally, we can adapt Prop. 110 and Prop. 111 to allow us to use the idiom "$\alpha_\Gamma(\underline{X},\underline{Y})$ is natural in Γ, $\underline{X}$ and $\underline{Y}$".

Proposition 112 Let $\mathcal{D}$ and $\mathcal{D}'$ be locally $\mathcal{C}$-indexed categories, and let $\mathcal{F}$ and $\mathcal{G}$ be functors from $\mathsf{opGroth}(\mathcal{D} \times \mathcal{D}')$ to **Set**. Suppose we are given a function $\mathcal{F}_\Gamma(\underline{X},\underline{Y}) \xrightarrow{\alpha_\Gamma(\underline{X},\underline{Y})} \mathcal{G}_\Gamma(\underline{X},\underline{Y})$, for each $\Gamma \in \mathsf{ob}\ \mathcal{C}$, $\underline{X} \in \mathsf{ob}\ \mathcal{D}$ and $\underline{Y} \in \mathsf{ob}\ \mathcal{D}'$. Then α is a natural transformation from $\mathcal{F}$ to $\mathcal{G}$ iff

- $\alpha_\Gamma(\underline{X},\underline{Y})$ is natural in $\Gamma \in \mathcal{C}^{\mathsf{op}}$ for each $\underline{X} \in \mathsf{ob}\ \mathcal{D}$ and $\underline{Y} \in \mathsf{ob}\ \mathcal{D}'$;
- $\alpha_\Gamma(\underline{X},\underline{Y})$ is natural in $\underline{X} \in \mathcal{D}_\Gamma$ for each $\Gamma \in \mathsf{ob}\ \mathcal{C}$ and $\underline{Y} \in \mathsf{ob}\ \mathcal{D}'$;
- $\alpha_\Gamma(\underline{X},\underline{Y})$ is natural in $\underline{Y} \in \mathcal{D}'_\Gamma$ for each $\Gamma \in \mathsf{ob}\ \mathcal{C}$ and $\underline{X} \in \mathsf{ob}\ \mathcal{D}$.

□

10.6.5 Representable and A-Representable Functors

Note We henceforth assume that the indexing category $\mathcal{C}$ is cartesian.

We recall from Sect. 10.3 that there are two definitions of representable functor: isomorphism style and element style, and that the former is more important for categorical semantics. So in this section, we define "representable functor" in the setting of locally indexed categories, using isomorphism style only. The element style definition, which is less important, is given in the next section.

The analogue of Def. 61 (isomorphism style) for locally indexed categories is as follows.

Definition 84 Let $\mathcal{D}$ be a locally $\mathcal{C}$-indexed category.

covariant Let $\mathcal{F}$ be a functor from $\mathsf{opGroth}\ \mathcal{D}$ to **Set**. A *representation* for $\mathcal{F}$ consists of a $\mathcal{D}$-object $\underline{V}$ (the *vertex*) together with an isomorphism

$$\mathcal{D}_\Gamma(\underline{V}, \underline{X}) \cong \mathcal{F}_\Gamma \underline{X} \quad \text{natural in } \Gamma \text{ and } \underline{X}.$$

contravariant Let $\mathcal{F}$ be a functor from $\mathsf{opGroth}(\mathcal{D}^{\mathsf{op}})$ to **Set**. A *representation* for $\mathcal{F}$ consists of $\mathcal{D}$-object $\underline{V}$ (the *vertex*) together with an isomorphism

$$\mathcal{D}_\Gamma(\underline{X}, \underline{V}) \cong \mathcal{F}_\Gamma \underline{X} \quad \text{natural in } \Gamma \text{ and } \underline{X}.$$

□

Isomorphism style definitions in the literature usually replace naturality in Γ with the *Beck-Chevalley condition*, to which it is equivalent. However, we consider naturality in Γ to be more intuitive.

We will also need the notion of *A-representable functor*, where A is an object of $\mathcal{C}$. As with representable functors, we provide only an isomorphism-style definition in this section.

Definition 85 Let $\mathcal{D}$ be a locally $\mathcal{C}$-indexed category and let A be an object of $\mathcal{C}$.

covariant Let $\mathcal{F}$ be a functor from $\mathsf{opGroth}\ \mathcal{D}$ to **Set**. An *A-representation* for $\mathcal{F}$ consists of a $\mathcal{D}$-object $\underline{V}$ (the *vertex*) together with an isomorphism

$$\mathcal{D}_\Gamma(\underline{V}, \underline{X}) \cong \mathcal{F}_{\Gamma \times A} \pi^*_{\Gamma, A} \underline{X} \quad \text{natural in } \Gamma \text{ and } \underline{X}.$$

contravariant Let $\mathcal{F}$ be a functor from $\mathsf{opGroth}(\mathcal{D}^{\mathsf{op}})$ to **Set**. An *A-representation* for $\mathcal{F}$ consists of $\mathcal{D}$-object $\underline{V}$ (the *vertex*) together with an isomorphism

$$\mathcal{D}_\Gamma(\underline{X}, \underline{V}) \cong \mathcal{F}_{\Gamma \times A} \pi^*_{\Gamma, A} \underline{X} \quad \text{natural in } \Gamma \text{ and } \underline{X}.$$

□

Def. 84 and Def. 85 are obtainable from each other:

- a representation for $\mathcal{F}$ is a 1-representation for $\mathcal{F}$;
- an A-representation for $\mathcal{F}$ is a representation for the functor

$$\mathcal{F}^A = \lambda_\Gamma \underline{X}. \mathcal{F}_{\Gamma \times A} \pi^*_{\Gamma, A} \underline{X}$$

Here are some examples of representations and A-representations.

Definition 86 Let $\mathcal{D}$ be a locally $\mathcal{C}$-indexed category.

1. A *coproduct* for a family of $\mathcal{D}$-objects $\{\underline{A}_i\}_{i \in I}$ is a representation for the functor $\lambda_\Gamma \underline{X}. \prod_{i \in I} \mathcal{D}_\Gamma(\underline{A}_i, \underline{X})$ from $\mathsf{opGroth}\ \mathcal{D}$ to **Set**. Explicitly, it is an isomorphism

$$\prod_{i \in I} \mathcal{D}_\Gamma(\underline{A}_i, \underline{Y}) \cong \mathcal{D}_\Gamma(\underline{V}, \underline{Y}) \text{ natural in } \Gamma \text{ and } \underline{Y}$$

2. A *product* for a family of $\mathcal{D}$-objects $\{\underline{A}_i\}_{i \in I}$ is a representation for the functor $\lambda_\Gamma \underline{X}. \prod_{i \in I} \mathcal{D}_\Gamma(\underline{X}, \underline{A}_i)$ from $\mathsf{opGroth}(\mathcal{D}^{\mathsf{op}})$ to **Set**. Explicitly, it is an isomorphism

$$\prod_{i \in I} \mathcal{D}_\Gamma(\underline{X}, \underline{A}_i) \cong \mathcal{D}_\Gamma(\underline{X}, \underline{V}) \quad \text{natural in } \Gamma \text{ and } \underline{X}$$

3. Let A be a $\mathcal{C}$-object. An *A-coproduct* for a $\mathcal{D}$ object $\underline{B}$ is an A-representation for the functor $\lambda_\Gamma \underline{X}.\mathcal{D}_\Gamma(\underline{B},\underline{X})$ from $\mathsf{opGroth}\ \mathcal{D}$ to **Set**. Explicitly, it is an isomorphism

$$\mathcal{D}_{\Gamma\times A}(\underline{B},\pi^*_{\Gamma,A}\underline{Y}) \cong \mathcal{D}_\Gamma(\underline{V},\underline{Y}) \text{ natural in } \Gamma \text{ and } \underline{Y}$$

4. Let A be a $\mathcal{C}$-object. An *A-product* for a $\mathcal{D}$-object $\underline{B}$ is an A-representation for the functor $\lambda_\Gamma \underline{X}.\mathcal{D}_\Gamma(\underline{X},\underline{B})$ from $\mathsf{opGroth}(\mathcal{D}^{\mathsf{op}})$ to **Set**. Explicitly, it is an isomorphism

$$\mathcal{D}_{\Gamma\times A}(\pi^*_{\Gamma,A}\underline{X},\underline{B}) \cong \mathcal{D}_\Gamma(\underline{X},\underline{V}) \text{ natural in } \Gamma \text{ and } \underline{X}$$

□

We examine what these structures mean in the special case of $\mathsf{self}\ \mathcal{C}$, the locally indexed category defined in Def. 77.

Proposition 113
1. A coproduct for $\{A_i\}_{i\in I}$ in $\mathsf{self}\ \mathcal{C}$ is a distributive coproduct for $\{A_i\}_{i\in I}$ in $\mathcal{C}$.
2. A product for $\{A_i\}_{i\in I}$ in $\mathsf{self}\ \mathcal{C}$ is a product for $\{A_i\}_{i\in I}$ in $\mathcal{C}$.
3. $\mathsf{self}\ \mathcal{C}$ has, for every A and B, an A-coproduct of B with vertex $A\times B$.
4. An A-product of B in $\mathsf{self}\ \mathcal{C}$ is an exponent from A to B in $\mathcal{C}$.

□

Prop. 113 is easy to prove using the element style definitions in the next section. For (1), the isomorphism-style definition of coproduct in $\mathsf{self}\ \mathcal{C}$ is exactly the isomorphism-style definition of distributive coproduct (Def. 74). We have thus achieved what was our main goal while studying locally indexed categories: to give a simple categorical semantics for sum types. The following will be useful in Chap. 14.

Definition 87 A locally $\mathcal{C}$-indexed category is *closed* if it has A-products for each $A \in \mathsf{ob}\ \mathcal{C}$. It is *countably closed* if it is closed and has all countable products. □

10.6.6 Yoneda Lemma

We give two forms of the Yoneda Lemma, one for representations and one for A-representations. Both are similar to the Yoneda Lemma for ordinary categories (Prop. 93).

Proposition 114 (cf. Prop. 93) Let $\mathcal{D}$ be a locally $\mathcal{C}$-indexed category.

covariant Let $\mathcal{F}$ be a functor from $\mathsf{opGroth}\ \mathcal{D}$ to **Set**, and let $\underline{V}$ be an object of $\mathcal{D}$. Then we have a canonical bijection

$$\mathcal{F}_1\underline{V} \quad \cong \quad [\mathsf{opGroth}\ \mathcal{D}, \mathbf{Set}](\lambda_\Gamma \underline{X}.\mathcal{D}_\Gamma(\underline{V},\underline{X}), \mathcal{F})$$

$$v \longmapsto \lambda_\Gamma \underline{X}.\lambda f.(\mathcal{F}_{()} f)v$$

$$(\alpha_1 \underline{V})\mathsf{id}_{\underline{V}} \longmapsfrom \alpha$$

Rider Given $v \in \mathsf{LHS}$, the corresponding $\alpha \in \mathsf{RHS}$ is an isomorphism iff $(\underline{V},v)$ satisfies the following "initiality" property: for any $\Gamma \in \mathsf{ob}\ \mathcal{C}$ and $\underline{X} \in \mathsf{ob}\ \mathcal{D}$ and any $x \in \mathcal{F}_\Gamma \underline{X}$ there is a unique $\underline{V} \xrightarrow[\Gamma]{f} \underline{X}$ such that $(\mathcal{F}_{()} f)v = x$.

contravariant Let $\mathcal{F}$ be a functor from $\mathsf{opGroth}(\mathcal{D}^{\mathsf{op}})$ to **Set**, and let $\underline{V}$ be an object of $\mathcal{D}$. Then we have a canonical bijection

$$\mathcal{F}_1\underline{V} \quad \cong \quad [\mathsf{opGroth}(\mathcal{D}^{\mathsf{op}}), \mathbf{Set}](\lambda_\Gamma \underline{X}.\mathcal{D}_\Gamma(\underline{X},\underline{V}), \mathcal{F})$$

$$v \longmapsto \lambda_\Gamma \underline{X}.\lambda f.(\mathcal{F}_{()}f)v$$

$$(\alpha_1\underline{V})\mathsf{id}_{\underline{V}} \longmapsto \alpha$$

Rider Given $v \in \mathsf{LHS}$, the corresponding $\alpha \in \mathsf{RHS}$ is an isomorphism iff $(\underline{V},v)$ satisfies the following "terminality" property: for any $\Gamma \in \mathsf{ob}\ \mathcal{C}$ and $\underline{X} \in \mathsf{ob}\ \mathcal{D}$ and any $x \in \mathcal{F}_\Gamma \underline{X}$ there is a unique $\underline{X} \xrightarrow[\Gamma]{f} \underline{V}$ such that $(\mathcal{F}_{()}f)v = x$.

□

As in Sect. 10.3, we can define a representation of $\mathcal{F}$ in element-style to be a pair $(\underline{V},v)$ satisfying the "initiality" property (or "terminality" property, in the contravariant case) stated in the rider, and the equivalence of the isomorphism-style and element-style definitions is immediate from the rider.

Prop. 107 is a corollary of this equivalence. *Proof* We set $\mathcal{D}$ to be $\mathsf{self}\ \mathcal{C}$ and we set $\mathcal{F}$ to be $\lambda_\Gamma \underline{X} \prod_{i\in I}(\mathsf{self}\ \mathcal{C})_\Gamma(A_i,\underline{X})$. A distributive coproduct in the sense of Def. 71(1) is an element-style representation for $\mathcal{F}$, while a distributive coproduct in the sense of Def. 74 is an isomorphism-style representation for $\mathcal{F}$. Hence the two definitions of distributive coproduct are equivalent, as claimed. □

The Yoneda Lemma for A-representations is given as follows. We recall from (10.6.5) that we write $\mathcal{F}^A$ is an abbreviation for $\lambda_\Gamma \underline{X}.\pi^*_{\Gamma,A}\underline{X}$.

Proposition 115 (cf. Prop. 93) Let $\mathcal{D}$ be a locally $\mathcal{C}$-indexed category, and let A be an object of $\mathcal{C}$. We write i for the isomorphism $A \xrightarrow{((),\mathsf{id}_A)} 1 \times A$.

covariant Let $\mathcal{F}$ be a functor from $\mathsf{opGroth}\ \mathcal{D}$ to **Set**, and let $\underline{V}$ be an object of $\mathcal{D}$. Then we have a canonical bijection

$$\mathcal{F}_A\underline{V} \quad \cong \quad [\mathsf{opGroth}\ \mathcal{D}, \mathbf{Set}](\lambda_\Gamma \underline{X}.\mathcal{D}_\Gamma(\underline{V},\underline{X}), \mathcal{F}^A)$$

$$v \longmapsto \lambda_\Gamma \underline{X}.\lambda f.(\mathcal{F}_{\pi'_{\Gamma,A}}\pi^*_{\Gamma,A}f)v$$

$$i^*((\alpha_1\underline{V})\mathsf{id}_{\underline{V}}) \longmapsto \alpha$$

Rider Given $v \in \mathsf{LHS}$, the corresponding $\alpha \in \mathsf{RHS}$ is an isomorphism iff $(\underline{V},v)$ satisfies the following "initiality" property: for any $\Gamma \in \mathsf{ob}\ \mathcal{C}$ and $\underline{X} \in \mathsf{ob}\ \mathcal{D}$ and any $x \in \mathcal{F}_{\Gamma\times A}\underline{X}$ there is a unique $\underline{V} \xrightarrow[\Gamma]{f} \underline{X}$ such that $(\mathcal{F}_{\pi'_{\Gamma,A}}\pi^*_{\Gamma,A}f)v = x$.

contravariant Let $\mathcal{F}$ be a functor from $\mathsf{opGroth}(\mathcal{D}^{\mathsf{op}})$ to **Set**, and let $\underline{V}$ be an object of $\mathcal{D}$. Then we have a canonical bijection

$$\mathcal{F}_A\underline{V} \quad \cong \quad [\mathsf{opGroth}(\mathcal{D}^{\mathsf{op}}), \mathbf{Set}](\lambda_\Gamma \underline{X}.\mathcal{D}_\Gamma(\underline{X},\underline{V}), \mathcal{F}^A)$$

$$v \longmapsto \lambda_\Gamma \underline{X}.\lambda f.(\mathcal{F}_{\pi'_{\Gamma,A}}\pi^*_{\Gamma,A}f)v$$

$$i^*((\alpha_1\underline{V})\mathsf{id}_{\underline{V}}) \longmapsto \alpha$$

Rider Given $v \in \mathsf{LHS}$, the corresponding $\alpha \in \mathsf{RHS}$ is an isomorphism iff $(\underline{V}, v)$ satisfies the following "terminality" property: for any $\Gamma \in \mathsf{ob}\ \mathcal{C}$ and $\underline{X} \in \mathsf{ob}\ \mathcal{D}$ and any $x \in \mathcal{F}_{\Gamma \times A}\underline{X}$ there is a unique $\underline{X} \xrightarrow[\Gamma]{f} \underline{V}$ such that $(\mathcal{F}_{\pi'_{\Gamma,A}} \pi^*_{\Gamma,A} f)v = x$.

□

Once again, we can define an A-representation of $\mathcal{F}$ in element-style to be a pair $(\underline{V}, v)$ satisfying the "initiality" property (or "terminality" property, in the contravariant case) stated in the rider, and the equivalence of the isomorphism-style and element-style definitions is immediate from the rider.

We adapt the Parametrized Representability Theorem from ordinary to locally indexed categories.

Proposition 116 (Parametrized Representability) (cf. Prop. 95) Let $\mathcal{I}$ and $\mathcal{D}$ be locally $\mathcal{C}$-indexed categories.

covariant Let $\mathcal{F} : \mathsf{opGroth}(\mathcal{I} \times \mathcal{D}) \longrightarrow$ **Set** be a functor. Suppose that for each $I \in \mathcal{I}$ there is a representation

$$\mathcal{D}_\Gamma(\underline{V}(I), \underline{X}) \cong \mathcal{F}_\Gamma(I, \underline{X}) \quad \text{natural in } \Gamma \text{ and } \underline{X}. \tag{10.7}$$

Then $\underline{V}$ extends uniquely to a functor from $\mathcal{I}$ to $\mathcal{D}$ so as to make (10.7) natural in I.

contravariant Let $\mathcal{F} : \mathsf{opGroth}(\mathcal{I} \times \mathcal{D}^{\mathsf{op}}) \longrightarrow$ **Set** be a functor. Suppose that for each $I \in \mathcal{I}$ there is a representation

$$\mathcal{D}_\Gamma(\underline{X}, \underline{V}(I)) \cong \mathcal{F}_\Gamma(I, \underline{X}) \quad \text{natural in } \Gamma \text{ and } \underline{X}. \tag{10.8}$$

Then $\underline{V}$ extends uniquely to a functor from $\mathcal{I}$ to $\mathcal{D}^{\mathsf{op}}$ so as to make (10.8) natural in I.

□

Finally, we adapt the Yoneda embedding from ordinary to locally indexed categories. This will be useful in Sect. 15.6.

Definition 88 Let $\mathcal{C}$ be a cartesian category, and let $\mathcal{D}$ be a locally $\mathcal{C}$-indexed category. We define $\hat{\mathcal{D}}$ to be the locally $\mathcal{C}$-indexed category where

- an object is a functor from $\mathsf{opGroth}(\mathcal{D}^{\mathsf{op}})$ to **Set**;
- a morphism from F to G over C is a natural transformation from F to G^C

with the evident identities, composition and reindexing. The *Yoneda embedding* is the functor

$$\mathcal{D} \xrightarrow{\quad\mathcal{Y}\quad} \hat{\mathcal{D}}$$

$$\underline{B} \longmapsto \lambda\Gamma\underline{X}.\mathcal{D}_\Gamma(\underline{X}, \underline{B})$$

$$\mathcal{D}_A(\underline{B}, \underline{C}) \xrightarrow{\mathcal{Y}_A(\underline{B},\underline{C})} [\mathsf{opGroth}(\mathcal{D}^{\mathsf{op}}), \mathbf{Set}](\lambda\Gamma\underline{X}.\mathcal{D}_\Gamma(\underline{X}, \underline{B}), \lambda\Gamma\underline{X}.\mathcal{D}_{\Gamma\times A}(\pi^*_{\Gamma,A}\underline{X}, \underline{C}))$$

$$g \longmapsto \lambda\Gamma\underline{X}.\lambda f.(\pi^*_{\Gamma,A} f; \pi'^*_{\Gamma,A} g)$$

This is easily checked to be a functor. □

Proposition 117 $\mathcal{Y}$ is fully faithful. □

Proof Putting $\underline{B}$ for $\underline{V}$ and $\mathcal{Y}\underline{C}$ for $\mathcal{F}$ in Prop. 115(contravariant), we learn that $\mathcal{Y}_A(\underline{B}, \underline{C})$ is a bijection. □

Chapter 11

Models Of CBPV: Overview

11.1 The Big Picture

Just as we defined object structure for $\times$-calculus (Def. 63), so we can define it for CBPV.

Definition 89 A *semantics of types for CBPV*, also called a *CBPV object structure*, is a tuple

$$\tau = (\tau_{\mathsf{valob}}, \tau_{\mathsf{compob}}, U, \sum, 1, \times, F, \prod, \to) \quad \text{consisting of}$$

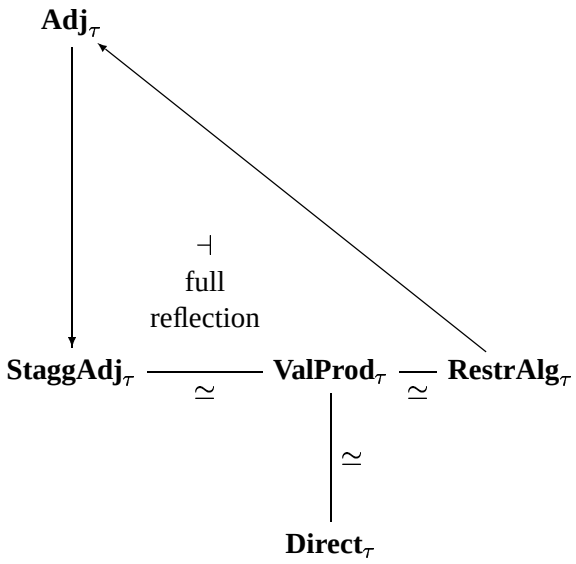
- a class τ_{valob}, whose elements we call *value objects*
- a class τ_{compob}, whose elements we call *computation objects*
- a function $U : \tau_{\mathsf{compob}} \longrightarrow \tau_{\mathsf{valob}}$
- a function $\sum_{i \in I} : \tau_{\mathsf{valob}}^I \longrightarrow \tau_{\mathsf{valob}}$ for every countable set I
- a value object 1
- a binary operation $\times : \tau_{\mathsf{valob}} \times \tau_{\mathsf{valob}} \longrightarrow \tau_{\mathsf{valob}}$
- a function $F : \tau_{\mathsf{valob}} \longrightarrow \tau_{\mathsf{compob}}$
- a function $\prod_{i \in I} : \tau_{\mathsf{compob}}^I \longrightarrow \tau_{\mathsf{compob}}$ for every countable set I
- a binary operation $\to$: $\tau_{\mathsf{valob}} \times \tau_{\mathsf{compob}} \longrightarrow \tau_{\mathsf{compob}}$.

□

For the remainder of Part III, we are going to construct and explain the diagram shown in Fig. 11.1, for any CBPV object structure τ.

Looking at this diagram, we see 4 equivalent categories:

- **Direct**$_\tau$ is the category of *direct models* for CBPV based on τ. We define these models from the CBPV equational theory, just as we defined direct models for $\times$ calculus—an outline is given in Sect. 11.2
- **RestrAlg**$_\tau$ is the category of *restricted algebra models* based on τ. These are explained in Chap. 12.
- **ValProd**$_\tau$ is the category of *value/producer models* based on τ. These are explained in Chap. 13.



Recall [Mac71] that a *full reflection* is an adjunction in which the counit is an isomorphism, or, equivalently, the right adjoint is fully faithful.

Figure 11.1: Models for CBPV

- **StaggAdj**$_\tau$ is the category of *staggered adjunction models* based on τ. These are explained in Sect. 14.8.

We also have **Adj**$_\tau$, the category of *adjunction models* based on τ. These models, which are explained in Chap. 14, are the most elegant. Unfortunately **Adj**$_\tau$ is not equivalent to the other categories; it is related to them by a full reflection.

The various kinds of models are similar to, and influenced by, the various models of CBV that are listed and shown equivalent in [PT99, PT97].

All our treatment of categorical semantics works with infinitely wide CBPV. The reader who prefers to work with finitely wide CBPV should substitute "finite" for "countable" throughout.

11.2 Direct Models For CBPV

Let τ be a CBPV object structure. We proceed as in Sect. 10.4.3. By analogy with Def. 66 we have

Definition 90 1. A *τ-multigraph s* consists of

- a set $s_{\mathsf{val}}(A_0, \ldots, A_{n-1}, B)$ (whose elements are called *value edges* from $A_0, \ldots, A_{n-1}$ to B) for each finite sequence of value objects $A_0, \ldots, A_{n-1}$ and each value object B;
- a set $s_{\mathsf{comp}}(A_0, \ldots, A_{n-1}, \underline{B})$ (whose elements are called *computation edges* from $A_0, \ldots, A_{n-1}$ to $\underline{B}$) for each finite sequence of value objects $A_0, \ldots, A_{n-1}$ and each computation object $\underline{B}$;

2. We write MGraph_τ for the category of τ-multigraphs for τ, with the obvious morphisms.

□

As in Sect. 10.4.3, the terms and equations of CBPV define a monad on MGraph_τ. We define inductively the "terms built from the signature s", using the term constructors of CBPV-with-

complex-values, with the additional rules

$$\frac{\Gamma \vdash^{\mathsf{v}} M_0 : B_0 \quad \cdots \quad \Gamma \vdash^{\mathsf{v}} M_{n-1} : B_{n-1}}{\Gamma \vdash^{\mathsf{v}} f(M_0, \ldots, M_{n-1}) : C}$$

for each value edge f from $B_0, \ldots, B_{n-1}$ to C in s, and

$$\frac{\Gamma \vdash^{\mathsf{v}} M_0 : B_0 \quad \cdots \quad \Gamma \vdash^{\mathsf{v}} M_{n-1} : B_{n-1}}{\Gamma \vdash^{\mathsf{c}} f(M_0, \ldots, M_{n-1}) : \underline{C}}$$

for each computation edge f from $B_0, \ldots, B_{n-1}$ to $\underline{C}$ in s. We define $\mathcal{T}s$ to be the τ-multigraph in which a value edge from Γ to A is an equivalence class (under provable equality) of values $\Gamma \vdash^{\mathsf{v}} V : B$ built from the signature s, and a computation edge from Γ to $\underline{B}$ is an equivalence class (under provable equality) of computations $\Gamma \vdash^{\mathsf{c}} M : \underline{B}$ built from the signature s. This new multigraph can be thought of as the "free CBPV model generated by s", keeping τ fixed throughout.

The rest of the monad structure is defined in the same way as in Sect. 10.4.3. By analogy with Def. 67 we have

Definition 91 1. A *direct model for CBPV* consists of

- a CBPV object structure τ;
- an algebra (s, θ) for the $\mathcal{T}$ monad on MGraph_τ.

2. We write $\mathbf{Direct}_\tau$ for the category of direct models for CBPV based on τ. Morphisms are algebra homomorphisms.

□

Like the CBPV term model discussed in Chap. 4, direct models have the reversible derivations listed in Prop. 23.

11.3 The Value Category

There is one part of the categorical semantics for CBPV that is already apparent from the results in Chap. 10. Since, as we explained in Sect. 4.8, $\times\sum$-calculus is a fragment of CBPV (replacing $\vdash$ by $\vdash^{\mathsf{c}}$), a CBPV model must include a countably distributive category. This category is called the *value category* of the model. For example, in each of the CBPV models of Chap. 6, the value category is either **Set** or **Cpo**. We use the terms *value objects* and *value morphisms* for the objects and morphisms of the value category.

The fact that the value category has countable distributive coproducts is all that we will need to say about the semantics of $\sum$ in CBPV. This may seem surprising, because $\sum$ has an elimination rule which is not included in the $\times\sum$-calculus fragment, viz.

$$\frac{\Gamma \vdash^{\mathsf{v}} V : \sum_{i \in I} A_i \quad \cdots \quad \Gamma, \mathtt{x} : A_i \vdash^{\mathsf{c}} M_i : \underline{B} \quad \cdots}{\Gamma \vdash^{\mathsf{c}} \mathtt{pm}\ V\ \mathtt{as}\ \{\ldots, (i, \mathtt{x}).M_i, \ldots\} : \underline{B}} \tag{11.1}$$

But we do not need to regard this as a primitive rule. Instead, we can regard the following provable equation

$$\mathtt{pm}\ V\ \mathtt{as}\ \{\ldots, (i, \mathtt{x}).M_i, \ldots\} = \mathtt{force}\ \mathtt{pm}\ V\ \mathtt{as}\ \{\ldots, (i, \mathtt{x}).\mathtt{thunk}\ M_i, \ldots\} \tag{11.2}$$

as providing a definition for the LHS. For this to be valid, we must be sure that the β- and η-laws for (11.1)

$$\begin{array}{rcl} \mathtt{pm}\ (\hat{\imath}, V)\ \mathtt{as}\ \{\ldots, (i, \mathtt{x}).M_i, \ldots\} & = & M_{\hat{\imath}}[V/\mathtt{x}] \\ M[V/\mathtt{z}] & = & \mathtt{pm}\ V\ \mathtt{as}\ \{\ldots, (i, \mathtt{x}).M[(i, \mathtt{x})/\mathtt{z}], \ldots\} \end{array}$$

are consequences of (11.2)—assuming all the other laws of Fig. 4.1—and this is easily verified.

This desugaring is unsuitable for operational semantics, because it uses complex values. But for our categorical purposes, it is acceptable. It can be seen as decomposing the reversible derivation

$$\frac{\cdots \quad \Gamma, A_i \vdash_c \underline{B} \quad \cdots}{\Gamma, \sum_{i \in I} A_i \vdash_c \underline{B}}$$

into the composite reversible derivation

$$\cfrac{\cfrac{\cdots \quad \Gamma, A_i \vdash_c \underline{B} \quad \cdots}{\cdots \quad \Gamma, A_i \vdash_v U\underline{B} \quad \cdots}}{\cfrac{\Gamma, \sum_{i \in I} A_i \vdash_v U\underline{B}}{\Gamma, \sum_{i \in I} A_i \vdash_c \underline{B}}}$$

11.4 Trivial CBPV Models

We explained in Sect. 4.8 that effect-free CBPV collapses into $\times\sum\prod\to$-calculus, via the *trivialization transform*. Consequently, every countably bicartesian closed category $\mathcal{C}$ gives a model for CBPV. The details are as follows.

- The value category is $\mathcal{C}$.
- A computation type (like a value type) denotes an object of $\mathcal{C}$.
- A computation $\Gamma \vdash_c M : \underline{B}$ denotes a $\mathcal{C}$-morphism from $[\![\Gamma]\!]$ to $[\![\underline{B}]\!]$.

The semantics of types is given by

$$\begin{array}{rcl} [\![U\underline{B}]\!] & = & [\![\underline{B}]\!] \\ [\![FA]\!] & = & [\![A]\!] \\ [\![\prod_{i \in I} \underline{B}_i]\!] & = & \prod_{i \in I} [\![\underline{B}_i]\!] \\ [\![A \to \underline{B}]\!] & = & [\![A]\!] \to [\![\underline{B}]\!] \end{array}$$

Such models of CBPV are called *trivial models*.

Chapter 12

Models In The Style Of Moggi

12.1 Introduction

Moggi [Mog91] explained how a *strong monad* can be used to give a semantics for CBV. In this chapter, we extend his work by showing how a strong monad can be used to construct an *algebra model* for CBPV. The reader may have noticed a striking similarity between the following CBPV models:

- printing
- divergence (the Scott model)
- errors
- printing + divergence

All these are examples of algebra models.

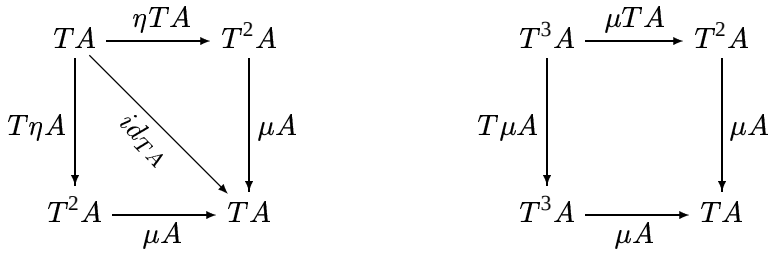
Moggi went further than these examples by suggesting that monads be used to model other effects such as global store. We explain and criticize this view (in the context of CBPV) in Sect. 12.6.

12.2 Strong Monads

Definition 92 Let $\mathcal{B}$ be a category or a locally indexed category. A *monad* on $\mathcal{B}$ consists of

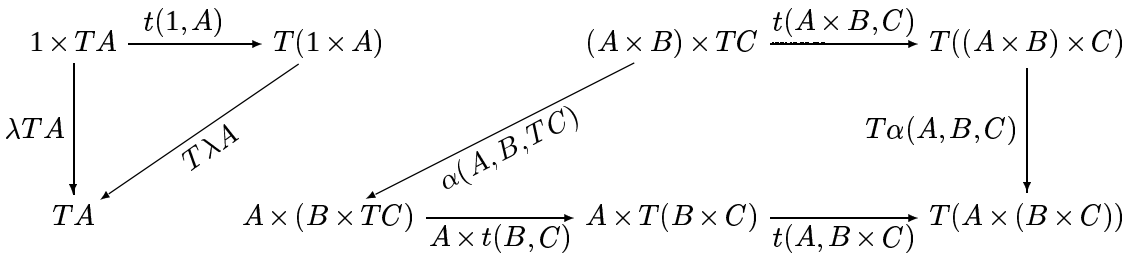
- an endofunctor T on $\mathcal{B}$;
- a natural transformation η from $\mathsf{id}_{\mathcal{B}}$ to T;
- a natural transformation μ from T^2 to T

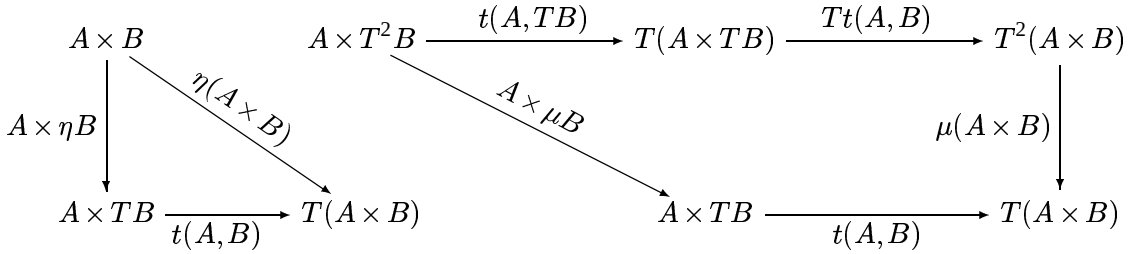
such that the following diagrams commute:



□

Definition 93 Let (T,η,μ) be a monad on a cartesian category $\mathcal{C}$. A *strength* for this monad consists of a natural transformation $t(A,B)$ from $A\times TB$ to $T(A\times B)$ such that the following diagrams commute:





A monad together with a strength is called a *strong monad*. We often refer to a strong monad (T,η,μ,t) just as T. □

Just as, in Prop. 113(1), we replaced "distributive coproduct in $\mathcal{C}$" by "coproduct in $\mathsf{self}\ \mathcal{C}$", so now we can replace "strong monad in $\mathcal{C}$" by "monad in $\mathsf{self}\ \mathcal{C}$". This result was remarked by Plotkin and stated in [Mog91]:

Proposition 118 Let $\mathcal{C}$ be a cartesian category. Then a strong monad on $\mathcal{C}$ is precisely a monad on $\mathsf{self}\ \mathcal{C}$. □

We make no further use of locally indexed categories in this chapter, although we mention them in Def. 95(3).

There are 4 strong monads we will be especially concerned with:

printing the $\mathcal{A}^*\times -$ strong monad on **Set**;

divergence the lifting strong monad on **Cpo**;

errors the $-+E$ strong monad on **Set**;

printing + divergence the $\mu X.(-+\mathcal{A}\times X)_\perp$ strong monad on **Cpo**.

12.3 Monad Models for CBV

It is helpful to look at Moggi's monad models for CBV before we look at monad models for CBPV.

Definition 94 A *CBV monad model* consists of

1. a countably distributive category $\mathcal{C}$;
2. a strong monad (T,η,μ,t) on $\mathcal{C}$;
3. *Kleisli exponents* i.e. for each pair of objects A and B an exponent from A to TB;

4. countable products of Kleisli exponents.

□

We now describe the semantics of CBV in such a model. Notice that the CBV printing semantics and CBV Scott semantics described in Chap. 2 are instances of this.

- A type denotes an object of $\mathcal{C}$.
- In particular $A \to B$ denotes $[\![A]\!] \to [\![TB]\!]$.
- A context $A_0, \ldots, A_{n-1}$ denotes $[\![A_0]\!] \times \cdots \times [\![A_{n-1}]\!]$.
- A value $\Gamma \vdash V : A$ denotes a morphism $[\![\Gamma]\!] \xrightarrow{[\![V]\!]^{\mathsf{val}}} [\![A]\!]$.
- A producer $\Gamma \vdash M : A$ denotes a morphism $[\![\Gamma]\!] \xrightarrow{[\![M]\!]^{\mathsf{prod}}} [\![A]\!]$.

Although the countable products of Kleisli exponents required by Def. 94 are not needed for the small CBV calculus of Chap. 2, and were not required by Moggi, they are needed for the infinitely wide CBV language of Appendix A.

12.4 Algebras

Our monad models for CBPV are based on the principle that *computation types denote algebras*.

Definition 95 [Mac71]

1. Let (T, η, μ) be a monad on a category $\mathcal{C}$. A *T-algebra* consists of a pair (X, θ), where X is an object of $\mathcal{C}$, θ is a morphism from TX to X, and

$$\begin{array}{ccccc} X & \xrightarrow{\eta X} & TX & \xleftarrow{\mu X} & T^2X \\ & \searrow^{\mathrm{id}} & \downarrow{\scriptstyle\theta} & & \downarrow{\scriptstyle T\theta} \\ & & X & \xleftarrow[\theta]{} & TX \end{array} \tag{12.1}$$

   commutes. X is called the *carrier* and θ the *structure map* of the algebra.

2. Let (X, θ) and (Y, ϕ) be T-algebras. A *T-algebra homomorphism* from (X, θ) to (Y, ϕ) is a $\mathcal{C}$-morphism $X \xrightarrow{f} Y$ such that

$$\begin{array}{ccc} TX & \xrightarrow{Tf} & TY \\ \downarrow{\scriptstyle\theta} & & \downarrow{\scriptstyle\phi} \\ X & \xrightarrow[f]{} & Y \end{array} \quad \text{commutes.}$$

   We write $\mathcal{C}^T$ for the *Eilenberg-Moore* category of T-algebras and T-algebra homomorphisms.

3. Given a strength t for the monad (T,η,μ), we can adapt (2) to the locally $\mathcal{C}$-indexed setting, as follows. Let (X,θ) and (Y,ϕ) be T-algebras and Γ a $\mathcal{C}$-object. A *T-algebra homomorphism* from (X,θ) to (Y,ϕ) over Γ is a $\mathcal{C}$-morphism $\Gamma \times X \xrightarrow{f} Y$ such that

$$\begin{array}{ccccc} \Gamma \times TX & \xrightarrow{t(\Gamma,X)} & T(\Gamma \times X) & \xrightarrow{Tf} & TY \\ \downarrow{\scriptstyle \Gamma\times\theta} & & & & \downarrow{\scriptstyle \phi} \\ \Gamma \times X & & \xrightarrow[f]{} & & Y \end{array} \quad \text{commutes.}$$

   We write $\mathcal{C}^T$ for the *Eilenberg-Moore* locally $\mathcal{C}$-indexed category of T-algebras and homomorphisms.

□

In this chapter, we will make no use of Def. 95(3), and the only use we will make of Def. 95 (2) is the notion of algebra isomorphism that it gives us.

We observe that, in the CBPV models listed in Sect. 12.1, computation types indeed denote algebras.

Proposition 119 1. An algebra for the $\mathcal{A}^* \times -$ monad on **Set** is precisely an $\mathcal{A}$-set. A homomorphism from $(X,*)$ to $(Y,*)$ over the set Γ is a function $\Gamma \times X \xrightarrow{f} Y$ such that $f(\rho, c*x) = c*f(\rho,x)$, as in Def. 10.

2. An algebra for the lifting monad on **Cpo** is precisely a cppo. A homomorphism from the cppo $\underline{X}$ to the cppo $\underline{Y}$ over the cpo Γ is a continuous function $\Gamma \times U\underline{X} \xrightarrow{f} U\underline{Y}$ such that $f(\rho,\bot) = \bot$, as in Def. 12.

3. An algebra for the $-+E$ monad on **Set** is precisely an E-set. A homomorphism from (X,error) to (Y,error) over the set Γ is a function $\Gamma \times X \xrightarrow{f} Y$ such that $f(\rho,\mathsf{error}\ e) = \mathsf{error}\ e$.

4. An algebra for the $\mu\mathtt{X}.(-+\mathcal{A}\times\mathtt{X})_\bot$ on **Cpo** is precisely an $\mathcal{A}$-cppo. A homomorphism from $(\underline{X},*)$ to $(\underline{Y},*)$ over the cpo Γ is a continuous function $\Gamma \times U\underline{X} \xrightarrow{f} U\underline{Y}$ such that $f(\rho,\bot) = \bot$ and $f(\rho,c*x) = c*f(\rho,x)$.

□

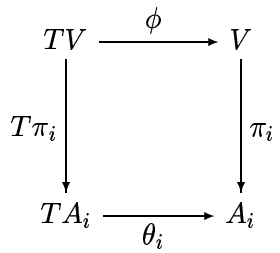
We thus require ways of constructing algebras analogous to the ways of constructing $\mathcal{A}$-sets etc.

Definition 96 Let (T,η,μ,t) be a strong monad on a cartesian category $\mathcal{C}$.

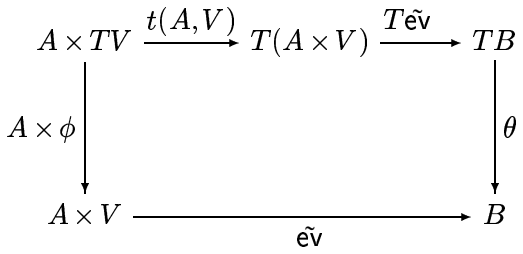
1. Let A be an object of $\mathcal{C}$. The *free T-algebra* on A has carrier TA and structure μA.

2. Let $\{(A_i,\theta_i)\}_{i\in I}$ be a family of T-algebras, and suppose their carriers $\{A_i\}_{i\in I}$ have a product $\mathsf{p} = (V,\{\pi_i\}_{i\in I})$. Then we write $\prod_{i\in I}(A_i,\theta_i)_{@\mathsf{p}}$ for the T-algebra whose carrier is V and whose structure is the unique $\mathcal{C}$-morphism $TV \xrightarrow{\phi} V$ such that for each

$i \in I$ the following commutes:



3. Let A be an object of $\mathcal{C}$, let (B,θ) be a T-algebra, and suppose there is an exponent $\mathsf{e} = (V,\mathsf{ev})$ from A to B. Then we write $A \to (B,\theta)_{@\mathsf{e}}$ for the T-algebra whose carrier is V and whose structure is the unique $\mathcal{C}$-morphism $TV \xrightarrow{\phi} V$ such that the following commutes:



where we write $\tilde{\mathsf{ev}}$ for the composite

$$A \times V \xrightarrow{\cong} V \times A \xrightarrow{\mathsf{ev}} B$$

□

Lemma 120 Def. 96(2)–(3) is given up to T-algebra isomorphism, in the following sense.

(2) Let $\{(A_i,\theta_i)\}_{i\in I}$ be a family of T-algebras, and suppose their carriers $\{A_i\}_{i\in I}$ have a product p in $\mathcal{C}$. Then for any T-algebra (W,ϕ) the following correspond:

- a T-algebra isomorphism $(W,\phi) \cong \prod_{i\in I}(A_i,\theta_i)_{@\mathsf{p}}$
- a product p' for the carriers $\{A_i\}_{i\in I}$ such that

$$(W,\phi) = \prod_{i\in I}(A_i,\theta_i)_{@\mathsf{p}'}$$

(3) Let A be an object of $\mathcal{C}$, let (B,θ) be a T-algebra, and suppose there is an exponent e from A to B. Then for any T-algebra (W,ϕ) the following correspond:

- a T-algebra isomorphism $(W,\phi) \cong A \to (B,\theta)_{@\mathsf{e}}$
- an exponent e' from A to B such that

$$(W,\phi) = A \to (B,\theta)_{@\mathsf{e}'}$$

□

12.5 Algebra Models

12.5.1 Unrestricted Algebra Models

Def. 96 suggests that, provided we have enough products and exponents in $\mathcal{C}$, we can interpret CBPV.

Definition 97 An *unrestricted algebra model* consists of

- a countably distributive category $\mathcal{C}$;
- a strong monad (T, η, μ, t) on $\mathcal{C}$;
- for each countable family $\{(A_i, \theta_i)\}_{i \in I}$ of T-algebras, a product for $\{A_i\}_{i \in I}$;
- for each $\mathcal{C}$-object A and T-algebra (B, θ), an exponent from A to B.

□

From an unrestricted algebra model we obtain a semantics for CBPV as follows:

- a value type (and hence a context) denotes an object of $\mathcal{C}$;
- a computation type denotes a T-algebra;
- a value $\Gamma \vdash^{\mathsf{v}} V : A$ denotes a $\mathcal{C}$-morphism from $[\![\Gamma]\!]$ to $[\![A]\!]$;
- if $\underline{B}$ denotes the algebra (Y, ϕ) then a computation $\Gamma \vdash^{\mathsf{c}} M : \underline{B}$ denotes a $\mathcal{C}$-morphism from $[\![\Gamma]\!]$ to Y.

The most important clauses in the semantics of terms are as follows:

- If $\Gamma \vdash^{\mathsf{v}} V : A$ then $\mathtt{produce}\, V$ denotes the composite
$$[\![\Gamma]\!] \xrightarrow{[\![V]\!]} [\![A]\!] \xrightarrow{\eta[\![A]\!]} T[\![A]\!]$$
- If $\Gamma \vdash^{\mathsf{c}} M : FA$ and $\Gamma, \mathtt{x} : A \vdash^{\mathsf{c}} N : \underline{B}$ and $\underline{B}$ denotes the algebra (Y, ϕ) then $M\ \mathtt{to}\ \mathtt{x}.\ N$ denotes the composite
$$[\![\Gamma]\!] \xrightarrow{(\mathsf{id}, [\![M]\!])} [\![\Gamma]\!] \times T[\![A]\!] \xrightarrow{t([\![\Gamma]\!], [\![A]\!])} T([\![\Gamma]\!] \times [\![A]\!]) \xrightarrow{T[\![N]\!]} TY \xrightarrow{\phi} Y$$
- For the $A \to \underline{B}$ constructs, suppose A denotes the object X and $\underline{B}$ denotes the algebra (Y, ϕ), and the given exponent from X to Y is
$$\mathcal{C}(\Gamma \times X, Y) \cong \mathcal{C}(\Gamma, V) \text{ natural in } \Gamma \tag{12.2}$$
  - If $\Gamma, \mathtt{x} : A \vdash^{\mathsf{c}} M : \underline{B}$ then $\lambda \mathtt{x}.M$ denotes the morphism $[\![\Gamma]\!] \longrightarrow V$ corresponding to $[\![M]\!]$ along (12.2).
  - If $\Gamma \vdash^{\mathsf{v}} W : A$ and $\Gamma \vdash^{\mathsf{c}} M : A \to \underline{B}$, then $W`M$ denotes the composite
$$[\![\Gamma]\!] \xrightarrow{(\mathsf{id}, [\![W]\!])} [\![\Gamma]\!] \times X \xrightarrow{f} Y$$
    where f corresponds to $[\![M]\!]$ along (12.2).
- $\mathtt{thunk}\, M$ denotes $[\![M]\!]$.
- $\mathtt{force}\, V$ denotes $[\![V]\!]$.

Notice that an algebra model does not distinguish between a computation and its thunk.

Proposition 121 An unrestricted algebra model for CBPV satisfies all the equations of the CBPV equational theory. □

This is straightforward to check, using a substitution lemma.

12.5.2 Examples of Unrestricted Algebra Models

Firstly, a trivial CBPV model, in the sense of Sect. 11.4, is obviously an unrestricted algebra model, using the identity strong monad on $\mathcal{C}$.

It is apparent that our running examples of strong monads,

- the $\mathcal{A}^* \times -$ monad on **Set**,
- the lifting monad on **Cpo**,
- the $- + E$ monad on **Set**,
- the $\mu X.(- + \mathcal{A} \times X)_\perp$ monad on **Cpo**

provide unrestricted algebra models, because **Set** and **Cpo** have all products and exponents, and (using Prop. 119) that these algebra models give, respectively, our semantics for

- printing,
- divergence (the Scott model),
- errors,
- printing + divergence.

Since **Set** and **Cpo** have *all* products and exponents, not just those required by Def. 97, these examples do not illustrate Def. 97 very well. Better examples are provided by

- the lifting monad on **SEAM**,
- the $\mu \mathtt{X}.(- + \mathcal{A} \times \mathtt{X})_\perp$ monad on **SEAM**.

The reader will recall from Sect. 5.2 that, unlike **Cpo**, the category **SEAM** does not have all countable products or all exponents. But it does have countable products of, and exponents to, SE domains, and this is precisely what we require for the lifting monad to give an unrestricted algebra model. The $\mu \mathtt{X}.(- + \mathcal{A} \times \mathtt{X})_\perp$ monad on **SEAM** also gives an unrestricted algebra model, because, roughly speaking, it is "bigger than" the lifting monad, and so an algebra for it (a SE $\mathcal{A}$-domain) is also an algebra for the lifting monad (a SE domain). This illustrates the general fact[1] that we can always "grow" the strong monad without having to check again for the required products and exponents in the value category $\mathcal{C}$.

12.5.3 Restricted Algebra Models

Given a strong monad (T, η, μ, t) on a countably distributive category $\mathcal{C}$, we want to construct an algebra model for CBPV. We know that we need certain products and exponents in $\mathcal{C}$, but just how many do we need?

- As in Def. 94, it is *necessary* for $\mathcal{C}$ to have all Kleisli exponents and all countable products of Kleisli exponents. We need this much to interpret CBV, so we certainly need it to interpret CBPV.

[1] The precise argument is this. Suppose we are given any strong monad morphism from a strong monad on (T, η, μ, t) to a strong monad (T', η', μ', t') on the same cartesian category $\mathcal{C}$—i.e. a natural transformation $T \xrightarrow{\alpha} T'$ satisfying the evident diagrams. Then every T'-algebra (X, θ) gives a T-algebra with the same carrier X and structure map $\alpha X; \theta$. Hence, given an unrestricted algebra model based on T we obtain an unrestricted algebra model based on T'. In the case above, there is a strong monad morphism from the lifting monad to the $\mu \mathtt{X}.(- + \mathcal{A} \times \mathtt{X})_\perp$ monad on **SEAM**.

- As in Def. 97, it is *sufficient* for $\mathcal{C}$ to have all countable products of, and all exponents to, carriers of T-algebras.

Sometimes $\mathcal{C}$ has the property that all idempotents split, and then these two conditions are equivalent, because every algebra is a retract of a free algebra. But in general $\mathcal{C}$ does not have this property[2] and we seek an answer that lies between these two extremes.

$$\begin{array}{ccccc} \text{unrestricted} & & & & \text{CBV} \\ \text{algebra} & \subset & ? & \subset & \text{monad} \\ \text{models} & & & & \text{models} \end{array}$$

To see the desired structure, suppose we have a family of algebras $\{j\underline{B}\}_{\underline{B}\in\mathfrak{I}}$ which contains all free algebras and is closed under countable product and $A \to -$ (for each A). Then we can interpret every computation type by an algebra $j\underline{B}$ (more precisely, by the index $\underline{B}$). So we do not need the ability to form products of, or exponents to, algebras outside this family.

We make this precise as follows.

Definition 98 A *restricted algebra model* consists of

- a countably distributive category $\mathcal{C}$;
- a strong monad (T, η, μ, t) on $\mathcal{C}$;
- a family $\{j\underline{B}\}_{\underline{B}\in\mathfrak{I}}$ of T-algebras, indexed by some class $\mathfrak{I}$—we write $U\underline{B}$ for the carrier and $\beta\underline{B}$ for the structure of the algebra $j\underline{B}$;
- for each $\mathcal{C}$-object A, an algebra index $FA \in \mathfrak{I}$ such that

$$\text{the free } T\text{-algebra on } A = jFA \tag{12.3}$$

- for each countable family of $\mathfrak{I}$-objects $\{\underline{B}_i\}_{i\in I}$, a product p for $\{U\underline{B}_i\}_{i\in I}$ and an algebra index $\prod_{i\in I}\underline{B}_i$ such that

$$\textstyle\prod_{i\in I} j\underline{B}_{i\ @\mathsf{p}} = j\prod_{i\in I}\underline{B}_i \tag{12.4}$$

- for each $\mathcal{C}$-object A and $\mathfrak{I}$-object $\underline{B}$, an exponent e from A to $U\underline{B}$ and an algebra index $A \to \underline{B}$ such that

$$A \to j\underline{B}_{@\mathsf{e}} = j(A \to \underline{B}) \tag{12.5}$$

□

Note that, by equating carriers of algebras,

- (12.3) implies $TA = UFA$;
- (12.4) implies the vertex of p is $U\prod_{i\in I}\underline{B}_i$;
- (12.5) implies the vertex of e is $U(A \to \underline{B})$.

The reader may ask: what if (12.3)–(12.5) are algebra isomorphisms rather than equations?

[2] Several people have pointed out that we can use Karoubi completion to fully faithfully embed a CBV monad model into another in which all idempotents do split in the value category.

- If (12.3) is an algebra isomorphism, then we can construct a new strong monad (T', η', μ', t') on $\mathcal{C}$ isomorphic to (T, η, μ, t), such that, w.r.t. this new monad, (12.3) is an equation.
- If (12.4)–(12.5) are algebra isomorphisms, then Lemma 120 tells us that we can choose suitable products and exponents in $\mathcal{C}$ such that, w.r.t. these products and exponents, (12.4)–(12.5) are equations.

In summary, it does not matter if (12.3)–(12.5) are only algebra isomorphisms, because we can turn them into equations.

From a restricted algebra model, we obtain a semantics for CBPV as follows:

- a value type (and hence a context) denotes an object of $\mathcal{C}$;
- a computation type $\underline{B}$ denotes an algebra index $[\![\underline{B}]\!] \in \mathfrak{I}$;
- a value $\Gamma \vdash^{\mathsf{v}} V : A$ denotes a $\mathcal{C}$-morphism from $[\![\Gamma]\!]$ to $[\![A]\!]$;
- if $j[\![\underline{B}]\!]$ is the algebra (Y, ϕ) then a computation $\Gamma \vdash^{\mathsf{c}} M : \underline{B}$ denotes a $\mathcal{C}$-morphism from $[\![\Gamma]\!]$ to Y.

In particular, $[\![U\underline{B}]\!]$ is the carrier of $j[\![\underline{B}]\!]$. We omit the semantics of terms, but note that once again

$$\begin{aligned} [\![\mathtt{thunk}\, M]\!] &= [\![M]\!] \\ [\![\mathtt{force}\, V]\!] &= [\![V]\!] \end{aligned}$$

Proposition 122 A restricted algebra model for CBPV satisfies all the equations of the CBPV equational theory. □

12.6 The Algebra Viewpoint

12.6.1 Explaining the Algebra Viewpoint

Using strong monads, we have seen that several of our CBPV models, viz.

- printing
- divergence (the Scott model)
- errors
- printing + divergence

are instances of a general construction: they are algebra models.

Moggi [Mog91] advocated a much wider use of strong monads than these examples. This stance can be justified by the fact that *every* model of the CBPV equational theory is equivalent to an algebra model. This is depicted in Fig. 11.1 and we prove it in Chap. 15. To grasp the idea, we just look at global store. The global store model of Chap. 6 is equivalent to the following restricted algebra model.

Take the $S \to (S \times -)$ strong monad on **Set**. For each set X let jX be the evident algebra with carrier $S \to X$. Then the family of such algebras contains all free algebras and is closed under countable product and $A \to -$ (for each set A), because we have

$$\begin{aligned} \text{the free algebra on } A &= j(S \times A) && (12.6) \\ \textstyle\prod_{i \in I} jB_i &\cong j \textstyle\prod_{i \in I} B_i && (12.7) \\ A \to jB &\cong j(A \to B) && (12.8) \end{aligned}$$

This gives us a restricted algebra model. As explained in Sect. 12.5.3, the fact that (12.7)–(12.8) are algebra isomorphisms rather than equations does not matter.

It is easy to see that this restricted algebra model and the global store model in Chap. 6 provide exactly the same semantics of types. The semantics of terms is also the same, except that a computation $\Gamma \vdash^{\mathtt{c}} M : \underline{B}$ denotes

- a function from $S \times [\![\Gamma]\!]$ to $[\![\underline{B}]\!]$ in the model of Chap. 6;
- a function from $[\![\Gamma]\!]$ to $S \to [\![\underline{B}]\!]$ in the restricted algebra model.

It is because of this difference that `thunk` and `force` are invisible in the algebra model but not in the model of Chap. 6.

12.6.2 Criticizing the Algebra Viewpoint

While it is possible to view every model of the CBPV equational theory as a restricted algebra model (as we illustrated for the global store model in Sect. 12.6.1), this viewpoint has two consequences:

1. there is no difference between a computation and its thunk;
2. for a computation type $\underline{B}$, its denotation $[\![\underline{B}]\!]$ is just an index, and not in itself important—what matters is the algebra $j[\![\underline{B}]\!]$ that $[\![\underline{B}]\!]$ identifies.

(1) is problematic because, operationally and conceptually, there *is* a difference between a computation and its thunk: computations are what operational semantics is defined on. If we want to describe a semantics (such as the global store semantics) that makes this difference apparent, then "restricted algebra model" is not an appropriate way to organize the description.

(2) is reasonable if we are modelling the pure CBPV equational theory. But in specific models of CBPV with effects, $[\![\underline{B}]\!]$ can be important.

- $[\![\underline{B}]\!]$ is used in soundness/adequacy statements, such as Prop. 36 where both sides of the equation $[\![M]\!]s = [\![T]\!]s'$ are elements of $[\![\underline{B}]\!]$.
- In models of control, $[\![\underline{B}]\!]$ is used in the clause $[\![\mathtt{os}\ \underline{B}]\!] = [\![\underline{B}]\!]$.

For these reasons, we will not take "restricted algebra model" as the last word on categorical semantics for CBPV, but seek more appropriate structures.

Chapter 13

Models In The Style Of Power-Robinson

13.1 Introduction

In Chap. 12 and Chap. 14, we pursue well-known mathematical notions of monad and adjunction to obtain categorical models for CBPV. In this chapter we use a different methodology, based on the work of Power and Robinson [PR97]. We take the term model of CBPV and ask: what sort of categorical structure is it?

Because CBPV is a big language, we proceed incrementally, first looking at the term model of the *value/producer fragment*, shown in Fig. 13.1, where the only computations are producers and the only type constructors are $1, \times$. (In fact, this fragment lies inside the *fine-grain CBV* language described in Sect. A.3.2.) What categories are present in the term model of this fragment?

Firstly, as we have seen, there is the *value category* $\mathcal{C}$, a cartesian category in which

- an object is a value type;
- a morphism from A to B is an equivalence class (modulo provable equality) of values $\mathtt{x} : A \vdash^{\mathtt{v}} V : B$;
- composition is given by substitution.

Secondly, there is the *producer category* $\mathcal{K}$, a category in which

- an object is a value type;
- a morphism from A to B is an equivalence class (modulo provable equality) of producers $\mathtt{x} : A \vdash^{\mathtt{p}} M : B$;
- composition is given by sequencing.

These two categories together form a *value/producer structure*, which we define in Sect. 13.2.2. This is essentially the structure described by Power and Robinson (it is called a "Freyd category" in [PT99]). This structure forms a precise categorical semantics for the value/producer fragment.

We will then move from the value/producer fragment to the whole of CBPV. Because the collection of all computations, unlike the collection of producers, does not form a category, we need to introduce the more general notion of *staggered category*. Using this together with value/producer structures, we will provide a categorical semantics for CBPV which is very close to the syntax.

Types

$$A ::= \quad 1 \mid A \times A$$

Judgements

$$\begin{array}{ll} \Gamma \vdash^{\mathsf{v}} V : A & \\ \Gamma \vdash^{\mathsf{p}} M : A & \text{(think of this as } \Gamma \vdash^{\mathsf{c}} M : F\,A\text{)} \end{array}$$

Terms

$$\frac{}{\Gamma, \mathtt{x} : A, \Gamma' \vdash^{\mathsf{v}} \mathtt{x} : A} \qquad \frac{\Gamma \vdash^{\mathsf{v}} V : A \quad \Gamma, \mathtt{x} : A \vdash^{\mathsf{v}} W : B}{\Gamma \vdash^{\mathsf{v}} \mathtt{let}\ \mathtt{x}\ \mathtt{be}\ V.\ W : B}$$

$$\frac{\Gamma \vdash^{\mathsf{v}} V : A \quad \Gamma, \mathtt{x} : A \vdash^{\mathsf{p}} M : B}{\Gamma \vdash^{\mathsf{p}} \mathtt{let}\ \mathtt{x}\ \mathtt{be}\ V.\ M : B}$$

$$\frac{\Gamma \vdash^{\mathsf{v}} V : A \quad \Gamma \vdash^{\mathsf{v}} V' : A'}{\Gamma \vdash^{\mathsf{v}} (V, V') : A \times A'} \qquad \frac{\Gamma \vdash^{\mathsf{v}} V : A \times A' \quad \Gamma, \mathtt{x} : A, \mathtt{y} : A' \vdash^{\mathsf{v}} W : B}{\Gamma \vdash^{\mathsf{v}} \mathtt{pm}\ V\ \mathtt{as}\ (\mathtt{x}, \mathtt{y}).W : B}$$

$$\frac{\Gamma \vdash^{\mathsf{v}} V : A \times A' \quad \Gamma, \mathtt{x} : A, \mathtt{y} : A' \vdash^{\mathsf{p}} M : B}{\Gamma \vdash^{\mathsf{p}} \mathtt{pm}\ V\ \mathtt{as}\ (\mathtt{x}, \mathtt{y}).M : B}$$

$$\frac{\Gamma \vdash^{\mathsf{v}} V : A}{\Gamma \vdash^{\mathsf{p}} \mathtt{produce}\ V : A} \qquad \frac{\Gamma \vdash^{\mathsf{p}} M : A \quad \Gamma, \mathtt{x} : A \vdash^{\mathsf{p}} N : B}{\Gamma \vdash^{\mathsf{p}} M\ \mathtt{to}\ \mathtt{x}.\ N : B}$$

Equations, using conventions of Sect. 1.4.2

$$\begin{array}{lrcl}
(\beta) & \mathtt{let}\ \mathtt{x}\ \mathtt{be}\ V.\ W & = & W[V/\mathtt{x}] \\
(\beta) & \mathtt{let}\ \mathtt{x}\ \mathtt{be}\ V.\ M & = & M[V/\mathtt{x}] \\
(\beta) & \mathtt{pm}\ (V, V')\ \mathtt{as}\ (\mathtt{x}, \mathtt{y}).W & = & W[V/\mathtt{x}, V'/\mathtt{y}] \\
(\beta) & \mathtt{pm}\ (V, V')\ \mathtt{as}\ (\mathtt{x}, \mathtt{y}).M & = & M[V/\mathtt{x}, V'/\mathtt{y}] \\
(\beta) & \mathtt{produce}\ V\ \mathtt{to}\ \mathtt{x}.\ M & = & M[V/\mathtt{x}] \\
(\eta) & W[V/\mathtt{z}] & = & \mathtt{pm}\ V\ \mathtt{as}\ (\mathtt{x}, \mathtt{y}).W[(\mathtt{x}, \mathtt{y})/\mathtt{z}] \\
(\eta) & M[V/\mathtt{z}] & = & \mathtt{pm}\ V\ \mathtt{as}\ (\mathtt{x}, \mathtt{y}).M[(\mathtt{x}, \mathtt{y})/\mathtt{z}] \\
(\eta) & M & = & M\ \mathtt{to}\ \mathtt{x}.\ \mathtt{produce}\ \mathtt{x} \\
 & (M\ \mathtt{to}\ \mathtt{x}.\ N)\ \mathtt{to}\ \mathtt{y}.\ P & = & M\ \mathtt{to}\ \mathtt{x}.\ (N\ \mathtt{to}\ \mathtt{y}.\ P)
\end{array}$$

Figure 13.1: The Value/Producer Fragment Of CBPV

13.2 Value/Producer Structures

13.2.1 Preliminaries

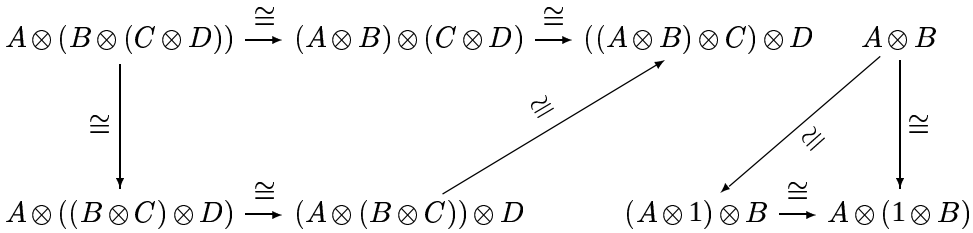
In this section, we review some well-known material.

Definition 99 1. A *monoidal category* consists of

- a category $\mathcal{C}$;
- an object 1;
- a functor $\otimes : \mathcal{C} \times \mathcal{C} \longrightarrow \mathcal{C}$;
- natural isomorphisms

$$\begin{aligned} A \otimes (B \otimes C) &\cong (A \otimes B) \otimes C \\ A &\cong 1 \otimes A \end{aligned}$$

such that the two structural isomorphisms from $1 \otimes 1$ to 1 are equal and the diagrams

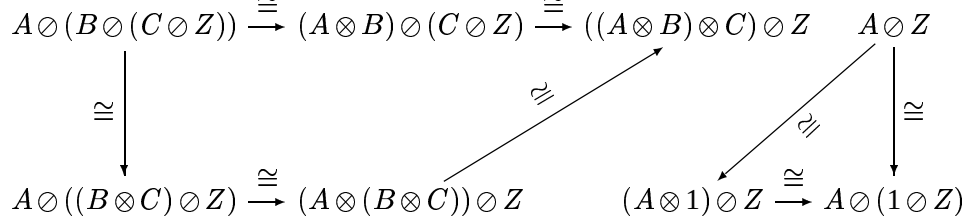


commute.

2. Let $(\mathcal{C}, 1, \otimes)$ be a monoidal category and $\mathcal{D}$ a category. A *left $\mathcal{C}$-action* on $\mathcal{D}$ consists of a functor $\oslash$ from $\mathcal{C} \times \mathcal{D}$ to $\mathcal{D}$, together with natural isomorphisms

$$\begin{aligned} A \oslash (B \oslash Z) &\cong (A \otimes B) \oslash Z \\ Z &\cong 1 \oslash Z \end{aligned}$$

such that the diagrams



commute. A *right $\mathcal{C}$-action* on $\mathcal{D}$ is defined similarly.

□

It is clear that

- every cartesian category is monoidal;
- every monoidal category has a canonical left action and right action on itself.

Proposition 123 (coherence) 1. Given a monoidal category $\mathcal{C}$, every diagram built from structural isomorphisms commutes.

2. Given a monoidal category $\mathcal{C}$ and a left $\mathcal{C}$-action on a category $\mathcal{D}$, every diagram built from structural isomorphisms commutes.

□

A precise statement and a proof of Prop. 123(1) can be found in [Mac71]. Prop. 123(2) is stated and proved similarly.

13.2.2 Value/Producer Structures

The structure relating the value category $\mathcal{C}$ and the producer category $\mathcal{K}$ was described by Power and Robinson [PR97]. For convenience, we reformulate it as follows.

Definition 100 Let $\mathcal{C}$ be a cartesian category and $\mathcal{K}$ a category such that $\mathsf{ob}\ \mathcal{K} = \mathsf{ob}\ \mathcal{C}$—we write a morphism in $\mathcal{K}$ as $A \xrightarrow{\quad f \quad} B$. A *value/producer structure*[1] from $\mathcal{C}$ to $\mathcal{K}$ consists of

- an identity-on-objects functor ι from $\mathcal{C}$ to $\mathcal{K}$;
- a left $\mathcal{C}$-action on $\mathcal{K}$, extending (along ι) the canonical left $\mathcal{C}$-action on $\mathcal{C}$.

We call the left action $\times$, because it is given on objects by $\times$. □

Suppose we are given a value/producer structure from $\mathcal{C}$ to $\mathcal{K}$. We can obtain a *right* $\mathcal{C}$-action on $\mathcal{K}$, extending (along ι) the canonical right $\mathcal{C}$-action on $\mathcal{C}$. This action too we call $\times$, because it is given on objects by $\times$. In summary, we write $f \times g$ in 3 situations:

- if f and g are both $\mathcal{C}$-morphisms, then $f \times g$ is a $\mathcal{C}$-morphism;
- if f is a $\mathcal{C}$-morphism and g is a $\mathcal{K}$-morphism, then $f \times g$ is a $\mathcal{K}$-morphism;
- if f is a $\mathcal{K}$-morphism and g is a $\mathcal{C}$-morphism, then $f \times g$ is a $\mathcal{K}$-morphism.

By contrast, if f and g are both $\mathcal{K}$-morphisms, then $f \times g$ is not defined. So it is not the case (in general) that $\mathcal{K}$ forms a monoidal category under $\times$. In the sense of Power and Robinson [PR97], $\mathcal{K}$ is a *symmetric premonoidal category* and ι is a *strict symmetric premonoidal functor*.

We interpret the value/producer fragment in a value/producer structure $(\mathcal{C}, \mathcal{K}, \iota, \ldots)$ as follows:

- a type (and hence a context) denotes an object;
- a value $\Gamma \vdash^{\mathsf{v}} V : A$ denotes a $\mathcal{C}$-morphism from $[\![\Gamma]\!]$ to $[\![A]\!]$;
- a producer $\Gamma \vdash^{\mathsf{p}} V : A$ denotes a $\mathcal{K}$-morphism from $[\![\Gamma]\!]$ to $[\![A]\!]$.

Some term constructors:

- If $\Gamma \vdash^{\mathsf{v}} V : A$ then $\mathtt{produce}\ V$ denotes $[\![\Gamma]\!] \xrightarrow{\quad \iota[\![V]\!] \quad} [\![A]\!]$.
- If $\Gamma \vdash^{\mathsf{p}} M : A$ and $\Gamma, \mathtt{x} : A \vdash^{\mathsf{p}} N : B$ then $M\ \mathtt{to}\ \mathtt{x}.\ N$ denotes

$$[\![\Gamma]\!] \xrightarrow{\quad \iota(\mathsf{id},\mathsf{id}) \quad} [\![\Gamma]\!] \times [\![\Gamma]\!] \xrightarrow{\quad [\![\Gamma]\!] \times [\![M]\!] \quad} [\![\Gamma]\!] \times [\![A]\!] \xrightarrow{\quad [\![N]\!] \quad} [\![B]\!]$$

- If $\Gamma \vdash^{\mathsf{v}} V : A$ and $\Gamma, \mathtt{x} : A \vdash^{\mathsf{p}} M : B$ then $\mathtt{let}\ \mathtt{x}\ \mathtt{be}\ V.\ M$ denotes

$$[\![\Gamma]\!] \xrightarrow{\quad \iota(\mathsf{id},[\![V]\!]) \quad} [\![\Gamma \times A]\!] \xrightarrow{\quad [\![M]\!] \quad} [\![B]\!]$$

We mention that the motivation for Def. 100 is the following:

Proposition 124 Models of the value/producer fragment of CBPV and value/producer structures are equivalent. □

[1] This is called a "Freyd category" in [PT99].

We can make this precise and prove it in the same way that we made Prop. 97 precise and proved it (in outline).

Def. 100 contains some redundancies, for the sake of elegance. We now give a bare-essentials characterization, which reduces the workload when constructing a value/producer structure.

Proposition 125 Let $\mathcal{C}$ be a cartesian category and $\mathcal{K}$ a category such that $\mathsf{ob}\ \mathcal{K} = \mathsf{ob}\ \mathcal{C}$. A value/producer structure from $\mathcal{C}$ to $\mathcal{K}$ is given by the following data:

- an identity-on-objects functor ι from $\mathcal{C}$ to $\mathcal{K}$
- for each $X \in \mathsf{ob}\ \mathcal{C}$ and $\mathcal{K}$-morphism $A \xrightarrow{\quad g \quad} B$, a $\mathcal{K}$-morphism $X \times A \xrightarrow{X \times g} X \times B$

such that

- the equations

$$\begin{aligned} X \times \mathsf{id} &= \mathsf{id} \\ X \times (f;g) &= (X \times f);(X \times g) \\ X \times (\iota f) &= \iota(X \times f) \end{aligned}$$

  are satisfied;

- for every $\mathcal{C}$-morphism $X \xrightarrow{\quad f \quad} Y$ and $\mathcal{K}$-morphism $A \xrightarrow{\quad g \quad} B$, the diagram

$$\begin{array}{ccc} X \times A & \xrightarrow{X \times g} & X \times B \\ {\scriptstyle \iota(f \times A)}\downarrow & & \downarrow{\scriptstyle \iota(f \times B)} \\ Y \times A & \xrightarrow[Y \times g]{} & Y \times B \end{array}$$

  commutes;

- the isomorphisms

$$\begin{aligned} A \times (B \times Z) &\cong (A \times B) \times Z \\ Z &\cong 1 \times Z \end{aligned}$$

  in $\mathcal{K}$, obtained by applying ι to the structural isomorphisms in $\mathcal{C}$, are natural in $Z \in \mathcal{K}$.

□

Note that it is not necessary to verify the coherence diagrams in $\mathcal{K}$, as these are obtained by applying ι to the coherence diagrams in $\mathcal{C}$.

13.3 Staggered Categories

We now wish to give a categorical semantics for the whole of CBPV in the spirit of value/producer structures. But the problem is that we cannot form a category of computations as we did for producers. Writing $\mathcal{E}(A, \underline{B})$ for the set of equivalence classes of computations $A \vdash^{\mathsf{c}} M : \underline{B}$, it is clear that $\mathcal{E}$ is not a category. Rather, it is a *staggered category*:

Definition 101 A *staggered category* $\mathcal{E}$ consists of

- a class of *source objects* $\mathsf{sourceob}\ \mathcal{E}$;

- a class of *target objects* $\mathsf{targetob}\ \mathcal{E}$, whose elements we underline;
- a *source-to-target function* $F : \mathsf{sourceob}\ \mathcal{E} \to \mathsf{targetob}\ \mathcal{E}$;
- for each $A \in \mathsf{sourceob}\ \mathcal{E}$ and $\underline{B} \in \mathsf{targetob}\ \mathcal{E}$ a set $\mathcal{E}(A, \underline{B})$ of *morphisms from A to $\underline{B}$*;
- for each $A \in \mathsf{sourceob}\ \mathcal{E}$, an identity morphism $A \xrightarrow{\mathsf{id}_A} FA$;
- for each $A \xrightarrow{f} FB$ and $B \xrightarrow{g} \underline{C}$, a composite morphism $A \xrightarrow{f;g} \underline{C}$, which we sometimes write as

$$A \xrightarrow{f} B \xrightarrow{g} \underline{C}$$

satisfying identity and associativity laws

$$\begin{array}{rcl} \mathsf{id}; f & = & f \\ f; \mathsf{id} & = & f \\ (f; g); h & = & f; (g; h) \end{array}$$

□

Definition 102 Let $\mathcal{E}$ be a staggered category. We write $\mathcal{E}_F$ for the ordinary category given by

$$\begin{array}{rcl} \mathsf{ob}\ \mathcal{E}_F & = & \mathsf{sourceob}\ \mathcal{E} \\ \mathcal{E}_F(A, B) & = & \mathcal{E}(A, FB) \end{array}$$

with the evident composition. □

With this construction $\mathcal{E}_F$ in mind, we often write $A \xrightarrow{f} B$ to say that f is a $\mathcal{E}$-morphism from A to FB. This agrees with the notation for composition in Def. 101.

If $\mathcal{E}$ is the staggered category of computations obtained from the term model as described above, then $\mathcal{E}_F$ is the producer category $\mathcal{K}$ described in Sect. 13.1.

Given a $\mathcal{E}$-morphism $A \xrightarrow{f} FA'$, we write $\mathcal{E}(f, \underline{B})$ for the morphism from $\mathcal{E}(A', \underline{B})$ to $\mathcal{E}(A, \underline{B})$ that takes g to $f; g$. Thus every target object $\underline{B}$ in a staggered category $\mathcal{E}$ induces a functor $\lambda X.\mathcal{E}(X, \underline{B})$ from $\mathcal{E}_F^{\mathsf{op}}$ to **Set**. This enables us to adapt the notion of "representable functor" to staggered categories, as follows.

Definition 103 (cf. Def. 61(contravariant, isomorphism style)) Let $\mathcal{E}$ be a staggered category, and let $\mathcal{F}$ be a functor from $\mathcal{E}_F^{\mathsf{op}}$ to **Set**. A *representation* for $\mathcal{F}$ in $\mathcal{E}$ consists of a target object $\underline{V}$ (the *vertex*) together with an isomorphism

$$\mathcal{F}X \cong \mathcal{E}(X, \underline{V}) \qquad \text{natural in } X \in \mathcal{E}_F^{\mathsf{op}}$$

□

Unlike for ordinary categories, there is no corresponding element-style definition.

13.4 Models For The Whole Of CBPV

Definition 104 A *CBPV value/producer model* consists of

- a countably distributive category $\mathcal{C}$;
- a staggered category $\mathcal{E}$ such that $\mathsf{sourceob}\ \mathcal{E} = \mathsf{ob}\ \mathcal{C}$;
- a value/producer structure $(\iota, \times)$ from $\mathcal{C}$ to $\mathcal{E}_F$.
- for each computation object $\underline{B}$, a representation of the functor $\lambda\Gamma.\mathcal{E}(\iota\Gamma, \underline{B})$, whose vertex we call $U\underline{B}$—explicitly, this is an isomorphism

$$\mathcal{E}(\iota\Gamma, \underline{B}) \cong \mathcal{C}(\Gamma, U\underline{B}) \text{ natural in } \Gamma \in \mathcal{C}^{\mathsf{op}} \tag{13.1}$$

- for each countable family of computation objects $\{\underline{B}_i\}_{i\in I}$, a representation in $\mathcal{E}$ of the functor $\lambda\Gamma.\prod_{i\in I}\mathcal{E}(\Gamma, \underline{B}_i)$, whose vertex we call $\prod_{i\in I}\underline{B}_i$—explicitly, this is an isomorphism

$$\prod_{i\in I}\mathcal{E}(\Gamma, \underline{B}_i) \cong \mathcal{E}(\Gamma, \prod_{i\in I}\underline{B}_i) \text{ natural in } \Gamma \in \mathcal{E}_F^{\mathsf{op}} \tag{13.2}$$

- for each value object A and computation object $\underline{B}$, a representation in $\mathcal{E}$ of the functor $\lambda\Gamma.\mathcal{E}(\Gamma \times A, \underline{B})$, whose vertex we call $A \to \underline{B}$—explicitly, this is an isomorphism

$$\mathcal{E}(\Gamma \times A, \underline{B}) \cong \mathcal{E}(\Gamma, A \to \underline{B}) \text{ natural in } \Gamma \in \mathcal{E}_F^{\mathsf{op}} \tag{13.3}$$

□

Given a CBPV value/producer model $(\mathcal{C}, \mathcal{E}, \iota, \dots)$, we can interpret CBPV. In particular, a computation $\Gamma \vdash^{\mathsf{c}} M : \underline{B}$ denotes a morphism from $[\![\Gamma]\!]$ to $[\![\underline{B}]\!]$ in $\mathcal{E}$. The basic constructs (those with analogues in the value/producer fragment) are interpreted as follows.

- Given $\Gamma \vdash^{\mathsf{v}} V : A$, the computation $\mathtt{produce}\ V$ denotes $[\![\Gamma]\!] \xrightarrow{\iota[\![V]\!]} F[\![A]\!]$.
- Given $\Gamma \vdash M : FA$ and $\Gamma, \mathtt{x} : A \vdash N : \underline{B}$, the computation $M\ \mathtt{to}\ \mathtt{x}.\ N$ denotes the composite

$$[\![\Gamma]\!] \xrightarrow{\iota(\mathsf{id},\mathsf{id})} [\![\Gamma]\!] \times [\![\Gamma]\!] \xrightarrow{[\![\Gamma]\!] \times [\![M]\!]} [\![\Gamma]\!] \times [\![A]\!] \xrightarrow{[\![N]\!]} [\![\underline{B}]\!]$$

  using the fact that $[\![N]\!]$ is a $\mathcal{E}_F$-morphism from $[\![\Gamma]\!]$ to $[\![A]\!]$.
- Given $\Gamma \vdash V : A$ and $\Gamma, \mathtt{x} : A \vdash^{\mathsf{c}} M : \underline{B}$, the computation $\mathtt{let\ x\ be}\ V.\ M$ denotes the composite

$$[\![\Gamma]\!] \xrightarrow{\iota(\mathsf{id}_\Gamma, [\![V]\!])} [\![\Gamma]\!] \times [\![A]\!] \xrightarrow{[\![M]\!]} [\![\underline{B}]\!]$$

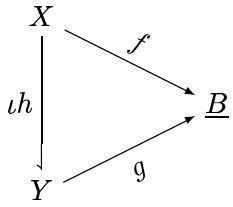
To interpret the constructs for U, we recall from Def. 104 that $U\underline{B}$ is the vertex of a representation for $\lambda\Gamma.\mathcal{E}(\iota\Gamma, \underline{B})$. As we saw in Sect. 10.3, this representation can be expressed in two ways:

isomorphism style as an isomorphism (13.1), which we write as

$$\mathcal{E}(\iota\Gamma, \underline{B}) \xrightarrow[\cong]{\mathsf{thunk}_{\Gamma,\underline{B}}} \mathcal{C}(\Gamma, U\underline{B}) \text{ natural in } \Gamma \quad \text{natural in } \Gamma$$

element style as a terminal object $U\underline{B} \xrightarrow{\mathsf{force}_{\underline{B}}} \underline{B}$ in the category where

- an object is a pair (X, f) where $X \xrightarrow{f} \underline{B}$ in $\mathcal{E}$;
- a morphism from (X, f) to (Y, g) is a $\mathcal{C}$-morphism $X \xrightarrow{h} Y$ such that



We see from Prop. 93(contravariant) that the two forms of the representation are related as follows:

- $\mathsf{force}_{\underline{B}}$ is obtained by applying $\mathsf{thunk}^{-1}_{U\underline{B},\underline{B}}$ to the identity on $U\underline{B}$ in $\mathcal{C}$.
- $\mathsf{thunk}^{-1}_{\Gamma\underline{B}}$ takes $\Gamma \xrightarrow{f} U\underline{B}$ to $(\iota f);\mathsf{force}_{\underline{B}}$.

We interpret the constructs for $U\underline{B}$ as follows:

- If $\Gamma \vdash^{\mathsf{c}} M : \underline{B}$ then $\mathtt{thunk}\, M$ denotes $[\![\Gamma]\!] \xrightarrow{\mathsf{thunk}\, M} U[\![\underline{B}]\!]$.
- If $\Gamma \vdash^{\mathsf{v}} V : U\underline{B}$ then $\mathtt{force}\, V$ denotes the composite

$$[\![\Gamma]\!] \xrightarrow{\iota[\![V]\!]} [\![U\underline{B}]\!] \xrightarrow{\mathsf{force}_{[\![\underline{B}]\!]}} [\![\underline{B}]\!]$$

It may seem more straightforward to ignore the element-style description of the representation and simply say that $\mathtt{force}\, V$ denotes $\mathsf{thunk}^{-1}[\![V]\!]$. This is perfectly valid, but the advantages of using $\mathsf{force}_{\underline{B}}$ become evident in Sect. 15.4.2.

To interpret the constructs for $A \to \underline{B}$ we use the isomorphism (13.3).

- If $\Gamma, \mathtt{x} : A \vdash^{\mathsf{c}} M : \underline{B}$ then $\lambda \mathtt{x}.M$ denotes the morphism $[\![\Gamma]\!] \longrightarrow [\![A]\!] \to [\![\underline{B}]\!]$ corresponding to $[\![M]\!]$ along (13.3).
- If $\Gamma \vdash^{\mathsf{v}} V : A$ and $\Gamma \vdash^{\mathsf{c}} M : A \to \underline{B}$ then $V \text{`} M$ denotes the composite

$$[\![\Gamma]\!] \xrightarrow{\iota(\mathsf{id}, [\![V]\!])} [\![\Gamma]\!] \times [\![A]\!] \xrightarrow{f} [\![\underline{B}]\!]$$

where f corresponds to $[\![M]\!]$ along (13.3).

The isomorphisms in Def. 104 for $U, \to, \prod$ correspond to the reversible derivations for these type constructors. The naturality of (13.1) in Γ corresponds to the fact that the reversible derivation for U preserves substitution in Γ. The naturality of (13.2) and (13.3) in $\Gamma \in \mathcal{E}_F^{\mathsf{op}}$ corresponds to the fact that the reversible derivations for $\prod$ and $\to$ preserve with sequencing in Γ as well as substitution .

Proposition 126 All the CBPV equations are validated in a value/producer model. □

The proof is straightforward, using a substitution lemma.

13.5 Examples of Value/Producer Models

It is easy to see that each of the models summarized in Fig. 6.3 are value/producer models.

In the set models, a source object of $\mathcal{E}$ is a set. The rest of $\mathcal{E}$ is given as follows:

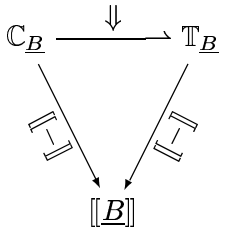
effect	a target object is	a morphism from A to $\underline{B}$ is	FA
printing	an $\mathcal{A}$-set	a function from A to $\underline{B}$	free $\mathcal{A}$-set on A
global store	a set	a function from $S \times A$ to $\underline{B}$	$S \times A$
global store + printing	an $\mathcal{A}$-set	a function from $S \times A$ to $\underline{B}$	$F(S \times A)$
control	a set	a function from $A \times \underline{B}$ to Ans	$A \to \mathsf{Ans}$
control + printing	a set	a function from $A \times \underline{B}$ to $\underline{\mathsf{Ans}}$	$U(A \to \underline{\mathsf{Ans}})$
erratic choice	a set	a relation from A to $\underline{B}$	A
errors	an E-set	a function from A to $\underline{B}$	free E-set on A

In the cpo models, a source object of $\mathcal{E}$ is a cpo. The rest of $\mathcal{E}$ is given as follows:

effect	targ. obj.	a morphism from A to $\underline{B}$ is	FA
divergence	a cppo	a continuous function from A to $\underline{B}$	lift of A
divergence + printing	an $\mathcal{A}$-cpo	a continuous function from $[\![\Gamma]\!]$ to $[\![\underline{B}]\!]$	free $\mathcal{A}$-set on A

13.6 Soundness w.r.t. Big-Step Semantics

It is a curious fact that each of our theorems stating the soundness and adequacy of denotational semantics w.r.t. big-step semantics is an instance of the following assertion: the diagram in $\mathcal{E}$



commutes for every computation type $\underline{B}$.

This may seem surreal, but if we look at examples, it becomes clear. (Note that we cannot use control as an example, because there is no big-step semantics for it.)

For a first example, consider divergence.

- $\Downarrow$ is a function from $\mathbb{C}_{\underline{B}}$ to the lift of $\mathbb{T}_{\underline{B}}$, so it is an $\mathcal{E}$-morphism from $\mathbb{C}_{\underline{B}}$ to $F\mathbb{T}_{\underline{B}}$.
- $[\![-]\!]$ is a function, and hence an $\mathcal{E}$-morphism, from $\mathcal{E}$-morphism from $\mathbb{C}_{\underline{B}}$ to $[\![\underline{B}]\!]$. Similarly from $\mathbb{T}_{\underline{B}}$.
- The diagram says that for any $M \in \mathbb{C}_{\underline{B}}$,
  - if M diverges, then $[\![M]\!] = \bot$;
  - if $M \Downarrow T$, then $[\![M]\!] = [\![T]\!]$.

  This is precisely Prop. 29.

For another example, consider global store.

- $\Downarrow$ is a function from $S \times \mathbb{C}_{\underline{B}}$ to $S \times \mathbb{T}_{\underline{B}}$, so it is an $\mathcal{E}$-morphism from $\mathbb{C}_{\underline{B}}$ to $F\mathbb{T}_{\underline{B}}$.
- $[\![-]\!]$ is a function from $S \times \mathbb{C}_{\underline{B}}$ to $[\![\underline{B}]\!]$, so it is an $\mathcal{E}$-morphism from $\mathbb{C}_{\underline{B}}$ to $[\![\underline{B}]\!]$. Similarly from $\mathbb{T}_{\underline{B}}$.
- The diagram says that for any $(s, M) \in S \times \mathbb{C}_{\underline{B}}$, if $s, M \Downarrow s', T$, then $[\![M]\!]s = [\![T]\!]s'$. This is precisely Prop. 36.

13.7 Technical Material

The technical results in this section will be needed for Chap. 15. Suppose we have a value/producer model $(\mathcal{C}, \mathcal{E}, \ldots)$.

Lemma 127 1. Let $\underline{B}$ and $\underline{C}$ be computation objects. Suppose we have a morphism

$$\mathcal{E}(\Gamma, \underline{B}) \xrightarrow{\alpha_\Gamma} \mathcal{E}(\Gamma, \underline{C}) \text{ natural in } \Gamma \in \mathcal{E}_F^{\mathsf{op}}$$

Then the diagram

$$\begin{array}{ccc} UFU\underline{B} & \xrightarrow{\mathsf{force}_{FU\underline{B}}} & U\underline{B} \\ {\scriptstyle \iota\mathsf{thunk}\,(\mathsf{force}_{FU\underline{B}};\mathsf{force}_{\underline{B}})}\downarrow & & \downarrow{\scriptstyle \alpha_{U\underline{B}}\mathsf{force}_{\underline{B}}} \\ U\underline{B} & \xrightarrow[\alpha_{U\underline{B}}\mathsf{force}_{\underline{B}}]{} & \underline{C} \end{array} \quad \text{commutes.}$$

2. Let $\underline{B}$ and $\underline{C}$ be computation objects, and let A be a value object. Suppose we have a morphism

$$\mathcal{E}(\Gamma, \underline{B}) \xrightarrow{\alpha_\Gamma} \mathcal{E}(\Gamma \times A, \underline{C}) \text{ natural in } \Gamma \in \mathcal{E}_F^{\mathsf{op}}$$

Then the diagram

$$\begin{array}{ccc} UFU\underline{B} \times A & \xrightarrow{\mathsf{force}_{FU\underline{B}} \times A} & U\underline{B} \times A \\ {\scriptstyle \iota\mathsf{thunk}\,(\mathsf{force}_{FU\underline{B}};\mathsf{force}_{\underline{B}}) \times A}\downarrow & & \downarrow{\scriptstyle \alpha_{U\underline{B}}\mathsf{force}_{\underline{B}}} \\ U\underline{B} \times A & \xrightarrow[\alpha_{U\underline{B}}\mathsf{force}_{\underline{B}}]{} & \underline{C} \end{array} \quad \text{commutes.}$$

□

Proof To prove (2), we show that both composites are equal to $\alpha_{UFU\underline{B}}(\mathsf{force}_{FU\underline{B}};\mathsf{force}_{\underline{B}})$. For the top-then-right composite, we apply the commutative diagram

$$\begin{array}{ccc} \mathcal{E}(U\underline{B}, \underline{B}) & \xrightarrow{\alpha_{U\underline{B}}} & \mathcal{E}(U\underline{B} \times A, \underline{C}) \\ {\scriptstyle \mathcal{E}(\mathsf{force}_{FU\underline{B}}, \underline{B})}\downarrow & & \downarrow{\scriptstyle \mathcal{E}(\mathsf{force}_{FU\underline{B}} \times A, \underline{C})} \\ \mathcal{E}(UFU\underline{B}, \underline{B}) & \xrightarrow{\alpha_{UFU\underline{B}}} & \mathcal{E}(UFU\underline{B} \times A, \underline{C}) \end{array}$$

to $\mathsf{force}_{\underline{B}}$ in the top left corner. For the left-then-bottom composite, we apply the commutative diagram

$$\begin{array}{ccc} \mathcal{E}(U\underline{B}, \underline{B}) & \xrightarrow{\alpha_{U\underline{B}}} & \mathcal{E}(U\underline{B} \times A, \underline{C}) \\ {\scriptstyle \mathcal{E}(\iota\mathsf{thunk}\,(\mathsf{force}_{FU\underline{B}};\mathsf{force}_{\underline{B}}), \underline{B})}\downarrow & & \downarrow{\scriptstyle \mathcal{E}(\iota\mathsf{thunk}\,(\mathsf{force}_{FU\underline{B}};\mathsf{force}_{\underline{B}}) \times A, \underline{C})} \\ \mathcal{E}(UFU\underline{B}, \underline{B}) & \xrightarrow{\alpha_{UFU\underline{B}}} & \mathcal{E}(UFU\underline{B} \times A, \underline{C}) \end{array}$$

to $\mathsf{force}_{\underline{B}}$ in the top left corner, and use the fact that $(\iota\mathsf{thunk}\ f);\mathsf{force}_{\underline{Y}} = f$ for any $X \xrightarrow{f} \underline{Y}$.

The proof of (1) is similar. □

Chapter 14

Adjunction Models For CBPV

14.1 Introduction

We have seen that Moggi's type constructor T decomposes into UF in CBPV. This prompts the question: surely it is the decomposition of a monad into an adjunction? In this chapter we answer this by describing a categorical structure called *CBPV adjunction model*. It is the most elegant, and arguably the most important, of the various categorical structures in Fig. 11.1, and every concrete CBPV model we have studied is an instance of it.

We first discuss informally the key idea of *oblique morphism* in an adjunction. This discussion foreshadows many of the formal definitions later in the chapter. We then define *strong adjunction* from a cartesian category $\mathcal{C}$ to a locally $\mathcal{C}$-indexed category $\mathcal{D}$, and use this to define CBPV adjunction model. We see how to interpret CBPV in such a model and how all the models we have looked at are instances of this notion.

In Sect. 14.6, we survey the various definitions of adjunction, with the aim of proving, in Sect. 14.6.4, that strong adjunctions from $\mathcal{C}$ to $\mathcal{D}$ are equivalent to adjunctions from $\mathsf{self}\ \mathcal{C}$ to $\mathcal{D}$.

We look at CBV and CBN in Sect. 14.7, and conclude with a treatment of *staggered adjunction models*—a related but much less elegant structure—and some technical material.

14.2 Oblique Morphisms

The key idea on which adjunction models are based is *oblique morphism* in an adjunction. In this section, we will explain oblique morphisms in an informal way only. The formal account will be given in Sect. 14.3.1 and Sect. 14.6.

14.2.1 Ordinary Categories

Before considering locally indexed categories, we will start by explaining oblique morphisms in the setting of ordinary categories.

Suppose we have an adjunction between categories $\mathcal{B}$ and $\mathcal{D}$ (we underline the objects of $\mathcal{D}$). Adjunction can be defined in many ways—we will give a list of equivalent definitions in Sect. 14.6.2—but for the sake of familiarity we will say that we have functors $U : \mathcal{D} \longrightarrow \mathcal{B}$ and $F : \mathcal{B} \longrightarrow \mathcal{D}$ and an isomorphism

$$\mathcal{B}(X, U\underline{Y}) \cong \mathcal{D}(FX, \underline{Y}) \quad \text{natural in } X \text{ and } \underline{Y}.$$

For $X \in \mathsf{ob}\ \mathcal{B}$ and $\underline{Y} \in \mathsf{ob}\ \mathcal{D}$, we say that an *oblique* morphism from X to $\underline{Y}$ is a $\mathcal{B}$-morphism from X to $U\underline{Y}$, or, equivalently, a $\mathcal{D}$-morphism from FX to $\underline{Y}$. Although an oblique morphism can be thought of as a $\mathcal{B}$-morphism or as a $\mathcal{D}$-morphism, we tend to think of it as neither of

these, but rather as a morphism that goes across from an object in $\mathcal{B}$ to an object in $\mathcal{D}$. For if g is an oblique morphism from X to $\underline{Y}$, then we can pre-compose g with any $\mathcal{B}$-morphism $X' \xrightarrow{f} X$ and we can post-compose g with any $\mathcal{D}$-morphism $\underline{Y} \xrightarrow{h} \underline{Y'}$. Consequently, if we write $O(X,\underline{Y})$ for the set of oblique morphisms from X to $\underline{Y}$—we call this set an "oblique homset"—then O is a functor from $\mathcal{B}^{\mathsf{op}} \times \mathcal{D}$ to **Set**.

This structure, very roughly speaking (the precise account is given in Sect. 14.3.2), gives us a model for CBPV:

- a value $\Gamma \vdash^{\mathsf{v}} V : A$ denotes a $\mathcal{B}$-morphism from $[\![\Gamma]\!]$ to $[\![A]\!]$
- a computation $\Gamma \vdash^{\mathsf{c}} M : \underline{B}$ denotes an oblique morphism from $[\![\Gamma]\!]$ to $[\![\underline{B}]\!]$.

So important are these oblique morphisms to our use of adjunctions that we are going to use a definition of adjunction that emphasizes them. To see why this is valuable, consider the continuation model using sets (Sect. 6.4.4). We first fix a set Ans and, for any set X, we write $\neg X$ for $X \to \mathsf{Ans}$. The continuation model is based on the adjunction where $\mathcal{B} = \mathbf{Set}$ and $\mathcal{D} = \mathbf{Set}^{\mathsf{op}}$:

$$\mathbf{Set}(X, \neg Y) \cong \mathbf{Set}(Y, \neg X) \tag{14.1}$$

Now an oblique morphism from X to Y can be described as a $\mathcal{B}$-morphism, a function from X to $\neg Y$. Or it can be described as a $\mathcal{D}$-morphism, a function from Y to $\neg X$. But it is surely simplest to describe it as a function from $X \times Y$ to Ans. Moreover, this agrees with our semantics of $\vdash^{\mathsf{c}}$ in Sect. 6.4.4: a computation $\Gamma \vdash^{\mathsf{c}} M : \underline{B}$ denotes a function from $[\![\Gamma]\!] \times [\![\underline{B}]\!]$ to Ans. We want to have the freedom, when we construct the above adjunction, to describe the oblique homsets in this way.

We will thus require an adjunction to provide not one isomorphism, as above, but two isomorphisms:

$$\begin{array}{rcll} \mathcal{B}(X, U\underline{Y}) & \cong & O(X,\underline{Y}) & \text{natural in } X \\ O(X,\underline{Y}) & \cong & \mathcal{D}(FX,\underline{Y}) & \text{natural in } \underline{Y} \end{array}$$

Here U and F are specified on objects only. By parametrized representability (Prop. 95), there is a unique way of extending U and F to functors so as to make these isomorphisms natural in both X and $\underline{Y}$, but we will have no need to do this.

In the case of the continuation model, these two isomorphisms are

$$\begin{array}{rcl} \mathbf{Set}(X, \neg Y) & \cong & \mathbf{Set}(X \times Y, \mathsf{Ans}) \\ \mathbf{Set}(X \times Y, \mathsf{Ans}) & \cong & \mathbf{Set}(Y, \neg X) \end{array}$$

This is a natural way to decompose (14.1).

14.2.2 Locally Indexed Categories

Moving to the locally $\mathcal{C}$-indexed setting, suppose $\mathcal{B}$ and $\mathcal{D}$ are locally $\mathcal{C}$-indexed categories and we are describing an adjunction between them. We can now speak of oblique morphisms from $X \in \mathsf{ob}\ \mathcal{B}$ to $\underline{Y} \in \mathsf{ob}\ \mathcal{D}$ over $\Gamma \in \mathsf{ob}\ \mathcal{C}$. These can be pre-composed, post-composed and reindexed, so, if we write $O_\Gamma(X,\underline{Y})$ for the set of oblique morphisms from X to $\underline{Y}$ over Γ, then O is a functor from $\mathsf{opGroth}(\mathcal{B}^{\mathsf{op}} \times \mathcal{D})$ to **Set**. The two isomorphisms look like this:

$$\begin{array}{rcll} \mathcal{B}_\Gamma(X, U\underline{Y}) & \cong & O_\Gamma(X,\underline{Y}) & \text{natural in } \Gamma \text{ and } X \\ O_\Gamma(X,\underline{Y}) & \cong & \mathcal{D}_\Gamma(FX,\underline{Y}) & \text{natural in } \Gamma \text{ and } \underline{Y} \end{array}$$

For the purposes of CBPV, we shall be concerned with an extremely special case of this situation: $\mathcal{C}$ is cartesian and $\mathcal{B}$ is $\mathsf{self}\ \mathcal{C}$, so $\mathcal{C}$ and $\mathcal{B}$ have the same objects. We can exploit this

to simplify our terminology. Instead of saying that g is an oblique morphism from X to $\underline{Y}$ over Γ, we can say, by combining Γ and X, that g is an oblique morphism over $\Gamma \times X$ to $\underline{Y}$. Thus, oblique morphisms do not require a source object—we can refer to oblique morphisms over Γ to $\underline{Y}$.

If g is an oblique morphism over Γ to $\underline{Y}$, we can reindex g by any $\mathcal{C}$-morphism $\Gamma' \xrightarrow{k} \Gamma$ and post-compose g with any $\mathcal{D}$-morphism $\underline{Y} \xrightarrow[\Gamma]{h} \underline{Y}'$. Consequently, if we write $O_\Gamma \underline{Y}$ for the set of oblique morphisms over Γ to $\underline{Y}$—and we again call this an "oblique homset"—then O is a functor from $\mathsf{opGroth}\ \mathcal{D}$ to **Set**.

14.3 Defining The Structure

14.3.1 Strong Adjunctions

Our notion of adjunction model is based on the following definition. As we explained in Sect. 14.2, it looks quite different from the usual definitions of adjunction because both U (the right adjoint) and F (the left adjoint) are given on objects only.

Definition 105 Let $\mathcal{C}$ be a cartesian category and let $\mathcal{D}$ be a locally $\mathcal{C}$-indexed category. A *strong adjunction* from $\mathcal{C}$ to $\mathcal{D}$ consists of

- a functor O from $\mathsf{opGroth}\ \mathcal{D}$ to **Set**—we call an element $g \in O_\Gamma \underline{Y}$ an *oblique morphism* over Γ to $\underline{Y}$ and we write $\xrightarrow[\Gamma]{g} \underline{Y}$
- for each $\underline{B} \in \mathsf{ob}\ \mathcal{D}$, a representation for the functor $\lambda\Gamma.O_\Gamma \underline{B}$, whose vertex we call $U\underline{B}$—explicitly, this is an isomorphism

$$\mathcal{C}(\Gamma, U\underline{B}) \cong O_\Gamma \underline{B} \quad \text{natural in } \Gamma \tag{14.2}$$

- for each $A \in \mathsf{ob}\ \mathcal{C}$, an A-representation for the functor O, whose vertex we call FA—explicitly, this is an isomorphism

$$O_{\Gamma \times A} \pi^*_{\Gamma,A} \underline{Y} \cong \mathcal{D}_\Gamma(FA, \underline{Y}) \quad \text{natural in } \Gamma \text{ and } \underline{Y} \tag{14.3}$$

□

The functoriality of O gives us reindexing and composition for oblique morphisms.

- For each oblique morphism $\xrightarrow[\Gamma]{g} \underline{Y}$ and $\mathcal{C}$-morphism $\Delta \xrightarrow{k} \Gamma$, we define the *reindexed* oblique morphism $\xrightarrow[\Delta]{k^*g} \underline{Y}$ to be $(O_k \underline{Y})g$.
- For each oblique morphism $\xrightarrow[\Gamma]{g} \underline{Y}$ and $\mathcal{D}$-morphism $\underline{Y} \xrightarrow[\Gamma]{h} \underline{Y}'$, we define the *composite* oblique morphism $\xrightarrow[\Gamma]{g;h} \underline{Y}'$ to be $(O_\Gamma h)g$.

These operations satisfy identity, associativity and reindexing laws:

$$\begin{aligned} g;\mathsf{id} &= g \\ g;(h;h') &= (g;h);h' \\ \mathsf{id}^*g &= g \\ (k';k)^*g &= k'^*(k^*g) \\ k^*(g;h) &= (k^*g);(k^*h) \end{aligned}$$

where g is an oblique morphism. Conversely, these operations and equations give us a functor O from $\mathsf{opGroth}\ \mathcal{D}$ to **Set**.

We will show (Prop. 133) that a strong adjunction from $\mathcal{C}$ to $\mathcal{D}$ is precisely an adjunction from $\mathsf{self}\ \mathcal{C}$ to $\mathcal{D}$. This can be added to our list of similar results:

- a distributive coproduct in $\mathcal{C}$ is precisely a coproduct in $\mathsf{self}\ \mathcal{C}$ (Prop. 113(1));
- a strong monad on $\mathcal{C}$ is precisely a monad on $\mathsf{self}\ \mathcal{C}$ (Prop. 118)

for a cartesian category $\mathcal{C}$. A consequence of these results is that a strong adjunction from $\mathcal{C}$ gives rise to a strong monad on $\mathcal{C}$.

14.3.2 Adjunction Models and CBPV

Def. 105 allows us to make the key definition of the chapter.

Definition 106 A *CBPV adjunction model* consists of

- a countably distributive category $\mathcal{C}$;
- a countably closed, locally $\mathcal{C}$-indexed category $\mathcal{D}$;
- a strong adjunction $(O, U, F, \dots)$ from $\mathcal{C}$ to $\mathcal{D}$.

□

We will see examples of this structure in Sect. 14.4.

Given an adjunction model $(\mathcal{C}, \mathcal{D}, O, \dots)$, we can interpret CBPV. The semantics is organized as follows:

- a value $\Gamma \vdash^{\mathsf{v}} V : A$ denotes a $\mathcal{C}$-morphism from $[\![\Gamma]\!]$ to $[\![A]\!]$;
- a computation $\Gamma \vdash^{\mathsf{c}} M : \underline{B}$ denotes an oblique morphism over $[\![\Gamma]\!]$ to $[\![\underline{B}]\!]$.

(We will explain this in detail in Sect. 14.5.) Notice that no term of CBPV denotes a $\mathcal{D}$-morphism—whereas there are 3 kinds of morphism in an adjunction model, there are only 2 judgements in CBPV. To see how we can think of the $\mathcal{D}$-morphisms from a CBPV perspective, define a *homomorphic context* from $\underline{A}$ to $\underline{B}$ over Γ to be a context of the form

$$\mathcal{C}[] = \lambda \overrightarrow{\mathtt{x}}.((\overrightarrow{V}\text{`}[])\ \mathtt{to}\ \mathtt{x}.\ M)$$

where $\underline{A}$ is the type of the hole, $\underline{B}$ is the type of the whole expression and all the free identifiers appear in Γ. (The operands $\overrightarrow{\mathtt{x}}$ and $\overrightarrow{V}$ can include both values and tags.) It is easy to see that the operation $M \mapsto \mathcal{C}[M]$ preserves all effects and sequencing. For example, the equations

$$\begin{aligned} \mathcal{C}[\mathtt{print}\ c;\ N] &= \mathtt{print}\ c;\ \mathcal{C}[N] \\ \mathcal{C}[\mathtt{diverge}] &= \mathtt{diverge} \\ \mathcal{C}[M\ \mathtt{to}\ \mathtt{x}.\ N] &= M\ \mathtt{to}\ \mathtt{x}.\ \mathcal{C}[N] \end{aligned}$$

are provable.

Given an adjunction model, a homomorphic context $\mathcal{C}[]$ will denote a $\mathcal{D}$-morphism from $[\![\underline{A}]\!]$ to $[\![\underline{B}]\!]$ over $[\![\Gamma]\!]$. In the Scott model, for example, $\mathcal{C}[]$ will denote a continuous function $[\![\Gamma]\!] \times [\![\underline{A}]\!] \xrightarrow{f} [\![\underline{B}]\!]$ which is *strict* in the sense that $f(\rho, \bot) = \bot$ for all $\rho \in [\![\Gamma]\!]$. This suggests that we add to CBPV a judgement

$$\Gamma | \underline{A} \vdash^{\mathsf{h}} H : \underline{B}$$

meaning that H is a homomorphic context from $\underline{A}$ to $\underline{B}$ over Γ. The attraction of adding such a judgement is that we would at last obtain a reversible derivation for F:

$$\frac{\Gamma, A \vdash^{\mathsf{c}} \underline{B}}{\Gamma | FA \vdash^{\mathsf{h}} \underline{B}}$$

We can then see that the isomorphism (14.3) corresponds to this reversible derivation, just as the isomorphism (14.2) corresponds to the reversible derivation for U:

$$\frac{\Gamma \vdash^{\mathsf{c}} \underline{B}}{\Gamma \vdash^{\mathsf{v}} U\underline{B}}$$

However, we cannot see that these homomorphic contexts have any special operational significance, and so we cannot justify adding this judgement to CBPV, even though the models of the augmented theory are indeed equivalent to adjunction models.

Since we will not augment the CBPV theory, if we want an equivalence theorem we have no choice but to remove some of the structure of adjunction models. The resulting mutilated structure is *staggered adjunction model*, discussed in Sect. 14.8.

14.4 Examples of Adjunction Models

14.4.1 Trivial Models

It is clear that a countably bicartesian closed category $\mathcal{C}$ gives a CBPV adjunction model: we set $\mathcal{D}$ to be $\mathsf{self}\ \mathcal{C}$ and we set $O_\Gamma B$ to be $\mathcal{C}(\Gamma, B)$. U and F are both identity (on objects).

It is worth comparing adjunction models to countably bicartesian closed categories. In the former, the requirement of finite products and countable distributive coproducts is imposed on $\mathcal{C}$, while the requirement of countable products and exponents is imposed on $\mathcal{D}$. In the latter, all these requirements are imposed on the same category. Thus the consequence of adding computational effects is to separate out these requirements into two categories related by an adjunction.

14.4.2 Eilenberg-Moore Models

If $(\mathcal{C}, T, \dots)$ is an unrestricted algebra model then we obtain an adjunction model $(\mathcal{C}, \mathcal{D}, O, \dots)$ by setting $\mathcal{D}$ to be the *Eilenberg-Moore category*: an object is a T-algebra and a $\mathcal{D}$-morphism from $\underline{A}$ to $\underline{B}$ over Γ is a T-algebra homomorphism in the sense of Def. 95. An oblique morphism from Γ to (Y, ϕ) is a $\mathcal{C}$-morphism from Γ to Y. It is easy to check all the required structure for an adjunction model.

As we saw in Chap. 12, the models for printing, divergence, errors and printing+divergence are all instances of this. For example, in the Scott model, a $\mathcal{D}$-morphism from $\underline{A}$ to $\underline{B}$ over Γ is a strict continuous function, as defined in Def. 12.

More generally, given a restricted algebra model $(\mathcal{C}, T, \{j\underline{B}\}_{B \in \mathfrak{I}}, \dots)$, we let an object of $\mathcal{D}$ be an algebra index (i.e. element of $\mathfrak{I}$) and a morphism from $\underline{A}$ to $\underline{B}$ over Γ be a T-algebra homomorphism from $j\underline{A}$ to $j\underline{B}$ over Γ. An oblique morphism from Γ to $\underline{B}$ is defined to be a $\mathcal{C}$-morphism from Γ to the carrier of $j\underline{B}$. We describe the construction in detail in Sect. 15.5.1.

14.4.3 Global Store

Suppose we have a CBPV adjunction model $(\mathcal{C}, \mathcal{D}, O, \dots)$ and a value object $S \in \mathsf{ob}\ \mathcal{C}$. We construct a new CBPV adjunction model $(\mathcal{C}, \mathcal{D}, O', \dots)$ where we set $O'_\Gamma \underline{Y}$ to be $O_{S \times \Gamma} \underline{Y}$. The new right adjoint is $U' = U(S \to -)$, with the isomorphism

$$O_{S \times \Gamma} \underline{B} \cong \mathcal{C}(\Gamma, U(S \to \underline{B})) \quad \text{natural in } \Gamma$$

The new left adjoint is $F' = F(S \times -)$, with the isomorphism

$$O_{S\times(\Gamma\times A)}\pi'^{*}_{S,\Gamma\times A}\pi^{*}_{\Gamma,A}\underline{Y} \cong \mathcal{D}_\Gamma(F(S\times A),\underline{Y}) \quad \text{natural in } \Gamma \text{ and } \underline{Y}$$

The global store model of Sect. 6.3.2 is obtained by applying this construction to the trivial adjunction model given by **Set**. The global store + printing model of Sect. 6.3.3 is obtained by applying this construction to the printing adjunction model described in Sect. 14.4.2.

14.4.4 Cell Generation

For any countable poset $\mathcal{W}$ we can define a construction on adjunction models, generalizing the global store construction in Sect. 14.4.3 (for the global store construction, $\mathcal{W}$ is the singleton poset).

Suppose we have a CBPV adjunction model $(\mathcal{C},\mathcal{D},O,\dots)$, and a value object $Sw \in \mathsf{ob}\ \mathcal{C}$ for each $w \in \mathsf{ob}\ \mathcal{W}$. We construct a new CBPV adjunction model $(\mathcal{C}',\mathcal{D}',O',\dots)$:

- The value category $\mathcal{C}'$ is $[\mathcal{W},\mathcal{C}]$. Clearly $\mathcal{C}'$ is countably distributive, by setting

$$\begin{aligned}(A\times A')w &= Aw \times A'w\\ (\textstyle\sum_{i\in I}A_i)w &= \textstyle\sum_{i\in I}A_i w\end{aligned}$$

- A $\mathcal{D}'$-object is a contravariant functor from $\mathcal{W}$ to $\mathcal{D}_1$.

- A $\mathcal{D}'$-morphism f from $\underline{A}$ to $\underline{B}$ over Γ provides, for each w, a $\mathcal{D}$-morphism fw from $\underline{A}w$ to $\underline{B}w$ over Γ w, in such a way that if $w \leqslant x$ then

$$\begin{array}{ccc}\underline{A}w & \xrightarrow{fw} & \underline{B}w\\ {\scriptstyle ()^*\underline{A}^w_x}\uparrow & & \uparrow{\scriptstyle ()^*\underline{B}^w_x}\\ \underline{A}x & \xrightarrow[\Gamma^w_x]{} & \underline{B}x\end{array} \quad \text{commutes.}$$

- Clearly $\mathcal{D}'$ is countably closed, by setting

$$\begin{aligned}(\textstyle\prod_{i\in I}\underline{B}_i)w &= \textstyle\prod_{i\in I}\underline{B}_i w\\ (A\to\underline{B})w &= [\![A]\!]w \to [\![\underline{B}]\!]w\end{aligned}$$

- An O'-oblique morphism over Γ to $\underline{B}$ provides, for each w, an O-oblique morphism over $Sw\times\Gamma w$ to $\underline{B}w$.

- Right and left adjoints are given by

$$\begin{aligned}(U\underline{B})w &= U\textstyle\prod_{w'\geqslant w}(Sw'\to\underline{B}w')\\ (FA)w &= F\textstyle\sum_{w'\geqslant w}(Sw'\times Aw')\end{aligned}$$

The remaining details are straightforward.

The model for cell generation in Chap. 7 is obtained by applying this construction to the trivial adjunction model given by **Set**. The cell generation + printing model of Sect. 7.8.2 is obtained by applying this construction to the printing adjunction model described in Sect. 14.4.2. Similarly, the cell generation + divergence of Sect. 7.8.3 is obtained by applying this construction to the Scott adjunction model described in Sect. 14.4.2.

14.4.5 Control

In Sect. 8.8 we explained how to obtain a *non-return model* from any kind of CBPV model (e.g. a value/producer model) with a chosen computation object $\underline{\mathsf{Ans}}$.

Now suppose we have a non-return model $(\mathcal{C}, \mathcal{G}, \ldots)$. We construct from this an adjunction model $(\mathcal{C}, \mathcal{D}, O, \ldots)$, as follows. Recalling the *opposite* of a locally $\mathcal{C}$-indexed category (Def. 78), we set $\mathcal{D}$ to be $(\mathsf{self}\ \mathcal{C})^{\mathsf{op}}$. Because $\mathcal{C}$ is countably distributive, $\mathcal{D}$ must be countably closed, using Prop. 113. We set $O_\Gamma B$ to be $\mathcal{G}(\Gamma \times B)$. Then the right adjoint U is $\neg$, with the isomorphism

$$O_\Gamma B \cong \mathcal{C}(\Gamma, \neg B) \qquad \text{natural in } \Gamma \in \mathcal{C}^{\mathsf{op}}$$

and the left adjoint F is $\neg$ too, with the isomorphism

$$O_{\Gamma \times A} \pi^*_{\Gamma,A} Y \cong (\mathsf{self}\ \mathcal{C})^{\mathsf{op}}_\Gamma (\neg A, Y) \quad \text{natural in } \Gamma \in \mathcal{C}^{\mathsf{op}} \text{ and } Y \in (\mathsf{self}\ \mathcal{C})^{\mathsf{op}}_\Gamma$$

Putting these constructions together, we can obtain an adjunction model from a CBPV model with a chosen computation object $\underline{\mathsf{Ans}}$. The model for control in Sect. 6.4.4 is obtained by applying this composite construction to the trivial CBPV model built from **Set**, with chosen set Ans. Similarly, the model for control+printing in Sect. 6.4.5 is obtained by applying this composite construction to the CBPV printing model with chosen $\mathcal{A}$-set $\underline{\mathsf{Ans}}$.

14.4.6 Erratic Choice

The CBPV erratic choice model in Sect. 6.5.2 is an adjunction model from **Set** to **Rel**, where **Rel** is the locally **Set**-indexed category in which an object is a set and a morphism from A to B over Γ is a relation from $\Gamma \times A$ to B, with the evident identities, composition and reindexing. **Rel** is countably closed because of the isomorphisms

$$\begin{array}{rcll} \prod_{i \in I} \mathbf{Rel}_\Gamma(X, B_i) & \cong & \mathbf{Rel}_\Gamma(X, \sum_{i \in I} B_i) & \text{natural in } \Gamma \text{ and } X \\ \mathbf{Rel}_{\Gamma \times A}(\pi^*_{\Gamma,A} X, B) & \cong & \mathbf{Rel}_\Gamma(X, A \times B) & \text{natural in } \Gamma \text{ and } X \end{array}$$

An oblique morphism from Γ to Y is a relation from Γ to Y, with the evident composition and reindexing operations. The right adjoint (U) is $\mathcal{P}$, with isomorphism

$$O_\Gamma B \cong \mathbf{Set}(\Gamma, \mathcal{P} B) \text{ natural in } \Gamma$$

The left adjoint (F) on objects is identity, with isomorphism

$$O_{\Gamma \times A} \pi^*_{\Gamma,A} Y \cong \mathbf{Rel}_\Gamma(A, Y) \text{ natural in } \Gamma \text{ and } Y$$

We comment that this adjunction is actually the Kleisli adjunction for the strong monad $\mathcal{P}$ on **Set**.

14.4.7 Pointer Game Model: The Families Construction

The pointer game model for CBPV given in Sect. 9.2 is an adjunction model obtained using a construction called *families*, based on [AM98a]. We start from the following structure.

Definition 107 A *pre-families adjunction model* consists of

- a cartesian category $\acute{\mathcal{C}}$
- a closed, locally $\acute{\mathcal{C}}$-indexed category $\acute{\mathcal{D}}$
- a functor $\acute{O}$ from $\mathsf{opGroth}\ \acute{\mathcal{D}}$ to **Set**

- for each countable family of $\acute{\mathcal{D}}$-objects $\{\underline{R}_i\}_{i\in I}$, a representation for the functor $\lambda\Gamma.\prod_{i\in I}\acute{O}_\Gamma\underline{R}_i$, whose vertex we call $U_{i\in I}\underline{R}_i$—explicitly, this is an isomorphism

$$\prod_{i\in I}\acute{O}_\Gamma\underline{R}_i \cong \acute{\mathcal{C}}(\Gamma, U_{i\in I}\underline{R}_i) \quad \text{natural in } \Gamma$$

- for each countable family of $\acute{\mathcal{C}}$-objects $\{R_i\}_{i\in I}$, a representation for the functor $\lambda_\Gamma\underline{Y}.\prod_{i\in I}\acute{O}_{\Gamma\times R_i}\pi^*_{\Gamma,R}\underline{Y}$, whose vertex we call $F_{i\in I}R_i$—explicitly, this is an isomorphism

$$\prod_{i\in I}\acute{O}_{\Gamma\times R_i}\pi^*_{\Gamma,R}\underline{Y} \cong \acute{\mathcal{D}}_\Gamma(F_{i\in I}R_i, \underline{Y}) \quad \text{natural in } \Gamma \text{ and } \underline{Y}$$

□

Definition 108 (families construction) Let $(\acute{\mathcal{C}}, \acute{\mathcal{D}}, \acute{O}, \dots)$ be a pre-families adjunction model. We obtain an adjunction model $(\mathcal{C}, \mathcal{D}, O, \dots)$ as follows.

- A $\mathcal{C}$-object is a countable family of $\acute{\mathcal{C}}$-objects.
- The $\mathcal{C}$ homset from $\{R_i\}_{i\in I}$ to $\{S_j\}_{j\in J}$ is the set $\prod_{i\in I}\sum_{j\in J}\acute{\mathcal{C}}(R_i, S_j)$ with the evident identities and composition.
- Thus $\mathcal{C}$ is a countably distributive category, with

$$\begin{array}{rll} \{R_i\}_{i\in I}\times\{S_j\}_{j\in J} & \text{given by} & \{R_i\times S_j\}_{(i,j)\in I\times J} \\ \sum_{i\in I}\{R_{ij}\}_{j\in J_i} & \text{given by} & \{R_{ij}\}_{(i,j)\in \sum_{i\in I}J_i} \end{array}$$

- A $\mathcal{D}$-object is a countable family of $\acute{\mathcal{D}}$-objects.
- The $\mathcal{D}$ homset over $\{\Gamma_i\}_{i\in I}$ from $\{\underline{R}_j\}_{j\in J}$ to $\{\underline{S}_k\}_{k\in K}$ is the set $\prod_{i\in I}\prod_{k\in K}\sum_{j\in J}\acute{\mathcal{D}}_{\Gamma_i}(\underline{R}_j, \underline{S}_k)$ with the evident identities, composition and reindexing.
- Thus $\mathcal{D}$ is a countably closed category, with

$$\begin{array}{rll} \{R_i\}_{i\in I}\to\{\underline{S}_j\}_{j\in J} & \text{given by} & \{R_i\to\underline{S}_j\}_{(i,j)\in I\times J} \\ \prod_{i\in I}\{\underline{R}_{ij}\}_{j\in J_i} & \text{given by} & \{\underline{R}_{ij}\}_{(i,j)\in \sum_{i\in I}J_i} \end{array}$$

The O homset over $\{\Gamma_i\}_{i\in I}$ to $\{\underline{R}_j\}_{j\in J}$ is the set $\prod_{i\in I}\prod_{j\in J}\acute{O}_{\Gamma_i}\underline{R}_j$ with the evident reindexing and composition.

This gives an adjunction model, with

$$\begin{array}{rll} U\{\underline{R}_i\}_{i\in I} & \text{given by the singleton family} & \{U_{i\in I}\underline{R}_i\} \\ F\{R_i\}_{i\in I} & \text{given by the singleton family} & \{F_{i\in I}R_i\} \end{array}$$

□

To give a game example of this, we will need yet another kind of pointer game. Suppose that Γ, R and S are arenas. The O-*first game* over Γ^{P} from R^{O} to S^{P} is the game whose rules are as follows:

- Play alternates between Player and Opponent. Opponent moves first.
- In each move, either a token of Γ (a *context token*), a token of R (a *source token*) or a token of S (a *target token*) is passed.

- In the initial move, Opponent passes a root of R.
- Player moves by either
  - passing a root of Γ or S, or
  - pointing to a previous O-move m and passing a successor of the token passed in move m.
- Opponent moves (except in the initial move) by pointing to a previous P-move m and passing a successor of the token passed in move m.

It is also permitted for either player to diverge instead of moving, except for the initial move, which Opponent must play. A consequence of these rules is that Player can pass only a P-token of Γ^{P}, R^{O} and S^{P}, and Opponent can pass only an O-token. We write $\mathsf{Ostrat}_{\Gamma^{\mathsf{P}}}(R^{\mathsf{O}}, S^{\mathsf{P}})$ for the set of strategies for the O-first game over Γ^{P} from R^{O} to S^{P}. This can be defined formally as in Sect. 9.2.4.

These strategies give us the following pre-families adjunction model, from which we obtain, by the families construction, the pointer game model for CBPV described in Sect. 9.2. We describe the homsets here: the operations on strategies such as composition, reindexing etc. are described in Sect. B.5.2.

- An object of $\acute{\mathcal{C}}$ is an arena.
- A $\acute{\mathcal{C}}$-morphism from R to S is an O-first strategy from R^{P} to S^{O} .
- $R \times R'$ is given as $R \uplus R'$.
- An object of $\acute{\mathcal{D}}$ is an arena.
- A $\acute{\mathcal{D}}$-morphism from $\underline{R}$ to $\underline{S}$ over Γ is an O-first strategy over Γ^{P} from $\underline{R}^{\mathsf{O}}$ to $\underline{S}^{\mathsf{P}}$.
- $R \to \underline{S}$ is given as $R \uplus \underline{S}$.
- An $\acute{O}$-morphism over Γ to $\underline{R}$ (i.e. an element of $\acute{O}_{\Gamma}\underline{R}$) is a P-first strategy from Γ^{P} to $\underline{R}^{\mathsf{P}}$.
- $U_{i\in I}\underline{R}_i$ is given as $\mathsf{pt}^{\mathsf{Q}}_{i\in I}\underline{R}_i$.
- $F_{i\in I}R_i$ is given as $\mathsf{pt}^{\mathsf{A}}_{i\in I}R_i$.

Similar pre-families models are obtained by imposing constraints of bracketing, visibility, innocence etc.

14.5 Interpreting CBPV In An Adjunction Model

Suppose we are given a CBPV adjunction model $(\mathcal{C}, \mathcal{D}, O, \dots)$. As we said in Sect. 14.3.2, this gives us a semantics for CBPV, organized as follows:

- a value $\Gamma \vdash^{\mathsf{v}} V : A$ denotes a $\mathcal{C}$-morphism from $[\![\Gamma]\!]$ to $[\![A]\!]$;
- a computation $\Gamma \vdash^{\mathsf{c}} M : \underline{B}$ denotes an oblique morphism over $[\![\Gamma]\!]$ to $[\![\underline{B}]\!]$.

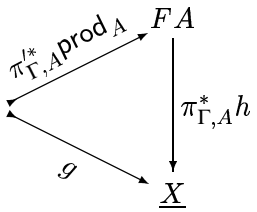
To interpret the constructs for F, we recall from Def. 106 that FA is the vertex of an A-representation for O. As we saw in Sect. 10.6.5, this A-representation can be expressed in two ways:

isomorphism style as an isomorphism (14.3), which we write as

$$O_{\Gamma\times A}\pi^{*}_{\Gamma,A}\underline{Y} \xrightarrow[\cong]{\mathsf{str}_{\Gamma A\underline{Y}}} \mathcal{D}_{\Gamma}(FA, \underline{Y}) \text{ natural in } \Gamma \text{ and } \underline{Y}$$

element style as an oblique morphism $\xrightarrow[A]{\mathsf{prod}_A} FA$ with the following "initiality" property:

for any $\xrightarrow[\Gamma\times A]{g} \underline{X}$ there is a unique $FA \xrightarrow[\Gamma]{h} \underline{X}$ such that the diagram



over $\Gamma\times A$ commutes.

We see from Prop. 115(covariant) that the two forms of the A-representation are related as follows:

- prod_A is obtained by applying str^{-1} to the $\mathcal{D}$-morphism $FA \xrightarrow[1]{\mathsf{id}} FA$, giving an oblique morphism over $1\times A$ to FA, and then reindexing along the isomorphism $A \xrightarrow{i} 1\times A$.
- for a $\mathcal{D}$-morphism $FA \xrightarrow[\Gamma]{h} \underline{B}$ we have

$$\mathsf{str}^{-1}_{\Gamma A\underline{B}}h = (\pi'^{*}_{\Gamma,A}\mathsf{prod}_A);(\pi^{*}_{\Gamma,A}h) \tag{14.4}$$

We use str and prod to describe the semantics of FA constructs:

- If $\Gamma \vdash^{\mathsf{v}} V : A$, then $\mathtt{produce}\ V$ denotes $\xrightarrow[[\![\Gamma]\!]]{[\![V]\!]^*\mathsf{prod}_{[\![A]\!]}} F[\![A]\!]$.
- If $\Gamma \vdash^{\mathsf{c}} M : FA$ and $\Gamma, \mathtt{x} : A \vdash^{\mathsf{c}} N : \underline{B}$, then $M\ \mathtt{to}\ \mathtt{x}.\ N$ denotes the composite

$$\xrightarrow[[\![\Gamma]\!]]{[\![M]\!]} F[\![A]\!] \xrightarrow[[\![\Gamma]\!]]{\mathsf{str}\ [\![N]\!]} [\![\underline{B}]\!]$$

To interpret the constructs for U, we recall from Def. 106 that $U\underline{B}$ is the vertex of a representation for $\lambda\Gamma.\, O_\Gamma\underline{B}$. As we saw in Sect. 10.3, this representation can be expressed in two ways:

isomorphism style as an isomorphism (13.1), which we write as

$$O_\Gamma\underline{B} \xrightarrow[\cong]{\mathsf{thunk}_{\Gamma,\underline{B}}} \mathcal{C}(\Gamma, U\underline{B}) \text{ natural in } \Gamma$$

element style as a terminal object $\xrightarrow[U\underline{B}]{\mathsf{force}_{\underline{B}}} \underline{B}$ in the category where

- an object is a pair (X, f) where $\xrightarrow[X]{f} \underline{B}$
- a morphism from (X, f) to (Y, g) is a $\mathcal{C}$-morphism $X \xrightarrow{h} Y$ such that $f = h^*g$.

We see from Prop. 93(contravariant) that the two forms of the representation are related as follows:

- $\mathsf{force}_{\underline{B}}$ is obtained by applying $\mathsf{thunk}^{-1}_{U\underline{B},\underline{B}}$ to the identity on $U\underline{B}$ in $\mathcal{C}$.
- $\mathsf{thunk}^{-1}_{\Gamma\underline{B}}$ takes $\Gamma \xrightarrow{f} U\underline{B}$ to $f^*\mathsf{force}_{\underline{B}}$.

We interpret the constructs for $U\underline{B}$ as follows:

- If $\Gamma \vdash^{\mathsf{c}} M : \underline{B}$ then $\mathtt{thunk}\ M$ denotes $\mathsf{thunk}\,[\![M]\!]$.
- If $\Gamma \vdash^{\mathsf{v}} V : U\underline{B}$ then $\mathtt{force}\ V$ denotes $[\![V]\!]^*\mathsf{force}_{[\![\underline{B}]\!]}$

It may seem more straightforward to ignore the element-style description of the representation and simply say that $\mathtt{force}\ V$ denotes $\mathsf{thunk}^{-1}[\![V]\!]$, but the advantages of using $\mathsf{force}_{\underline{B}}$ become evident in Sect. 15.5.2.

To give the semantics of $A \to \underline{B}$, we will construct an isomorphism

$$\mathcal{O}_{\Gamma\times A}\underline{B} \cong \mathcal{O}_\Gamma A \to \underline{B} \qquad \text{natural in } \Gamma \tag{14.5}$$

corresponding to the reversible derivation

$$\frac{\Gamma, A \vdash^{\mathsf{c}} \underline{B}}{\Gamma \vdash^{\mathsf{c}} A \to \underline{B}}$$

Using (14.5), we interpret the $\to$ constructs as follows:

- If $\Gamma, \mathtt{x} : A \vdash^{\mathsf{c}} M : \underline{B}$ then $\lambda\mathtt{x}.M$ denotes the oblique morphism $\xrightarrow[[\![\Gamma]\!]]{} [\![A]\!] \to [\![\underline{B}]\!]$ corresponding to $[\![M]\!]$ along (14.5).
- If $\Gamma \vdash^{\mathsf{v}} V : A$ and $\Gamma \vdash^{\mathsf{c}} M : A \to \underline{B}$ then $V\text{`}M$ denotes $\xrightarrow[\Gamma]{(\mathsf{id}_\Gamma, [\![V]\!])^*g} [\![\underline{B}]\!]$ where g corresponds to $[\![M]\!]$ along (14.5)

To construct (14.5), we define a *division* of Γ (an object in a cartesian category) to be a triple (X, Y, i), where X and Y are value objects and i is an isomorphism $\Gamma \cong X \times Y$. We choose a division of Γ and construct (14.5) as the composite

$$\begin{array}{ccccc} \mathcal{O}_{\Gamma\times A}\underline{B} & \xrightarrow{\cong} & \mathcal{O}_{(X\times A)\times Y}\underline{B} & \xrightarrow{\cong} & \mathcal{D}_{X\times A}(FY, \underline{B}) \\ & & & & \Big\downarrow \cong \\ \mathcal{O}_\Gamma A \to \underline{B} & \underset{\cong}{\longrightarrow} & \mathcal{O}_{X\times Y} A \to \underline{B} & \underset{\cong}{\longrightarrow} & \mathcal{D}_X(FY, A \to \underline{B}) \end{array} \tag{14.6}$$

This construction appears to depend on the choice of division of Γ, but this is not the case:

Proposition 128 1. For any Γ, the composite isomorphism (14.6) is independent of the particular division $\Gamma \cong X \times Y$ used.

2. The isomorphism (14.5) is natural in $\Gamma \in \mathcal{C}^{\mathsf{op}}$.

□

Proof

(1) we defer to Sect. 14.9.3.

(2) follows from (1). As we are free to choose any division of Γ, we choose the division $\Gamma \cong \Gamma \times 1$, and then (14.6) is a composite of isomorphisms each of which is natural in $\Gamma \in \mathcal{C}^{\mathsf{op}}$.

□

To interpret the constructs for $\prod_{i\in I}\underline{B}_i$ we provide, for each value object Γ, an isomorphism

$$\prod_{i\in I} O_\Gamma \underline{B}_i \cong O_\Gamma \prod_{i\in I}\underline{B}_i \qquad \text{natural in } \Gamma \tag{14.7}$$

corresponding to the reversible derivation

$$\frac{\cdots \quad \Gamma \vdash^{\mathsf{c}} \underline{B}_i \quad \cdots}{\Gamma \vdash^{\mathsf{c}} \prod_{i\in I}\underline{B}_i}$$

(14.7) is constructed, and its naturality in $\Gamma \in \mathcal{C}^{\mathsf{op}}$ proved, in a similar way to (14.5), although the proofs are easier.

Proposition 129 The interpretation of CBPV in a CBPV adjunction model satisfies all the equations of the CBPV equational theory. □

We defer the proof of this to Sect. 14.9.3.

14.6 Adjunctions

14.6.1 Basic Definition

The aim of this section is to explain and prove the following claim that we made in Sect. 14.3.2, that a strong adjunction from $\mathcal{C}$ to $\mathcal{D}$ is the same as an adjunction from $\mathsf{self}\ \mathcal{C}$ to $\mathcal{D}$. In the course of doing this we will examine numerous definitions of adjunction, and make precise the discussion in Sect. 14.2.

We take as our basic definition of "adjunction" the following. It is widely accepted because, like Def. 92, it makes sense in any 2-category.

Definition 109 Let $\mathcal{B}$ and $\mathcal{D}$ be either

- both categories, or
- both locally $\mathcal{C}$-indexed categories, for the same category $\mathcal{C}$.

Then an *adjunction* from $\mathcal{B}$ to $\mathcal{D}$ consists of

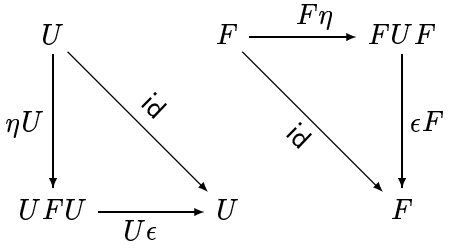
- two functors

$$\mathcal{B} \underset{U}{\overset{F}{\rightleftarrows}} \mathcal{D}$$

  (U is called the *right adjoint*, F is called the *left adjoint*);

- natural transformations $1 \xrightarrow{\eta} UF$ (the *unit*) and $FU \xrightarrow{\epsilon} 1$ (the *counit*)

satisfying the *triangle laws*



□

Proposition 130 An adjunction from $\mathcal{B}$ gives rise to a monad on $\mathcal{B}$. □

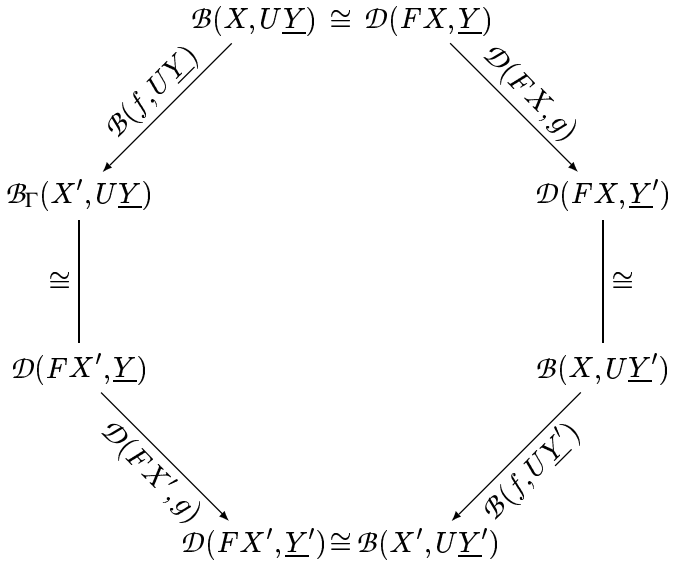
Proof We set $T = UF$ and $\mu = U\epsilon F$. The required diagrams are easily verified. □

14.6.2 Adjunction Between Ordinary Categories

As we said in Sect. 14.3.1, there are many equivalent definitions of adjunction. In the setting of ordinary categories, we make a list in which we collect some familiar definitions (1)–(3) and some new definitions (4)–(5).

Definition 110 Let $\mathcal{B}$ and $\mathcal{D}$ be categories. We underline objects of $\mathcal{D}$.

1. An *adjunction* from $\mathcal{B}$ to $\mathcal{D}$ consists of
   - two functors
   $$\mathcal{B} \underset{U}{\overset{F}{\rightleftarrows}} \mathcal{D}$$
   - an isomorphism
   $$\mathcal{B}(X, U\underline{Y}) \cong \mathcal{D}(FX, \underline{Y}) \text{ natural in } X \text{ and } \underline{Y}.$$
2. An *adjunction* from $\mathcal{B}$ to $\mathcal{D}$ consists of
   - a functor $\mathcal{D} \xrightarrow{U} \mathcal{B}$
   - for each $X \in \mathsf{ob}\,\mathcal{B}$, a representation for $\lambda\underline{Y}.\mathcal{B}(X,\underline{Y})$, whose vertex we call FX.
3. An *adjunction* from $\mathcal{B}$ to $\mathcal{D}$ consists of
   - a functor $\mathcal{B} \xrightarrow{F} \mathcal{D}$
   - for each $\underline{Y} \in \mathsf{ob}\,\mathcal{D}$ a representation for $\lambda X.\mathcal{D}(FX,\underline{Y})$, whose vertex we call $U\underline{Y}$.
4. An *adjunction* from $\mathcal{B}$ to $\mathcal{D}$ consists of
   - two functions
   $$\mathsf{ob}\,\mathcal{B} \underset{U}{\overset{F}{\rightleftarrows}} \mathsf{ob}\,\mathcal{D}$$
   - for each $X \in \mathsf{ob}\,\mathcal{B}$ and $\underline{Y} \in \mathsf{ob}\,\mathcal{D}$ a bijection
   $$\mathcal{B}(X, U\underline{Y}) \cong \mathcal{D}(FX, \underline{Y})$$
   such that, for each $\mathcal{B}$-morphism $X' \xrightarrow{f} X$ and each $\mathcal{C}$-morphism $\underline{Y} \xrightarrow{g} \underline{Y}'$, the following commutes:



5. An *adjunction* from $\mathcal{B}$ to $\mathcal{D}$ consists of

   - a functor O from $\mathcal{B}^{\mathrm{op}} \times \mathcal{D}$ to **Set**—we call an element $g \in O(X, \underline{Y})$ an *oblique morphism* from X to $\underline{Y}$ and write $X \xrightarrow{\quad g \quad} \underline{Y}$
   - for each object $\underline{Y} \in \mathcal{D}$, a representation for $\lambda X.O(X, \underline{Y})$, whose vertex we call $U\underline{Y}$
   - for each object $X \in \mathcal{B}$, a representation for $\lambda Y.O(X, \underline{Y})$, whose vertex we call FX.

□

We compare these definitions as follows.

- In Def. 110(1)—like in Def. 109—both F and U are given on both objects and morphisms. Accordingly, the isomorphism

$$\mathcal{B}(X, U\underline{Y}) \cong \mathcal{D}(FX, \underline{Y}) \tag{14.8}$$

  is required to be natural in both X and $\underline{Y}$.

- In Def. 110(2), U is given on both objects and morphisms but F is given only on objects. Accordingly, the isomorphism (14.8) is required to be natural in $\underline{Y}$ but not in X.

- In Def. 110(3), F is given on both objects and morphisms but U is given only on objects. Accordingly, the isomorphism (14.8) is required to be natural in X but not in $\underline{Y}$.

- in Def. 110(4)—(5), both U and F are given only on objects. Accordingly, in Def. 110(4) the isomorphism (14.8) is not required to be natural in either X or $\underline{Y}$, while in Def. 110(5), it is divided into two isomorphisms:

$$\begin{array}{rcll} \mathcal{B}(X, U\underline{Y}) & \cong & O(X, \underline{Y}) & \text{natural in } X \\ O(X, \underline{Y}) & \cong & \mathcal{D}(FX, \underline{Y}) & \text{natural in } \underline{Y} \end{array}$$

It is clear that Def. 110(5) is most similar to our notion of strong adjunction. An oblique morphism can be pre-composed with a $\mathcal{C}$-morphism or post-composed with a $\mathcal{D}$-morphism, and these operations satisfy identity and associativity laws.

Proposition 131 Def. 110(1)–(5) and Def. 109 are all equivalent. □

Proof The equivalence of Def.109) and Def. 110(1) is standard. The equivalence of (1)–(3) and (5) is a consequence of parametrized representability (Prop. 95). In moving from (1) to (5) we can either set $O(X, \underline{Y})$ to be $\mathcal{B}(X, U\underline{Y})$ or we can set it to be $\mathcal{D}(FX, \underline{Y})$. The equivalence of (4) and (5) is straightforward. □

14.6.3 Adjunction Between Locally Indexed Categories

In Def. 110 we gave many definitions for "adjunction" in the setting of ordinary categories. It is easy to adapt these definitions to the setting of locally $\mathcal{C}$-indexed categories:

Definition 111 (cf.110) Let $\mathcal{B}$ and $\mathcal{D}$ be locally $\mathcal{C}$-indexed categories. We underline objects of $\mathcal{D}$.

1. An *adjunction* from $\mathcal{B}$ to $\mathcal{D}$ consists of

   - two functors

$$\mathcal{B} \underset{U}{\overset{F}{\rightleftarrows}} \mathcal{D}$$

- a bijection

$$\mathcal{B}_\Gamma(X, U\underline{Y}) \cong \mathcal{D}_\Gamma(FX, \underline{Y}) \quad \text{natural in } \Gamma, X \text{ and } \underline{Y}.$$

Note In the literature, the condition of naturality in Γ is usually replaced by the *Beck-Chevalley condition*. The two conditions are equivalent (assuming naturality in X and $\underline{Y}$), but we use the former because we consider it to be more intuitive.

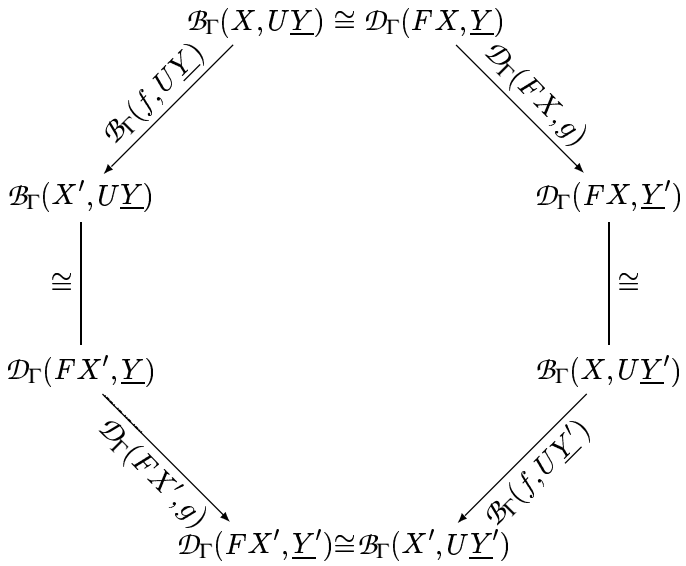
2. An *adjunction* from $\mathcal{B}$ to $\mathcal{D}$ consists of
   - a functor $\mathcal{D} \xrightarrow{U} \mathcal{B}$
   - for each $X \in \mathsf{ob}\ \mathcal{B}$, a representation for $\lambda_\Gamma \underline{Y}.\mathcal{B}_\Gamma(X, \underline{Y})$, whose vertex we call FX.
3. An *adjunction* from $\mathcal{B}$ to $\mathcal{D}$ consists of
   - a functor $\mathcal{B} \xrightarrow{F} \mathcal{D}$
   - for each $\underline{Y} \in \mathsf{ob}\ \mathcal{D}$ a representation for $\lambda_\Gamma X.\mathcal{D}_\Gamma(FX, \underline{Y})$, whose vertex we call $U\underline{Y}$.
4. An *adjunction* from $\mathcal{B}$ to $\mathcal{D}$ consists of
   - two functions

$$\mathsf{ob}\ \mathcal{B} \underset{U}{\overset{F}{\rightleftarrows}} \mathsf{ob}\ \mathcal{D}$$

   - for each $X \in \mathsf{ob}\ \mathcal{B}$ and $\underline{Y} \in \mathsf{ob}\ \mathcal{D}$ a bijection

$$\mathcal{B}_\Gamma(X, U\underline{Y}) \cong \mathcal{D}_\Gamma(FX, \underline{Y}) \text{ natural in } \Gamma$$

   such that, for each $\mathcal{B}$-morphism $X' \xrightarrow[\Gamma]{f} X$ and each $\mathcal{C}$-morphism $\underline{Y} \xrightarrow[\Gamma]{g} \underline{Y}'$, the following commutes:



5. An *adjunction* from $\mathcal{B}$ to $\mathcal{D}$ consists of
   - a functor O from $\mathsf{opGroth}(\mathcal{B}^{\mathsf{op}} \times \mathcal{D})$ to **Set**—we call an element $g \in O_\Gamma(X, \underline{Y})$ an *oblique morphism* from X to $\underline{Y}$ over Γ and write $X \xrightarrow[\Gamma]{g} \underline{Y}$

- for each object $\underline{Y} \in \mathcal{D}$, a representation for $\lambda_\Gamma X.\mathcal{O}_\Gamma(X,\underline{Y})$, whose vertex we call $U\underline{Y}$
- for each object $X \in \mathcal{B}$, a representation for $\lambda_\Gamma Y.\mathcal{O}_\Gamma(X,\underline{Y})$, whose vertex we call FX.

□

It is Def. 111(5) which is closest to our notion of strong adjunction. An oblique morphism can be pre-composed with a morphism in $\mathcal{B}$, post-composed with a morphism in $\mathcal{D}$ or reindexed along a morphism in $\mathcal{C}$. These operations satisfy identity, associativity and reindexing laws.

Proposition 132 Def. 109 and Def. 111(1)–(5) are equivalent. □

This is proved the same way as Prop. 131.

14.6.4 Proof of Equivalence Theorem

Let $\mathcal{C}$ be a cartesian category and let $\mathcal{D}$ be a locally $\mathcal{C}$-indexed category. Our aim is to prove the following:

Proposition 133 strong adjunctions from $\mathcal{C}$ to $\mathcal{D}$ and adjunctions from $\mathsf{self}\ \mathcal{C}$ to $\mathcal{D}$ are equivalent. □

We will use Def. 111(5) to characterize adjunctions from $\mathsf{self}\ \mathcal{C}$ to $\mathcal{D}$.

Our proof will make use of various functors involving $\mathsf{self}\ \mathcal{C}$. We introduce a useful notation:

Definition 112

- Given $\mathcal{C}$-objects Γ and X, we write $\Gamma \cdot X$ for $\Gamma \times X$;
- Given $\mathcal{C}$-morphisms $\Gamma' \xrightarrow{k} \Gamma$ and $\Gamma' \times X \xrightarrow{f} Y$ we write $k \cdot f$ for the $\mathcal{C}$-morphism $\Gamma' \times X \xrightarrow{((\pi;k),f)} \Gamma \times Y$.

□

This notation gives us

- the functor

$$\mathsf{opGroth}\ \mathsf{self}\ \mathcal{C} \longrightarrow \mathcal{C}^{\mathsf{op}}$$

$$_\Gamma X \longmapsto \Gamma \cdot X$$

- the functor

$$\mathsf{opGroth}((\mathsf{self}\ \mathcal{C})^{\mathsf{op}} \times \mathcal{D}) \longrightarrow \mathsf{opGroth}\ \mathcal{D}$$

$$_\Gamma(X,\underline{Y}) \longmapsto {}_{\Gamma\cdot X}\pi^*_{\Gamma,X}\underline{Y}$$

It is easily verified that these preserve identities and composition.

We give a homset functor for $\mathcal{C}$, which is more general than the basic one from $\mathcal{C}^{\mathsf{op}} \times \mathcal{C}$ to **Set**.

Definition 113 We write $\mathsf{Hom}_\mathcal{C}$ or just $\mathcal{C}$ for the functor

$$\mathsf{opGroth}\ \mathsf{self}\ \mathcal{C} \longrightarrow \mathbf{Set}$$

$$_\Gamma Y \longmapsto \mathcal{C}(\Gamma, Y)$$

$$(_\Gamma Y \xrightarrow{{}_k f} {}_\Gamma Z) \mapsto \lambda g.((\mathsf{id},k);(\Gamma' \times g);f)$$

It is easily verified that this preserves identities and composition. □

The advantage of this more general homset functor for $\mathcal{C}$ is that the homset functor for $\mathsf{self}\ \mathcal{C}$ can be recovered from it:

$$(\mathsf{self}\ \mathcal{C})_\Gamma(X,Y) = \mathcal{C}_{\Gamma\cdot X}\pi^*_{\Gamma,X}Y \text{ natural in } \Gamma,\ X \text{ and } Y$$

Lemma 134 Suppose we have an adjunction $(O,\dots)$ from $\mathsf{self}\ \mathcal{C}$ to $\mathcal{D}$. Then we obtain an isomorphism

$$O_\Gamma(X,\underline{Y}) \cong O_{\Gamma\cdot X}(1,\pi^*_{\Gamma,X}\underline{Y}) \text{ natural in } \Gamma, X, \underline{Y} \tag{14.9}$$

□

Proof Using parametrized representability (Prop. 116(contravariant)), we extend U to a functor from $\mathcal{D}$ to $\mathsf{self}\ \mathcal{C}$ in the unique way that makes the isomorphism

$$O_\Gamma(X,\underline{Y}) \cong (\mathsf{self}\ \mathcal{C})_\Gamma(X,U\underline{Y})$$

—which is required to be natural in Γ and X—natural also in $\underline{Y}$. Then (14.9) is given as the composite

$$\begin{array}{ccccc}
O_\Gamma(X,\underline{Y}) & \xrightarrow{\cong} & (\mathsf{self}\ \mathcal{C})_\Gamma(X,U\underline{Y}) & \xrightarrow{\cong} & \mathcal{C}_{\Gamma\cdot X}\pi^*_{\Gamma,X}U\underline{Y} \\
 & & & & \Big| = \\
 & & & & \mathcal{C}_{\Gamma\cdot X}U\pi^*_{\Gamma,X}\underline{Y} \\
 & & & & \Big| \cong \text{ (by Lemma 109)} \\
O_{\Gamma\cdot X}(1,\pi^*_{\Gamma,X}\underline{Y}) & \underset{\cong}{-} & (\mathsf{self}\ \mathcal{C})_{\Gamma\cdot X}(1,U\pi^*_{\Gamma,X}\underline{Y}) & \underset{=}{-} & \mathcal{C}_{(\Gamma\cdot X)\cdot 1}\pi^*_{\Gamma\cdot X,1}U\pi^*_{\Gamma,X}\underline{Y}
\end{array}$$

□

Proof of Prop. 133.

- Suppose we have a strong adjunction $(O,\dots)$ from $\mathcal{C}$ to $\mathcal{D}$. We construct an adjunction $(O,\dots)$ from $\mathsf{self}\ \mathcal{C}$ to $\mathcal{D}$ by setting $O_\Gamma(X,\underline{Y})$ to be $O_{\Gamma\cdot X}\pi^*_{\Gamma,X}\underline{Y}$. The isomorphism for $U\underline{B}$ is given by

  $$O_\Gamma(X,\underline{B}) = O_{\Gamma\cdot X}\underline{B} \cong \mathcal{C}_{\Gamma\cdot X}U\underline{B} = (\mathsf{self}\ \mathcal{C})_\Gamma(X,U\underline{B})$$

  natural in Γ and X. The isomorphism for FA is given by

  $$O_\Gamma(A,\underline{Y}) = O_{\Gamma\cdot A}\pi^*_{\Gamma,A}\underline{Y} \cong \mathcal{D}_\Gamma(FA,\underline{Y})$$

  natural in Γ and $\underline{Y}$.

- Suppose we have an adjunction $(O,\dots)$ from $\mathsf{self}\ \mathcal{C}$ to $\mathcal{D}$. We construct a strong adjunction $(O,\dots)$ by setting $O_\Gamma\underline{Y}$ to be $O_\Gamma(1,\underline{Y})$. The isomorphism for $U\underline{B}$ is given by

  $$O_\Gamma\underline{B} = O_\Gamma(1,\underline{B}) \cong (\mathsf{self}\ \mathcal{C})_\Gamma(1,\underline{B}) \cong \mathcal{C}_{\Gamma\cdot 1}\underline{B} \cong \mathcal{C}_\Gamma U\underline{B}$$

  natural in Γ. The isomorphism for FA is given by

  $$O_{\Gamma\cdot A}\pi^*_{\Gamma,A}\underline{Y} = O_{\Gamma\cdot A}(1,\pi^*_{\Gamma,A}\underline{Y}) \cong O_\Gamma(A,\underline{Y}) \cong \mathcal{D}_\Gamma(FA,\underline{Y})$$

  natural in Γ and $\underline{Y}$.

- Suppose we have a strong adjunction $(O,\dots)$. We obtain an adjunction $(O,\dots)$ and then another strong adjunction $(O',\dots)$. We have

$$O'_\Gamma \underline{Y} = O_\Gamma(1,\underline{Y}) = O_{\Gamma\cdot 1}\pi^*_{\Gamma,1}\underline{Y} \cong O_\Gamma \underline{Y}$$

  natural in Γ and $\underline{Y}$. (The isomorphism on the right is obtained by Lemma 109.) It is easily verified that this isomorphism preserves the isomorphisms for U and F.

- Suppose we have an adjunction $(O,\dots)$. We obtain a strong adjunction $(O,\dots)$ and then another adjunction $(O',\dots)$. We have

$$O'_\Gamma(X,\underline{Y}) = O_{\Gamma\cdot X}\pi^*_{\Gamma,X}\underline{Y} = O_{\Gamma\cdot X}(1,\pi^*_{\Gamma,X}\underline{Y}) \cong O_\Gamma(X,\underline{Y})$$

  natural in Γ, X and $\underline{Y}$. It is easily verified that this isomorphism preserves the isomorphisms for U and F.

□

14.7 CBV is Kleisli, CBN is co-Kleisli

We recall from Sect. 3.7 that

- a CBV producer $A_0,\dots,A_{n-1} \vdash M : B$ translates into the CBPV computation $A_0,\dots,A_{n-1} \vdash^{\mathsf{c}} M : FB$
- a CBN term $\underline{A}_0,\dots,\underline{A}_{n-1} \vdash M : \underline{B}$ translates into the CBPV computation $U\underline{A}_0,\dots,U\underline{A}_{n-1} \vdash^{\mathsf{c}} M : \underline{B}$

As we shall see below, this means that CBV is interpreted in the *Kleisli part* of a CBPV adjunction model, while CBN is interpreted in the *co-Kleisli part*.

The reader is probably used to thinking of a Kleisli adjunction as generated from a monad, and a co-Kleisli adjunction as generated from a comonad. We use a different, but equivalent, formulation.

Definition 114 1. An adjunction between ordinary categories (or an adjunction between locally $\mathcal{C}$-indexed categories or a strong adjunction) is *Kleisli* when its left adjoint on objects is identity. We customarily write the right adjoint (on objects) of a Kleisli adjunction as T rather than U.

2. An adjunction between ordinary categories (or an adjunction between locally $\mathcal{C}$-indexed categories) is *co-Kleisli* when its right adjoint (on objects) is identity. We customarily write the left adjoint on objects of a co-Kleisli adjunction as L rather than F.

□

An example of a Kleisli adjunction is the adjunction model for erratic choice given in Sect. 14.4.6.

Definition 115 1. Given an adjunction $(\mathcal{B},\mathcal{D},O,\dots)$ between ordinary categories—in the sense of Def. 110(5)—its *Kleisli part* is the Kleisli adjunction $(\mathcal{C},\mathcal{D}^{\mathsf{Kl}},O^{\mathsf{Kl}})$ obtained by restricting the objects of $\mathcal{D}$ to the family $\{FA\}_{A\in \mathsf{ob}\ \mathcal{B}}$. Explicitly,

$$\begin{aligned} \mathsf{ob}\ \mathcal{D}^{\mathsf{Kl}} &= \mathsf{ob}\ \mathcal{C} \\ \mathcal{D}^{\mathsf{Kl}}(X,Y) &= \mathcal{D}(FX,FY) \\ O^{\mathsf{Kl}}(X,Y) &= O(X,FY) \end{aligned}$$

Its right adjoint is given (on objects) by $T = UF$ with the evident isomorphisms for left and right adjoints. This is clearly a sub-adjunction of the adjunction $(\mathcal{B},\mathcal{D},O,\dots)$.

We similarly define the Kleisli part of an adjunction between locally $\mathcal{C}$-indexed categories, and the Kleisli part of a strong adjunction.

2. Given an adjunction $(\mathcal{B}, \mathcal{D}, O, \dots)$ between ordinary categories—in the sense of Def. 110(5)—its *co-Kleisli part* is the co-Kleisli adjunction $(\mathcal{B}^{\mathsf{coKl}}, \mathcal{D}, O^{\mathsf{coKl}}$ obtained by restricting the $\mathcal{B}$-objects to the family $\{U\underline{B}\}_{\underline{B} \in \mathsf{ob}\ \mathcal{D}}$. Explicitly,

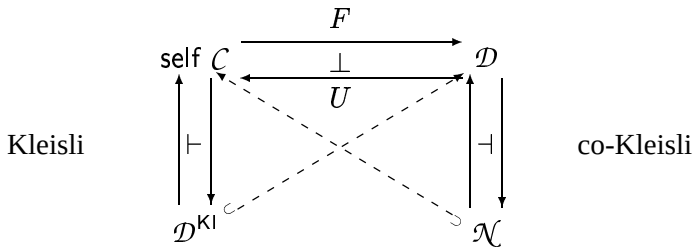
$$\begin{aligned} \mathsf{ob}\ \mathcal{B}^{\mathsf{coKl}} &= \mathsf{ob}\ \mathcal{D} \\ \mathcal{B}^{\mathsf{coKl}}(\underline{X}, \underline{Y}) &= \mathcal{B}(U\underline{X}, U\underline{Y}) \\ O^{\mathsf{coKl}}(\underline{X}, \underline{Y}) &= O(U\underline{X}, \underline{Y}) \end{aligned}$$

   Its left adjoint is given (on objects) by $L = FU$ with the evident isomorphisms for left and right adjoints. This is clearly a sub-adjunction of the adjunction $(\mathcal{B}, \mathcal{D}, O, \dots)$.

   We similarly define the co-Kleisli part of an adjunction between locally $\mathcal{C}$-indexed categories, but not of a strong adjunction.

□

If we take a CBPV adjunction model, we have the following diagram of locally $\mathcal{C}$-indexed adjunctions and embeddings:



It is worth considering these various categories and adjunctions in the setting of, say, the Scott model:

- $\mathcal{C}$ is the category of cpos and continuous functions;
- $\mathcal{D}$ is the **Cpo**-indexed category of cppos and strict continuous functions;
- $\mathcal{D}^{\mathsf{Kl}}$ is an indexed form of the category of cpos and partial continuous functions, used to interpret CBV;
- $\mathcal{N}$ is the **Cpo**-indexed category of cppos and continuous functions, whose fibre over 1 is used to interpret CBN.

In this example, the CBPV adjunction model along the top is Eilenberg-Moore, but this, of course, will not be the case in general.

With this example in mind, our main result is unsurprising:

Proposition 135 Let $(\mathcal{C}, \mathcal{D}, O, \dots)$ be an adjunction model.

1. The denotation of a CBV producer is an oblique morphism in the Kleisli part.
2. The denotation of a CBN term is (more accurately, it corresponds to) an oblique morphism over 1 in the co-Kleisli part.

□

Proof

1. A CBV producer $\Gamma \vdash M : B$ denotes $\llbracket \Gamma \rrbracket \xrightarrow{\llbracket M \rrbracket^{\mathsf{prod}}} F\llbracket B \rrbracket$, which is an oblique morphism over $\llbracket \Gamma \rrbracket$ to $\llbracket B \rrbracket$ in the Kleisli part.

2. A CBN term $\underline{A}_0, \ldots, \underline{A}_{n-1} \vdash M : \underline{B}$ denotes $\xrightarrow{\llbracket M \rrbracket} \llbracket \underline{B} \rrbracket$. We have

$$U\llbracket \underline{A}_0 \rrbracket \times \cdots \times U\llbracket \underline{A}_{n-1} \rrbracket \cong 1 \times U(\llbracket \underline{A}_0 \rrbracket \sqcap \cdots \sqcap \llbracket \underline{A}_{n-1} \rrbracket)$$

so $\llbracket M \rrbracket$ corresponds to an oblique morphism from $\llbracket \underline{A}_0 \rrbracket \sqcap \cdots \sqcap \llbracket \underline{A}_{n-1} \rrbracket$ to $\llbracket \underline{B} \rrbracket$ over 1 in the co-Kleisli part.

□

14.8 Staggered Adjunction Models

14.8.1 Adjunction Models Contain Superfluous Data

In Sect. 14.3.2, we considered the semantics of CBPV in an adjunction model $(\mathcal{C}, \mathcal{D}, O, \ldots)$. We said that a term of CBPV denotes either a $\mathcal{C}$-morphism (if it is a value) or an oblique morphism (if it is a computation): therefore, no term denotes a $\mathcal{D}$-morphism. But $\mathcal{D}$ is useful, because it helps us to organize the model.

We ask the question: which morphisms of $\mathcal{D}$ are genuinely involved in the semantics of CBPV? Inspection of the semantic equations[1] reveals the answer: only those whose source is of the form FA. From the viewpoint of CBPV, all the other homsets of $\mathcal{D}$ are superfluous. This suggests that perhaps

1. some model of CBPV does not arise from any CBPV adjunction model;
2. some model of CBPV arises from 2 non-equivalent CBPV adjunction models.

While (1) is actually false (as is evident from Fig. 11.1), (2) is true.

To remedy this situation, we will modify Def. 106 so that only the appropriate homsets are included. Unfortunately, the resulting structure, *staggered adjunction model*, is not at all elegant. It is not clear at present how significant it is.

14.8.2 Removing The Superfluous Data Gives Staggered Adjunction Models

Our problem is that removing the superfluous homsets from $\mathcal{D}$ in Def. 106 takes us outside the realm of category theory, because, for each Γ, if $\mathcal{D}_\Gamma$ is to be a category then it must specify a homset $\mathcal{D}_\Gamma(\underline{A}, \underline{B})$ for *all* objects $\underline{A}$ and $\underline{B}$. We thus require a generalization of "category" where the homset $\mathcal{B}(\underline{A}, \underline{B})$ is required only for $\underline{A}$ in a certain family $\{FA\}_{A \in \mathfrak{I}}$ of objects. In the following definition, we use the notation ${}^F A$ to mean "technically the index A, but intuitively the object FA". The intuition is not quite accurate because if $FA = FA'$ but $A \neq A'$ then $\mathcal{D}({}^F A, \underline{B})$ and $\mathcal{D}({}^F A', \underline{B})$ can be different; but in practice F is usually injective.

Definition 116 A *staggered category* $\mathcal{D}$ consists of

- a class $\mathsf{targetob}\ \mathcal{D}$ of *target objects* (whose objects we underline)
- a family $\{FA\}_{A \in \mathfrak{I}}$ in $\mathsf{targetob}\ \mathcal{D}$, indexed by some class $\mathfrak{I}$—this is called the *source family*
- for each $A \in \mathfrak{I}$ and each $\underline{B} \in \mathsf{targetob}\ \mathcal{D}$, a set written $\mathcal{D}({}^F A, \underline{B})$—its elements are called *morphisms from ${}^F A$ to $\underline{B}$*
- for each $A \in \mathfrak{I}$ an identity morphism ${}^F A \xrightarrow{\mathsf{id}_{FA}} FA$

[1] and of the proof of Prop. 129

- for each ${}^F A \xrightarrow{f} FB$ and ${}^F B \xrightarrow{g} \underline{C}$ a composite morphism ${}^F A \xrightarrow{f;g} \underline{C}$ also written as

$$ {}^F A \xrightarrow{f} {}^F B \xrightarrow{g} \underline{C} $$

satisfying identity and associativity laws:

$$\begin{aligned} \mathsf{id}; f &= f \\ f; \mathsf{id} &= f \\ (f;g);h &= f;(g;h) \end{aligned}$$

□

Technically, Def. 116 and Def. 101 are equivalent, but the notation and terminology is different because the purpose is different.

The required definition, which involves terminology we have not yet defined, is as follows:

Definition 117 A *CBPV staggered adjunction model* consists of

- a countably distributive category $\mathcal{C}$;
- a countably closed, locally $\mathcal{C}$-indexed staggered category $\mathcal{D}$, whose source family is indexed by $\mathsf{ob}\ \mathcal{C}$;
- a staggered strong adjunction from $\mathcal{C}$ to $\mathcal{D}$.

□

The only difference between Def. 106 and Def. 117 is that in the latter $\mathcal{D}$ is staggered. We must now define, step by step, the terminology used in Def. 117.

Definition 118 (cf. Def. 75) Let $\mathcal{C}$ be a category. A *locally $\mathcal{C}$-indexed staggered category $\mathcal{D}$* consists of

- a class $\mathsf{targetob}\ \mathcal{D}$ of *target objects* (whose objects we underline)
- a family $\{FA\}_{A \in \mathfrak{I}}$ in $\mathsf{targetob}\ \mathcal{D}$, indexed by some class $\mathfrak{I}$—this is called the *source family*
- for each $\Gamma \in \mathsf{ob}\ \mathcal{C}$, each $A \in \mathfrak{I}$ and each $\underline{B} \in \mathsf{targetob}\ \mathcal{D}$, a set written $\mathcal{D}_\Gamma({}^F A, \underline{B})$—its elements are called *morphisms from ${}^F A$ to $\underline{B}$ over Γ*
- for each $A \in \mathfrak{I}$, an identity morphism ${}^F A \xrightarrow[\Gamma]{\mathsf{id}_{FA}} FA$
- for each ${}^F A \xrightarrow[\Gamma]{f} FB$ and ${}^F B \xrightarrow[\Gamma]{g} \underline{C}$, a composite morphism ${}^F A \xrightarrow[\Gamma]{f;g} \underline{C}$
- for each ${}^F A \xrightarrow[\Gamma]{f} \underline{B}$ and each $\mathcal{C}$-morphism $\Gamma' \xrightarrow{k} \Gamma$, a reindexed morphism ${}^F A \xrightarrow[\Gamma']{k^* f} \underline{B}$

such that

$$\begin{array}{rcl} \mathsf{id};f & = & f \\ f;\mathsf{id} & = & f \\ (f;g);h & = & f;(g;h) \\ k^*\mathsf{id} & = & \mathsf{id} \\ k^*(f;g) & = & (k^*f);(k^*g) \end{array}$$

□

We first have to define products and exponents in a locally $\mathcal{C}$-indexed staggered category $\mathcal{D}$. This can only be done in isomorphism style—there is no element style definition.

Definition 119 Let $\mathcal{D}$ be a locally $\mathcal{C}$-indexed staggered category.

1. (cf. Def. 86(2)) A *product* for a family of target objects $\{\underline{B}_i\}_{i\in I}$ consists of a target object $\underline{V}$ (the *vertex*) together with an isomorphism

$$\textstyle\prod_{i\in I}\mathcal{D}_\Gamma({}^F X,\underline{B}_i) \cong \mathcal{D}_\Gamma({}^F X,\underline{V}) \tag{14.10}$$

   natural in Γ and ${}^F X$, in the sense that the diagram

$$\begin{array}{ccc} \prod_{i\in I}\mathcal{D}_\Gamma({}^F X,\underline{B}_i) & \cong & \mathcal{D}_\Gamma({}^F X,\underline{V}) \\ \Big\downarrow{\scriptstyle \prod_{i\in I}\mathcal{D}_k(f,\underline{B}_i)} & & \Big\downarrow{\scriptstyle \mathcal{D}_k(f,\underline{V})} \\ \prod_{i\in I}\mathcal{D}_{\Gamma'}({}^F X',\underline{B}_i) & \cong & \mathcal{D}_{\Gamma'}({}^F X',\underline{V}) \end{array} \qquad \text{commutes for each} \qquad \begin{array}{ccc} \Gamma & & FX \\ \Big\uparrow{\scriptstyle k} & & \Big\uparrow{\scriptstyle f} \\ \Gamma' & & {}^F X' \end{array}$$

2. (cf. Def. 86(4)) Suppose $\mathcal{C}$ is cartesian and A is a $\mathcal{C}$-object. Then an A-product for a target object $\underline{B}$ consists of a target object $\underline{V}$ (the *vertex*) together with an isomorphism

$$\mathcal{D}_{\Gamma\times A}(\pi^*_{\Gamma,A}{}^F X,\underline{B}) \cong \mathcal{D}_\Gamma({}^F X,\underline{V}) \tag{14.11}$$

   natural in Γ and ${}^F X$, in the sense that the diagram

$$\begin{array}{ccc} \mathcal{D}_{\Gamma\times A}({}^F X,\underline{B}) & \cong & \mathcal{D}_\Gamma({}^F X,\underline{V}) \\ \Big\downarrow{\scriptstyle \mathcal{D}_{k\times A}(\pi^*_{\Gamma,A}f,\underline{B})} & & \Big\downarrow{\scriptstyle \mathcal{D}_k(f,\underline{V})} \\ \mathcal{D}_{\Gamma'\times A}({}^F X',\underline{B}) & \cong & \mathcal{D}_\Gamma({}^F X',\underline{V}) \end{array} \qquad \text{commutes for each} \qquad \begin{array}{ccc} \Gamma & & FX \\ \Big\uparrow{\scriptstyle k} & & \Big\uparrow{\scriptstyle f} \\ \Gamma' & & {}^F X' \end{array}$$

3. (cf. Def. 87 Suppose $\mathcal{C}$ is cartesian. $\mathcal{D}$ is *countably closed* when it is equipped with all countable products and A-products, for every $A \in \mathsf{ob}\ \mathcal{C}$.

□

It is easy to see that the joint naturality condition for (14.10) is equivalent to separate naturality in Γ and ${}^F X$. Similarly for (14.11).

Definition 120 (cf. Def. 105) Let $\mathcal{C}$ be a cartesian category and let $\mathcal{D}$ be a locally $\mathcal{C}$-indexed staggered category with source family indexed by $\mathsf{ob}\ \mathcal{C}$. A *staggered strong adjunction* from $\mathcal{C}$ to $\mathcal{D}$ consists of

- "a functor O from $\mathsf{opGroth}\ \mathcal{D}$ to **Set**" i.e. the following structure
  - for each $\Gamma \in \mathsf{ob}\ \mathcal{C}$ and $\underline{B} \in \mathsf{targetob}\ \mathcal{D}$ a set $O_\Gamma \underline{B}$, whose elements we call *oblique morphisms* over Γ to $\underline{B}$;
  - for each oblique morphism $\xrightarrow[\Gamma]{g} \underline{Y}$ and $\mathcal{C}$-morphism $\Gamma' \xrightarrow{k} \Gamma$, a reindexed oblique morphism $\xrightarrow[\Gamma]{k^*g} \underline{Y}$;
  - for each oblique morphism $\xrightarrow[\Gamma]{g} FY$ and $\mathcal{D}$-morphism ${}^F Y \xrightarrow[\Gamma]{h} \underline{Y'}$, a composite oblique morphism $\xrightarrow[\Gamma]{g;h} \underline{Y'}$

  satisfying identity, associativity and reindexing laws:

$$\begin{aligned} g;\mathsf{id} &= g \\ g;(h;h') &= (g;h);h' \\ \mathsf{id}^* g &= g \\ (k';k)^* g &= k'^*(k^* g) \\ k^*(g;h) &= (k^* g);(k^* h) \end{aligned}$$

  where g is an oblique morphism;

- for each $\underline{B} \in \mathsf{targetob}\ \mathcal{D}$, a representation for the functor $\lambda\Gamma.O_\Gamma\underline{B}$ from $\mathcal{C}^{\mathsf{op}}$ to **Set**, whose vertex we call $U\underline{B}$;

- for each $A \in \mathsf{ob}\ \mathcal{C}$, an "$A$-representation for the functor O" i.e. the following structure

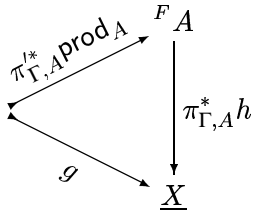
  isomorphism style an isomorphism

$$O_{\Gamma\times A}\pi^*_{\Gamma,A}\underline{Y} \xrightarrow[\cong]{\mathsf{str}_{\Gamma A \underline{Y}}} \mathcal{D}_\Gamma({}^F A, \underline{Y}) \tag{14.12}$$

  jointly natural in Γ and $\underline{Y}$ in the sense that the diagram

$$\begin{array}{ccc} O_{\Gamma\times A}FY & \cong & \mathcal{D}_\Gamma({}^F A, FY) \\ \downarrow{\scriptstyle O_{k\times A}\pi^*_{\Gamma,A}h} & & \downarrow{\scriptstyle \mathcal{D}_k({}^F A, h)} \\ O_{\Gamma'\times A}\underline{Y'} & \cong & \mathcal{D}_{\Gamma'}({}^F A, \underline{Y'}) \end{array} \qquad \text{commutes for each} \qquad \begin{array}{cc} \Gamma & {}^F Y \\ \uparrow{\scriptstyle k} & \downarrow{\scriptstyle h} \\ \Gamma' & \underline{Y'} \end{array}$$

  element style an oblique morphism $\xrightarrow[A]{\mathsf{prod}_A} FA$ with the following "initiality" property: for any $\xrightarrow[\Gamma\times A]{g} \underline{X}$ there is a unique ${}^F A \xrightarrow[\Gamma]{h} \underline{X}$ such that the diagram



  over $\Gamma \times A$ commutes.

□

As we saw in Sect. 14.5 for adjunction models, the isomorphism style characterization and the element style characterization of the structure for FA are equivalent:

- prod_A is obtained by applying str^{-1} to the $\mathcal{D}$-morphism ${}^F A \xrightarrow[1]{\mathsf{id}} FA$, giving an oblique morphism over $1 \times A$ to FA, and then reindexing along the isomorphism $A \xrightarrow{i} 1 \times A$.
- for a $\mathcal{D}$-morphism ${}^F A \xrightarrow[\Gamma]{h} \underline{B}$ we have

$$\mathsf{str}^{-1}_{\Gamma A \underline{B}} h = \pi'^{*}_{\Gamma,A} \mathsf{prod}_A ; \pi^{*}_{\Gamma,A} h \tag{14.13}$$

The analogous equivalence in Sect. 14.5 (between isomorphism style and element style definitions) was an instance of the Yoneda Lemma, and we can use essentially the same Yoneda-style argument to prove this equivalence here.

It is a corollary of (14.13) that, for (14.12), joint naturality and separate naturality in Γ and $\underline{Y}$ are equivalent. (We cannot use the usual proof that joint naturality and separate naturality are equivalent, because it does not work for staggered categories.)

Proposition 136 Suppose $(\mathcal{C}, \mathcal{D}, O, \dots)$ is a CBPV adjunction model. We obtain a CBPV staggered adjunction model $(\mathcal{C}, \mathcal{D}', O, \dots)$ by setting $\mathcal{D}'_\Gamma({}^F X, \underline{Y})$ to be $\mathcal{D}_\Gamma(FX, \underline{Y})$. □

This "forgetful" construction shows that a staggered adjunction model is, as we intended, an adjunction model without the superfluous homsets.

We can adapt Sect. 14.5 (replacing FA by ${}^F A$ where required), to interpret CBPV in any staggered adjunction model. Prop. 128, and Prop. 129 apply to staggered adjunction models as well as to adjunction models—the proofs are easily adapted.

14.9 Technical Material

14.9.1 Introduction

The aim of this section is to prove Prop. 128(1) and Prop. 129. Before we do this, we give some lemmas about strong adjunctions that will be used also in Chap. 15.

We shall present the results and proofs in this section for adjunction models. They can all be adapted to staggered adjunction models by replacing FA by ${}^F A$ where appropriate.

14.9.2 Lemmas About Strong Adjunctions

Suppose that we have a strong adjunction $(\mathcal{C}, \mathcal{D}, O, \dots)$. It must have the following properties. In fact, the structure for U is not used at all here, just the structure for F.

Lemma 137

$$\pi'^{*}_{\Gamma,A}\mathsf{prod}_A;\pi^{*}_{\Gamma,A}\mathsf{str}\ g = g \text{ for } \xrightarrow[\Gamma\times A]{g} \underline{B} \tag{14.14}$$

$$\mathsf{str}\ (\pi'^{*}_{\Gamma,A}\mathsf{prod}_A;\pi^{*}_{\Gamma,A}h) = h \text{ for } {}^{F}A \xrightarrow[\Gamma]{h} \underline{B} \tag{14.15}$$

$$\mathsf{str}\ (k\times A)^{*}g = k^{*}\mathsf{str}\ g \text{ for } \Gamma' \xrightarrow{k} \Gamma \text{ and } \xrightarrow[\Gamma\times A]{g} \underline{B} \tag{14.16}$$

$$\mathsf{str}\ (g;\pi^{*}_{\Gamma,A}h) = (\mathsf{str}\ g);h \text{ for } \xrightarrow[\Gamma\times A]{g} \underline{B} \text{ and } \underline{B} \xrightarrow[\Gamma]{h} \underline{B'} \tag{14.17}$$

$$\mathsf{str}\ \pi'^{*}_{\Gamma,A}\mathsf{prod}_A = \mathsf{id}_A \tag{14.18}$$

$$\mathsf{str}\ (f;\pi^{*}_{\Gamma,A}\mathsf{str}\ g) = \mathsf{str}\ f;\mathsf{str}\ g \text{ for } \xrightarrow[\Gamma\times A]{f} FB \text{ and } \xrightarrow[\Gamma\times B]{g} \underline{C} \tag{14.19}$$

$$k^{*}\mathsf{prod}_A;\mathsf{str}\ g = (\mathsf{id}_\Gamma,k)^{*}g \text{ for } \Gamma \xrightarrow{k} A \text{ and } \xrightarrow[\Gamma\times A]{g} \underline{B} \tag{14.20}$$

$$(\mathsf{str}\ k^{*}\mathsf{prod}_B);(\mathsf{str}\ g) = \mathsf{str}\ (\pi_{\Gamma,A},k)^{*}g \text{ for } \Gamma\times A \xrightarrow{k} B \text{ and } \xrightarrow[\Gamma\times B]{g} \underline{C} \tag{14.21}$$

$$\pi'^{*}_{\Gamma,A}\mathsf{str}\ \pi'^{*}_{A,B}g = \pi^{*}_{\Gamma,A}\mathsf{str}\ \pi'^{*}_{\Gamma,B}g \text{ for } \xrightarrow[B]{g} \underline{C} \tag{14.22}$$

□

Proof (14.14)–(14.15) state that str^{-1}, as described in (14.4), is the inverse of str. (14.16) and (14.17) state that $\mathsf{str}_{\Gamma A\underline{B}}$ is natural in Γ and $\underline{B}$ respectively. (14.18) is a special case of (14.15), putting id_A for h. For (14.19) we see that the LHS (by (14.14)) and the RHS (by (14.15)) are both equal to

$$\mathsf{str}\ (\pi'^{*}_{\Gamma,A}\mathsf{prod}_A;\pi^{*}_{\Gamma,A}\mathsf{str}\ f;\pi^{*}_{\Gamma,A}\mathsf{str}\ g)$$

For (14.20), we see that the RHS (by (14.14)) and the LHS are both equal to

$$(\mathsf{id}_\Gamma,k)^{*}(\pi'^{*}_{\Gamma,A}\mathsf{prod}_A;\pi^{*}_{\Gamma,A}\mathsf{str}\ g)$$

For (14.21), we see that

$$\begin{array}{rcl} & & \text{LHS} \\ \text{by (14.19)} & = & \mathsf{str}\ (k^{*}\mathsf{prod}_B;\pi^{*}_{\Gamma,A}\mathsf{str}g) \\ \text{by (14.16)} & = & \mathsf{str}\ (k^{*}\mathsf{prod}_B;\mathsf{str}\ (\pi_{\Gamma,A}\times B)^{*}g) \\ \text{by (14.20)} & = & \mathsf{str}\ ((\mathsf{id}_{\Gamma\times A},k)^{*}(\pi_{\Gamma,A}\times B)^{*}g \\ & = & \text{RHS} \end{array}$$

For (14.22), both sides are equal by (14.16) to $\mathsf{str}\ \pi'^{*}_{\Gamma\times A,B}g$. □

Recall that a *division* of Γ (an object in a cartesian category) is defined to be a triple (X,Y,i), where X and Y are value objects and i is an isomorphism $\Gamma\cong X\times Y$.

Lemma 138 For any division $\Gamma \xrightarrow[\cong]{i} X\times Y$, there is a $\mathcal{C}$-morphism $\Gamma \xrightarrow{k} X$ and a $\mathcal{D}$-morphism $F1 \xrightarrow[\Gamma]{h} FY$ with the following properties:

- for every $\underline{B} \in \mathsf{ob}\ \mathcal{D}$, the diagram

$$\begin{array}{ccc} \mathcal{D}_X(FY, \underline{B}) & \xrightarrow{\mathcal{D}_k(h, \underline{B})} & \mathcal{D}_\Gamma(F1, \underline{B}) \\ \cong \Big\downarrow & & \Big\downarrow \cong \\ \mathcal{O}_{X \times Y} \underline{B} & \xrightarrow[\mathcal{O}_{\pi;i} \underline{B}]{} & \mathcal{O}_{\Gamma \times 1} \underline{B} \end{array} \quad \text{commutes;} \tag{14.23}$$

- for every $\underline{B} \in \mathsf{ob}\ \mathcal{D}$ and $A \in \mathsf{ob}\ \mathcal{D}$, writing i_A for the evident isomorphism from $(\Gamma \times A) \times 1$ to $(X \times A) \times Y$, the composite

$$\begin{array}{ccc} \mathcal{D}_{X \times A}(FY, \underline{B}) & \xrightarrow{\mathcal{D}_{k \times A}(\pi^*_{\Gamma, A} h, \underline{B})} & \mathcal{D}_{\Gamma \times A}(F1, \underline{B}) \\ \cong \Big\downarrow & & \Big\downarrow \cong \\ \mathcal{O}_{(X \times A) \times Y} \underline{B} & \xrightarrow[\mathcal{O}_{i_A} \underline{B}]{} & \mathcal{O}_{(\Gamma \times A) \times 1} \underline{B} \end{array} \quad \text{commutes.} \tag{14.24}$$

□

Proof We set k to be $i; \pi_{X,Y}$ and we set h to be $\mathsf{str}\ (\pi_{\Gamma,1}; i; \pi'_X Y)^* \mathsf{prod}_Y$. We then verify (14.23)–(14.24) by following an arbitrary morphism from top left to bottom right, using (14.4) and Lemma 137. □

Lemma 139 Suppose we have an oblique morphism $\Gamma \xrightarrow[]{f} FC$. Then the following commutes:

$$\begin{array}{ccc} \mathcal{D}_\Gamma(FC, \underline{B}) & \xrightarrow{\cong} & \mathcal{O}_{\Gamma \times C} \underline{B} \\ \mathcal{D}_\Gamma(\mathsf{str}\ \pi^*_{\Gamma,1} f, \underline{B}) \Big\downarrow & & \Big\downarrow f; \mathsf{str}\ - \\ \mathcal{D}_\Gamma(F1, \underline{B}) & \xrightarrow[\cong]{} & \mathcal{O}_\Gamma \underline{B} \end{array}$$

□

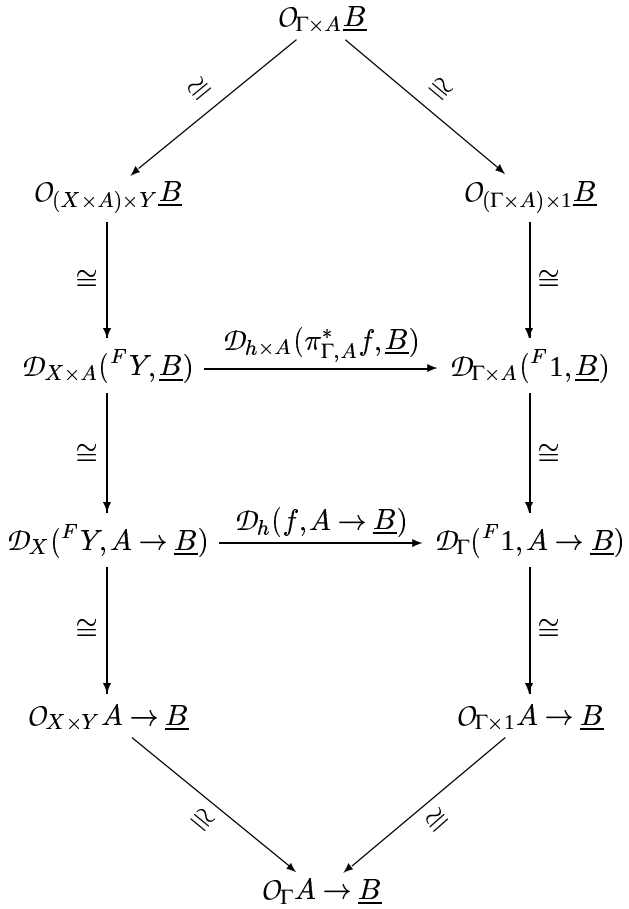
Proof Follow a typical morphism from top left to bottom right. □

14.9.3 Proof of Semantics Theorems

Proof of Prop. 128(1). It is sufficient to show that every division $\Gamma \cong X \times Y$ gives the same isomorphisms (14.6) as the "canonical" division $\Gamma \cong \Gamma \times 1$.

From the division $\Gamma \cong X \times Y$, we obtain a $\mathcal{C}$-morphism $\Gamma \xrightarrow{k} X$ and a $\mathcal{D}$-morphism ${}^F 1 \xrightarrow[\Gamma]{h} FY$ with the properties given in Lemma 138. Then the required result for the

isomorphism (14.6) is given by the diagram



The top and the bottom pentagons commute by the construction of k and h, according to Lemma 138. The middle square commutes because the isomorphism

$$\mathcal{D}_{\Gamma\times A}(\pi^{*}_{\Gamma,A}\underline{X},B)\cong\mathcal{D}_{\Gamma}(\underline{X},\underline{V})\text{ is natural in }\Gamma\text{ and }\underline{X}.$$

□

We prove Prop. 129. *Proof* We first show that substitution is interpreted by reindexing. This is straightforward, using Prop. 128(2).

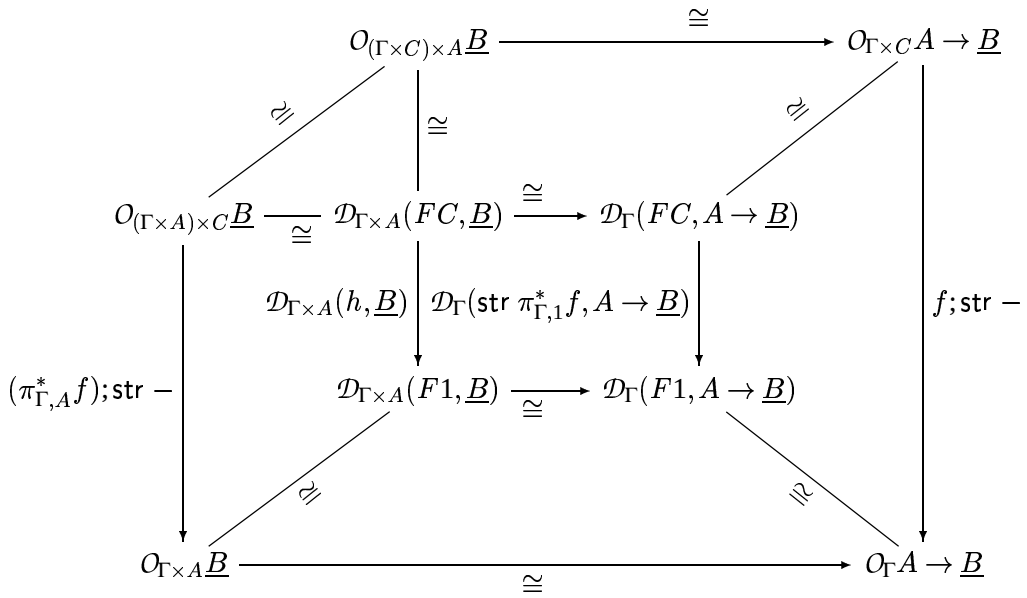
Most of the equations then follow directly from Lemma 137. For example

- the β-law for F follows from (14.20);
- the η-law for F follows from (14.18);
- the associativity law for F follows from (14.19).

To prove the equation

$$M\ \mathtt{to}\ \mathtt{x}.\ \lambda\mathtt{y}.N=\lambda\mathtt{y}.(M\ \mathtt{to}\ \mathtt{x}.\ N)\tag{14.25}$$

we reason as follows. Fix an oblique morphism $\Gamma \xrightarrow{\quad f \quad} FC$. Consider the diagram



The right quadrilateral, like the left, commutes by Lemma 139. The top and bottom quadrilaterals commute by Prop. 128(1), choosing the division of $\Gamma \times A$ into Γ and A. Whether the left quadrilateral and the middle square commute depends on what we choose the morphism $F1 \xrightarrow{\quad h \quad} \Gamma \times A$ FC to be. We want it to be

- $\mathsf{str}\ \pi^*_{\Gamma\times A,1}\pi^*_{\Gamma,A}f$ to make the left quadrilateral commute (by Lemma 139);
- $\pi^*_{\Gamma,A}\mathsf{str}\ \pi^*_{\Gamma,1}f$ to make the middle square commute.

Fortunately, Lemma 137(14.16) tells us that these two morphisms are equal. So we set h to be this morphism and the entire diagram then commutes.

Now, if f is $[\![M]\!]$ and $(\Gamma \times C) \times A \xrightarrow{\quad g \quad} \underline{B}$ is $[\![N]\!]$ then the image of g along the top and right edges is the LHS of (14.25) while the image of g along the left and bottom edges is the RHS of (14.25).

The proof of the equation

$$M\ \mathtt{to}\ \mathtt{x}.\ \lambda\{\ldots,i.N_i,\ldots\} = \lambda\{\ldots,M\ \mathtt{to}\ \mathtt{x}.\ N_i,\ldots\}$$

is similar but easier. □

Chapter 15

Relating Categorical Models

15.1 Introduction

In this chapter, we construct the various relationships depicted in Fig. 11.1. In the first part of the chapter (Sect. 15.2–15.4) we show that direct models, staggered adjunction models and restricted algebra models and value/producer models are all equivalent. The easiest way to do this, as depicted in Fig. 11.1, is to show that value/producer models are equivalent to each of the others. Equivalence does not means that the various models correspond exactly, only that they correspond up to isomorphism, and there are significant differences contained within these isomorphisms:

- A value/producer model corresponds to a staggered adjunction model where the isomorphism

  $$\mathcal{O}_{\Gamma \times A} \underline{Y} \cong \mathcal{D}_{\Gamma}({}^{F}A, \underline{Y})$$

  is identity.

- A restricted algebra model corresponds to a value/producer model where the isomorphism

  $$\mathcal{E}(\Gamma, \underline{B}) \cong \mathcal{C}(\Gamma, U\underline{B})$$

  is identity.

Consequently

- staggered adjunction models provide the most flexibility, which is helpful when constructing particular models of CBPV;
- restricted algebra models are the most rigid, which is helpful when proving results about "all models of CBPV".

We then in Sect. 15.5–15.6 turn to the full reflection between restricted algebra models and adjunction models, depicted in Fig. 11.1. Because of the equivalences, this gives us also a full reflection between

- value/producer models and adjunction models;
- staggered adjunction models and adjunction models;
- direct models and adjunction models.

In particular, this means that in any of the various equivalent models, we have a notion of homomorphism:

Definition 121 A *homomorphism* from $\underline{A}$ to $\underline{B}$ over C is

in a restricted algebra model $(\mathcal{C}, T, \{j\underline{B}\}_{\underline{B}\in\mathfrak{I}}, \dots)$ an algebra homomorphism from $j\underline{A}$ to $j\underline{B}$ over C

in a value/producer model $(\mathcal{C}, \mathcal{E}, \dots)$ a morphism

$$\mathcal{E}(\Gamma, \underline{A}) \longrightarrow \mathcal{E}(\Gamma \times C, \underline{B}) \text{ natural in } \Gamma \in \mathcal{E}_F^{\mathsf{op}}$$

in a staggered adjunction model $(\mathcal{C}, \mathcal{D}, O, \dots)$ a morphism

$$\mathcal{D}_\Gamma({}^F X, \underline{A}) \longrightarrow \mathcal{D}_{\Gamma\times C}(\pi^{*F}_{\Gamma,C} X, \underline{B}) \text{ natural in } \Gamma \text{ and } {}^F X$$

in a direct model (s, θ) a derivation

$$\frac{\Gamma \vdash^{\mathsf{c}} \underline{A}}{\Gamma, C \vdash^{\mathsf{c}} \underline{B}}$$

—i.e. a function $s(\Gamma; \underline{A}) \xrightarrow{\theta_\Gamma} s(\Gamma, C; \underline{B}))$ for each sequence Γ of value objects—that preserves substitution and sequencing in Γ.

□

We will see in Sect. 15.6 that these definitions accurately characterize the homomorphisms in the induced adjunction model.

Def. 121 gives us also a notion of isomorphism in each of the models. For example, we are now able to say that, in any CBPV model, $A \to B \to \underline{C}$ is isomorphic to $(A \times B) \to \underline{C}$—a statement that was previously meaningless. An isomorphism from $\underline{A}$ to $\underline{B}$ is

- in a restricted algebra model $(\mathcal{C}, T, \{j\underline{B}\}_{\underline{B}\in\mathfrak{I}}, \dots)$: an algebra isomorphism from $j\underline{A}$ to $j\underline{B}$;
- in a value/producer model $(\mathcal{C}, \mathcal{E}, \dots)$: an isomorphism

$$\mathcal{E}(\Gamma, \underline{A}) \xrightarrow{\cong} \mathcal{E}(\Gamma, \underline{B}) \text{ natural in } \Gamma \in \mathcal{E}_F^{\mathsf{op}}$$

- in a staggered adjunction model $(\mathcal{C}, \mathcal{D}, O, \dots)$: an isomorphism

$$\mathcal{D}_\Gamma({}^F X, \underline{A}) \xrightarrow{\cong} \mathcal{D}_\Gamma({}^F X, \underline{B}) \text{ natural in } \Gamma \text{ and } {}^F X$$

- in a direct model (s, θ): a reversible derivation

$$\frac{\Gamma \vdash^{\mathsf{c}} \underline{A}}{\Gamma \vdash^{\mathsf{c}} \underline{B}}$$

—i.e. a bijection $s(\Gamma; \underline{A}) \xrightarrow{\theta_\Gamma} s(\Gamma; \underline{B}))$ for each sequence Γ of value objects—that preserves substitution and sequencing in Γ.

This explains Def. 22(2), because the term model of CBPV is an example of a direct model.

We recall that in an adjunction model, a homomorphism ($\mathcal{D}$-morphism) from $\underline{A}$ to $\underline{B}$ over C corresponds, by the Yoneda embedding (Def. 88), to a natural transformation

$$\mathcal{D}_\Gamma(\underline{X}, \underline{A}) \longrightarrow \mathcal{D}_{\Gamma\times C}(\pi^*_{\Gamma,C}\underline{X}, \underline{B}) \text{ natural in } \Gamma \text{ and } \underline{X}$$

Our definition of homomorphism in a staggered adjunction model is a variant of this.

15.2 Equivalence: Direct Models and Value/Producer Models

This equivalence is straightforward to construct, and is along the lines of the proof of Prop. 99. We give it in outline.

- Let $(\mathcal{C}, \mathcal{E}, \dots)$ be a value/producer model based on τ. Then we construct a τ-multigraph s by setting
  - $s_{\mathsf{val}}(A_0, \dots, A_{n-1}; B)$ to be $\mathcal{C}(A_0 \times \cdots \times A_{n-1}, B)$;
  - $s_{\mathsf{comp}}(A_0, \dots, A_{n-1}; \underline{B})$ to be $\mathcal{E}(A_0 \times \cdots \times A_{n-1}, \underline{B})$

  We define a structure θ on s in the evident way, by induction over terms, and verify all the required equations.

- Let (s, θ) be a direct model based on τ. Then we construct a value/producer model by setting
  - $\mathcal{C}(A, B)$ to be $s_{\mathsf{val}}(A; B)$;
  - $\mathcal{E}(A, \underline{B})$ to be $s_{\mathsf{comp}}(A; \underline{B})$.

  The structure is defined in the evident way and all required equations verified.

- Let $(\mathcal{C}, \mathcal{E}, \dots)$ be a value/producer model. Obtain a direct model (s, θ) and then a value/producer model $(\mathcal{C}', \mathcal{E}', \dots)$ as above. We see that

$$\begin{aligned}
\mathcal{C}'(A, B) &= s_{\mathsf{val}}(A, B) \\
&= \mathcal{C}(1 \times A, B) \\
&\cong \mathcal{C}(A, B) \\
\mathcal{E}'(A, \underline{B}) &= s_{\mathsf{comp}}(A, \underline{B}) \\
&= \mathcal{E}(1 \times A, \underline{B}) \\
&\cong \mathcal{E}(A, \underline{B})
\end{aligned}$$

  This bijection can be shown to preserve all structure, as required.

- Let (s, θ) be a direct model. Obtain a value/producer model $(\mathcal{C}, \mathcal{E}, \dots)$ and then a direct model (s', θ') as above. We see that

$$\begin{aligned}
s'_{\mathsf{val}}(A_0, \dots, A_{n-1}; B) &= \mathcal{C}(A_0 \times \cdots \times A_{n-1}, B) \\
&= s_{\mathsf{val}}(A_0 \times \cdots \times A_{n-1}; B) \\
&\cong s_{\mathsf{val}}(A_0, \dots, A_{n-1}; B) \\
s'_{\mathsf{comp}}(A_0, \dots, A_{n-1}; \underline{B}) &= \mathcal{E}(A_0 \times \cdots \times A_{n-1}, \underline{B}) \\
&= s_{\mathsf{comp}}(A_0 \times \cdots \times A_{n-1}; \underline{B}) \\
&\cong s_{\mathsf{comp}}(A_0, \dots, A_{n-1}; \underline{B})
\end{aligned}$$

  This bijection is easily constructed, and shown to be structure preserving.

15.3 Equivalence: Value/Producer Models and Staggered Adjunction Models

15.3.1 From Value/Producer Model To Staggered Adjunction Model

Given a value/producer model $(\mathcal{C}, \mathcal{E}, \dots)$, we construct a staggered adjunction model by setting

- $O_\Gamma \underline{B}$ to be $\mathcal{E}(\Gamma, \underline{B})$;
- $\mathcal{D}_\Gamma({}^F A, \underline{B})$ to be $\mathcal{E}(\Gamma \times A, \underline{B})$.

with identities, composition and reindexing defined as follows.

- The $\mathcal{D}$-morphism $\mathsf{id}_{{}^F A}$ over Γ is given as $\Gamma \times A \xrightarrow{\iota\pi'} A$.
- For $\mathcal{D}$-morphisms ${}^F A \xrightarrow[\Gamma]{f} FB$ and ${}^F B \xrightarrow[\Gamma]{g} \underline{C}$, the composite $f;g$ is given as

$$\Gamma \times A \xrightarrow{\iota(\pi, \mathsf{id})} \Gamma \times (\Gamma \times A) \xrightarrow{\Gamma \times f} \Gamma \times B \xrightarrow{g} \underline{C}$$

- For $\mathcal{C}$-morphism $\Gamma' \xrightarrow{k} \Gamma$ and $\mathcal{D}$-morphism ${}^F A \xrightarrow[\Gamma]{h} \underline{B}$, the $\mathcal{D}$-morphism $k^* h$ is defined to be

$$\Gamma' \times A \xrightarrow{\iota(k \times A)} \Gamma \times A \xrightarrow{h} \underline{B}$$

- For oblique morphism $\xrightarrow[\Gamma]{g} FB$ and $\mathcal{D}$-morphism ${}^F B \xrightarrow[\Gamma]{h} \underline{C}$, the oblique morphism $g;h$ is given by

$$\Gamma \xrightarrow{\iota(\mathsf{id}, \mathsf{id})} \Gamma \times \Gamma \xrightarrow{\Gamma \times g} \Gamma \times B \xrightarrow{h} \underline{C}$$

- For $\mathcal{C}$-morphism $\Gamma' \xrightarrow{k} \Gamma$ and oblique morphism $\xrightarrow[\Gamma]{g} \underline{B}$, the oblique morphism $k^* g$ is given by

$$\Gamma' \xrightarrow{\iota k} \Gamma \xrightarrow{g} \underline{B}$$

We set the isomorphism

$$O_{\Gamma \times A} \pi^*_{\Gamma, A} \underline{Y} \cong \mathcal{D}_\Gamma({}^F A, \underline{Y})$$

to be the identity, as the two sides are the same. It follows that prod_A is given as id_A in $\mathcal{E}$.

The structure for U, $\prod$ and $\to$ in the staggered adjunction model is exactly given by the structure for U,$\prod$ and $\to$ (respectively) in the value/producer model.

The verification of equations is entirely straightforward.

15.3.2 From Staggered Adjunction Model to Value/Producer Model

Given a staggered adjunction model $(\mathcal{C}, \mathcal{D}, O, \dots)$, we want to obtain a value/producer model.

We construct a staggered category $\mathcal{E}$ as follows.

- We set $\mathcal{E}(A, \underline{B})$ to be $O_A \underline{B}$.
- $A \xrightarrow{\mathsf{id}_A} FA$ is defined to be prod_A.

- Given $A \xrightarrow{\quad f \quad} FB$ and $B \xrightarrow{\quad g \quad} \underline{C}$, the composite $f;g$ is defined to be

$$\xrightarrow[A]{f} {}^{F}B \xrightarrow[A]{\mathsf{str}\ \pi'^{*}_{A,B}g} FC$$

Associativity and identity laws are easily deduced from Lemma 137.

Given a $\mathcal{C}$-morphism $A \xrightarrow{\quad f \quad} B$, the $\mathcal{E}$-morphism $A \xrightarrow{\quad \iota f \quad} FB$ is given as $f^{*}\mathsf{prod}_{B}$.

Given $X \in \mathsf{ob}\ \mathcal{C}$ and a $\mathcal{E}$-morphism $A \xrightarrow{\quad f \quad} FB$, the $\mathcal{E}$-morphism $X \times A \xrightarrow{X \times f} F(X \times B)$ is defined to be

$$\xrightarrow[X \times A]{\pi'^{*}_{X,A}f} {}^{F}B \xrightarrow[X \times A]{\pi^{*}_{X,A}\mathsf{str}\ \mathsf{prod}_{X \times B}} F(X \times B)$$

To be sure that we have described a value/producer structure from $\mathcal{C}$ to $\mathcal{E}_F$, we must verify the equations listed in Prop. 125. As an example—the most difficult, in fact—we must prove

$$X \times (f;g) = (X \times f);(X \times g)$$

for $A \xrightarrow{\quad f \quad} FB$ and $B \xrightarrow{\quad g \quad} FC$. Here is a proof:

$$\begin{array}{rcl}
\text{RHS} & = & \pi'^{*}_{X,A}f;\pi^{*}_{X,A}\mathsf{str}\ \mathsf{prod}_{X \times B};\mathsf{str}\ \pi'^{*}_{X \times A,X \times B}(\pi'^{*}_{X,B}g;\pi^{*}_{X,B}\mathsf{str}\ \mathsf{prod}_{X \times C}) \\
\text{by (14.16)} & = & \pi'^{*}_{X,A}f;\mathsf{str}\ (\pi_{X,A} \times B)^{*}\mathsf{prod}_{X \times B};\mathsf{str}\ \pi'^{*}_{X \times A,X \times B}(\pi'^{*}_{X,B}g;\pi^{*}_{X,B}\mathsf{str}\ \mathsf{prod}_{X \times C}) \\
\text{by (14.21)} & = & \pi'^{*}_{X,A}f;\mathsf{str}\ (\pi_{X,A} \times B)^{*}(\pi'^{*}_{X,B}g;\pi^{*}_{X,B}\mathsf{str}\ \mathsf{prod}_{X \times C}) \\
\text{by (14.16)} & = & \pi'^{*}_{X,A}f;\pi^{*}_{X,A}\mathsf{str}\ (\pi'^{*}_{X,B}g;\pi^{*}_{X,B}\mathsf{str}\ \mathsf{prod}_{X \times C}) \\
\text{by (14.19)} & = & \pi'^{*}_{X,A}f;\pi^{*}_{X,A}(\mathsf{str}\ \pi'^{*}_{X,B}g;\mathsf{str}\ \mathsf{prod}_{X \times C}) \\
\text{by (14.22)} & = & \pi'^{*}_{X,A}(f;\mathsf{str}\ \pi'^{*}_{A,B}g);\pi^{*}_{X,A}\mathsf{str}\ \mathsf{prod}_{X \times C} \\
 & = & \text{LHS}
\end{array}$$

The structure for U, $\to$ and $\prod$ is straightforward, but to prove the required naturality conditions we need some lemmas.

Lemma 140 Given $\Gamma' \xrightarrow{\quad f \quad} \Gamma$ in $\mathcal{C}$ and $\Gamma \xrightarrow{\quad f \quad} \underline{B}$ in $\mathcal{E}$, the composite $(\iota f);g$ in $\mathcal{E}$ is the oblique morphism $f^{*}g$. □

Lemma 141 Let $\Gamma' \xrightarrow{\quad g \quad} F\Gamma$ be a producer in $\mathcal{E}$. Then

1. $g \times A$, for any value object A is given by

$$\xrightarrow[\Gamma' \times A]{\pi^{*}_{\Gamma',A}g} {}^{F}\Gamma \xrightarrow[\Gamma' \times A]{\mathsf{str}\ (\pi'_{\Gamma' \times A,\Gamma},(\pi_{\Gamma' \times A,\Gamma};\pi'^{*}_{\Gamma',A}\mathsf{prod}_{\Gamma \times A}} F(\Gamma \times A)$$

2. $\mathcal{E}(g,\underline{B})$, for any computation object $\underline{B}$, is given by

$$\begin{array}{ccc}
O_{\Gamma}\underline{B} & \xrightarrow{\ \cong\ } & \mathcal{D}_1({}^{F}\Gamma,\underline{B}) \\
 & & \Big\downarrow{\scriptstyle \mathcal{D}_1(\mathsf{str}\ \pi'^{*}_{1,\Gamma'}g,\underline{B})} \\
O_{\Gamma'}\underline{B} & \xleftarrow{\ \cong\ } & \mathcal{D}_1({}^{F}\Gamma',\underline{B})
\end{array}$$

3. $\mathcal{E}(g \times A, \underline{B})$, for any value object A and computation object $\underline{B}$, is given by

$$\begin{array}{ccc} \mathcal{O}_{\Gamma \times A}\underline{B} & \xrightarrow{\cong} & \mathcal{D}_{1\times A}({}^{F}\Gamma, \underline{B}) \\ & & \downarrow{\scriptstyle \mathcal{D}_{1\times A}(\mathsf{str}\ \pi'^{*}_{1\times A,\Gamma'} g, \underline{B})} \\ \mathcal{O}_{\Gamma' \times A}\underline{B} & \xleftarrow{\cong} & \mathcal{D}_{1\times A}({}^{F}\Gamma', \underline{B}) \end{array}$$

□

Proof (1) is straightforward. For (2)–(3), follow a typical morphism around, and use Lemma 137. □

For U, we require an isomorphism

$$\mathcal{E}(\iota\Gamma, \underline{B}) \cong \mathcal{C}(\Gamma, U\underline{B}) \text{ natural in } \Gamma \in \mathcal{C}^{\mathsf{op}}$$

This is precisely the isomorphism

$$\mathcal{O}_\Gamma \underline{B} \cong \mathcal{C}(\Gamma, U\underline{B}) \text{ natural in } \Gamma \in \mathcal{C}^{\mathsf{op}}$$

Lemma 140 shows that the naturality conditions are the same.

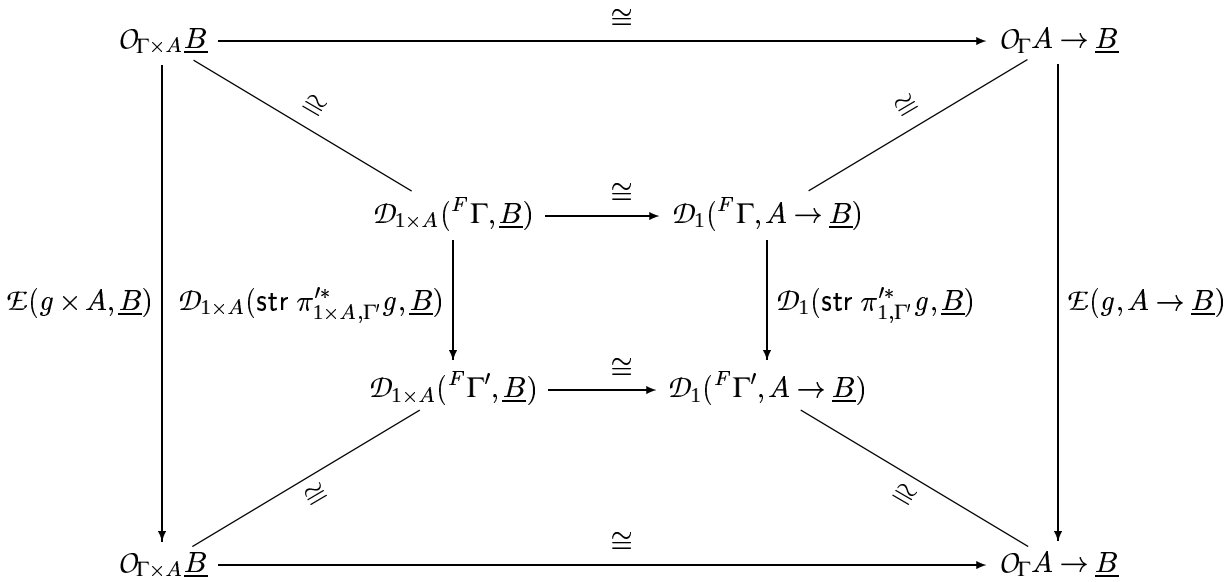
For $\to$, we require an isomorphism

$$\mathcal{E}(\Gamma \times A, \underline{B}) \cong \mathcal{E}(\Gamma, A \to \underline{B}) \text{ natural in } \Gamma \in \mathcal{E}_F^{\mathsf{op}}$$

This is precisely the isomorphism

$$\mathcal{O}_{\Gamma \times A}\underline{B} \cong \mathcal{O}_\Gamma A \to \underline{B}$$

constructed in Sect. 14.5. To prove the required naturality in $\Gamma \in \mathcal{E}_F^{\mathsf{op}}$, let $\Gamma' \xrightarrow{\ g\ } F\Gamma$ be a producer in $\mathcal{E}$. Then the diagram



commutes as required: the left and right quadrilaterals by Lemma 141, the top and bottom quadrilaterals by Prop. 128(1), using the division of Γ into 1 and Γ and the division of Γ' into 1 and Γ', and the central square by the naturality of the closure isomorphism and the equation

$$\mathsf{str}\ \pi'^{*}_{1\times A,\Gamma'} g = \pi^{*}_{1,A} \mathsf{str}\ \pi'^{*}_{1,\Gamma'} g$$

The required structure for $\prod$ is constructed similarly.

Proposition 142 The semantics of CBPV in the staggered adjunction model $(\mathcal{C}, \mathcal{D}, O, \dots)$, as described in Chap. 14, is precisely that obtained, as in Chap. 13, from the value/producer model $(\mathcal{C}, \mathcal{E}, \dots)$ that we have just constructed. □

15.3.3 Value/Producer to Staggered Adjunction to Value/Producer

Given a value/producer model $(\mathcal{C}, \mathcal{E}, \dots)$, first obtain a staggered adjunction model $(\mathcal{C}, \mathcal{D}, O, \dots)$ and then obtain another model $(\mathcal{C}, \mathcal{E}', \dots)$. We see that

$$\mathcal{E}'(A, \underline{B}) = \mathcal{E}(A, \underline{B})$$

so we set the identity morphism to be the required isomorphism from the original value/producer model to the new value/producer model. Of course, we must verify that all the structure is the same, so that this isomorphism is structure preserving, but this verification is entirely straightforward.

15.3.4 Staggered Adjunction to Value/Producer to Staggered Adjunction

Given a staggered adjunction model $(\mathcal{C}, \mathcal{D}, O, \dots)$, first obtain a value/producer model $(\mathcal{C}, \mathcal{E}, \dots)$ and then obtain another staggered adjunction model $(\mathcal{C}, \mathcal{D}', O', \dots)$. We see that

$$\begin{array}{rcccl} O'_\Gamma \underline{B} & = & O_\Gamma \underline{B} & & \\ \mathcal{D}'_\Gamma({}^F A, \underline{B}) & = & O_{\Gamma \times A} \underline{B} & \cong & \mathcal{D}_\Gamma({}^F A, \underline{B}) \end{array}$$

This gives the required isomorphism between the old and the new model. We must prove that this isomorphism is structure preserving, and this is straightforward with the help of Lemma 137.

15.4 Equivalence: Restricted Algebra Models and Value/Producer Models

15.4.1 From Restricted Algebra Model To Value/Producer Model

Given a restricted algebra model $(\mathcal{C}, T, \{j\underline{B}\}_{\underline{B} \in \mathcal{I}}, \dots)$, we want to obtain a value/producer model.

We construct a staggered category $\mathcal{E}$ as follows. Clearly $\mathsf{sourceob}\ \mathcal{E} = \mathsf{ob}\ \mathcal{C}$ and we set $\mathsf{targetob}\ \mathcal{E} = \mathcal{I}$. We define the homset $\mathcal{E}(A, \underline{B})$ to be $\mathcal{C}(A, U\underline{B})$. In particular, a producer from A to B will be a $\mathcal{C}$-morphism from A to $UFB = TB$.

- The $\mathcal{E}$-morphism id_A is given as ηA.
- For $\mathcal{E}$-morphisms $A \xrightarrow{\quad f \quad} B$ and $B \xrightarrow{\quad g \quad} \underline{C}$, the composite $f; g$ is given as

$$A \xrightarrow{\quad f \quad} TB \xrightarrow{\quad Tg \quad} TU\underline{C} \xrightarrow{\quad \beta \underline{C} \quad} U\underline{C}$$

- For $\mathcal{C}$-morphism $A \xrightarrow{\quad f \quad} B$, ιf is given as

$$A \xrightarrow{\quad f \quad} B \xrightarrow{\quad \eta B \quad} TB$$

- For $\mathcal{E}$-morphism $A \xrightarrow{\quad f \quad} B$, $X \times f$ is given as

$$X \times A \xrightarrow{\quad X \times f \quad} X \times TB \xrightarrow{\quad t(X, B) \quad} T(X \times B)$$

It is easy to verify that $\mathcal{E}$ is a staggered category and that we have defined a value/producer structure from $\mathcal{C}$ to $\mathcal{E}_F$, and to prove the following lemma.

Lemma 143 Given $A \xrightarrow{f} B$ in $\mathcal{C}$ and $B \xrightarrow{g} \underline{C}$ in $\mathcal{E}$, the composite $(\iota f); g$ in $\mathcal{E}$ is given as $f; g$ in $\mathcal{C}$. □

We set the U isomorphism

$$\mathcal{E}(\iota\Gamma, \underline{B}) \cong \mathcal{C}(\Gamma, U\underline{B})$$

to be the identity—its naturality in $\Gamma \in \mathcal{C}^{\mathsf{op}}$ follows from Lemma 143. Consequently $\mathsf{force}_{\underline{B}}$ is $\mathsf{id}_{\underline{B}}$.

For any A and $\underline{B}$, we have an exponent from A to $U\underline{B}$ with vertex $U(A \to \underline{B})$, i.e. we have an isomorphism

$$\mathcal{C}(\Gamma \times A, U\underline{B}) \cong \mathcal{C}(\Gamma, U(A \to \underline{B}))$$

This serves for our required isomorphism

$$\mathcal{E}(\Gamma \times A, \underline{B}) \cong \mathcal{E}(\Gamma, A \to \underline{B})$$

It is straightforward to show that this is natural in $\Gamma \in \mathcal{E}_F^{\mathsf{op}}$ as required. The structure for $\prod$ is similar.

Proposition 144 The semantics of CBPV in the restricted algebra model $(\mathcal{C}, T, \{j\underline{B}\}_{\underline{B} \in \mathfrak{I}}, \dots)$, as described in Chap. 12, is precisely that obtained, as in Chap. 13, from the value/producer model $(\mathcal{C}, \mathcal{E}, \dots)$ that we have just constructed. □

15.4.2 From Value/Producer Model To Restricted Algebra Model

Given a value/producer model $(\mathcal{C}, \mathcal{E}, \dots)$ we want to obtain a restricted algebra model.

The strong monad is constructed as follows.

- For $A \in \mathsf{ob}\ \mathcal{C}$, TA is defined to be UFA.
- For $A \xrightarrow{f} B$, Tf is defined to be $\mathsf{thunk}\ (\mathsf{force}_{FA}; \iota f)$.
- ηA is defined to be $\mathsf{thunk}\ \mathsf{id}_A$.
- μA is defined to be $\mathsf{thunk}\ (\mathsf{force}_{FUFA}; \mathsf{force}_{FA})$.
- $t(A, B)$ is defined to be $\mathsf{thunk}\ (A \times \mathsf{force}_{FB})$.

The family of algebras is $\{j\underline{B}\}_{\underline{B} \in \mathsf{targetob}\ \mathcal{E}}$, where $j\underline{B}$ is the T-algebra with carrier $U\underline{B}$ and structure $\mathsf{thunk}\ (\mathsf{force}_{FU\underline{B}}; \mathsf{force}_{\underline{B}})$.

Checking the various strong monad and algebra equations is easy when we observe that to prove $f = g$, where

$$A \underset{g}{\overset{f}{\rightrightarrows}} U\underline{B}$$

it suffices to prove that

$$\begin{array}{ccc} A & \xrightarrow{\iota f} & U\underline{B} \\ {\scriptstyle \iota g}\downarrow & & \downarrow{\scriptstyle \mathsf{force}_{\underline{B}}} \\ U\underline{B} & \xrightarrow[\mathsf{force}_{\underline{B}}]{} & \underline{B} \end{array} \quad \text{commutes.}$$

For $\to$, we must first construct an exponent from A to $U\underline{B}$, with vertex $U(A \to \underline{B})$. This is given by

$$\begin{array}{rcl} \mathcal{C}(\Gamma, U(A \to \underline{B})) & \cong & \mathcal{E}(\iota\Gamma, A \to \underline{B}) \\ & \cong & \mathcal{E}((\iota\Gamma) \times A, \underline{B}) \\ & = & \mathcal{E}(\iota(\Gamma \times A), \underline{B}) \\ & \cong & \mathcal{C}(\Gamma \times A, U\underline{B}) \end{array}$$

which is natural in Γ because each factor is. If we write α_Γ for the isomorphism from $\mathcal{E}(\Gamma, A \to \underline{B})$ to $\mathcal{E}(\Gamma \times A, \underline{B})$, then the evaluation map ev is given by $\mathsf{thunk}\ \alpha_{U(A \to \underline{B})}\mathsf{force}\ _{A \to \underline{B}}$. We must show that the structure of $j(A \to \underline{B})$ is that given by this exponent; this follows straightforwardly from Lemma 127(2).

The structure for $\prod$ is constructed similarly.

Lemma 145 (used in proof of Prop. 148) Given $\Gamma' \xrightarrow{\quad h \quad} \Gamma$ and $\Gamma \xrightarrow{\quad g \quad} \underline{A}$ then

$$\begin{array}{ccc} UF\Gamma & \xrightarrow{T\mathsf{thunk}\, g} & UFU\underline{A} \\ {\scriptstyle \mathsf{thunk}\, h}\uparrow & & \downarrow{\scriptstyle \beta\underline{A}} \\ \Gamma' & \xrightarrow{\mathsf{thunk}\,(h;g)} & U\underline{A} \end{array}$$

□

15.4.3 Value/Producer to Restricted Algebra to Value/Producer

Given a value/producer model $(\mathcal{C}, \mathcal{E}, \dots)$, first obtain a restricted algebra model $(\mathcal{C}, T, \{j\underline{B}\}_{\underline{B} \in \mathcal{I}}, \dots)$ and then obtain another model $(\mathcal{C}, \mathcal{E}', \dots)$. We see that

$$\mathcal{E}'(A, \underline{B}) = \mathcal{C}(A, U\underline{B}) \cong \mathcal{E}(A, \underline{B})$$

It is easy to check that this is structure preserving.

15.4.4 Restricted Algebra to Value/Producer to Restricted Algebra

Given a restricted algebra model $(\mathcal{C}, T, \{j\underline{B}\}_{\underline{B} \in \mathcal{I}}, \dots)$, we obtain a value/producer model $(\mathcal{C}, \mathcal{E}, \dots)$ and then another restricted algebra model. It is easy to check that this latter restricted algebra model is exactly the same as the former, so the required structure preserving isomorphism is the identity.

15.5 Full Reflection: Restricted Algebra Models and Adjunction Models

15.5.1 From Restricted Algebra Model To Adjunction Model

Given a restricted algebra model $\mathcal{M} = (\mathcal{C}, T, \{j\underline{B}\}_{\underline{B} \in \mathcal{I}}, \dots)$, we obtain an adjunction model $\mathcal{UM}$ as outlined in Sect. 14.4.2. Here are the details:

- $\mathcal{D}_\Gamma(\underline{A}, \underline{B})$ is defined to be the set of T-algebra homomorphisms from $j\underline{A}$ to $j\underline{B}$ over Γ.
- The identity $\underline{A} \xrightarrow[\Gamma]{\mathsf{id}} \underline{A}$ is $\pi_{\Gamma, \underline{A}}$.
- The composite $\underline{A} \xrightarrow[\Gamma]{h} \underline{B} \xrightarrow[\Gamma]{h'} \underline{C}$ is given by $(\pi_{\Gamma, U\underline{A}}, h); h'$.

- The reindexing of $\underline{A} \xrightarrow[\Gamma]{h} \underline{B}$ by $\Delta \xrightarrow{k} \Gamma$ is given by $(k \times U\underline{A}); h$.
- $O_\Gamma \underline{B}$ is defined to be $\mathcal{C}(\Gamma, U\underline{B})$.
- The composite $\xrightarrow[\Gamma]{g} \underline{A} \xrightarrow[\Gamma]{h} \underline{B}$ is given by $(\mathsf{id}_\Gamma, g); h$
- The reindexing of $\xrightarrow[\Gamma]{g} \underline{A}$ by $\Delta \xrightarrow{k} \Gamma$ is given by $k; g$.
- Given $\xrightarrow[\Gamma \times A]{g} \underline{B}$, the $\mathcal{D}$-morphism $\mathsf{str}\ g$ is given by $t(\Gamma, A); Tf; \beta\underline{B}$. Equivalently, prod_A is given by η_A.
- Given $\xrightarrow[\Gamma]{g} \underline{B}$, the $\mathcal{C}$-morphism $\mathsf{thunk}\ g$ is just g. Equivalently, $\mathsf{force}\ _{\underline{B}}$ is given by $\mathsf{id}_{U\underline{B}}$.
- The required isomorphism

$$\mathcal{D}_{\Gamma \times A}(\pi^*_{\Gamma,A}\underline{X}, \underline{B}) \cong \mathcal{D}_\Gamma(\underline{X}, A \to \underline{B}) \tag{15.1}$$

  is given by

$$\begin{array}{lcl} \mathcal{C}((\Gamma \times A) \times U\underline{X}, U\underline{B}) & \cong & \\ \mathcal{C}((\Gamma \times U\underline{X}) \times A, U\underline{B}) & \cong & \mathcal{C}(\Gamma \times U\underline{X}, U(A \to \underline{B})) \end{array}$$

  It is easily verified that this bijection restricts to a bijection between algebra homomorphisms, as required; similarly for $\prod$.

All the required equations for an adjunction model are easy to prove. It is also easy to show that the isomorphism

$$O_{\Gamma \times A}\underline{B} \cong O_\Gamma A \to \underline{B}$$

derived from (15.1) is just the isomorphism

$$\mathcal{C}(\Gamma \times A, U\underline{B}) \cong \mathcal{C}(\Gamma, U(A \to \underline{B}))$$

expressing $U(A \to \underline{B})$ as the vertex of an exponent from A to $U\underline{B}$; similarly for $\prod$.

15.5.2 From Adjunction Model To Restricted Algebra Model

Let $\mathcal{A} = (\mathcal{C}, \mathcal{D}, O, \ldots)$ be an adjunction model. Obtain a staggered adjunction model (by forgetting most of the homsets), then a value/producer model (by Sect. 15.3.2), then a restricted algebra model (by Sect. 15.4.2)—call this $\mathcal{FA} = (\mathcal{C}, T, \{j\underline{B}\}_{\underline{B} \in \mathsf{ob}\ \mathcal{D}}, \ldots)$.

Explicitly:

$$\begin{array}{rcl} T(A \xrightarrow{f} B) & = & \mathsf{thunk}\ (\mathsf{force}\ _{FA}; \mathsf{str}\ \pi'^*_{UFA,A} f^* \mathsf{prod}_B) \\ \eta A & = & \mathsf{thunk}\ \mathsf{prod}_A \\ \mu A & = & \mathsf{thunk}\ (\mathsf{force}\ _{FUFA}; \mathsf{str}\ \pi'^*_{UFUFA,UFA} \mathsf{force}\ _{FA}) \\ t(A,B) & = & \mathsf{thunk}\ (\pi'^*_{A,UFB} \mathsf{force}\ _{FB}; \pi^*_{A,UFB} \mathsf{str}\ \mathsf{prod}_{A \times B}) \\ \beta\underline{B} & = & \mathsf{thunk}\ (\mathsf{force}\ _{FU\underline{B}}; \mathsf{str}\ \pi'^*_{UFU\underline{B},U\underline{B}} \mathsf{force}\ _{\underline{B}}) \end{array}$$

The exponent from A to $U\underline{B}$ with vertex $U(A \to \underline{B})$ is given as the composite

$$\begin{array}{rcl} \mathcal{C}(\Gamma \times A, \underline{B}) & \cong & \\ \mathcal{O}_{\Gamma \times A}\underline{B} & \cong & \mathcal{O}_\Gamma A \to \underline{B} \\ & \cong & \mathcal{C}(\Gamma, U(A \to \underline{B})) \end{array}$$

Similarly for $\prod$.

15.5.3 Restricted Algebra To Adjunction To Restricted Algebra

Proposition 146 Let $\mathcal{M}$ be a restricted algebra model. Then $\mathcal{F}\,\mathcal{U}\mathcal{M} = \mathcal{M}$. □

This is straightforward to prove.

We can thus set the counit of our full reflection to be the identity.

15.5.4 Adjunction To Restricted Algebra To Adjunction

Let $\mathcal{A} = (\mathcal{C}, \mathcal{D}, \mathcal{O}, \ldots)$ be an adjunction model. We obtain a restricted algebra model $\mathcal{F}\mathcal{A} = (\mathcal{C}, T, \{j\underline{B}\}_{\underline{B} \in \mathsf{ob}\ \mathcal{D}}, \ldots)$. We then obtain another adjunction model $\mathcal{U}\mathcal{F}\mathcal{A}$. We wish to construct an adjunction-model-morphism $\mathsf{unit}_{\mathcal{A}}$ from $\mathcal{A}$ to $\mathcal{U}\mathcal{F}\mathcal{A}$—this will be the unit of our full reflection.

- $\mathsf{unit}_{\mathcal{A}}$ leaves $\mathcal{C}$-morphisms unchanged.
- $\mathsf{unit}_{\mathcal{A}}$ takes an oblique morphism $\xrightarrow[\Gamma]{g} \underline{A}$ to $\Gamma \xrightarrow{\mathsf{thunk}\ g} U\underline{A}$.
- $\mathsf{unit}_{\mathcal{A}}$ takes a $\mathcal{D}$-morphism $\underline{A} \xrightarrow[\Gamma]{h} \underline{B}$ to $\Gamma \times U\underline{A} \xrightarrow{\mathsf{thunk}\ (\pi'^{*}_{\Gamma,U\underline{A}}\mathsf{force}_{\underline{A}}; \pi^{*}_{\Gamma,U\underline{A}}h)} U\underline{B}$.

In the last case, we must check that $k = \mathsf{unit}_{\mathcal{A}}h$ is a T-algebra homomorphism from $j\underline{A}$ to $j\underline{B}$ over Γ, i.e. that the diagram

$$\begin{array}{ccccc} \Gamma \times UFU\underline{A} & \xrightarrow{t(\Gamma, U\underline{A})} & UF(\Gamma \times U\underline{A}) & \xrightarrow{Tk} & UFU\underline{B} \\ \Big\downarrow{\scriptstyle \Gamma \times \beta\underline{A}} & & & & \Big\downarrow{\scriptstyle \beta\underline{B}} \\ \Gamma \times U\underline{A} & & \xrightarrow[k]{} & & U\underline{B} \end{array} \quad \text{commutes.}$$

It is sufficient to prove that

$$t(\Gamma, U\underline{A})^*)(Tk)^*(\beta\underline{B})^*\mathsf{force}_{\underline{B}} = (\Gamma \times \beta\underline{A})^*(k)^*\mathsf{force}_{\underline{B}}$$

Expanding and distributing the LHS gives $(\pi'^{*}_{\Gamma,UFU\underline{A}}\mathsf{force}_{FU\underline{A}}); l$ where l is the $\mathcal{D}$-morphism

$$\begin{array}{l} (\pi^{*}_{\Gamma,UFU\underline{A}}\mathsf{str}\ \mathsf{prod}_{\Gamma \times U\underline{A}}); (t(\Gamma, U\underline{A})^*\mathsf{str}\ \pi'^{*}_{UF(\Gamma \times U\underline{A}),\Gamma \times U\underline{A}}k^*\mathsf{prod}_{U\underline{B}}); \\ \qquad\qquad (t(\Gamma, U\underline{A})^*(Tk)^*\mathsf{str}\ \pi'^{*}_{UFU\underline{B},U\underline{B}}\mathsf{force}_{\underline{B}}) \\ \text{by (14.16)} \quad = \quad (\pi^{*}_{\Gamma,UFU\underline{A}}\mathsf{str}\ \mathsf{prod}_{\Gamma \times U\underline{A}}); (\mathsf{str}\ \pi'^{*}_{\Gamma \times UFU\underline{A},\Gamma \times U\underline{A}}k^*\mathsf{prod}_{U\underline{B}}); (\mathsf{str}\ \pi'^{*}_{\Gamma \times UFU\underline{A},U\underline{B}}\mathsf{force}_{\underline{B}}) \\ \text{by (14.19)} \quad = \quad (\mathsf{str}(\pi_{\Gamma,UFU\underline{A}} \times U\underline{A})^*\mathsf{prod}_{\Gamma \times U\underline{A}}); (\mathsf{str}\ \pi'^{*}_{\Gamma \times UFU\underline{A},\Gamma \times U\underline{A}}(\pi'^{*}_{\Gamma,U\underline{A}}\mathsf{force}_{\underline{A}}; \pi^{*}_{\Gamma,U\underline{A}}h)) \\ \text{by (14.19)} \quad = \quad \mathsf{str}(\pi_{\Gamma,UFU\underline{A}} \times U\underline{A})^*(\pi'^{*}_{\Gamma,U\underline{A}}\mathsf{force}_{\underline{A}}; \pi^{*}_{\Gamma,U\underline{A}}h) \\ \text{by (14.16)} \quad = \quad \pi^{*}_{\Gamma,UFU\underline{A}}\mathsf{str}(\pi'^{*}_{\Gamma,U\underline{A}}\mathsf{force}_{\underline{A}}; \pi^{*}_{\Gamma,U\underline{A}}h) \\ \text{by (14.17)} \quad = \quad (\pi^{*}_{\Gamma,UFU\underline{A}}\mathsf{str}\ \pi'^{*}_{\Gamma,U\underline{A}}\mathsf{force}_{\underline{A}}); (\pi^{*}_{\Gamma,UFU\underline{A}}h) \end{array}$$

Expanding and distributing the RHS gives $(\pi'^{*}_{\Gamma,UFU\underline{A}}\mathsf{force}_{FU\underline{A}});l'$ where l' is the $\mathcal{D}$-morphism

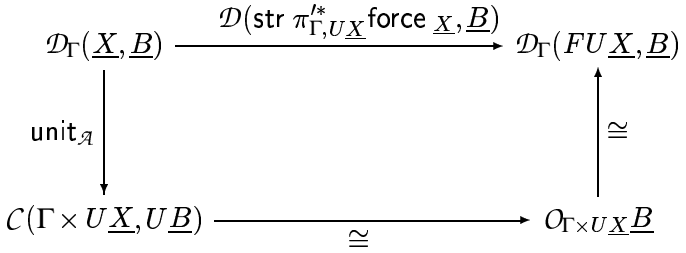
$$\begin{array}{rcl} & & (\pi'^{*}_{\Gamma,UFU\underline{A}}\mathsf{str}\ \pi'^{*}_{UFU\underline{A},U\underline{A}}\mathsf{force}_{\underline{A}});(\pi^{*}_{\Gamma,UFU\underline{A}}h) \\ \text{by (14.22)} & = & (\pi^{*}_{\Gamma,UFU\underline{A}}\mathsf{str}\ \pi'^{*}_{\Gamma,U\underline{A}}\mathsf{force}_{\underline{A}});(\pi^{*}_{\Gamma,UFU\underline{A}}h) \end{array}$$

Thus $l = l'$. For brevity, we have omitted all the instances of reducing $(\mathsf{thunk}\ g)^{*}\mathsf{force}_{\underline{B}}$ to g and of distributing reindexing over composition.

Showing that unit_{A} preserves identities, composition and reindexing is straightforward. Preservation of the U isomorphism is trivial, and for preservation of the F isomorphism it suffices to show preservation of $\mathsf{produce}_{A}$, which is trivial.

For the $\to$ and $\prod$ isomorphisms, we need

Lemma 147 The diagram

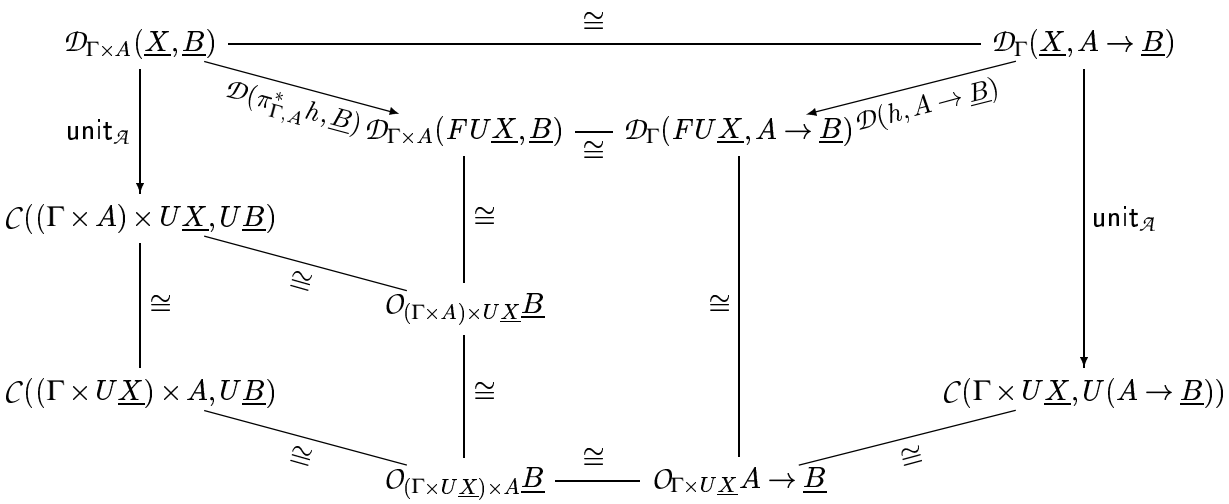


commutes.

□

Proof Following a morphism $\underline{X} \xrightarrow[\Gamma]{h} \underline{B}$ down, right and up gives $\mathsf{str}(\pi'^{*}_{\Gamma,U\underline{X}}\mathsf{force}_{\underline{X}};\pi^{*}_{\Gamma,U\underline{X}}h)$ which by Lemma 137(14.17) is equal to $(\mathsf{str}\ \pi'^{*}_{\Gamma,U\underline{X}}\mathsf{force}_{\underline{X}});h$. □

Now the fact that $\mathsf{unit}_{\mathcal{A}}$ preserves the $\to$ isomorphism is given by the commutativity of



where h is $\mathsf{str}\ \pi'^{*}_{\Gamma,U\underline{X}}\mathsf{force}_{\underline{X}}$. The right quadrilateral commutes by Lemma 147, as does the upper left quadrilateral, using the fact that $\pi^{*}_{\Gamma,A}h$ is $\mathsf{str}\ \pi'^{*}_{\Gamma\times A,U\underline{X}}\mathsf{force}\ \underline{X}$ by Lemma 137(14.16). The central pentagon commutes by Prop. 128(1)—we have chosen to divide $\Gamma\times U\underline{X}$ into Γ and $U\underline{X}$. The upper quadrilateral commutes by the naturality of the $\to$ isomorphism in $\mathcal{A}$. The bottom left quadrilateral commutes by the naturality of the U isomorphism in $\mathcal{A}$.

Similarly, we can prove that $\mathsf{unit}_{\mathcal{A}}$ preserves the isomorphism for $\prod$.

15.5.5 Triangle Laws

To complete our construction of the full reflection, we must verify the triangle laws for an adjunction. Because the counit is identity, these are quite simple.

1. $\mathcal{F}\,\mathsf{unit}_{\mathcal{A}} = \mathsf{id}_{\mathcal{FA}}$. This simply says that unit_A is identity on $\mathcal{C}$-morphisms, which is true by definition.

2. $\mathsf{unit}_{\mathcal{UM}} = \mathsf{id}_{\mathcal{UM}}$. This is easily verified.

15.6 Notions Of Homomorphism

We now show that the various definitions of homomorphism given in Def. 121 accurately characterize the homomorphisms in the induced adjunction model.

Proposition 148 Let $\mathcal{M}$ be a restricted algebra model, value/producer model, staggered adjunction model or direct model giving rise to the adjunction model $(\mathcal{C}, \mathcal{D}, O, \ldots)$, as in Fig. 11.1. Then there is a bijection between the homomorphisms from $\underline{A}$ to $\underline{B}$ over C in $\mathcal{M}$, as defined in Def. 121, and $\mathcal{D}_C(\underline{A}, \underline{B})$. □

Proof

restricted algebra model This is just the construction of the adjunction model.

value/producer model Let $(\mathcal{C}, \mathcal{E}, \ldots)$ be a value/producer model giving rise to the restricted algebra model $(\mathcal{C}, T, \{j\underline{B}\}_{\underline{B} \in \mathsf{targetob}\ \mathcal{E}}, \ldots)$. We have to construct a bijection between morphisms

$$\mathcal{E}(\Gamma, \underline{A}) \longrightarrow \mathcal{E}(\Gamma \times C, \underline{B}) \text{ natural in } \Gamma \in \mathcal{E}_F^{\mathsf{op}}$$

and algebra homomorphisms from $j\underline{A}$ to $j\underline{B}$ over C.

- Given a morphism

  $$\mathcal{E}(\Gamma, \underline{A}) \xrightarrow{\alpha_\Gamma} \mathcal{E}(\Gamma \times C, \underline{B}) \text{ natural in } \Gamma \in \mathcal{E}_F^{\mathsf{op}}$$

  we set f to be

  $$C \times U\underline{A} \xrightarrow{(\pi', \pi)} U\underline{A} \times C \xrightarrow{\mathsf{thunk}\ \alpha_{U\underline{A}} \mathsf{force}\ \underline{A}} U\underline{B}$$

  Using Lemma 127(2), we can prove that this is an algebra homomorphism from $j\underline{A}$ to $j\underline{B}$ over C.

- Given an algebra homomorphism f from $j\underline{A}$ to $j\underline{B}$ over C, we obtain the morphism α_Γ taking $\Gamma \xrightarrow{g} \underline{A}$ to

  $$\Gamma \times C \xrightarrow{\iota \mathsf{thunk}\ g \times C} U\underline{A} \times C \xrightarrow{\iota(\pi', \pi)} C \times U\underline{A} \xrightarrow{\iota f} U\underline{B} \xrightarrow{\mathsf{force}\ \underline{B}} \underline{B}$$

  The naturality of α_Γ in $\Gamma \in \mathcal{E}_F^{\mathsf{op}}$ is proved using Lemma 145.

That these are inverse constructions follows from a Yoneda-style argument.

staggered adjunction model Let $(\mathcal{C}, \mathcal{D}, O)$ be a staggered adjunction model giving rise to the value/producer model $(\mathcal{C}, \mathcal{E}, \ldots)$, We have to construct a bijection between morphisms

$$\mathcal{D}_\Gamma({}^F X, \underline{A}) \longrightarrow \mathcal{D}_{\Gamma \times C}(\pi^{*F} X, \underline{B}) \text{ natural in } \Gamma \text{ and } {}^F X$$

and morphisms

$$\mathcal{E}(\Gamma, \underline{A}) \longrightarrow \mathcal{E}(\Gamma \times C, \underline{B}) \text{ natural in } \Gamma \in \mathcal{E}_F^{\mathsf{op}}$$

- Given a morphism

$$\mathcal{D}_\Gamma({}^F X, \underline{A}) \longrightarrow \mathcal{D}_{\Gamma\times C}(\pi^*_{\Gamma,C}{}^F X, \underline{B}) \text{ natural in } \Gamma \text{ and } {}^F X$$

  we use an arbitrary division of Γ to obtain a morphism from $\mathcal{E}_\Gamma \underline{A}$ to $\mathcal{E}_{\Gamma\times C}\underline{B}$—as in Prop. 128(1), this morphism is independent of the particular division used. We show that it is natural in $\Gamma \in \mathcal{E}_F^{\mathsf{op}}$ just as we proved the naturality of

$$\mathcal{E}(\Gamma \times A, \underline{B}) \cong \mathcal{E}(\Gamma, A \to \underline{B})$$

  in Sect. 15.3.2.

- Given a morphism

$$\mathcal{E}(\Gamma, \underline{A}) \xrightarrow{\alpha_\Gamma} \mathcal{E}(\Gamma \times C, \underline{B}) \text{ natural in } \Gamma \in \mathcal{E}_F^{\mathsf{op}}$$

  we obtain

$$\begin{array}{ccc} \mathcal{D}_\Gamma({}^F X, \underline{A}) & \xrightarrow{\cong} & \mathcal{O}_{\Gamma\times X}\underline{A} \\ & & \Big\downarrow{\scriptstyle \alpha_{\Gamma\times X}} \\ \mathcal{D}_{\Gamma\times C}({}^F X, \underline{B}) & \xleftarrow[\cong]{} & \mathcal{O}_{(\Gamma\times X)\times C}\underline{B} \end{array}$$

  This can be shown natural in Γ and ${}^F X$ using Lemma 137.

It is easy to show that these constructions are inverse.

direct model Let (s,θ) be a direct model giving rise to the value/producer model $(\mathcal{C},\mathcal{E},\dots)$. We have to construct a bijection between derivations

$$\frac{\Gamma \vdash^{\mathsf{c}} \underline{A}}{\Gamma, C \vdash^{\mathsf{c}} \underline{B}}$$

that commutes with sequencing in Γ, and morphisms

$$\mathcal{E}(\Gamma, \underline{A}) \longrightarrow \mathcal{E}(\Gamma \times C, \underline{B}) \text{ natural in } \Gamma \in \mathcal{E}_F^{\mathsf{op}}$$

We omit the details, which are straightforward.

□

Part IV

Conclusions

Chapter 16

Conclusions, Comparisons and Further Work

16.1 Summary of Achievements and Drawbacks

In Part I, we introduced the CBPV paradigm as a language of semantic primitives for higher order programming with computational effects (even just divergence). In Part II we saw a vast range of semantics that support CBPV's claim to be such a language, subsuming both CBV and CBN. That range includes models for printing, storage, divergence, erratic choice, errors and control effects; it also includes possible world models and interaction-based semantics such as jumping implementations (using continuations) and pointer game models. Again and again, we saw the advantages of using CBPV as a language of study. For example, in the interaction-based semantics, the explicit control flow in CBPV makes it closer to the detailed behaviour present in the model than CBN or CBV are.

However, there were some examples of CBV models that we were unable to decompose into CBPV in a natural way:

- the model for input based on Moggi's input monad [Mog91]—note also the related difficulty in formulating an operational semantics for this effect;
- the constrained model for finite erratic choice based on the strong monad $\mathcal{P}_{\mathsf{fin}}$ on **Set**, described in Sect. 6.5.3;
- the constrained model for store that incorporates parametricity in initializations, mentioned in Sect. 7.9.

In Part III, we saw that all of our semantics for CBPV are instances of adjunction models. It is therefore reasonable to say that what CBPV achieves is to decompose Moggi's strong monads into strong adjunctions. However, as we explained in Sect. 14.3.2, whilst all these semantics exhibit adjunction structure, the CBPV language itself does not. We therefore provided other forms of categorical semantics that, although less elegant, agree exactly with the CBPV equational theory.

16.2 Contrast With Other Decompositions

We contrast the success of the CBPV decompositions of CBV and CBN type constructors with 2 other well-known decompositions:

- Moggi's decomposition of $A \to_{\mathbf{CBV}} B$ as $A \to TB$ by contrast with our $U(A \to FB)$
- the linear logic decomposition of $\underline{A} \to_{\mathbf{CBN}} \underline{B}$ as $!\underline{A} \multimap \underline{B}$ by contrast with our $(U\underline{A}) \to \underline{B}$.

Both these decompositions/translations are given in [BW96].

For discussion's sake, we will say that the types of Moggi's target language include

$$A ::= \quad \texttt{bool} \mid \texttt{nat} \mid A \times A \mid A \to A \mid TA \mid \cdots$$

although Moggi does not explicitly include `bool` and `nat`. In categorical terms, Moggi's decomposition of $\to_{\mathbf{CBN}}$ is isomorphic to ours, because $U(A \to FB)$ must be the vertex of an exponent from A to TB. But much is lost in Moggi's translation, as we now explain.

- As we said in Sect. 1.2.3, the monadic language does not have an operational semantics (at least not a simple one in the same way as CBV, CBN and CBPV) because it is a language of values.
- Although the monadic language does have a denotational semantics using cpos, in which $A \to B$ denotes the cpo of total functions from $[\![A]\!]$ to $[\![B]\!]$, this is not a SEAM predomain semantics (Def. 23). For example, $\texttt{nat} \to \texttt{nat}$ denotes an uncountable flat cpo, which is not a SEAM predomain.
- We cannot add general type recursion to the monadic language, because for example we could not interpret $\mu\texttt{X}.(\texttt{X} \to \texttt{bool})$—there is no cpo X such that $X \cong X \to 2$. The heart of the problem is that total function space between cpos does not define a locally continuous functor.
- We cannot translate the monadic language into Jump-With-Argument (preserving semantics), so we do not have a jumping implementation for it. To put this another way, look at the continuation semantics for $A \to_{\mathbf{CBV}} B$ given by the two decompositions:

  $$\begin{array}{rcl} A \to TB & \text{gives} & A \to \neg\neg B \\ U(A \to FB) & \text{gives} & \neg(A \times \neg B) \end{array}$$

  These are isomorphic. But the latter is meaningful in a jumping sense—it says that a CBV function is a point to which we jump taking both an argument and a return address for the result. By contrast, the $\to$ used in the former cannot be understood in jumping terms.

The target language of the linear logic decomposition is a linear λ-calculus with types including

$$\underline{A} ::= \quad !\underline{A} \mid \underline{A} \otimes \underline{A} \mid \underline{A} \multimap \underline{A} \mid \cdots$$

(We are underlining the types because they denote pointed cpos.)

Much is lost in the translation from CBN into this language. For example, O'Hearn's semantics of global store

$$[\![\underline{A} \to_{\mathbf{CBN}} \underline{B}]\!] = (S \to [\![\underline{A}]\!]) \to [\![\underline{B}]\!]$$

and the Streicher-Reus continuation semantics

$$[\![\underline{A} \to_{\mathbf{CBN}} \underline{B}]\!] = (\neg[\![\underline{A}]\!]) \times [\![\underline{B}]\!]$$

do not exhibit the linear decomposition. One reason for this is that, as we said in Sect. 1.2.3, the linear λ-calculus assumes commutativity of effects.

For the same reason, the pointer game model for CBN (given by the CBPV model of Chap. 9) does not exhibit the linear decomposition of $\to_{\mathbf{CBN}}$. This is perhaps surprising, given the influence of linear logic on the development of game semantics.

16.3 Beyond Simple Types

In this thesis we have studied only simply typed languages, together with recursive and infinitary types. We have not looked at polymorphism or dependent typing.

16.3.1 Dependent Types

Dependent types and computational effects are a problematic combination, as Moggi found. From a CBPV perspective, the problem is as follows.

On the one hand, it is easy and even beneficial to add dependent types to CBPV. $\sum$ and $\times$ can both be seen as instances of dependent sum, while $\prod$ and $\to$ can both be seen as instances of dependent product. The one typing rule that requires care is the sequencing rule

$$\frac{\Gamma \vdash_{\mathtt{c}} M : FA \quad \Gamma, \mathtt{x} : A \vdash_{\mathtt{c}} N : \underline{B}}{\Gamma \vdash_{\mathtt{c}} M \ \mathtt{to}\ \mathtt{x}.\ N : \underline{B}}$$

In this rule, we must stipulate that $\underline{B}$ does not depend on $\mathtt{x}$, only on Γ. This restriction is indispensable if we want to retain our denotational models. The categorical semantics of this language remain to be studied, but we do not expect difficulties.

The problem is that there is no obvious translation from λ-calculus with dependent sums and dependent products into this "dependently typed CBPV". In the simply typed case we have 2 translations (CBV and CBN); but the CBV translation of application and the CBN translation of pattern-matching both use sequencing, and when we try extend to the dependently typed setting, we violate the above restriction.

Thus it appears that this "dependently typed CBPV" may not be as expressive as a dependently typed language without effects. However, further investigation is required.

16.3.2 Polymorphism

From an operational perspective, we speculate that we could add polymorphism to CBPV by adding the following type constructors:

$$\begin{array}{rcl} A & ::= & \cdots \mid X \mid \sum X.A \mid \sum \underline{X}.A \\ \underline{B} & ::= & \cdots \mid \underline{X} \mid \prod X.\underline{B} \mid \prod \underline{X}.\underline{B} \end{array}$$

However, whether this is suitable from the point of view of impredicative denotational semantics remains to be investigated.

16.4 Further Work

In addition to the investigation of dependent and polymorphic types mentioned above, there are a number of holes in various places in the thesis that need to be filled, and the problems involved are by no means trivial.

- Development of a CBPV model combining erratic choice and divergence.
- Formulation of a big-step semantics for the combination of divergence and printing, as discussed in Sect. 6.7.1.
- Development of parametric models for cell generation.
- In the pointer game model, presenting the interpretation of term constructors in a clean way.

Part V

Appendices

Appendix A

Technical Treatment of CBV and CBN

A.1 The Revised Simply Typed λ-Calculus

A.1.1 Introduction

This chapter looks in detail at the relationship between CBV, CBN and CBPV. We will prove various technical results, including universality and full abstraction of transforms. The various languages, translations and results are listed in Sect. A.2.

Before we come to these, it is helpful to recollect what we were aiming to achieve in Chap. 2–3:

> We take a purely functional language, add a computational effect (e.g. printing) and explore various possible choices: about evaluation order, about observational equivalence etc. Then we seek a single language that includes all of these possible choices.

The wider the range of language possibilities that we explore and show to be included within CBPV, the stronger our claim that CBPV is a "subsuming paradigm".

In this chapter we carefully retread this path. In order that our exploration of possibilities be as thorough as possible, we want our starting point (the purely functional language) to have a wider range of type constructors than was provided in $\lambda\,\mathtt{bool}{+}$.

For example, we want to look at product types. Rather than decide *a priori* whether these will be projection products or pattern-match products (in the sense of Sect. 2.3.2), we will provide both. We will then see how each is affected by the addition of computational effects, and eventually verify that each possibility is included within CBPV.

Similarly, we will provide not just unary but multi-ary functions. For although there are various isomorphisms that justify the decomposition of multi-ary function types into unary function types, it is not clear *a priori* that these will continue to be valid when effects are added.

The purely functional language that we use is called the *revised simply typed λ-calculus*, or Revised-λ for short. Because it is intended as an exploratory tool, as we have explained, Revised-λ has extremely general type constructors:

tuple types include both sums and pattern-match products as special cases.

function types include both unary function types and projection products as special cases.

We work with infinitely wide languages (see Sect. 5.1 for a discussion). The reader wishing to consider only finitely wide languages should substitute "finite" for "countable" throughout this chapter.

A.1.2 Tuple Types

Tuple types are a generalization of the usual sum types. Something of sum type is formed as a pair consisting of a tag and a term; the type of this term depends on the tag. By contrast, something of tuple type is formed as a finite tuple consisting of a tag and *several* terms; the number of terms and their types depend on the tag. This is the most liberal tuple type formation possible within a simply typed language, because simple typing requires that the type of a term cannot depend on the type of another term, only on a tag.

Tuple types are built as follows. Let I be a countable set—this will be the set of tags. Suppose that for each $i \in I$ we have a finite sequence of types $A_{i\,0}, \ldots, A_{i\,(r_i-1)}$. Then we form the tuple type $\sum_{i\in I}^{\times_{j\in\$r_i}} A_{i\,j}$. This denotes the set $\sum_{i\in I}([\![A_{i\,0}]\!] \times \cdots \times [\![A_{i\,(r_i-1)}]\!])$, or, isomorphically, the set of tuples $(i, a_0, \ldots, a_{r_i-1})$, where $a_j \in [\![A_{i\,j}]\!]$.

When each $r_i = 1$ we obtain the usual sum type written $\sum_{i\in I} A_i$. In particular, when $I = \{0,1\}$ we obtain the binary sum $A + B$ and when I is empty we obtain the zero type 0.

When I is a singleton set $\{*\}$ and $r_* = 2$, we obtain a pattern-match binary product type $A \times B$. Similarly, when I is a singleton set $\{*\}$ and $r_* = 0$, we obtain the *pattern-match-unit* type 1.

In the effect-free setting, $\sum$,$\times$ and 1 are sufficient to give all tuple types because of this decomposition:

$$\sum_{i\in I}^{\times_{j\in\$r_i}} A_{i\,j} \cong \sum_{i\in I}(A_{i\,0} \times \cdots \times A_{i\,(r_i-1)}) \tag{A.1}$$

However, it is not apparent *a priori* whether (A.1) will remain valid when we add effects. This is the reason for providing general tuple types. (In fact, (A.1) is valid in CBV but not in CBN.)

Another special case of tuple types is when each $r_i = 0$. Such a tuple type is called a *ground type*. Its closed terms are of the form (i), for $i \in I$; sometimes we write this just as i. In particular when $I = \{\mathsf{true}, \mathsf{false}\}$ we call this type `bool` and when $I = \mathbb{N}$ we call this type `nat`. We write `true` for (true), `false` for (false) and `if` M `then` N `else` N' for `pm` M `as` $\{(\mathsf{true}).\ N, (\mathsf{false}).\ N'\}$.

A.1.3 Function Types

Revised-λ function types are a generalization of the usual unary function types. Something of unary function type is applied to a single operand. By contrast, something of Revised-λ function type is applied to several operands. The first operand is a tag and the rest are terms. The number of terms and their types depend on the tag, and the type of the result also depends on the tag. This is the most liberal function type formation possible within a simply typed language, because simple typing requires that the type of a term cannot depend on the type of another term, only on a tag.

We use a novel notation: when we apply a function to several operands, we delimit these operands on the left with the symbol $\angle$. For example N applied to M_0 and M_1 is written $\angle M_0, M_1\text{`}N$. The reason we do not write it $(M_0, M_1)\text{`}N$ is that this notation suggests that (M_0, M_1) is a subterm, which it is not.

Function types are built as follows. Let I be a countable set—this will be the set of tags. Suppose that for each $i \in I$ we have a finite sequence of types $A_{i0}, \ldots, A_{i\,(r_i-1)}$ and another type B_i. Then we form the function type $\prod_{i\in I}^{\to_{j\in\$r_i}} A_{i\,j} B_i$. This denotes the set $\prod_{i\in I}([\![A_{i\,0}]\!] \to \cdots \to [\![A_{i\,(r_i-1)}]\!] \to [\![B_i]\!])$, or isomorphically the set of functions that takes the sequence of operands $\angle i, a_0, \ldots, a_{r_i-1}$, where $a_j \in [\![A_{i\,j}]\!]$, to an element of $[\![B_i]\!]$.

Where each $r_i = 1$ we obtain a projection product type written $\prod_{i\in I} A_i$. In particular, when $I = \{0,1\}$ we obtain the binary projection-product which we write $A \sqcap B$, and when I is empty we obtain the projection-unit 1_Π.

When I is a singleton set, say $\{*\}$, and $r_* = 1$ we obtain the usual unary function type $A \to B$.

In the effect-free setting $\prod$ and $\to$ are sufficient to give all function types because of this decomposition:

$$\prod_{i\in I}^{\to_{j\in \$r_i}} A_{ij} B_i \cong \prod_{i\in I}(A_{i0} \to \cdots \to A_{i(r_i-1)} \to B_i) \tag{A.2}$$

However, it is not apparent *a priori* whether (A.2) will remain valid when we add effects. This is the reason for providing general function types. (In fact, (A.2) is valid in CBN but not in CBV.)

Types $\quad A ::= \quad \sum_{i\in I}^{\times_{j\in \$r_i}} A_{ij} \;\mid\; \prod_{i\in I}^{\to_{j\in \$r_i}} A_{ij} A_i$

where each set I is countable.

Terms

$$\frac{}{\Gamma, \mathtt{x} : A, \Gamma' \vdash \mathtt{x} : A} \qquad \frac{\Gamma \vdash M : A \quad \Gamma, \mathtt{x} : A \vdash N : B}{\Gamma \vdash \mathtt{let\ x\ be}\ M.\ N : B}$$

$$\frac{\Gamma \vdash M_0 : A_{\hat{\imath}0} \;\cdots\; \Gamma \vdash M_{r_i-1} : A_{\hat{\imath}(r_i-1)}}{\Gamma \vdash (\hat{\imath}, M_0, \ldots, M_{r_i-1}) : \sum_{i\in I}^{\times_{j\in \$r_i}} A_{ij}}$$

$$\frac{\Gamma \vdash M : \sum_{i\in I}^{\times_{j\in \$r_i}} A_{ij} \quad \cdots\ \Gamma, \mathtt{x}_0 : A_{i0}, \ldots, \mathtt{x}_{r_i-1} : A_{i(r_i-1)} \vdash N_i : B \ \cdots}{\Gamma \vdash \mathtt{pm}\ M\ \mathtt{as}\ \{\ldots, (i, \mathtt{x}_0, \ldots, \mathtt{x}_{r_i-1}).N_i, \ldots\} : B}$$

$$\frac{\cdots\ \Gamma, \mathtt{x}_0 : A_{i\,0}, \ldots, \mathtt{x}_{r_i-1} : A_{i\,r_i-1} \vdash M_i : B_i \ \cdots}{\Gamma \vdash \lambda\{\ldots, \angle i, \mathtt{x}_0, \ldots, \mathtt{x}_{r_i-1}.M_i, \ldots\} : \prod_{i\in I}^{\to_{j\in \$r_i}} A_{ij} B_i}$$

$$\frac{\Gamma \vdash M_0 : A_{\hat{\imath}0} \;\cdots\; \Gamma \vdash M_{r_i-1} : A_{\hat{\imath}(r_i-1)} \quad \Gamma \vdash N : \prod_{i\in I}^{\to_{j\in \$r_i}} A_{ij} B_i}{\Gamma \vdash \angle\hat{\imath}, M_0, \ldots, M_{r_{i-1}}\text{`}N : B_{\hat{\imath}}}$$

where $\hat{\imath}$ is any element of I.

β-laws

$$\begin{array}{rcl}
\mathtt{let\ x\ be}\ M.\ N & = & N[M/\mathtt{x}] \\
\mathtt{pm}\ (\hat{\imath}, \overrightarrow{M_j})\ \mathtt{as}\ \{\ldots, (i, \overrightarrow{\mathtt{x}_j}).N_i, \ldots\} & = & N_{\hat{\imath}}[\overrightarrow{M_j/\mathtt{x}_j}] \\
\angle\hat{\imath}, \overrightarrow{M_j}\text{`}\lambda\{\ldots, \angle i, \overrightarrow{\mathtt{x}_j}.N_i, \ldots\} & = & N_{\hat{\imath}}[\overrightarrow{M_j/\mathtt{x}_j}]
\end{array}$$

η-laws

$$\begin{array}{rcl}
N[M/\mathtt{z}] & = & \mathtt{pm}\ M\ \mathtt{as}\ \{\ldots, (i, \overrightarrow{\mathtt{x}_j}).N[(i, \overrightarrow{\mathtt{x}_j})/\mathtt{z}], \ldots\} \\
M & = & \lambda\{\ldots, \angle i, \overrightarrow{\mathtt{x}_j}.(\angle i, \overrightarrow{\mathtt{x}_j}\text{`}M), \ldots\}
\end{array}$$

Figure A.1: Syntax and Equations of Revised-λ

A.2 Languages and Translations

In the rest of this chapter we will look in detail at various languages involving effects, and the translations between them, shown in Fig. A.2. Arrows of the form $\hookrightarrow$ indicate language extensions. For the three languages marked with an asterisk, we provide an equational theory.

Because we are considering so many languages, we have many variants of the same results. To reduce clutter, we omit many of these propositions, where they are straightforward. Furthermore, when treating CBV and CBN, we omit proofs when they are similar to the corresponding proofs for CBPV. We give a summary of the results here.

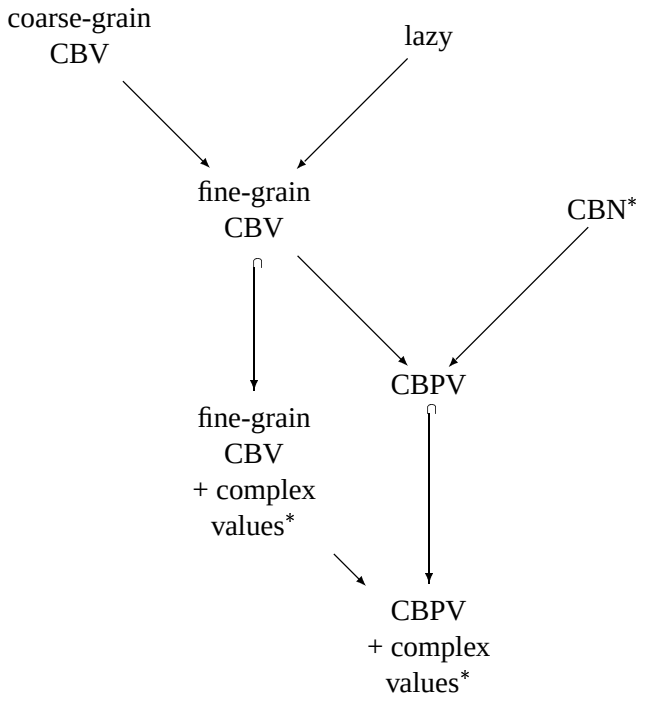


Figure A.2: Effectful Languages

Each language has the following properties:

- the big-step semantics is total, because the only effect we are using in our languages is printing;
- denotational semantics commutes with substitution and with weakening;
- denotational semantics agrees with operational semantics (soundness);
- denotational equality implies observational equivalence.

Each equational theory has the following properties:

- provable equality commutes with substitution and weakening;
- provable equality implies denotational equality;
- provable equality implies observational equivalence.

Each translation has the following properties:

- the translation preserves denotational semantics;
- the translation commutes with substitution and weakening up to provable equality;
- the translation preserves provable equality (where the source language has an equational theory);
- the translation preserves and reflects operational semantics for ground terms;
- the translation reflects observational equivalence.

We would like each translation to have the following properties:

- the translation commutes exactly with substitution and weakening;
- the translation preserves and reflects operational semantics for all terms.

However, these properties usually fail, and to achieve them we have to extend the translation from a function between terms to a relation between terms.

The two translations into CBPV[1] have the following properties:

- every type in the target language is isomorphic to the translation of some type in the source language;
- every term in the target language (of appropriate type and context) is provably equal to the translation of some term in the source language;
- the translation reflects provable equality;
- the translation preserves observational equivalence (full abstraction).

A.3 Call-By-Value

A.3.1 Coarse-Grain Call-By-Value

For CG-CBV we evaluate to the following *terminal* closed terms:

$$T ::= \quad (\hat{\imath}, T_0, \ldots, T_{r_{\hat{\imath}}-1}) \;\mid\; \lambda\{\ldots, \angle i, \overrightarrow{\mathtt{x}_j}.M_i, \ldots\}$$

The relation $M \Downarrow T$ is given inductively by the rules of Fig. A.3. We extend it to printing as in Sect. 3.4.1.

$$\frac{M \Downarrow T \quad N[T/\mathtt{x}] \Downarrow T'}{\mathtt{let}\ \mathtt{x}\ \mathtt{be}\ M.\ N \Downarrow T'}$$

$$\frac{M_0 \Downarrow T_0 \quad \cdots \quad M_{r_{\hat{\imath}}-1} \Downarrow T_{r_{\hat{\imath}}-1}}{(\hat{\imath}, M_0, \ldots, M_{r_{\hat{\imath}}-1}) \Downarrow (\hat{\imath}, T_0, \ldots, T_{r_{\hat{\imath}}-1})}$$

$$\frac{M \Downarrow (\hat{\imath}, \overrightarrow{T_j}) \quad N_{\hat{\imath}}[\overrightarrow{T_j/\mathtt{x}_j}] \Downarrow T}{\mathtt{pm}\ M\ \mathtt{as}\ \{\ldots, (i, \overrightarrow{\mathtt{x}_j}).N_i, \ldots\} \Downarrow T}$$

$$\frac{}{\lambda\{\ldots, \angle i, \overrightarrow{\mathtt{x}_j}.M_i, \ldots\} \Downarrow \lambda\{\ldots, \angle i, \overrightarrow{\mathtt{x}_j}.M_i, \ldots\}}$$

$$\frac{M_0 \Downarrow T_0 \quad \cdots \quad M_{r_{\hat{\imath}}-1} \Downarrow T_{r_{\hat{\imath}}-1} \quad N \Downarrow \lambda\{\ldots, \angle i, \overrightarrow{\mathtt{x}_j}.N_i, \ldots\} \quad N_{\hat{\imath}}[\overrightarrow{T_j/\mathtt{x}_j}] \Downarrow T}{\angle \hat{\imath}, M_0, \ldots, M_{r_{\hat{\imath}}-1}\text{`}N \Downarrow T}$$

Figure A.3: Big-Step Semantics for CG-CBV—No Effects

Proposition 149 Properties of big-step semantics:

[1] We have not investigated these properties for the other translations.

1. If T is terminal, then $T \Downarrow T$.
2. For every closed term M, $M \Downarrow T$ for a unique T.

□

Denotational semantics for printing is given as in Sect. 2.6.2. The type constructors are interpreted as follows.

- $\sum_{i\in I}^{\times_{j\in \$r_i}} A_{ij}$ denotes $\sum_{i\in I}([\![A_{i0}]\!] \times \cdots \times [\![A_{i(r_i-1)}]\!])$.
- $\prod_{i\in I}^{\to_{j\in \$r_i}} A_{ij}B_i$ denotes $\prod_{i\in I}([\![A_{i0}]\!] \to \cdots \to [\![A_{i(r_i-1)}]\!] \to (\mathcal{A}^* \times [\![B_i]\!]))$.

A.3.2 Fine-Grain Call-By-Value

The Language

The coarse-grain CBV language that we have seen suffers from two problems.

1. We had to make an arbitrary choice as to the order of evaluation of tuples and applications.
2. A value V has two denotations: $[\![V]\!]^{\mathsf{val}}$, its denotation as a value, and $[\![V]\!]^{\mathsf{prod}}$, its denotation as a producer.

How can we refine the language to avoid these problems, while leaving unchanged the types and their denotations?

In order to have a single denotation function $[\![-]\!]$, we make a syntactic distinction between values and producers, so that every term is either a value or a producer but not both. Thus we have two judgements

$$\Gamma \vdash^{\mathsf{v}} V : A \qquad \Gamma \vdash^{\mathsf{p}} M : A$$

which respectively say that V is a value of type A and that M is a producer of type A (i.e. M produces a value of type A).

The calculus that this leads to is called *fine-grain CBV*. The terms and big-step semantics are given in Fig. A.4. We do not evaluate values, so the operational semantics is defined only on producers.

It is convenient in FG-CBV to write TA as syntactic sugar for $\prod_{i\in\{*\}} A$. Thus in the printing semantics TA denotes $\mathcal{A}^* \times [\![A]\!]$, and in the Scott semantics TA denotes $[\![A]\!]_\perp$. We write $\texttt{thunk}\ M$ for $\lambda *.M$ and $\texttt{force}\ V$ for $*\text{`}V$. Because of the importance of TA, we present rules and equations for it explicitly, even though they are just special cases of the rules and equations for function types.

Note that in FG-CBV `let` is used only for binding identifiers. The sequencing of producers, which was written in CG-CBV as $\texttt{let x be}\ M.\ N$ is in FG-CBV written more suggestively as $M\ \texttt{to x}.\ N$. Furthermore this is the *only* construct in FG-CBV that sequences producers, so both operational and denotational semantics for other terms (such as application) is much simpler than before. In particular, the arbitrariness present in CG-CBV (evaluation order for application and for tupling) is no longer present in FG-CBV.

FG-CBV is based on Moggi's "monadic metalanguage" [Mog91]. But unlike Moggi's calculus, FG-CBV distinguishes between a producer M of type A and its thunk, a value of type TA.

To add our example effect to the language, we add the rule

$$\frac{\Gamma \vdash^{\mathsf{p}} M : A}{\Gamma \vdash^{\mathsf{p}} \texttt{print}\ c;\ M : A}$$

and adapt and then extend the big-step semantics exactly as in Sect. 3.4.1. We then obtain:

Proposition 150 For every closed producer M, there is a unique m, V such that $M \Downarrow m, V$. □

$$\frac{}{\Gamma, \mathtt{x} : A, \Gamma' \vdash^{\mathsf{v}} \mathtt{x} : A} \qquad \frac{\Gamma \vdash^{\mathsf{v}} V : A \quad \Gamma, \mathtt{x} : A \vdash^{\mathsf{p}} M : B}{\Gamma \vdash^{\mathsf{v}} \mathtt{let}\ \mathtt{x}\ \mathtt{be}\ V.\ M : B}$$

$$\frac{\Gamma \vdash^{\mathsf{v}} V : A}{\Gamma \vdash^{\mathsf{p}} \mathtt{produce}\ V : A} \qquad \frac{\Gamma \vdash^{\mathsf{p}} M : A \quad \Gamma, \mathtt{x} : A \vdash^{\mathsf{p}} N : B}{\Gamma \vdash^{\mathsf{p}} M\ \mathtt{to}\ \mathtt{x}.\ N : B}$$

$$\frac{\overrightarrow{\Gamma \vdash^{\mathsf{v}} V_j : A_{\hat{\imath} j}}}{\Gamma \vdash^{\mathsf{v}} (\hat{\imath}, \overrightarrow{V_j}) : \sum_{i \in I}^{\times_{j \in \$r_i}} A_{ij}} \qquad \frac{\Gamma \vdash^{\mathsf{v}} V : \sum_{i \in I}^{\times_{j \in \$r_i}} A_{ij} \quad \cdots\ \Gamma, \overrightarrow{\mathtt{x}_j : A_{ij}} \vdash^{\mathsf{p}} M_i : B\ \cdots}{\Gamma \vdash^{\mathsf{p}} \mathtt{pm}\ V\ \mathtt{as}\ \{\ldots (i, \overrightarrow{\mathtt{x}}).M_i, \ldots\} : B}$$

$$\frac{\cdots\ \Gamma, \overrightarrow{\mathtt{x}_j : A_{ij}} \vdash^{\mathsf{p}} M_i : B_i\ \cdots}{\Gamma \vdash^{\mathsf{v}} \lambda\{\ldots, \angle i, \overrightarrow{\mathtt{x}_j}.M, \ldots\} : \prod_{i \in I}^{\to_{j \in \$r_i}} A_{ij} B_i} \qquad \frac{\overrightarrow{\Gamma \vdash^{\mathsf{v}} V_j : A_{\hat{\imath} j}} \quad \Gamma \vdash^{\mathsf{v}} W : \prod_{i \in I}^{\to_{j \in \$r_i}} A_{ij} B_i}{\Gamma \vdash^{\mathsf{p}} \angle \hat{\imath}, \overrightarrow{V_j}\text{`}W : B_{\hat{\imath}}}$$

$$\frac{\Gamma \vdash^{\mathsf{p}} M : A}{\Gamma \vdash^{\mathsf{v}} \mathtt{thunk}\ M : TA} \qquad \frac{\Gamma \vdash^{\mathsf{v}} V : TA}{\Gamma \vdash^{\mathsf{p}} \mathtt{force}\ V : A}$$

$$\frac{M[V/\mathtt{x}] \Downarrow W}{\mathtt{let}\ \mathtt{x}\ \mathtt{be}\ V.\ M \Downarrow W}$$

$$\frac{}{\mathtt{produce}\ V \Downarrow V} \qquad \frac{M \Downarrow V \quad N[V/\mathtt{x}] \Downarrow W}{M\ \mathtt{to}\ \mathtt{x}.\ N \Downarrow W}$$

$$\frac{M_{\hat{\imath}}[\overrightarrow{V_j/\mathtt{x}_j}] \Downarrow W}{\mathtt{pm}\ (\hat{\imath}, \overrightarrow{V_j})\ \mathtt{as}\ \{\ldots, (i, \overrightarrow{\mathtt{x}_j}).M_i, \ldots\} \Downarrow W}$$

$$\frac{M_{\hat{\imath}}[\overrightarrow{V_j/\mathtt{x}_j}] \Downarrow W}{\angle \hat{\imath}, \overrightarrow{V_j}\text{`}\lambda\{\ldots, \angle i, \overrightarrow{\mathtt{x}_j}.M_i, \ldots\} \Downarrow W}$$

$$\frac{M \Downarrow W}{\mathtt{force}\ \mathtt{thunk}\ M \Downarrow W}$$

Figure A.4: Terms and Big-Step Semantics for FG-CBV—No Effects

Observational Equivalence

Definition 122 Given two producers $\Gamma \vdash^{\mathsf{p}} M, M' : B$, we say that

1. $M \simeq_{\mathsf{ground}} M'$ when for all ground producer contexts $\mathcal{C}[]$, $\mathcal{C}[M] \Downarrow m, i$ iff $\mathcal{C}[M'] \Downarrow m, i$;
2. $M \simeq_{\mathsf{anytype}} M'$ when for all producer contexts $\mathcal{C}[]$ of any type, $\mathcal{C}[M] \Downarrow m, T$ for some T iff $\mathcal{C}[M'] \Downarrow m, T$ for some T

Similarly for values. □

Proposition 151 The two relations $\simeq_{\mathsf{ground}}$ and $\simeq_{\mathsf{anytype}}$ are the same. □

Denotational Semantics for Printing

The semantics of types is the same as CG-CBV.

If $\Gamma \vdash^{\mathsf{v}} V : A$ then V denotes a function from $[\![\Gamma]\!]$ to $[\![A]\!]$, whereas if $\Gamma \vdash^{\mathsf{p}} M : A$ then M denotes a function from $[\![\Gamma]\!]$ to $\mathcal{A}^* \times [\![A]\!]$. We omit the semantics of terms.

Complex Values and Equational Theory

For all purposes except operational semantics, we extend the FG-CBV calculus with the following rules for *complex values*:

$$\frac{\Gamma \vdash^{\mathsf{v}} V : A \quad \Gamma, \mathtt{x} : A \vdash^{\mathsf{v}} W : A}{\Gamma \vdash^{\mathsf{v}} \mathtt{let\ x\ be}\ V.\ W : A}$$

$$\frac{\Gamma \vdash^{\mathsf{v}} V : \sum_{i \in I}^{\times_{j \in \$r_i}} A_{ij} \quad \cdots \quad \Gamma, \overrightarrow{\mathtt{x}_j : A_{ij}} \vdash^{\mathsf{v}} W_i : B}{\Gamma \vdash^{\mathsf{v}} \mathtt{pm}\ V\ \mathtt{as}\ \{\ldots, (i, \overrightarrow{\mathtt{x}_j}).W_i, \ldots\} : B}$$

We then form the equational theory shown in Fig. A.5. M,N and P range over producers, while V and W range over values.

The equation for `print` is of course specific to our example effect, but there are directly analogous equations for many other effects. For example, if we were considering divergence, we would have an equation

$$\mathtt{diverge\ to\ x}.\ M = \mathtt{diverge}$$

We call this theory (without the `print` equation) the *CBV equational theory*.

Proposition 152 1. There is an effective procedure that given a producer $\Gamma \vdash^{\mathsf{p}} M : A$, possibly containing complex values, returns a producer $\Gamma \vdash^{\mathsf{p}} \tilde{M} : A$ without complex values, such that $M = \tilde{M}$ is provable.

2. There is an effective procedure that, given a closed value $\vdash^{\mathsf{v}} V : A$, possibly containing complex values, returns a closed value $\vdash^{\mathsf{v}} \tilde{V} : A$ without complex values, such that $V = \tilde{V}$ is provable.

□

This is proved like Prop. 26.

β-laws

$$
\begin{array}{rcc}
\texttt{let x be } V.\ M & = & M[V/\texttt{x}] \\
\texttt{let x be } V.\ W & = & W[V/\texttt{x}] \\
(\texttt{produce } V)\ \texttt{to x}.\ M & = & M[V/\texttt{x}] \\
\texttt{pm } (\hat{\imath}, \overrightarrow{V_j})\ \texttt{as } \{\ldots, (i, \overrightarrow{\texttt{x}_j}).M_i, \ldots\} & = & M_{\hat{\imath}}[\overrightarrow{V_j/\texttt{x}_j}] \\
\texttt{pm } (\hat{\imath}, \overrightarrow{V_j})\ \texttt{as } \{\ldots, (i, \overrightarrow{\texttt{x}_j}).W_i, \ldots\} & = & W_{\hat{\imath}}[\overrightarrow{V_j/\texttt{x}_j}] \\
\angle\hat{\imath}, \overrightarrow{V_j}\text{`}\lambda\{\ldots, \angle i, \overrightarrow{\texttt{x}_j}.M_i, \ldots\} & = & M_{\hat{\imath}}[\overrightarrow{V_j/\texttt{x}_j}] \\
\texttt{force thunk } M & = & M
\end{array}
$$

η-laws

$$
\begin{array}{rcc}
M & = & M\ \texttt{to x}.\ \texttt{produce x} \\
M[V/\texttt{z}] & = & \texttt{pm } V\ \texttt{as } \{\ldots, (i, \overrightarrow{\texttt{x}_j}).M[(i, \overrightarrow{\texttt{x}_j})/\texttt{z}], \ldots\} \\
W[V/\texttt{z}] & = & \texttt{pm } V\ \texttt{as } \{\ldots, (i, \overrightarrow{\texttt{x}_j}).W[(i, \overrightarrow{\texttt{x}_j})/\texttt{z}], \ldots\} \\
V & = & \lambda\{\ldots, \angle i, \overrightarrow{\texttt{x}_j}.(\angle i, \overrightarrow{\texttt{x}_j}\text{`}V), \ldots\} \\
V & = & \texttt{thunk force } V
\end{array}
$$

sequencing laws

$$
(M\ \texttt{to x}.\ N)\ \texttt{to y}.\ P \quad = \quad M\ \texttt{to x}.\ (N\ \texttt{to y}.\ P)
$$

`print` laws

$$
(\texttt{print } c;\ M)\ \texttt{to x}.\ N \quad = \quad \texttt{print } c;\ (M\ \texttt{to x}.\ N)
$$

Figure A.5: CBV equations, using conventions of Sect. 1.4.2

$\Gamma \vdash M : A$	$\Gamma \vdash^{\mathsf{p}} M^{\mathsf{cg}} : A$
$\texttt{x}$	$\texttt{produce x}$
$\texttt{let x be } M.\ N$	$M^{\mathsf{cg}}\ \texttt{to x}.\ N^{\mathsf{cg}}$
$(\hat{\imath}, M_0, \ldots, M_{r_{\hat{\imath}}-1})$	$M_0^{\mathsf{cg}}\ \texttt{to x}_0.\ \ldots M_{r_{\hat{\imath}}-1}^{\mathsf{cg}}\ \texttt{to x}_{r_{\hat{\imath}}-1}.\ \texttt{produce } (\hat{\imath}, \overrightarrow{\texttt{x}_j})$
$\texttt{pm } M\ \texttt{as } \{\ldots, (i, \overrightarrow{\texttt{x}_j}).N_i, \ldots\}$	$M^{\mathsf{cg}}\ \texttt{to z}.\ \texttt{pm z as } \{\ldots, (i, \overrightarrow{\texttt{x}_j}).N_i^{\mathsf{cg}}, \ldots\}$
$\lambda\{\ldots, \angle i, \overrightarrow{\texttt{x}_j}.M_i, \ldots\}$	$\texttt{produce } \lambda\{\ldots, \angle i, \overrightarrow{\texttt{x}_j}.M_i^{\mathsf{cg}}, \ldots\}$
$\angle\hat{\imath}, M_0, \ldots, M_{r_{\hat{\imath}}-1}\text{`}N$	$M_0^{\mathsf{cg}}\ \texttt{to x}_0.\ \ldots M_{r_{\hat{\imath}}-1}^{\mathsf{cg}}\ \texttt{to x}_{r_{\hat{\imath}}-1}.\ N^{\mathsf{cg}}\ \texttt{to } f.\ \angle\hat{\imath}, \overrightarrow{\texttt{x}_j}\text{`}f$
$\texttt{print } c;\ M$	$\texttt{print } c;\ M^{\mathsf{cg}}$

$\Gamma \vdash V : A$	$\Gamma \vdash^{\mathsf{v}} V^{\mathsf{cgval}} : A$
$\texttt{x}$	$\texttt{x}$
$(\hat{\imath}, V_0, \ldots, V_{r_{\hat{\imath}}-1})$	$(\hat{\imath}, V_0^{\mathsf{cgval}}, \ldots, V_{r_{\hat{\imath}}-1}^{\mathsf{cgval}})$
$\lambda\{\ldots, \angle i, \overrightarrow{\texttt{x}_j}.M_i, \ldots\}$	$\lambda\{\ldots, \angle i, \overrightarrow{\texttt{x}_j}.M_i^{\mathsf{cg}}, \ldots\}$

Figure A.6: Translation from CG-CBV to FG-CBV

A.3.3 From CG-CBV To FG-CBV

The translation from CG-CBV to FG-CBV is given in Fig. A.6. It is in two parts: $-^{\mathsf{cg}}$ is defined on producers and $-^{\mathsf{cgval}}$ on values. The translation $-^{\mathsf{cg}}$ reflects our choice of evaluation order for CG-CBV.

We now address the technical properties of the translation. It neither commutes with substitution nor preserves operational semantics.

substitution It is not the case that $(M[V/\mathtt{x}])^{\mathsf{cg}}$ and $M^{\mathsf{cg}}[V^{\mathsf{cgval}}/\mathtt{x}]$ are the same term (although they will are provably equal in the CBV equational theory of Fig. A.5). To see this, let M be $\mathtt{x}$ and V be $(0,(0))$, where 0 is a tag. Then

$$\begin{array}{rcl}(M[V/\mathtt{x}])^{\mathsf{cg}} & = & \mathtt{produce}\,(0)\,\mathtt{to}\,\mathtt{x}.\,\mathtt{produce}\,(0,\mathtt{x}) \\ M^{\mathsf{cg}}[V^{\mathsf{cgval}}/\mathtt{x}] & = & \mathtt{produce}\,(0,(0))\end{array}$$

operational semantics It is not the case that $M \Downarrow m, V$ implies $M^{\mathsf{cg}} \Downarrow m, V^{\mathsf{cgval}}$. For example, let M be $\mathtt{let}\ \mathtt{x}\ \mathtt{be}\ (0,(0)).\ \lambda *.\mathtt{x}$.

The issue is that CG-CBV cannot recognize that what is substituted is not a certain kind of producer (to be trivially reevaluated when required) but something genuinely different—a value.

To salvage what we can, we proceed as follows. First, we write the two translations as relations $\mapsto^{\mathsf{cg}}$ and $\mapsto^{\mathsf{cgval}}$ from CG-CBV to FG-CBV terms. We can present Fig. A.6 by rules such as these:

$$\frac{M \mapsto^{\mathsf{cg}} M' \quad N \mapsto^{\mathsf{cg}} N'}{\mathtt{let}\ \mathtt{x}\ \mathtt{be}\ M.\ N \mapsto^{\mathsf{cg}} M'\ \mathtt{to}\ \mathtt{x}.\ N'}$$

$$\frac{\cdots\ P_i \mapsto^{\mathsf{cg}} P_i'\ \cdots}{\lambda\{\ldots,\angle i, \overrightarrow{\mathtt{x}_j}.P_i,\ldots\} \mapsto^{\mathsf{cgval}} \lambda\{\ldots,\angle i, \overrightarrow{\mathtt{x}_j}.P_i',\ldots\}}$$

To these rules we add the following

$$\frac{M \mapsto^{\mathsf{cg}} \mathtt{produce}\ V_0\ \mathtt{to}\ \mathtt{x}_0.\ \ldots \mathtt{produce}\ V_{r_i-1}\ \mathtt{to}\ \mathtt{x}_{r_i-1}.\ \mathtt{produce}\ (\hat{\imath}, \overrightarrow{\mathtt{x}_j})}{M \mapsto^{\mathsf{cg}} \mathtt{produce}\ (\hat{\imath}, \overrightarrow{V_j})}$$

(By analogy with convention (3) in Sect. 1.4.2, it is assumed that $\mathtt{x}_j$ is not in the context of $V_{j'}$.) We have thus defined non-functional relations $\mapsto^{\mathsf{cg}}$ and $\mapsto^{\mathsf{cgval}}$, and we will show that they commute with substitution and preserve and reflect operational semantics.

Proposition 153 For any producer M, we have $M \mapsto^{\mathsf{cg}} M^{\mathsf{cg}}$, and if $M \mapsto^{\mathsf{cg}} N$ then $N = M^{\mathsf{cg}}$ is provable in the CBV equational theory of Fig. A.5; similarly for values. □

Proposition 154

1. If $V \mapsto^{\mathsf{cgval}} V'$ then $V \mapsto^{\mathsf{cg}} \mathtt{produce}\ V'$.
2. If $M \mapsto^{\mathsf{cg}} M'$ and $V \mapsto^{\mathsf{cgval}} V'$ then $M[V/\mathtt{x}] \mapsto^{\mathsf{cg}} M'[V'/\mathtt{x}]$.
3. If $W \mapsto^{\mathsf{cg}} W'$ and $V \mapsto^{\mathsf{cgval}} V'$ then $W[V/\mathtt{x}] \mapsto^{\mathsf{cg}} W'[V'/\mathtt{x}]$.

□

Proposition 155

1. If $M \Downarrow V$ and $M \mapsto^{\mathsf{cg}} M'$, then, for some V', $M' \Downarrow V'$ and $V \mapsto^{\mathsf{cgval}} V'$.
2. If $M \mapsto^{\mathsf{cg}} M'$ and $M' \Downarrow V'$, then, for some V, $M \Downarrow V$ and $V \mapsto^{\mathsf{cgval}} V'$.

□

To prove this, we introduce the following.

Definition 123 1. In CG-CBV, the following producers are *safe*:

$$\begin{array}{rcl} S & ::= & \mathtt{x} \mid \mathtt{let\ x\ be}\ S.\ S \mid (\hat{\imath}, \overrightarrow{S_j}) \\ & & \mid \mathtt{pm}\ S\ \mathtt{as}\ \{\ldots, (i, \overrightarrow{\mathtt{x}_j}).S_i, \ldots\} \mid \lambda\{\ldots, \angle i, \overrightarrow{\mathtt{x}_j}.M_i, \ldots\} \end{array}$$

2. In FG-CBV the following producers are *safe*:

$$\begin{array}{rcl} S & ::= & \mathtt{produce}\ V \mid \mathtt{let\ x\ be}\ V.\ S \mid S\ \mathtt{to\ x}.\ S \\ & & \mid \mathtt{pm}\ V\ \mathtt{as}\ \{\ldots, (i, \overrightarrow{\mathtt{x}_j}).S_i, \ldots\} \end{array}$$

□

In summary, a producer is safe iff, inside it, every application occurs in the scope of a λ.

Lemma 156 Suppose $A_0, \ldots, A_{n-1} \vdash M \mapsto^{\mathsf{cg}} M' : B$. Then

1. M is safe iff M' is safe.
2. Suppose M is safe and $U_0 \mapsto^{\mathsf{cgval}} U_0', \ldots, U_{n-1} \mapsto^{\mathsf{cgval}} U_{n-1}'$ (U_i and U_i' may not be safe).
   - If $M[\overrightarrow{U_i/\mathtt{x}_i}] \Downarrow V$, then, for some V', $M'[\overrightarrow{U_i'/\mathtt{x}_i}] \Downarrow V'$ and $V \mapsto^{\mathsf{cgval}} V'$.
   - If $M'[\overrightarrow{U_i'/\mathtt{x}_i}] \Downarrow V'$, then, for some V, $M[\overrightarrow{U_i/\mathtt{x}_i}] \Downarrow V$ and $V \mapsto^{\mathsf{cgval}} V'$.

□

We prove this by induction on $M \mapsto^{\mathsf{cg}} M'$. Finally we prove Prop. 155 by induction on $M \Downarrow V$ (for (1)) and on $M' \Downarrow V'$ (for (2)).

A.4 Call-By-Name

For CBN we evaluate to the following *terminal* closed terms:

$$T ::= \quad (\hat{\imath}, \overrightarrow{M_j}) \mid \quad \mid \lambda\{\ldots, \angle i, \overrightarrow{\mathtt{x}_j}.M_i, \ldots\}$$

The relation $M \Downarrow T$ is defined in Fig. A.7.

Proposition 157 For each closed term M, we have $M \Downarrow T$ for a unique terminal term T. □

$$\frac{}{(\hat{\imath}, \overrightarrow{M_j}) \Downarrow (\hat{\imath}, \overrightarrow{M_j})} \qquad \frac{N[M/\mathtt{x}] \Downarrow T}{\mathtt{let\ x\ be}\ M.\ N \Downarrow T}$$

$$\frac{M \Downarrow (\hat{\imath}, \overrightarrow{N_j}) \quad P_{\hat{\imath}}[\overrightarrow{N_j/\mathtt{x}_j}] \Downarrow T}{\mathtt{pm}\ M\ \mathtt{as}\ \{\ldots, (i, \overrightarrow{\mathtt{x}_j}).P_i, \ldots\} \Downarrow T}$$

$$\frac{}{\lambda\{\ldots, \angle i, \overrightarrow{\mathtt{x}_j}.M_i, \ldots\} \Downarrow \lambda\{\ldots, \angle i, \overrightarrow{\mathtt{x}_j}.M_i, \ldots\}} \qquad \frac{N \Downarrow \lambda\{\ldots, \angle i, \overrightarrow{\mathtt{x}_j}.N_i, \ldots\} \quad N_{\hat{\imath}}[\overrightarrow{M_j/\mathtt{x}_j}] \Downarrow T}{\angle \hat{\imath}, \overrightarrow{M_j}\text{`}N \Downarrow T}$$

Figure A.7: Big-Step Semantics for CBN—No Effects

To add our example effect, we adapt and extend Fig. A.7 exactly as in Sect. 3.4.1. We then have

Proposition 158 For every closed term M, there is a unique m, T such that $M \Downarrow m, T$. □

The printing semantics follows Sect. 2.7.4. The interpretation of type constructors is given by

- If $[\![A_{ij}]\!] = (X_{ij}, *)$, then $[\![\sum_{i\in I}^{\times_{j\in \$r_i}} A_{ij}]\!]$ is the free $\mathcal{A}$-set on $\sum_{i\in I}(X_{i0} \times \cdots \times X_{i(r_i-1)})$.
- If $[\![A_{ij}]\!] = (X_{ij}, *)$ and $[\![B_i]\!] = (Y_i, *)$ then $[\![\prod_{i\in I}^{\rightarrow_{j\in \$r_i}} A_{ij} B_i]\!]$ is the $\mathcal{A}$-set $\prod_{i\in I}(X_{i0} \to \cdots \to X_{i(r_i-1)} \to (Y_i, *))$

We define an equational theory for CBN , whose axioms are the equations in Fig. A.8.

$$\beta\textbf{-laws}$$

$$\begin{array}{rcl}
\mathtt{let\ x\ be}\ M.\ N & = & N[M/\mathtt{x}] \\
\mathtt{pm}\ (\hat{\imath}, \overrightarrow{M_j})\ \mathtt{as}\ \{\ldots, (i, \overrightarrow{\mathtt{x}_j}).N_i, \ldots\} & = & N_{\hat{\imath}}[\overrightarrow{M_j/\mathtt{x}_j}] \\
\angle\hat{\imath}, \overrightarrow{M_j}\text{`}\lambda\{\ldots, \angle i, \overrightarrow{\mathtt{x}_j}.N_i, \ldots\} & = & N_{\hat{\imath}}[\overrightarrow{M_j/\mathtt{x}_j}]
\end{array}$$

$$\eta\textbf{-laws}$$

$$M = \lambda\{\ldots, \angle i, \overrightarrow{\mathtt{x}_j}.(\angle i, \overrightarrow{\mathtt{x}_j}\text{`}M), \ldots\}$$

pattern-matching laws

$$\begin{array}{rcl}
M & = & \mathtt{pm}\ M\ \mathtt{as}\ \{\ldots, (i, \overrightarrow{\mathtt{x}_j}).(i, \overrightarrow{\mathtt{x}_j}), \ldots\} \\
\mathtt{pm}\ M\ \mathtt{as}\ \{\ldots, (i, \overrightarrow{\mathtt{x}_j}).(\mathtt{pm}\ N_i\ \mathtt{as}\ \{\ldots, (k, \overrightarrow{\mathtt{y}_l}.P_k, \ldots\}), \ldots\} & & \\
\quad = & \mathtt{pm}\ (\mathtt{pm}\ M\ \mathtt{as}\ \{\ldots, (i, \overrightarrow{\mathtt{x}_j}).N_i, \ldots\})\ \mathtt{as}\ \{\ldots, (k, \overrightarrow{\mathtt{y}_l}).P_k, \ldots\} & \\
\mathtt{pm}\ M\ \mathtt{as}\ \{\ldots, (i, \overrightarrow{\mathtt{x}_j}).\lambda\{\ldots, \angle k, \overrightarrow{\mathtt{y}_l}.N_{ik}, \ldots\}, \ldots\} & & \\
\quad = & \lambda\{\ldots, \angle k, \overrightarrow{\mathtt{y}_l}.(\mathtt{pm}\ M\ \mathtt{as}\ \{\ldots, (i, \overrightarrow{\mathtt{x}_j}).N_{ik}, \ldots\}), \ldots\} &
\end{array}$$

$$\mathtt{print}\ \textbf{laws}$$

$$\begin{array}{rcl}
\mathtt{print}\ c;\ (\mathtt{pm}\ M\ \mathtt{as}\ \{\ldots, (i, \overrightarrow{\mathtt{x}_j}).N_i, \ldots\}) & = & \mathtt{pm}\ (\mathtt{print}\ c;\ M)\ \mathtt{as}\ \{\ldots, (i, \overrightarrow{\mathtt{x}_j}).N_i, \ldots\} \\
\mathtt{print}\ c;\ \lambda\{\ldots, \angle i, \overrightarrow{\mathtt{x}_j}.M_i, \ldots\} & = & \lambda\{\ldots, \angle i, \overrightarrow{\mathtt{x}_j}.(\mathtt{print}\ c;\ M_i), \ldots\}
\end{array}$$

Figure A.8: CBN equations, using conventions of Sect. 1.4.2

The equations for `print` are of course specific to our example effect, but there would be directly analogous equations for many other effects. For example, if we were considering divergence, we would have equations:

$$\begin{array}{rcl}
\mathtt{diverge} & = & \mathtt{pm}\ \mathtt{diverge}\ \mathtt{as}\ \{\ldots, (i, \overrightarrow{\mathtt{x}_j}).N_i, \ldots\} \\
\mathtt{diverge} & = & \lambda\{\ldots, \angle i, \overrightarrow{\mathtt{x}_j}.\mathtt{diverge}, \ldots\}
\end{array}$$

We call this theory (without the `print` equations) the *CBN equational theory*.

A.5 The Lazy Paradigm

Recall from Sect. 2.7.3 that the lazy paradigm is defined to have the same operational semantics as CBN, but to use $\simeq_{\mathsf{anytype}}$ rather than $\simeq_{\mathsf{ground}}$ as its observational equivalence. Thus its models must not satisfy, for example, the η-law for function types.

We shall say little about the lazy paradigm, because it is subsumed within FG-CBV. First, as in CG-CBV, we introduce a value/producer terminology.

Definition 124

A *value* is a lazy term whose substitution instances are all terminal. Unlike in CBV an identifier is not a value because any term can be substituted for it, so values are just the following:

$$V ::= \quad (\hat{\imath}, \overrightarrow{M_j}) \mid \quad \mid \lambda\{\ldots, \angle i, \overrightarrow{\mathtt{x}_j}.M_i, \ldots\}$$

A *producer* is a lazy term (so-called because a closed producer *produces* a closed value). □

We translate the lazy language into FG-CBV in Fig. A.9. The translation is essentially that given in [HD97]. Like the translation from CG-CBV to FG-CBV, the translation on terms is defined in two parts: $-^{\mathsf{lazy}}$ is defined on producers, and $-^{\mathsf{lazyval}}$ is defined on values.

The translation is motivated as follows.

- Identifers in the lazy language are bound to unevaluated terms, so we regard them (from a CBV perspective) as bound to thunks.
- Similarly, tuple-components and operands in the lazy language are unevaluated terms, so we regard them as thunks.
- Consequently, identifiers, tuple-components and operands all have type of the form TA—a thunk type.

The translation from lazy to FG-CBV is very similar to the translation from CBN to CBPV.

C	C^{lazy}
$\Sigma_{i\in I}^{\times_{j\in \$r_i}} A_{ij}$	$\Sigma_{i\in I}^{\times_{j\in \$r_i}} TA_{ij}^{\mathsf{lazy}}$
$\prod_{i\in I}^{\rightarrow_{j\in \$r_i}} A_{ij}B_i$	$\prod_{i\in I}^{\rightarrow_{j\in \$r_i}} TA_{ij}^{\mathsf{lazy}} B_i^{\mathsf{lazy}}$

$A_0, \ldots, A_{n-1} \vdash M : B$	$TA_0^{\mathsf{lazy}}, \ldots, TA_{n-1}^{\mathsf{lazy}} \vdash^{\mathsf{p}} M^{\mathsf{lazy}} : B^{\mathsf{lazy}}$
$\mathtt{x}$	$\mathtt{force}\ \mathtt{x}$
$\mathtt{let}\ \mathtt{x}\ \mathtt{be}\ M.\ N$	$\mathtt{let}\ \mathtt{x}\ \mathtt{be}\ \mathtt{thunk}\ M^{\mathsf{lazy}}.\ N^{\mathsf{lazy}}$
$(\hat{\imath}, M_0, \ldots, M_{r_i-1})$	$\mathtt{produce}\ (\hat{\imath}, \mathtt{thunk}\ M_0^{\mathsf{lazy}}, \ldots, \mathtt{thunk}\ M_{r_i-1}^{\mathsf{lazy}})$
$\mathtt{pm}\ M\ \mathtt{as}\ \{\ldots, (i, \overrightarrow{\mathtt{x}_j}).N_j, \ldots\}$	$M^{\mathsf{lazy}}\ \mathtt{to}\ \mathtt{z}.\ \mathtt{pm}\ \mathtt{z}\ \mathtt{as}\ \{\ldots, (i, \overrightarrow{\mathtt{x}_j}).N_j^{\mathsf{lazy}}, \ldots\}$
$\lambda\{\ldots, \angle i, \overrightarrow{\mathtt{x}_j}.M_i, \ldots\}$	$\mathtt{produce}\ \lambda\{\ldots, \angle i, \overrightarrow{\mathtt{x}_j}.M_i^{\mathsf{lazy}}, \ldots\}$
$\angle\hat{\imath}, M_0, \ldots, M_{r_i-1}\text{`}N$	$N^{\mathsf{lazy}}\ \mathtt{to}\ \mathtt{f}.\ \angle\hat{\imath}, (\mathtt{thunk}\ M_0), \ldots, (\mathtt{thunk}\ M_{r_i-1})\text{`}f$
$\mathtt{print}\ c;\ M$	$\mathtt{print}\ c;\ M^{\mathsf{lazy}}$

$A_0, \ldots, A_{n-1} \vdash V : B$	$TA_0^{\mathsf{lazy}}, \ldots, TA_{n-1}^{\mathsf{lazy}} \vdash^{\mathsf{v}} V^{\mathsf{lazyval}} : B^{\mathsf{lazy}}$
$(\hat{\imath}, M_0, \ldots, M_{r_i-1})$	$(\hat{\imath}, \mathtt{thunk}\ M_0^{\mathsf{lazy}}, \ldots, \mathtt{thunk}\ M_{r_i-1}^{\mathsf{lazy}})$
$\lambda\{\ldots, \angle i, \overrightarrow{\mathtt{x}_j}.M_i, \ldots\}$	$\lambda\{\ldots, \angle i, \overrightarrow{\mathtt{x}_j}.M_i^{\mathsf{lazy}}, \ldots\}$

Figure A.9: Translation from lazy to FG-CBV: types, producers, values

This translation does not commute with substitution or preserve operational semantics precisely—only up to the prefix `force thunk`. (Recall that in the CBV equational theory, we have $\mathtt{force}\ \mathtt{thunk}\ M = M$.)

substitution It is not the case that $(M[N/\mathtt{x}])^{\mathsf{lazy}}$ and $M^{\mathsf{lazy}}[\mathtt{thunk}\ N^{\mathsf{lazy}}/\mathtt{x}]$ are the same term (although they will be provably equal in the CBV equational theory).. To see this, let M be $\mathtt{x}$. Then $(M[N/\mathtt{x}])^{\mathsf{lazy}} = N^{\mathsf{lazy}}$ but $M^{\mathsf{lazy}}[\mathtt{thunk}\ N^{\mathsf{lazy}}/\mathtt{x}] = \mathtt{force}\ \mathtt{thunk}\ N^{\mathsf{lazy}}$.

operational semantics It is not the case that $M \Downarrow m, V$ implies $M^{\mathsf{lazy}} \Downarrow m, V^{\mathsf{lazyval}}$. For example, let M be $\mathtt{let}\ \mathtt{x}\ \mathtt{be}\ (0).\ (0, \mathtt{x})$.

To make the relationship precise, we proceed as follows. First, we write the two translations as relations $\mapsto^{\mathsf{lazy}}$ and $\mapsto^{\mathsf{lazyval}}$ from lazy to FG-CBV terms. We can present Fig. A.9 by rules such as these:

$$\frac{M \mapsto^{\mathsf{lazy}} M' \quad N \mapsto^{\mathsf{lazy}} N'}{\mathtt{let\ x\ be}\ M.\ N \mapsto^{\mathsf{lazy}} \mathtt{let\ x\ be}\ (\mathtt{thunk}\ M').\ N'}$$

$$\frac{\cdots\ P_i \mapsto^{\mathsf{lazy}} P_i'\ \cdots}{\lambda\{\ldots, \angle i, \overrightarrow{\mathtt{x}_j}.P_i, \ldots\} \mapsto^{\mathsf{lazyval}} \lambda\{\ldots, \angle i, \overrightarrow{\mathtt{x}_j}.P_i', \ldots\}}$$

To these rules we add the following:

$$\frac{M \mapsto^{\mathsf{lazy}} M'}{M \mapsto^{\mathsf{lazy}} \mathtt{force\ thunk}\ M'}$$

Proposition 159 For any producer M, we have $M \mapsto^{\mathsf{lazy}} M^{\mathsf{lazy}}$, and if $M \mapsto^{\mathsf{lazy}} M'$ then $M' = M^{\mathsf{lazy}}$ is provable in the CBV equational theory; similarly for values. □

Proposition 160

1. If $V \mapsto^{\mathsf{lazyval}} V'$ then $V \mapsto^{\mathsf{lazy}} \mathtt{produce}\ V'$.
2. If $M \mapsto^{\mathsf{lazy}} M'$ and $N \mapsto^{\mathsf{lazyval}} N'$ then $M[N/\mathtt{x}] \mapsto^{\mathsf{lazy}} M'[\mathtt{thunk}\ N'/\mathtt{x}]$.
3. If $W \mapsto^{\mathsf{lazyval}} W'$ and $N \mapsto^{\mathsf{lazyval}} N'$ then $W[N/\mathtt{x}] \mapsto^{\mathsf{lazyval}} W'[\mathtt{thunk}\ N'/\mathtt{x}]$.

□

Proposition 161

1. If $M \Downarrow V$ and $M \mapsto^{\mathsf{lazy}} M'$, then, for some V', $M' \Downarrow V'$ and $V \mapsto^{\mathsf{lazyval}} V'$.
2. If $M \mapsto^{\mathsf{lazy}} M'$ and $M' \Downarrow V'$, then, for some V, $M \Downarrow V$ and $V \mapsto^{\mathsf{lazyval}} V'$.

□

We prove these by induction primarily on $\Downarrow$ and secondarily on $\mapsto^{\mathsf{lazy}}$.

A.6 Subsuming FG-CBV and CBN

The translations into CBPV are easily obtained by considering denotational semantics.

A.6.1 From FG-CBV to CBPV

The translation from FG-CBV is given in Fig. A.10.

Proposition 162 For a producer M, $(M[V/\mathtt{x}])^{\mathsf{v}}$ and $M^{\mathsf{v}}[V^{\mathsf{v}}/\mathtt{x}]$ are the same term; and similarly for values. □

Proposition 163 For producers $\Gamma \vdash^{\mathsf{p}} M, N : A$, if $M = N$ is provable in the CBV theory then $M^{\mathsf{v}} = N^{\mathsf{v}}$ is provable in the CBPV theory; and similarly for values. □

Proposition 164 The translation $-^{\mathsf{v}}$ preserves and reflects operational semantics:

1. if $M \Downarrow V$ then $M^{\mathsf{v}} \Downarrow \mathtt{produce}\ V^{\mathsf{v}}$;
2. if $M^{\mathsf{v}} \Downarrow \mathtt{produce}\ V'$ then $M \Downarrow V$ for some V such that $V^{\mathsf{v}} = V'$.

□

C	C^{v} (a value type)
$\sum_{i\in I}^{\times_{j\in \$r_i}} A_{ij}$	$\sum_{i\in I}(A^{\mathsf{v}}_{i0}\times\cdots\times A^{\mathsf{v}}_{i(r_i-1)})$
$\prod_{i\in I}^{\to_{j\in \$r_i}} A_{ij}B_i$	$U\prod_{i\in I}(A^{\mathsf{v}}_{i0}\to\cdots\to A^{\mathsf{v}}_{i(r_i-1)}\to FB^{\mathsf{v}}_i)$
TA	UFA^{v}

$A_0,\dots,A_{n-1}\vdash^{\mathsf{v}} V : B$	$A^{\mathsf{v}}_0,\dots,A^{\mathsf{v}}_{n-1}\vdash^{\mathsf{v}} V^{\mathsf{v}} : B^{\mathsf{v}}$
$\mathtt{x}$	$\mathtt{x}$
$(\hat{\imath}, V_0,\dots,V_{r_{\hat{\imath}}-1})$	$(\hat{\imath},(V^{\mathsf{v}}_0,\dots,V^{\mathsf{v}}_{r_{\hat{\imath}}-1}))$
$\lambda\{\dots,\angle i,\mathtt{x}_0,\dots,\mathtt{x}_{r_i-1}.M_i,\dots\}$	$\mathtt{thunk}\,\lambda\{\dots,i.\lambda\mathtt{x}_0.\dots.\lambda\mathtt{x}_{r_i-1}.M^{\mathsf{v}}_i,\dots\}$
$\mathtt{thunk}\,M$	$\mathtt{thunk}\,M^{\mathsf{v}}$

$A_0,\dots,A_{n-1}\vdash^{\mathsf{p}} M : C$	$A^{\mathsf{v}}_0,\dots,A^{\mathsf{v}}_{n-1}\vdash^{\mathsf{c}} M^{\mathsf{v}} : FC^{\mathsf{v}}$
$\mathtt{let\ x\ be}\ V.\ M$	$\mathtt{let\ x\ be}\ V^{\mathsf{v}}.\ M^{\mathsf{v}}$
$\mathtt{produce}\,V$	$\mathtt{produce}\,V^{\mathsf{v}}$
$M\ \mathtt{to\ x}.\ N$	$M^{\mathsf{v}}\ \mathtt{to\ x}.\ N^{\mathsf{v}}$
$\mathtt{pm}\ V\ \mathtt{as}\ \{\dots,(i,\overrightarrow{\mathtt{x}_j}).M_i,\dots\}$	$\mathtt{pm}\ V^{\mathsf{v}}\ \mathtt{as}\ \{\dots,(i,(\overrightarrow{\mathtt{x}_j})).M^{\mathsf{v}}_i,\dots\}$
$\angle\hat{\imath},V_0,\dots,V_{r_{\hat{\imath}}-1}\text{`}W$	$V^{\mathsf{v}}_{r_{\hat{\imath}}-1}\text{`}\dots\text{`}V^{\mathsf{v}}_0\text{`}\hat{\imath}\text{`}\mathtt{force}\,W^{\mathsf{v}}$
$\mathtt{force}\,V$	$\mathtt{force}\,V^{\mathsf{v}}$
$\mathtt{print}\,c;\ M$	$\mathtt{print}\,c;\ M^{\mathsf{v}}$

When we add complex values to the source language FG-CBV, we must add them also to the target language CBPV, and we then extend the translation as follows:

$A_0,\dots,A_{n-1}\vdash^{\mathsf{v}} V : B$	$A^{\mathsf{v}}_0,\dots,A^{\mathsf{v}}_{n-1}\vdash^{\mathsf{v}} V^{\mathsf{v}} : B^{\mathsf{v}}$
$\mathtt{let\ x\ be}\ V.\ W$	$\mathtt{let\ x\ be}\ V^{\mathsf{v}}.\ W^{\mathsf{v}}$
$\mathtt{pm}\ V\ \mathtt{as}\ \{\dots,(i,\overrightarrow{\mathtt{x}_j}).W_i,\dots\}$	$\mathtt{pm}\ V^{\mathsf{v}}\ \mathtt{as}\ \{\dots,(i,(\overrightarrow{\mathtt{x}_j})).W^{\mathsf{v}}_i,\dots\}$

Figure A.10: Translation of FG-CBV types, values and producers

A.6.2 From CBPV Back To FG-CBV

Our aim is to prove the following.

Proposition 165 1. Any CBPV value type A is isomorphic to B^{v} for some CBV type B.

2. For any CBPV producer $A_0^{\mathsf{v}}, \ldots, A_{n-1}^{\mathsf{v}} \vdash^{\mathsf{c}} N : FB^{\mathsf{v}}$ there is an FG-CBV producer $A_0, \ldots, A_{n-1} \vdash^{\mathsf{p}} M : B$ such that $M^{\mathsf{v}} = N$ is provable in CBPV; and similarly for values.

3. For any FG-CBV producers $\Gamma \vdash^{\mathsf{p}} M, M' : B$, if $M^{\mathsf{v}} = M'^{\mathsf{v}}$ is provable in CBPV then $M = M'$ is provable in FG-CBV; and similarly for values.

(2)–(3) can be extended to terms with holes i.e. contexts. □

Corollary 166 (Full Abstraction) For any FG-CBV producers $A_0, \ldots, A_{n-1} \vdash^{\mathsf{p}} M, M' : B$, if $M \simeq M'$ then $M^{\mathsf{v}} \simeq M'^{\mathsf{v}}$; and similarly for values. (The converse is trivial.) □

Proof Suppose $\mathcal{C}[M^{\mathsf{v}}] \Downarrow m, \mathtt{produce}\ i$, for some CBPV ground context $\mathcal{C}$ of type $F\sum_{i\in I} 1$. Construct a FG-CBV context $\mathcal{C}'$ such that $\mathcal{C}'^{\mathsf{v}} = \mathcal{C}$ is provable in CBPV. Then we reason as follows.

1. Since in CBPV provable equality implies observational equivalence, $(\mathcal{C}'[M])^{\mathsf{v}} \Downarrow m, \mathtt{produce}\, i$ using the fact that $(\mathcal{C}'[M])^{\mathsf{v}}$ is precisely $\mathcal{C}'^{\mathsf{v}}[M^{\mathsf{v}}]$.

2. By Prop. 164(2), $\mathcal{C}'[M] \Downarrow m, i$.

3. Since $M \simeq M'$, we have $\mathcal{C}'[M'] \Downarrow m, i$.

4. By Prop. 164(1), $(\mathcal{C}'[M'])^{\mathsf{v}} \Downarrow m, i$.

5. Since in CBPV provable equality implies observational equivalence, $\mathcal{C}[M'^{\mathsf{v}}] \Downarrow m, \mathtt{produce}\, i$.

The proof for values is similar. □

To prove Prop. 165, we give a translation $-^{\mathsf{v}^{-1}}$ from CBPV to FG-CBV. (We shall see that, up to isomorphism, it is inverse to $-^{\mathsf{v}}$.) At first glance, it is not apparent how to make such a translation. For while it will translate a value type into a CBV type, what will it translate a computation type into? The answer is: a family of pairs of CBV types $\{(B_i, C_i)\}_{i\in I}$. To see the principle of the translation, we notice that in CBPV any computation type must be isomorphic to a type of the form $\prod_{i\in I}(B_i \to FC_i)$. (This is discussed in Sect. 4.7.3.)

This motivates the following.

Definition 125

- A *CBV pseudo-computation-type* is a family $\{(B_i, C_i)\}_{i\in I}$ of pairs of CBV types.
- Let $A_0, \ldots, , A_{n-1}$ be a sequence of CBV types and let $\{(B_i, C_i)\}_{i\in I}$ be a CBV pseudo-computation-type. We write

$$A_0, \ldots, A_{n-1} \vdash^{\mathsf{c}}_{\mathsf{CBV}} \{M_i\}_{i\in I} : \{(B_i, C_i)\}_{i\in I}$$

to mean that, for each $i \in I$, M_i is an FG-CBV producer $A_0, \ldots, A_{n-1}, \mathtt{n} : B_i \vdash M_i : C_i$. We say that $\{M_i\}_{i\in I}$ is an *FG-CBV pseudo-computation* of type $\{(B_i, C_i)\}_{i\in I}$ on the context $A_0, \ldots, , A_{n-1}$. Given two such pseudo-computations $\{M_i\}_{i\in I}$ and $\{N_i\}_{i\in I}$, we say that they are *provably equal in CBV* when for each $i \in I$ the equation $M_i = N_i$ is provable in CBV.

□

Before presenting the reverse translation, we extend the forward translation as follows:

- Given a CBV pseudo-computation-type $\{(A_i, B_i)\}_{i\in I}$, we write $\{(A_i, B_i)\}^{\mathsf{v}}_{i\in I}$ for $\prod_{i\in I}(A_i^{\mathsf{v}} \to FB_i^{\mathsf{v}})$.
- Given an FG-CBV pseudo-computation

$$A_0, \ldots, A_{n-1} \vdash^{\mathsf{c}}_{\mathsf{CBV}} \{M_i\}_{i\in I} : \{(B_i, C_i)\}_{i\in I}$$

  we write $\{M_i\}^{\mathsf{v}}_{i\in I}$ for

$$A_0^{\mathsf{v}}, \ldots, A_{n-1}^{\mathsf{v}} \vdash^{\mathsf{c}} \lambda\{\ldots, i.\lambda \mathtt{n} M_i^{\mathsf{v}}, \ldots\} : \{(A_i, B_i)\}^{\mathsf{v}}_{i\in I}$$

It is clear that the extended translation preserves provable equality.

The reverse translation, presented in Fig. A.11 is organized as follows:

- a value type A is translated into a CBV type $A^{\mathsf{v}^{-1}}$;
- a computation type $\underline{B}$ is translated into a CBV pseudo-computation-type $\underline{B}^{\mathsf{v}^{-1}}$
- a CBPV value $A_0, \ldots, A_{n-1} \vdash^{\mathsf{v}} V : B$ is translated into a CBV value $A_0^{\mathsf{v}^{-1}}, \ldots, A_{n-1}^{\mathsf{v}^{-1}} \vdash^{\mathsf{v}} V^{\mathsf{v}^{-1}} : B^{\mathsf{v}^{-1}}$;
- a computation $A_0, \ldots, A_{n-1} \vdash^{\mathsf{c}} M : \underline{B}$ is translated into an FG-CBV pseudo-computation $A_0^{\mathsf{v}^{-1}}, \ldots, A_{n-1}^{\mathsf{v}^{-1}} \vdash^{\mathsf{c}}_{\mathsf{CBV}} M^{\mathsf{v}^{-1}} : \underline{B}^{\mathsf{v}^{-1}}$.

It is easy to show that the reverse translation commutes with substitution (of values), and thence that it preserves provable equality.

Using this translation (and a result that it agrees with operational semantics in a certain sense) we can obtain from the printing denotational semantics for FG-CBV an alternative semantics for CBPV. Thus a computation type will denote a family of pairs of sets and a computation will denote a family of functions. More generally we can obtain a CBPV semantics from any FG-CBV semantics. However, the construction is artificial and we are not aware of any natural model for CBPV that arises in this way.

To show that the two translations are inverse up to isomorphism, we first construct the isomorphisms.

1. For each CBV type A we define a function α_A from CBV values $\Gamma \vdash^{\mathsf{v}} V : A$ to CBV values $\Gamma \vdash^{\mathsf{v}} W : A^{\mathsf{v}\mathsf{v}^{-1}}$, and a function α_A^{-1} in the opposite direction.
2. For each CBPV value type A we define a function β_A from CBPV values $\Gamma \vdash^{\mathsf{v}} V : A$ to values $\Gamma \vdash^{\mathsf{v}} W : A^{\mathsf{v}^{-1}\mathsf{v}}$, and a function β_A^{-1} in the opposite direction.
3. For each CBPV computation type $\underline{B}$ we define a function $\beta_{\underline{B}}$ from CBPV computations $\Gamma \vdash^{\mathsf{c}} M : \underline{B}$ to CBPV computations $\Gamma \vdash^{\mathsf{c}} N : \underline{B}^{\mathsf{v}^{-1}\mathsf{v}}$ and a function $\beta_{\underline{B}}^{-1}$ in the opposite direction.

(1) is defined by induction on A. (2) and (3) are defined by mutual induction on A and $\underline{B}$. We omit the definitions, which are straightforward.

Lemma 167 (properties of α and β)

The following equations are provable in the CBV equational theory.

$$\begin{aligned} \alpha_A(W[V/\mathtt{x}]) &= (\alpha_A W)[V/\mathtt{x}] \\ \alpha_A \alpha_A^{-1} V &= V \\ \alpha_A^{-1} \alpha_A V &= V \end{aligned}$$

A	$A^{\mathsf{v}^{-1}}$	where
$U\underline{B}$	$\prod^{\rightarrow}_{i\in I} A_i B_i$	$\underline{B}^{\mathsf{v}^{-1}} = \{(A_i, B_i)\}_{i\in I}$
$\sum_{k\in K} A_k$	$\sum_{k\in K} A_k^{\mathsf{v}^{-1}}$	
$A \times A'$	$A^{\mathsf{v}^{-1}} \times A'^{\mathsf{v}^{-1}}$	

$\underline{B}$	$\underline{B}^{\mathsf{v}^{-1}}$	where
FA	$\{(1, A^{\mathsf{v}^{-1}})\}_{i\in\{*\}}$	
$\prod_{k\in K} \underline{B}_k$	$\{(A_{kl}, B_{kl})\}_{k\in K, l\in L_k}$	$\underline{B}_k^{\mathsf{v}^{-1}} = \{(A_{kl}, B_{kl})\}_{l\in L_k}$
$A \to \underline{B}$	$\{(A^{\mathsf{v}^{-1}} \times A_i, B_i)\}_{i\in I}$	$\underline{B}^{\mathsf{v}^{-1}} = \{(A_i, B_i)\}_{i\in I}$

V	$V^{\mathsf{v}^{-1}}$
$\mathtt{x}$	$\mathtt{x}$
$\mathtt{let\ x\ be}\ V.\ W$	$\mathtt{let\ x\ be}\ V^{\mathsf{v}^{-1}}.\ W^{\mathsf{v}^{-1}}$
$(\hat{k}, V)$	$(\hat{k}, V^{\mathsf{v}^{-1}})$
$\mathtt{pm}\ V\ \mathtt{as}\ \{\ldots, (k, \mathtt{x}).W_k, \ldots\}$	$\mathtt{pm}\ V^{\mathsf{v}^{-1}}\ \mathtt{as}\ \{\ldots, (k, \mathtt{x}).W_k^{\mathsf{v}^{-1}}, \ldots\}$
(V, V')	$(V^{\mathsf{v}^{-1}}, V'^{\mathsf{v}^{-1}})$
$\mathtt{pm}\ V\ \mathtt{as}\ (\mathtt{x}, \mathtt{y}).W$	$\mathtt{pm}\ V^{\mathsf{v}^{-1}}\ \mathtt{as}\ (\mathtt{x}, \mathtt{y}).W^{\mathsf{v}^{-1}}$
$\mathtt{thunk}\ M$	$\lambda\{\ldots, \angle i, \mathtt{n}.M_i^{\mathsf{v}^{-1}}, \ldots\}$

M	$M_{\hat{\imath}}^{\mathsf{v}^{-1}}$	where
$\mathtt{let\ x\ be}\ V.\ M$	$\mathtt{let\ x\ be}\ V^{\mathsf{v}^{-1}}.\ M_{\hat{\imath}}^{\mathsf{v}^{-1}}$	
$\mathtt{produce}\ V$	$\mathtt{produce}\ V^{\mathsf{v}^{-1}}$	
$M\ \mathtt{to\ x}.\ N$	$M_*^{\mathsf{v}^{-1}}[()/\mathtt{n}]\ \mathtt{to\ x}.\ N_{\hat{\imath}}^{\mathsf{v}^{-1}}$	
$\mathtt{force}\ V$	$\angle i, \mathtt{n}\text{`}\mathtt{force}\ V^{\mathsf{v}^{-1}}$	
$\mathtt{pm}\ V\ \mathtt{as}\ \{\ldots, (k, \mathtt{x}).M_k, \ldots\}$	$\mathtt{pm}\ V^{\mathsf{v}^{-1}}\ \mathtt{as}\ \{\ldots, (k, \mathtt{x}).(M_k)_{\hat{\imath}}^{\mathsf{v}^{-1}}, \ldots\}$	
$\mathtt{pm}\ V\ \mathtt{as}\ (\mathtt{x}, \mathtt{y}).M$	$\mathtt{pm}\ V^{\mathsf{v}^{-1}}\ \mathtt{as}\ (\mathtt{x}, \mathtt{y}).M_{\hat{\imath}}^{\mathsf{v}^{-1}}$	
$\lambda\{\ldots, k.M_k, \ldots\}$	$(M_{\hat{k}})_{\hat{l}}^{\mathsf{v}^{-1}}$	$\hat{\imath} = (\hat{k}, \hat{l})$
$\hat{k}\text{`}M$	$M_{\hat{k}\hat{\imath}}^{\mathsf{v}^{-1}}$	
$\lambda\mathtt{x}.M$	$\mathtt{pm\ n\ as}\ (\mathtt{x}, \mathtt{n}).\ M_{\hat{\imath}}^{\mathsf{v}^{-1}}$	
$V\text{`}M$	$M_{\hat{\imath}}^{\mathsf{v}^{-1}}[(V^{\mathsf{v}^{-1}}, \mathtt{n})/\mathtt{n}]$	
$\mathtt{print}\ c;\ M$	$\mathtt{print}\ c;\ M_{\hat{\imath}}^{\mathsf{v}^{-1}}$	

Figure A.11: The Reverse Translation $-^{\mathsf{v}^{-1}}$

The following equations are provable in the CBPV equational theory.

$$\begin{array}{rcl}
\beta_A(W[V/\mathtt{x}]) & = & (\beta_A W)[V/\mathtt{x}] \\
\beta_{\underline{B}}(M[V/\mathtt{x}]) & = & (\beta_{\underline{B}} M)[W/\mathtt{x}] \\
\beta_A \beta_A^{-1} V & = & V \\
\beta_A^{-1} \beta_A V & = & V \\
\beta_{\underline{B}} \beta_{\underline{B}}^{-1} M & = & M \\
\beta_{\underline{B}}^{-1} \beta_{\underline{B}} M & = & M \\
\beta_{\underline{B}}(M \ \mathtt{to}\ \mathtt{x}.\ N) & = & M \ \mathtt{to}\ \mathtt{x}.\ \beta_{\underline{B}} N \\
\beta_{\underline{B}}(\mathtt{print}\ c;\ N) & = & \mathtt{print}\ c;\ \beta_{\underline{B}} N \\
(\alpha_A V)^{\mathsf{v}} & = & \beta_{A^{\mathsf{v}}}(V^{\mathsf{v}})
\end{array}$$

□

These are each proved by induction over types. Using β_A, we have now proved Prop. 165(1).

The translations are inverse up to these isomorphisms, in the following sense:

Lemma 168 For an FG-CBV value $A_0, \dots, A_{n-1} \vdash^{\mathsf{v}} V : B$, we can prove in CBV

$$V^{\mathsf{v}\mathsf{v}^{-1}}[\overrightarrow{\alpha_{A_i}\mathtt{x}_i/\mathtt{x}_i}] = \alpha_B V$$

For an FG-CBV producer $A_0, \dots, A_{n-1} \vdash^{\mathsf{p}} M : B$, we can prove in CBV

$$M_*^{\mathsf{v}\mathsf{v}^{-1}}[\overrightarrow{\alpha_{A_i}\mathtt{x}_i/\mathtt{x}_i}, ()/\mathtt{n}] = M \ \mathtt{to}\ \mathtt{x}.\ \mathtt{produce}\ \alpha_B \mathtt{x}$$

For a CBPV value $A_0, \dots, A_{n-1} \vdash^{\mathsf{v}} V : B$, we can prove in CBPV

$$V^{\mathsf{v}^{-1}\mathsf{v}}[\overrightarrow{\beta_{A_i}\mathtt{x}_i/\mathtt{x}_i}] = \beta_B V$$

For a CBPV computation $A_0, \dots, A_{n-1} \vdash^{\mathsf{c}} M : \underline{B}$, we can prove in CBPV

$$M^{\mathsf{v}^{-1}\mathsf{v}}[\overrightarrow{\beta_{A_i}\mathtt{x}_i/\mathtt{x}_i}] = \beta_{\underline{B}} M$$

These results can be extended to terms with holes i.e. contexts. □

This is proved by induction over terms using Lemma 167.

We are now in a position to prove Prop. 165(2)–(3). Fix a CBV context $A_0, \dots, A_{n-1}$ and CBV type B.

Definition 126 1. For any CBPV value $A_0^{\mathsf{v}}, \dots, A_{n-1}^{\mathsf{v}} \vdash^{\mathsf{v}} W : B^{\mathsf{v}}$ define the FG-CBV value θW to be

$$A_0, \dots, A_{n-1} \vdash^{\mathsf{v}} \alpha_B^{-1} W^{\mathsf{v}^{-1}}[\overrightarrow{\alpha A_i \mathtt{x}_i/\mathtt{x}_i}] : B$$

2. For any CBPV producer $A_0^{\mathsf{v}}, \dots, A_{n-1}^{\mathsf{v}} \vdash^{\mathsf{c}} N : FB^{\mathsf{v}}$ define the FG-CBV producer θN to be

$$A_0, \dots, A_{n-1} \vdash^{\mathsf{p}} N_*^{\mathsf{v}^{-1}}[\overrightarrow{\alpha A_i \mathtt{x}_i/\mathtt{x}_i}, ()/\mathtt{n}] \ \mathtt{to}\ \mathtt{y}.\ \mathtt{produce}\ \alpha_B^{-1}\mathtt{y} : B$$

□

Lemma 169 1. For any CBPV producer $A_0^{\mathsf{v}}, \dots, A_{n-1}^{\mathsf{v}} \vdash^{\mathsf{c}} N : FB^{\mathsf{v}}$, the equation $(\theta N)^{\mathsf{v}} = N$ is provable in CBPV.

2. For any FG-CBV producer $A_0, \dots, A_{n-1} \vdash^{\mathsf{p}} M : B$, the equation $\theta(M^{\mathsf{v}}) = M$ is provable in CBV.

Similarly for values. This can all be extended to terms with holes i.e. contexts. □

Proof

(1) By Lemma 168, $N = \beta^{-1}_{FB^{\mathsf{v}}} N^{\mathsf{v}^{-1}\mathsf{v}}[\overrightarrow{\beta_{A_i}\mathtt{x}_i/\mathtt{x}_i}]$ is provable, and we can see that $\beta^{-1}_{FB^{\mathsf{v}}} N^{\mathsf{v}^{-1}\mathsf{v}}[\overrightarrow{\beta_{A_i}\mathtt{x}_i/\mathtt{x}_i}] = (\theta N)^{\mathsf{v}}$ by expanding both sides and using Lemma 167.

(2) This follows directly from Lemma 168.

The proof for values is similar. □

Prop. 165(2)–(3) follows immediately from Lemma 169.

A.6.3 From CBN to CBPV

C	C^{n} (a computation type)
$\sum_{i\in I}^{\times_{j\in \$r_i}} A_{ij}$	$F\sum_{i\in I}(UA^{\mathsf{n}}_{i0} \times \cdots \times UA^{\mathsf{n}}_{i(r_i-1)})$
$\prod_{i\in I}^{\to_{j\in \$r_i}} A_{ij}B_i$	$\prod_{i\in I}(UA^{\mathsf{n}}_{i0} \to \cdots \to UA^{\mathsf{n}}_{i(r_i-1)} \to B^{\mathsf{n}}_i)$

$A_0, \dots, A_{n-1} \vdash M : C$	$UA^{\mathsf{n}}_0, \dots, UA^{\mathsf{n}}_{n-1} \vdash^{\mathsf{c}} M^{\mathsf{n}} : C^{\mathsf{n}}$
$\mathtt{x}$	$\mathtt{force}\ \mathtt{x}$
$\mathtt{let}\ \mathtt{x}\ \mathtt{be}\ M.\ N$	$\mathtt{let}\ \mathtt{x}\ \mathtt{be}\ \mathtt{thunk}\ M^{\mathsf{n}}.\ N^{\mathsf{n}}$
$(\hat{\imath}, M_0, \dots, M_{r_i-1})$	$\mathtt{produce}\ (\hat{\imath}, (\mathtt{thunk}\ M^{\mathsf{n}}_0, \dots, \mathtt{thunk}\ M^{\mathsf{n}}_{r_i-1}))$
$\mathtt{pm}\ M\ \mathtt{as}\ \{\dots, (i, \overrightarrow{\mathtt{x}_j}).N_i, \dots\}$	$M\ \mathtt{to}\ \mathtt{z}.\ \mathtt{pm}\ \mathtt{z}\ \mathtt{as}\ \{\dots, (i, (\overrightarrow{\mathtt{x}_j})).M_i, \dots\}$
$\lambda\{\dots, \angle i, \mathtt{x}_0, \dots, \mathtt{x}_{r_i-1}.M_i, \dots\}$	$\lambda\{\dots, i.\lambda\mathtt{x}_0. \dots \lambda\mathtt{x}_{r_i-1}.M^{\mathsf{n}}_i, \dots\}$
$\angle\hat{\imath}, M_0, \dots, M_{r_i-1}$‘$N$	$(\mathtt{thunk}\ M^{\mathsf{n}}_{r_i-1})$‘$\cdots$‘$(\mathtt{thunk}\ M^{\mathsf{n}}_0)$‘$\hat{\imath}$‘$N$

Figure A.12: Translation of CBN types and terms

Lemma 170 Given CBN terms $\Gamma \vdash N : A$ and $\Gamma, \mathtt{x} : A \vdash M : B$, the equation $M[N/\mathtt{x}]^{\mathsf{n}} = M^{\mathsf{n}}[\mathtt{thunk}\ N^{\mathsf{n}}/\mathtt{x}]$ is provable in CBPV. □

Proposition 171 If $M = N$ is provable in the CBN equational theory then $M^{\mathsf{n}} = N^{\mathsf{n}}$ is provable in the CBPV equational theory. □

The technical treatment of this translation is also very similar to the treatment of the translation in Sect. A.5.

This translation does not commute with substitution or preserve operational semantics precisely—only up to the prefix `force thunk`. (Recall that in the CBV equational theory, we have $\mathtt{force}\ \mathtt{thunk}\ M = M$.)

substitution It is not the case that $(M[N/\mathtt{x}])^{\mathsf{n}}$ and $M^{\mathsf{n}}[\mathtt{thunk}\ N^{\mathsf{n}}/\mathtt{x}]$ are the same term (although they will be provably equal in the CBPV equational theory). To see this, let M be $\mathtt{x}$. Then $(M[N/\mathtt{x}])^{\mathsf{n}} = N^{\mathsf{n}}$ but $M^{\mathsf{n}}[\mathtt{thunk}\ N^{\mathsf{n}}/\mathtt{x}] = \mathtt{force}\ \mathtt{thunk}\ N^{\mathsf{n}}$.

operational semantics It is not the case that $M \Downarrow m, T$ implies $M^{\mathsf{n}} \Downarrow m, T^{\mathsf{n}}$. For example, let M be $\mathtt{let}\ \mathtt{x}\ \mathtt{be}\ (0).\ (0, \mathtt{x})$.

To make the relationship precise, we proceed as follows. First, we write the translation as a relation $\mapsto^{\mathsf{n}}$ from CBN terms to CBPV computations. We can present Fig. A.12 by rules such as these:

$$\frac{M \mapsto^{\mathsf{n}} M' \quad N \mapsto^{\mathsf{n}} N'}{\mathtt{let\ x\ be}\ M.\ N \mapsto^{\mathsf{n}} \mathtt{let\ x\ be\ thunk}\ M'.\ N'}$$

To these rules we add the following:

$$\frac{M \mapsto^{\mathsf{n}} M'}{M \mapsto^{\mathsf{n}} \mathtt{force\ thunk}\ M'}$$

Lemma 172 For any term M, we have $M \mapsto^{\mathsf{n}} M^{\mathsf{n}}$, and if $M \mapsto^{\mathsf{n}} M'$ then $M' = M^{\mathsf{n}}$ is provable in the CBPV equational theory. □

Lemma 173 If $M \mapsto^{\mathsf{n}} M'$ and $N \mapsto^{\mathsf{n}} N'$ then $M[N/\mathtt{x}] \mapsto^{\mathsf{n}} M'[\mathtt{thunk}\ N'/\mathtt{x}]$, and similarly for multiple substitution. □

Proposition 174
1. If $M \Downarrow N$ and $M \mapsto^{\mathsf{n}} M'$, then, for some N', $M' \Downarrow N'$ and $N \mapsto^{\mathsf{n}} N'$.
2. If $M \mapsto^{\mathsf{n}} M'$ and $M' \Downarrow N'$, then, for some N, $M \Downarrow N$ and $N \mapsto^{\mathsf{n}} N'$.

□

We prove these by induction primarily on $\Downarrow$ and secondarily on $\mapsto^{\mathsf{n}}$.

A.6.4 From CBPV Back To CBN

Our aim is to prove the following.

Proposition 175
1. Every CBPV computation type $\underline{B}$ is isomorphic to B^{n} for some CBN type B.
2. For any CBPV computation $UA_0^{\mathsf{n}}, \ldots, UA_{n-1}^{\mathsf{n}} \vdash^{\mathsf{c}} N : B^{\mathsf{n}}$ there is a CBN term $A_0, \ldots, A_{n-1} \vdash M : B$ such that $M^{\mathsf{n}} = N$ is provable in CBPV.
3. For any CBN terms $\Gamma \vdash M, M' : B$, if $M^{\mathsf{n}} = M'^{\mathsf{n}}$ is provable in CBPV then $M = M'$ is provable in CBN.

(2)–(3) can be extended to terms with holes i.e. contexts. □

Corollary 176 (Full Abstraction) For any CBN terms $A_0, \ldots, A_{n-1} \vdash M, M' : B$, if $M \simeq M'$ then $M^{\mathsf{n}} \simeq M'^{\mathsf{n}}$. (The converse is trivial.) □

The proof is similar to that of Cor. 166.

To prove Prop. 175, we first give a translation $-^{\mathsf{n}^{-1}}$ from CBPV to CBN. (We shall see that, up to isomorphism, it is inverse to $-^{\mathsf{n}}$.)

At first glance, it is not apparent how to make such a translation. For while it will translate a computation type into a CBN type, what will it translate a value type into? The answer is: a family of CBN types $\{A_i\}_{i \in I}$. This approach is based on [AM98a].

To see the principle of the translation, we notice that in CBPV any value type must be isomorphic to a type of the form $\sum_{i \in I} U\underline{A}_i$. (This is discussed in Sect. 4.7.3.)

This motivates the following.

Definition 127
- A *CBN pseudo-value-type* is a family $\{A_i\}_{i \in I}$ of CBN types.

- Let $\{A_{0i}\}_{i\in I_0},\dots,\{A_{(n-1)i}\}_{i\in I_{n-1}}$ be a sequence of CBN pseudo-value-types and let $\{B_j\}_{j\in J}$ be another CBN pseudo-value-type. We write

$$\{A_{0i}\}_{i\in I_0},\dots,\{A_{(n-1)i}\}_{i\in I_{n-1}} \vdash^{\mathsf{v}}_{\mathsf{CBN}} \{(V \bullet \overrightarrow{i_j}, V_{\overrightarrow{i_j}})\}_{\overrightarrow{i_j\in I_j}} : \{B_j\}_{j\in J}$$

  to mean that, for each $i_0 \in I_0,\dots,i_{n-1}\in I_{n-1}$, we have $V \bullet \overrightarrow{i_j} \in J$ and $V_{\overrightarrow{i_j}}$ is a CBN term $A_{0i_0},\dots,A_{(n-1)i_{n-1}} \vdash V_{\overrightarrow{i_j}} : B_{V\bullet\overrightarrow{i_j}}$. We say that $\{(V \bullet \overrightarrow{i_j}, V_{\overrightarrow{i_j}})\}_{\overrightarrow{i_j\in I_j}}$ is a *CBN pseudo-value* of type $\{B_j\}_{j\in J}$ on the context $\{A_{0i}\}_{i\in I_0},\dots,\{A_{(n-1)i}\}_{i\in I_{n-1}}$. Given two such pseudo-values $\{(V \bullet \overrightarrow{i_j}, V_{\overrightarrow{i_j}})\}_{\overrightarrow{i_j\in I_j}}$ and $\{(W \bullet \overrightarrow{i_j}, W_{\overrightarrow{i_j}})\}_{\overrightarrow{i_j\in I_j}}$, we say that they are *provably equal in CBN* when for each $\overrightarrow{i_j \in I_j}$ we have $V \bullet \overrightarrow{i_j} = W \bullet \overrightarrow{i_j}$ and the equation $V_{\overrightarrow{i_j}} = W_{\overrightarrow{i_j}}$ is provable in CBN.

- Let $\{A_{0i}\}_{i\in I_0},\dots,\{A_{(n-1)i}\}_{i\in I_{n-1}}$ be a sequence of CBN pseudo-value-types and let B be a CBN type. We write

$$\{A_{0i}\}_{i\in I_0},\dots,\{A_{(n-1)i}\}_{i\in I_{n-1}} \vdash^{\mathsf{c}}_{\mathsf{CBN}} \{M_{\overrightarrow{i_j}}\}_{\overrightarrow{i_j\in I_j}} : B$$

  to mean that, for each $i_0 \in I_0,\dots,i_{n-1}\in I_{n-1}$, $M_{\overrightarrow{i_j}}$ is a CBN term $A_{0i_0},\dots,A_{(n-1)i_{n-1}} \vdash M_{\overrightarrow{i_j}} : B$. We say that $\{M_{\overrightarrow{i_j}}\}_{\overrightarrow{i_j\in I_j}}$ is a *CBN pseudo-computation* of type B on the context $\{A_{0i}\}_{i\in I_0},\dots,\{A_{(n-1)i}\}_{i\in I_{n-1}}$. Given two such pseudo-computations $\{M_{\overrightarrow{i_j}}\}_{\overrightarrow{i_j\in I_j}}$ and $\{N_{\overrightarrow{i_j}}\}_{\overrightarrow{i_j\in I_j}}$, we say that they are *provably equal in CBN* when for all $\overrightarrow{i_j \in I_j}$ the equation $M_{\overrightarrow{i_j}} = N_{\overrightarrow{i_j}}$ is provable in CBN.

□

Before presenting the reverse translation, we extend the forward translation as follows:

- Given a CBN pseudo-value-type $\{A_i\}_{i\in I}$ we write $\{A_i\}^{\mathsf{n}}_{i\in I}$ for $\sum_{i\in I} U A_i^{\mathsf{n}}$.
- Given a CBN pseudo-value

$$\{A_{0i}\}_{i\in I_0},\dots,\{A_{(n-1)i}\}_{i\in I_{n-1}} \vdash^{\mathsf{v}}_{\mathsf{CBN}} \{(V \bullet \overrightarrow{i_j}, V_{\overrightarrow{i_j}})\}_{\overrightarrow{i_j\in I_j}} : \{B_j\}_{j\in J}$$

  we write $\{(V \bullet \overrightarrow{i_j}, V_{\overrightarrow{i_j}})\}^{\mathsf{n}}_{\overrightarrow{i_j\in I_j}}$ to mean the CBPV value

$$\{A_{0i}\}^{\mathsf{n}}_{i\in I_0},\dots,\{A_{(n-1)i}\}^{\mathsf{n}}_{i\in I_{n-1}} \vdash^{\mathsf{v}} \mathtt{pm}\,(\overrightarrow{\mathtt{x}_j})\ \mathtt{as}\ \{\dots,(\overrightarrow{i_j},(\overrightarrow{\mathtt{x}_j})).(V \bullet \overrightarrow{i_j}, \mathtt{thunk}\, V_{\overrightarrow{i_j}}),\dots\} : \{B_j\}^{\mathsf{n}}_{j\in J}$$

- Given a CBN pseudo-computation

$$\{A_{0i}\}_{i\in I_0},\dots,\{A_{(n-1)i}\}_{i\in I_{n-1}} \vdash^{\mathsf{c}}_{\mathsf{CBN}} \{M_{\overrightarrow{i_j}}\}_{\overrightarrow{i_j\in I_j}} : B$$

  we write $\{M_{\overrightarrow{i_j}}\}^{\mathsf{n}}_{\overrightarrow{i_j\in I_j}}$ to mean the CBPV computation

$$\{A_{0i}\}^{\mathsf{n}}_{i\in I_0},\dots,\{A_{(n-1)i}\}^{\mathsf{n}}_{i\in I_{n-1}} \vdash^{\mathsf{c}} \mathtt{pm}\,(\overrightarrow{\mathtt{x}_j})\ \mathtt{as}\ \{\dots,(\overrightarrow{i_j},(\overrightarrow{\mathtt{x}_j})).M^{\mathsf{n}}_{\overrightarrow{i_j}},\dots\} : B^{\mathsf{n}}$$

It is clear that this extended translation preserves provable equality.

The reverse translation, presented in Fig. A.13 is organized as follows:

- a value type A is translated into a CBN pseudo-value-type $A^{\mathsf{n}^{-1}}$;
- a computation type $\underline{B}$ is translated into a CBN type $\underline{B}^{\mathsf{n}^{-1}}$;
- a CBPV value $A_0,\dots,A_{n-1} \vdash^{\mathsf{v}} B$ is translated into a CBN pseudo-value $A_0^{\mathsf{n}^{-1}},\dots,A_{n-1}^{\mathsf{n}^{-1}} \vdash^{\mathsf{v}}_{\mathsf{CBN}} V^{\mathsf{n}^{-1}} : B^{\mathsf{n}^{-1}}$

A	$A^{\mathsf{n}^{-1}}$	where
$U\underline{B}$	$\{\underline{B}^{\mathsf{n}^{-1}}\}$	
$\sum_{k\in K} A_k$	$\{A_{kl}\}_{k\in K, l\in L_k}$	$A_k^{\mathsf{n}^{-1}} = \{A_{kl}\}_{l\in L_k}$
$A \times A'$	$\{A_k \sqcap A'_l\}_{k\in K, l\in L}$	$A^{\mathsf{n}^{-1}} = \{A_k\}_{k\in K}$ and $A'^{\mathsf{n}^{-1}} = \{A_l\}_{l\in L}$

$\underline{B}$	$\underline{B}^{\mathsf{n}^{-1}}$	where
FA	$\sum_{i\in I} A_i$	$A^{\mathsf{n}^{-1}} = \{A_i\}_{i\in I}$
$\prod_{k\in K} \underline{B}_k$	$\prod_{k\in K} \underline{B}_k^{\mathsf{n}^{-1}}$	
$A \to \underline{B}$	$\prod_{i\in I}(A_i \to \underline{B}^{\mathsf{n}^{-1}})$	$A^{\mathsf{n}^{-1}} = \{A_i\}_{i\in I}$

V	$V^{\mathsf{n}^{-1}} \bullet \overrightarrow{i_j}$ $\qquad$ $V_{\overrightarrow{i_j}}^{\mathsf{n}^{-1}}$	where
$\mathtt{x}_r$	i_r $\qquad$ $\mathtt{x}_r$	
$\mathtt{let\ x\ be}\ V.\ W$	$W^{\mathsf{n}^{-1}} \bullet \overrightarrow{i_j}\hat{k}$	
	$\mathtt{let\ x\ be}\ V_{\overrightarrow{i_j}}^{\mathsf{n}^{-1}}.\ W_{\overrightarrow{i_j}\hat{k}}^{\mathsf{n}^{-1}}$	$V^{\mathsf{n}^{-1}} \bullet \overrightarrow{i_j} = \hat{k}$
(V, V')	$(V^{\mathsf{n}^{-1}} \bullet \overrightarrow{i_j}, V'^{\mathsf{n}^{-1}} \bullet \overrightarrow{i_j})$	
	$\lambda\{0.V_{\overrightarrow{i_j}}^{\mathsf{n}^{-1}}, 1.V'^{\mathsf{n}^{-1}}_{\overrightarrow{i_j}}\}$	
$\mathtt{pm}\ V\ \mathtt{as}\ (\mathtt{x},\mathtt{y}).W$	$W^{\mathsf{n}^{-1}} \bullet \overrightarrow{i_j}\hat{k}\hat{l}$	
	$\mathtt{let\ x\ be}\ 0\text{`}V_{\overrightarrow{i_j}}^{\mathsf{n}^{-1}}, \mathtt{y\ be}\ 1\text{`}V_{\overrightarrow{i_j}}^{\mathsf{n}^{-1}}.\ W_{\overrightarrow{i_j}\hat{k}\hat{l}}^{\mathsf{n}^{-1}}$	$V^{\mathsf{n}^{-1}} \bullet \overrightarrow{i_j} = (\hat{k}, \hat{l})$
$(\hat{k}, V)$	$(\hat{k}, V^{\mathsf{n}^{-1}} \bullet \overrightarrow{i_j})$ $\qquad$ $V_{\overrightarrow{i_j}}^{\mathsf{n}^{-1}}$	
$\mathtt{pm}\ V\ \mathtt{as}\ \{\ldots, (k,\mathtt{x}).W_k, \ldots\}$	$(W_{\hat{k}})^{\mathsf{n}^{-1}} \bullet \overrightarrow{i_j}\hat{l}$	
	$\mathtt{let\ x\ be}\ V_{\overrightarrow{i_j}}^{\mathsf{n}^{-1}}.\ (W_{\hat{k}})_{\overrightarrow{i_j}\hat{l}}^{\mathsf{n}^{-1}}$	$V^{\mathsf{n}^{-1}} \bullet \overrightarrow{i_j} = (\hat{k}, \hat{l})$
$\mathtt{thunk}\ M$	$*$ $\qquad$ $M_{\overrightarrow{i_j}}^{\mathsf{n}^{-1}}$	

M	$M_{\overrightarrow{i_j}}^{\mathsf{n}^{-1}}$	where
$\mathtt{let\ x\ be}\ V.\ M$	$\mathtt{let\ x\ be}\ V_{\overrightarrow{i_j}}^{\mathsf{n}^{-1}}.\ M_{\overrightarrow{i_j}\hat{k}}^{\mathsf{n}^{-1}}$	$V^{\mathsf{n}^{-1}} \bullet \overrightarrow{i_j} = \hat{k}$
$\mathtt{pm}\ V\ \mathtt{as}\ (\mathtt{x},\mathtt{y}).M$	$\mathtt{let\ x\ be}\ 0\text{`}V_{\overrightarrow{i_j}}^{\mathsf{n}^{-1}}, \mathtt{y\ be}\ 1\text{`}V_{\overrightarrow{i_j}}^{\mathsf{n}^{-1}}.\ M_{\overrightarrow{i_j}\hat{k}\hat{l}}^{\mathsf{n}^{-1}}$	$V^{\mathsf{n}^{-1}} \bullet \overrightarrow{i_j} = (\hat{k}, \hat{l})$
$\mathtt{pm}\ V\ \mathtt{as}\ \{\ldots, (k,\mathtt{x}).M_k, \ldots\}$	$\mathtt{let\ x\ be}\ V_{\overrightarrow{i_j}}^{\mathsf{n}^{-1}}.\ (M_{\hat{k}})_{\overrightarrow{i_j}\hat{l}}^{\mathsf{n}^{-1}}$	$V^{\mathsf{n}^{-1}} \bullet \overrightarrow{i_j} = (\hat{k}, \hat{l})$
$\mathtt{force}\ V$	$V_{\overrightarrow{i_j}}^{\mathsf{n}^{-1}}$	
$\mathtt{produce}\ V$	$(V^{\mathsf{n}^{-1}} \bullet \overrightarrow{i_j}, V_{\overrightarrow{i_j}}^{\mathsf{n}^{-1}})$	
$M\ \mathtt{to\ x}.\ N$	$\mathtt{pm}\ M_{\overrightarrow{i_j}}^{\mathsf{n}^{-1}}\ \mathtt{as}\ \{\ldots, (i,\mathtt{x}).N_{\overrightarrow{i_j}i}^{\mathsf{n}^{-1}}, \ldots\}$	
$\lambda\mathtt{x}.M$	$\lambda\{\ldots, i.\lambda\mathtt{x}.M_{\overrightarrow{i_j}i}^{\mathsf{n}^{-1}}, \ldots\}$	
$V\text{`}M$	$V_{\overrightarrow{i_j}}^{\mathsf{n}^{-1}}\text{`}V^{\mathsf{n}^{-1}} \bullet \overrightarrow{i_j}\text{`}M_{\overrightarrow{i_j}}^{\mathsf{n}^{-1}}$	
$\lambda\{\ldots, k.M_k, \ldots\}$	$\lambda\{\ldots, k.(M_k)_{\overrightarrow{i_j}}^{\mathsf{n}^{-1}}, \ldots\}$	
$\hat{k}\text{`}M$	$\hat{k}\text{`}M_{\overrightarrow{i_j}}^{\mathsf{n}^{-1}}$	
$\mathtt{print}\ c;\ M$	$\mathtt{print}\ c;\ M_{\overrightarrow{i_j}}^{\mathsf{n}^{-1}}$	

Figure A.13: The Reverse Translation $-^{\mathsf{n}^{-1}}$

- a CBPV computation $A_0, \ldots, A_{n-1} \vdash^{\mathsf{c}} M : \underline{B}$ is translated into a CBN pseudo-computation $A_0^{\mathsf{n}^{-1}}, \ldots, A_{n-1}^{\mathsf{n}^{-1}} \vdash^{\mathsf{c}}_{\mathsf{CBN}} M^{\mathsf{n}^{-1}} : \underline{B}^{\mathsf{n}^{-1}}$.

It is easy to show that the reverse translation commutes with substitution (of values) and thence that it preserves provable equality.

Using this translation (and a result that it agrees with operational semantics in a certain sense) we can obtain from the printing denotational semantics for CBN an alternative semantics for CBPV. Thus a computation type will denote a family of sets and a computation will denote a family of functions. More generally we can obtain a CBPV semantics from any CBN semantics. Several important models, such as the Scott model and the game model, can be seen as arising in this way [AM98a].

To show that the two translations are inverse up to isomorphism, we first construct the isomorphisms.

1. For each CBN type A we define a function α_A from CBN terms $\Gamma \vdash M : A$ to CBN terms $\Gamma \vdash N : A^{\mathsf{n}\mathsf{n}^{-1}}$, and a function α_A^{-1} in the opposite direction.

2. For each CBPV value type A we define a function β_A from CBPV values $\Gamma \vdash^{\mathsf{v}} V : A$ to values $\Gamma \vdash^{\mathsf{v}} W : A^{\mathsf{n}^{-1}\mathsf{n}}$, and a function β_A^{-1} in the opposite direction.

3. For each CBPV computation type $\underline{B}$ we define a function $\beta_{\underline{B}}$ from CBPV computations $\Gamma \vdash^{\mathsf{c}} M : \underline{B}$ to CBPV computations $\Gamma \vdash^{\mathsf{c}} N : \underline{B}^{\mathsf{n}^{-1}\mathsf{n}}$ and a function $\beta_{\underline{B}}^{-1}$ in the opposite direction.

(1) is defined by induction on A. (2) and (3) are defined by mutual induction on A and $\underline{B}$. We omit the definitions, which are straightforward.

Lemma 177 (properties of α and β)

The following equations are provable in the CBN equational theory.

$$\begin{aligned}
\alpha_A(N[M/\mathtt{x}]) &= (\alpha_A N)[M/\mathtt{x}] \\
\alpha_A \alpha_A^{-1} M &= M \\
\alpha_A^{-1} \alpha_A M &= M \\
\alpha_B(\mathtt{pm}\ M\ \mathtt{as}\ \{\ldots, (i, \overrightarrow{\mathtt{x}_j}).N_i, \ldots\}) &= \mathtt{pm}\ M\ \mathtt{as}\ \{\ldots, (i, \overrightarrow{\mathtt{x}_j}).\alpha_B N_i, \ldots\}
\end{aligned}$$

The following equations are provable in the CBPV equational theory.

$$\begin{aligned}
\beta_A(W[V/\mathtt{x}]) &= (\beta_A W)[V/\mathtt{x}] \\
\beta_{\underline{B}}(M[V/\mathtt{x}]) &= (\beta_{\underline{B}} M)[W/\mathtt{x}] \\
\beta_A \beta_A^{-1} V &= V \\
\beta_A^{-1} \beta_A V &= V \\
\beta_{\underline{B}} \beta_{\underline{B}}^{-1} M &= M \\
\beta_{\underline{B}}^{-1} \beta_{\underline{B}} M &= M \\
\beta_{\underline{B}}(M\ \mathtt{to}\ \mathtt{x}.\ N) &= M\ \mathtt{to}\ \mathtt{x}.\ \beta_{\underline{B}} N \\
\beta_{\underline{B}}(\mathtt{print}\ c;\ N) &= \mathtt{print}\ c;\ \beta_{\underline{B}} N \\
(\alpha_A M)^{\mathsf{n}} &= \beta_{A^{\mathsf{n}}}(M^{\mathsf{n}})
\end{aligned}$$

□

These are each proved by induction over types. Using $\beta_{\underline{B}}$, we have now proved Prop. 175(1).

The translations are inverse up to these isomorphisms, in the following sense:

Lemma 178 For a CBN term $A_0, \ldots, A_{n-1} \vdash M : B$, we can prove in CBV

$$M^{\mathsf{n}\mathsf{n}^{-1}}_{*\ldots*}[\overrightarrow{\alpha_{A_i}\mathtt{x}_i/\mathtt{x}_i}] = \alpha_B M$$

For a CBPV value $A_0, \ldots, A_{n-1} \vdash^{\mathsf{v}} V : B$, we can prove in CBPV

$$V^{\mathsf{n}^{-1}\mathsf{n}}[\overrightarrow{\beta_{A_i}\mathtt{x}_i/\mathtt{x}_i}] = \beta_B V$$

For a CBPV computation $A_0, \ldots, A_{n-1} \vdash^{\mathsf{c}} M : \underline{B}$, we can prove in CBPV

$$M^{\mathsf{n}^{-1}\mathsf{n}}[\overrightarrow{\beta_{A_i}\mathtt{x}_i/\mathtt{x}_i}] = \beta_{\underline{B}} M$$

These results can be extended to terms with holes i.e. contexts. □

This is proved by induction over terms using Lemma 177.

We are now in a position to prove Prop. 175(2)–(3). Fix a CBN context $A_0, \ldots, A_{n-1}$ and CBN type B.

Definition 128 For any CBPV computation $UA_0^{\mathsf{n}}, \ldots, UA_{n-1}^{\mathsf{n}} \vdash^{\mathsf{c}} N : B^{\mathsf{n}}$ define the CBN term ϕN to be

$$A_0, \ldots, A_{n-1} \vdash \alpha_B N^{\mathsf{n}^{-1}}_{*\ldots*}[\overrightarrow{\alpha_{A_i}\mathtt{x}_i/\mathtt{x}_i}] : B$$

□

Lemma 179
1. For any CBPV computation $UA_0^{\mathsf{n}}, \ldots, UA_{n-1}^{\mathsf{n}} \vdash^{\mathsf{c}} N : B^{\mathsf{n}}$, the equation $(\phi N)^{\mathsf{n}} = N$ is provable in CBPV.
2. For any CBN term $A_0, \ldots, A_{n-1} \vdash M : B$, the equation $\phi(M^{\mathsf{n}}) = M$ is provable in CBN.

□

Proof

(1) By Lemma 178, $N = \beta_B^{-1} N^{\mathsf{n}^{-1}\mathsf{n}}[\overrightarrow{\beta_{UA_i}\mathtt{x}_i/\mathtt{x}_i}]$ is provable, and we can see that $\beta_B^{-1} N^{\mathsf{n}^{-1}\mathsf{n}}[\overrightarrow{\beta_{UA_i}\mathtt{x}_i/\mathtt{x}_i}] = (\phi N)^{\mathsf{n}}$ by expanding both sides and using Lemma 177.

(2) This follows directly from Lemma 178.

This can all be extended to terms with holes i.e. contexts. □

Prop. 175(2)–(3) follows immediately from Lemma 179.

Appendix B

Technical Material For Games

B.1 Introduction

The am of this appendix (with the exception of Sect. B.6, which is a proof of the type definability result Prop. 81) is to describe semantics of type constructors directly. Earlier accounts of game semantics have given the interpretation of categorical combinators (especially composition), and then obtained the semantics of term constructors from this. We have avoided this because some readers may want to learn the game semantics for CBPV without learning the categorical semantics, and also because we consider it useful to have a general framework for constructing strategies from other strategies, of which categorical composition is just one instance.

As we said in Sect. 9.1.1, operations on strategies are messy to describe. The problem is partly a lack of appropriate idioms for talking about games and strategies in general, and partly specific to pointer games. We hope that future work will remedy this situation.

B.2 Strategies From Strategies

B.2.1 Discussion

All of Sect. B.2 applies to games in general, not just pointer games.

Suppose we have a family of games $\{G_i\}_{i\in I}$, and that for each G_i we have a strategy σ_i. We want to construct a strategy τ for another game H. We call G_i the *i-inner game* and we call H the *outer game* How do we construct τ?

Say for example that H is P-first. (This is the easiest case.) We have 3 choices:

- We can make a P-move in the outer game, and see how the outer Opponent will play;
- we can start a play of an O-first inner game G_i by playing an initial O-move, and see how the Player, following strategy σ_i, responds;
- we can create a play of a P-first inner game G_i, and see how the Player, following strategy σ_i, begins.

After this, we have 4 choices—the above three together with a fourth:

- we can continue a (previously started) play of an inner game G_i by playing an O-move, and seeing how the Player, following strategy σ_i responds.

The play continues in this way. Each cycle (a move by us, then a move by inner Player or outer Opponent) has one of these 4 forms. Of course, the inner Player or outer Opponent may diverge,

and so may we. At the end of each cycle, the outer play is awaiting-P and each inner play that has been started is awaiting-O.

Play thus develops as shown in Fig. B.1. What we have described is the *meta-game* from

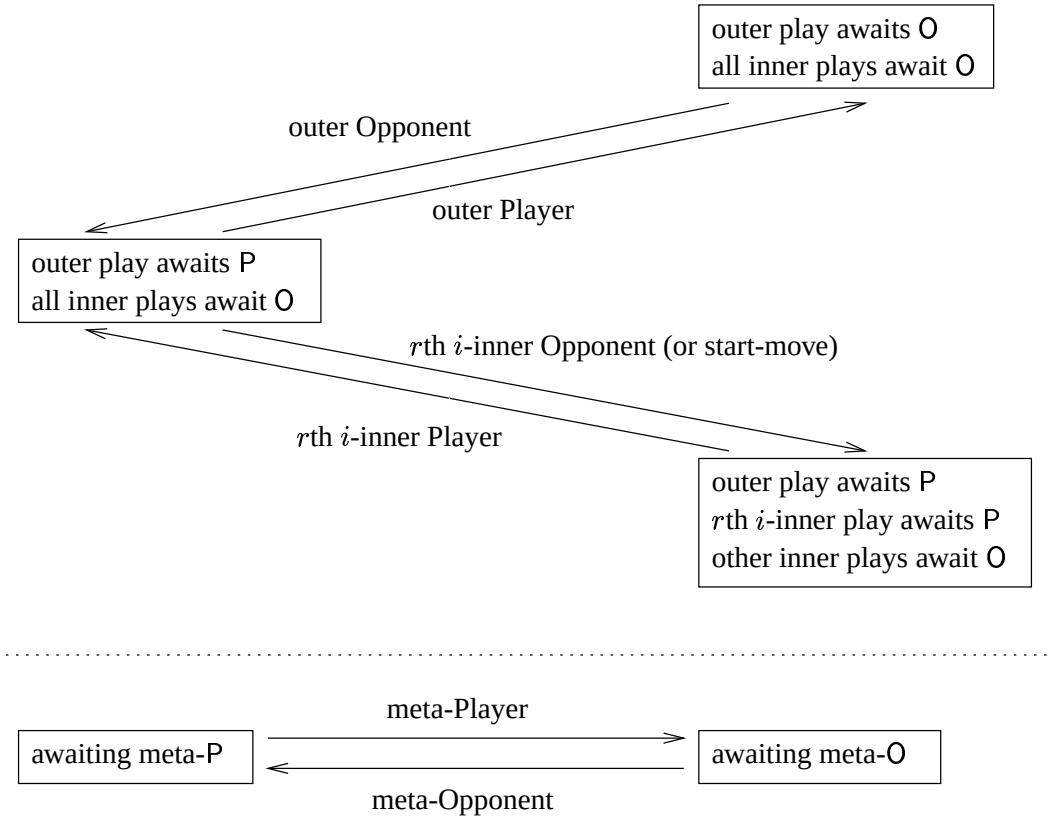


Figure B.1: The Meta-Game

$\{G_i\}_{i\in I}$ to H. In this game,

- a P-move consists of either an inner O-move, an outer P-move, or the creation of a new play for a P-first inner game
- a O-move consists of an inner P-move or an outer O-move.

Notice that it is only the meta-Player who can switch between different games; the meta-Opponent always plays a move in the same game that the meta-Player has just moved in. This is called the *switching condition*, variants of which appear throughout the games literature.

If the outer game H is O-first, then the first thing that happens is the outer Opponent's initial move. After that, we continue as above. Thus the meta-game is P-first if H is P-first, and O-first if H is O-first.

B.2.2 Meta-Strategies

We now formalize the discussion in Sect. B.2.1.

At any point in time, there is a single outer play and finitely many inner plays.

Definition 129 A *finite multi-play* a from inner games $\{G_i\}_{i\in I}$ to outer game H consists of

- a finite play a_{outer} for H (the *outer play*);

- for each $i \in I$ a finite sequence $a_{i0}, a_{i1}, \ldots$ of finite plays for G_i (the *i-inner plays*)—we write $|a_i|$ for the length of this sequence.

such that the total number of inner plays $\sum_{i \in I} |a_i|$ is finite. □

As a record of what has happened in the meta-game, a multi-play is missing one piece of information: it describes the ordering of the moves within each play, but not the global ordering across all plays.

Definition 130 Let a be a finite multi-play from inner games $\{G_i\}_{i \in I}$ to outer game H.

1. The *meta-moves* of a are

$\mathsf{outermove}\ n$	where n is a move in a_{outer}
$\mathsf{startmove}(i, r)$	where G_i is P-first and $r \in \$|a_i|$
$\mathsf{innermove}(i, r, n)$	where $r \in \$|a_i|$ and n is a move in a_{ir}

   The *length* of a, written $|a|$, is the number of meta-moves, which is necessarily finite.

2. A *global ordering* for a is a total ordering $\leqslant$ on the meta-moves of a such that

   - $\leqslant$ extends the ordering on each play in a, as follows:
     - if $m \leqslant n$ are moves in a_{outer} then $\mathsf{outermove}\ m \leqslant \mathsf{outermove}\ n$
     - if $m \leqslant n$ are moves in a_{ir} then $\mathsf{innermove}(i, r, m) \leqslant \mathsf{innermove}(i, r, n)$
   - for each $i \in I$, the various i-inner plays are numbered according to the order in which they begin, as follows:
     - if G_i is a P-first game and $r \leqslant s \in |a_i|$ then $\mathsf{startmove}(i, r) \leqslant \mathsf{startmove}(i, s)$
     - if G_i is an O-first game and $r \leqslant s \in |a_i|$ then $\mathsf{innermove}(i, r, 0) \leqslant \mathsf{innermove}(i, s, 0)$.

□

Definition 131 Let a be a finite multi-play from inner games $\{G_i\}_{i \in I}$ to outer game H. Suppose that H is P-first.

1. Given a global ordering $\leqslant$ on a, let θ be the (necessarily unique) order isomorphism from $\$|a|$ to $\leqslant$. We say that a meta-move of the form $\theta(2k)$ is a *meta-*P*-move* and a meta-move of the form $\theta(2k+1)$ is a *meta-*O*-move*. For example, the initial meta-move is a meta-P-move, unless a is empty.

2. A global ordering $\leqslant$ on a satisfies the *switching condition* when each meta-O-move q is in the same play as the preceding meta-P-move p i.e. either

   - $p = \mathsf{outermove}\ m$ and $q = \mathsf{outermove}\ m+1$, or
   - $p = \mathsf{innermove}(i, r, m)$ and $q = \mathsf{innermove}(i, r, m+1)$, or
   - $p = \mathsf{startmove}(i, r)$ and $q = \mathsf{innermove}(i, r, 0)$.

3. A *finite meta-play* from inner games $\{G_i\}_{i \in I}$ to outer game H is a finite multi-play a with a global ordering satisfying the switching condition. We say that a meta-play is

   - awaiting meta-P, when its length (i.e. the number of meta-moves) is even
   - awaiting meta-O, when its length is odd.

□

Def. 131 is given in the case that the outer game H is P-first. If H is O-first, we change Def. 131 as follows:

- In (1), a meta-move p of the form $\theta(2k)$ is a meta-O-move and a meta-move of the form $\theta(2k+1)$ is a meta-P-move. For example, the initial meta-move is a meta-O-move.
- In (2), we require the initial meta-move, which is a meta-O-move, to be $\mathsf{outermove}\ 0$.
- In (3), a meta-play is awaiting meta-P when its length is odd, and awaiting meta-O when its length is even.

Proposition 180 Let a be a finite meta-play from inner games $\{G_i\}_{i\in I}$ to outer game H. Each meta-P-move is either

- $\mathsf{outermove}\ m$, where m is a P-move in $a_{\mathsf{outerplay}}$, or
- $\mathsf{startmove}(i,r)$, where G_i is a P-first game
- $\mathsf{innermove}(i,r,m)$, where m is an O-move in a_{ir}.

Each meta-O-move is either

- $\mathsf{outermove}\ m$, where m is an O-move in $a_{\mathsf{outerplay}}$, or
- $\mathsf{innermove}(i,r,m)$, where m is a P-move in a_{ir}.

□

Definition 132 A *meta-strategy* from inner games $\{G_i\}_{i\in I}$ to outer game H is a set of finite meta-plays from $\{G_i\}_{i\in I}$ to H which is prefix-closed, contingent complete and deterministic (as in Def. 56). □

Definition 133 Let μ be a meta-strategy for inner games $\{G_i\}_{i\in I}$ and outer game H. If, for each $i \in I$ we are given a strategy σ_i for G_i, then we form a strategy for H called the μ*-composite* of $\{\sigma_i\}_{i\in I}$:

$$\mu\{\sigma_i\}_{i\in I} = \{a_{\mathsf{outer}} | a \in \mu, \forall i, r.\ s_{ir} \in \sigma_i\}$$

□

B.3 Descriptions of Moves in Games

We now return to pointer games. Suppose a is an O-first play on an arena R. For each move m in a, we say that move m is *described as*

- $\mathsf{rtmove}\ \ r$ if m is a root move passing the token r (a root of R)
- $\mathsf{ptr}\ n\ \ r$ if m is a non-root move pointing to move n and passing the token r (a successor of the token passed in move n)

Similarly if a is a P-first play on R.

Suppose a is an O-first play from R to S. Recall that every move is either a *source move*, where a token of R is passed, or a *target move*, where a token of S is passed. For each move m in a, we say that move m is *described as*

- $\begin{smallmatrix}\mathsf{source}\\ \mathsf{rtmove}\end{smallmatrix}\ \ r$ if m is a source root move passing the token r (a root of R)

- $\overset{\text{source}}{\underset{\text{ptr } n}{}}\ r$ if m is a source non-root move pointing to source move n and passing the token r (a successor of the token passed in move n)
- $\overset{\text{target}}{\underset{\text{rtmove}}{}}\ r$ if m is a target root move passing the token r (a root of S)
- $\overset{\text{target}}{\underset{\text{ptr } n}{}}\ r$ if m is a target non-root move pointing to target move n and passing the token r (a successor of the token passed in move n)

Similarly if a is a P-first play from R to S.

Finally, suppose a is an O-first play over Γ from R to S (Sect. 14.4.7). Recall that every move is either a *context move*, where a token of Γ is passed, a *source move*, where a token of R is passed, or a *target move*, where a token of S is passed. For each move m in a, we say that move m is *described as*

- $\overset{\text{context}}{\underset{\text{rtmove}}{}}\ r$ if m is a context root move passing the token r (a root of Γ)
- $\overset{\text{context}}{\underset{\text{ptr } n}{}}\ r$ if m is a context non-root move pointing to context move n and passing the token r (a successor of the token passed in move n)
- $\overset{\text{source}}{\underset{\text{rtmove}}{}}\ r$ if m is a source root move passing the token r (a root of R)
- $\overset{\text{source}}{\underset{\text{ptr } n}{}}\ r$ if m is a source non-root move pointing to source move n and passing the token r (a successor of the token passed in move n)
- $\overset{\text{target}}{\underset{\text{rtmove}}{}}\ r$ if m is a target root move passing the token r (a root of S)
- $\overset{\text{target}}{\underset{\text{ptr } n}{}}\ r$ if m is a target non-root move pointing to target move n and passing the token r (a successor of the token passed in move n)

We introduce a useful notation. If d is a description of a move in a game and a is a sequence of moves, we write $d \,.\, a$ for the play obtained by prefixing a move described d to a. Formally:

- the length of $d \,.\, a$ is $1 + |a|$ if $|a| < \infty$, and ∞ if $|a| = \infty$
- move 0 in $d \,.\, a$ is described as d
- move $m + 1$ in $d \,.\, a$ has the same description as move m in a, except that a pointer n is replaced by a pointer $n + 1$ because of the extra move.

B.4 Copycat Behaviour

In Sect. 9.2.6, we looked at *copycat* strategies, in which each O-move is mimicked by the following P-moved. This is a special case of a more general class of behaviour, which we now define. The definition is circular, but it has only one fixpoint because we consider finite plays only. (If we considered infinite plays, we would have to interpret the definition coinductively.)

Definition 134 Suppose that

- a is an O-first play on R, awaiting-O
- m is an O-move in a
- the token played in m is r

- the token played in $m+1$ is r'
- θ is an isomorphism from $R \upharpoonright_r$ to $R \upharpoonright_{r'}$

We say that the move-pair $m/m+1$ *begins copycat* along θ in a when, for every O-move n pointing to $m+1$ and passing the token s,

- the P-move $n+1$ points to m and passes the token $\theta^{-1}s$
- $n/n+1$ begins copycat along $\theta^{-1}s$ restricted to the isomorphism

$$R \upharpoonright_s \cong R \upharpoonright_{\theta^{-1}s}$$

in a.

□

An alternative characterization of copycat behaviour, used in e.g. [McC96], is to say that the subsequence of moves hereditarily pointing to move $m+1$ has the same structure as the subsequence of moves hereditarily pointing to move m, transformed by θ. However, we shall not require this formulation.

The definition can be adapted to the other 4 kinds of game listed in Sect. B.3. They can also be adapted to the meta-games of Sect. B.2 where the inner and outer games are pointer games of the 5 types listed in Sect. B.3. The definitions required are numerous, because m and $m+1$ could be from different games, one could be a source move and one a target move, and so forth.

B.5 Construction of Pointer Game Models

B.5.1 CBPV Term Constructors

In Sect. 9.2.8, we omitted the interpretation of term constructors. We now give some examples of these; the remaining clauses are similar.

Suppose Γ denotes $\{R_i\}_{i\in I}$ and A denotes $\{S_j\}_{j\in J}$. Then $\Gamma, \mathtt{x} : A \vdash^{\mathsf{v}} \mathtt{x} : A$ at (i,j) denotes j together with the strategy given by the set of O-first plays awaiting-O from $(R_i \uplus S_j)^{\mathsf{P}}$ to S_j^{O} such that:

- If O-move 0 is described as $\overset{\mathsf{target}}{\mathsf{rtmove}}\ s$ then P-move 1 is described as $\overset{\mathsf{source}}{\mathsf{rtmove}}\ \mathsf{inr}\ s$, beginning copycat along the isomorphism

$$R_i \uplus S_j \upharpoonright_{\mathsf{inr}\ s} \underset{\cong}{\overset{\mathsf{inr}}{\longleftarrow}} S_j \upharpoonright_s$$

Suppose Γ denotes $\{R_i\}_{i\in I}$ and A denotes $\{S_j\}_{j\in J}$ and $\underline{B}$ denotes $\{T_k\}_{k\in K}$. For given $i \in I$ and $k \in K$, suppose that $\Gamma \vdash^{\mathsf{c}} M : FA$ at i and $()$ denotes σ and that $\Gamma, \mathtt{x} : A \vdash^{\mathsf{v}} N : \underline{B}$ at (i,j) and k denotes τ_j for each $j \in J$. Then $M\ \mathtt{to}\ \mathtt{x}.\ N$ at i and k denotes $\mu(\sigma, \{\tau_j\}_{j\in J})$ where μ is the set of meta-plays awaiting meta-O with

- outer game the P-first game from R_i^{P} to T_k^{P}
- 0-inner game the P-first game from R_i^{P} to $(\mathsf{pt}^{\mathsf{A}}_{j\in J} S_j)^{\mathsf{P}}$
- $1j$-inner game the P-first game from $(R_i \uplus S_j)^{\mathsf{P}}$ to T_k^{P}

such that:

- The initial meta-P-move 0 is a start-move for a 0-inner game;

- **key clause** If a meta-O-move is a P-move m in a 0-inner-game described as $\overset{\text{target}}{\underset{\text{rtmove}}{}}\ \mathsf{root}\ j$ (an A-move), then the succeeding meta-P-move is a start-move for a $1j$-inner game. If a subsequent meta-O-move is a P-move in this $1j$-inner game described as $\overset{\text{source}}{\underset{\text{rtmove}}{}}\ \mathsf{inr}\ s$ then the succeeding meta-P-move $n+1$ is an O-move in that 0-inner game described as $\overset{\text{target}}{\underset{\text{ptr}\ m}{}}\ \mathsf{under}(j,s)$ beginning copycat along the isomorphism

$$R_i \uplus S_j \restriction_{\mathsf{inr}\ s} \xleftarrow[\cong]{\mathsf{inr}} S_j \restriction_s \xrightarrow[\cong]{\mathsf{under}(j,-)} \mathsf{pt}^{\mathsf{A}}_{j\in J} S_j \restriction_{\mathsf{under}(j,s)}$$

- If a meta-O-move is a P-move in a 0-inner game described as $\overset{\text{source}}{\underset{\text{rtmove}}{}}\ r$, then the succeeding meta-P-move is a P-move in the outer game described as $\overset{\text{source}}{\underset{\text{rtmove}}{}}\ r$, beginning copycat along the identity on $R_i \restriction_r$.

- If a meta-O-move is a P-move in a $1j$-inner game described as $\overset{\text{source}}{\underset{\text{rtmove}}{}}\ \mathsf{inl}\ r$, then the succeeding meta-P-move is a P-move in the outer game described as $\overset{\text{source}}{\underset{\text{rtmove}}{}}\ r$, beginning copycat along the isomorphism

$$R_i \uplus S_j \restriction_{\mathsf{inl}\ r} \xleftarrow[\cong]{\mathsf{inl}} R_i \restriction_r$$

- If a meta-O-move is a P-move in a $1j$-inner game described as $\overset{\text{target}}{\underset{\text{rtmove}}{}}\ t$ then the succeeding meta-P-move is a P-move in the outer game described as $\overset{\text{target}}{\underset{\text{rtmove}}{}}\ t$, beginning copycat along the identity on $T_k \restriction_t$.

Suppose Γ denotes $\{R_i\}_{i\in I}$ and $\underline{B}$ denotes $\{S_j\}_{j\in J}$. For given $i \in I$, suppose that for each $j \in J$, $\Gamma \vdash^{\mathsf{c}} M : \underline{B}$ at i and j denotes σ_j. Then $\mathtt{thunk}\ M$ at i denotes $()$ together with the strategy

$$\{\ \overset{\text{target}}{\underset{\text{rtmove}}{}}\ \mathsf{root}\ j\ .\ a \text{ replacing } \mathsf{target}\ s \mapsto \mathsf{target}\ \mathsf{under}(j,s) | a \in \sigma\}$$

The additional move at the start indicates the forcing of $\mathtt{thunk}\ M$ by the context.

Suppose Γ denotes $\{R_i\}_{i\in I}$ and $\underline{B}$ denotes $\{S_j\}_{j\in J}$. For given $i \in I$ suppose that $\Gamma \vdash^{\mathsf{v}} V : U\underline{B}$ at i denotes $((), \sigma)$. Then $\mathtt{force}\ M$ at i and j denotes the strategy

$$\{a |\ \overset{\text{target}}{\underset{\text{rtmove}}{}}\ \mathsf{root}\ j\ .\ a \text{ replacing } \mathsf{target}\ s \mapsto \mathsf{target}\ \mathsf{under}(j,s) \in \sigma\}$$

Suppose Γ denotes $\{R_i\}_{i\in I}$ and A denotes $\{S_j\}_{j\in J}$. For given $i \in I$ suppose that $\Gamma \vdash^{\mathsf{v}} V : A$ denotes (j, σ). Then $\mathtt{produce}\ V$ at i and $*$ denotes the strategy

$$\{\ \overset{\text{target}}{\underset{\text{rtmove}}{}}\ \mathsf{root}\ j\ .\ a \text{ replacing } \mathsf{target}\ s \mapsto \mathsf{target}\ \mathsf{under}(j,s) | a \in \sigma\}$$

Control Effects

As we mentioned in Sect. 9.2.2, $\mathtt{os}\ \underline{B}$ denotes the same arena family as $\underline{B}$. The semantics of terms for control effects is straightforward. As an example, we give the interpretation of the consumer $[]\ \mathtt{to}\ \mathtt{x}.\ M :: K$.

Suppose Γ denotes $\{R_i\}_{i\in I}$ and A denotes $\{S_j\}_{j\in J}$ and $\underline{B}$ denotes $\{T_k\}_{k\in K}$. For given $i \in I$, suppose that $\Gamma \vdash^{\mathsf{v}} K : \mathtt{os}\ \underline{B}$ at i denotes (k, τ) and that $\Gamma, \mathtt{x} : A \vdash^{\mathsf{c}} M : \underline{B}$ at (i, j) and k denotes σ_j for each $j \in J$. Then the consumer $[]\ \mathtt{to}\ \mathtt{x}.\ M :: K$ at i and denotes $()$ paired with $\mu(\{\sigma_j\}_{j\in J}, \tau)$ where μ is the meta-strategy defined by the meta-plays awaiting meta-O with

- outer game the P-first game from R_i^{P} to $(\mathsf{pt}^{\mathsf{A}}_{j\in J} S_j)^{\mathsf{P}}$;

- $0j$-inner game the P-first game from $(R_i \uplus S_j)^{\mathsf{P}}$ to T_k^{P};
- 1-inner game the O-first game from R_i^{P} to T_k^{O}.

such that:

- **key clause** If in the initial meta-O-move, which is the initial O-move 0 in the outer game, this O-move is described as $\overset{\mathsf{target}}{\underset{\mathsf{rtmove}}{}}\ \mathsf{root}\ j$—thus O is passing an A-token i.e. the context is producing a value to this consumer—then the succeeding meta-P-move is a start-move for a $0j$-inner game. If a subsequent meta-O-move is a P-move in this $0j$-inner game described as $\overset{\mathsf{source}}{\underset{\mathsf{rtmove}}{}}\ \mathsf{inr}\ s$ then the succeeding meta-P-move is an O-move in that 0-inner game described as $\overset{\mathsf{target}}{\underset{\mathsf{ptr}\,0}{}}\ \mathsf{under}(j,s)$ beginning copycat along the isomorphism

$$R_i \uplus S_j \restriction_{\mathsf{inr}\ s} \xleftarrow[\cong]{\mathsf{inr}} S_j \restriction_s \xrightarrow[\cong]{\mathsf{under}(j,-)} \mathsf{pt}^{\mathsf{A}}_{j\in J} S_j \restriction_{\mathsf{under}(j,s)}$$

- If a meta-O-move is a P-move in a 0-inner game described as $\overset{\mathsf{target}}{\underset{\mathsf{rtmove}}{}}\ t$ then the succeeding meta-O-move is a start-move in a 1-inner game described as $\overset{\mathsf{target}}{\underset{\mathsf{rtmove}}{}}\ t$, beginning copycat along the identity on $T_k \restriction_t$
- If a meta-O-move is a P-move in a 1-inner game described as $\overset{\mathsf{source}}{\underset{\mathsf{rtmove}}{}}\ r$, then the succeeding meta-P-move is a P-move in the outer game described as $\overset{\mathsf{source}}{\underset{\mathsf{rtmove}}{}}\ r$, beginning copycat along the identity on $R_i \restriction_r$
- If a meta-O-move is a P-move in a $0j$-inner game described as $\overset{\mathsf{source}}{\underset{\mathsf{rtmove}}{}}\ \mathsf{inl}\ r$, then the succeeding meta-P-move is a P-move in the outer game described as $\overset{\mathsf{source}}{\underset{\mathsf{rtmove}}{}}\ r$, beginning copycat along the isomorphism

$$R_i \uplus S_j \restriction_{\mathsf{inl}\ r} \xleftarrow[\cong]{\mathsf{inl}} R_i \restriction_r$$

Store

Suppose that A denotes the arena family $\{S_j\}_{j\in J}$. As we stated in Sect. 9.2.2, `ref` A denotes the same arena family as $U((FA) \sqcap (A \to \underline{\mathtt{comm}}))$, this is a singleton family whose sole arena we will call $\hat{S}$. The tokens of this arena are

Q	root read
A	under(read, root j)
Q/A	under(read, under(j, r))
Q	root write j
A	under(write j, inr done)
Q/A	under(write j, inl r)

where we write

read	for	inl ()
write j	for	inr(j, ())

`read` V `as x`. M is interpreted the same way as $(0$‘`force` $V)$ `to x`. M. $V := W$; M is interpreted the same way as $(1$‘`force` $V); M$.

Suppose Γ denotes $\{R_i\}_{i\in I}$ and A denotes $\{S_j\}_{j\in J}$ and $\underline{B}$ denotes $\{T_k\}_{k\in K}$. For given $i \in I$ and $k \in K$, suppose that $\Gamma \vdash^{\mathsf{v}} V : A$ denotes (j_0, σ) and that $\Gamma, \mathtt{x} : \mathtt{ref}\ A \vdash^{\mathsf{c}} M : \underline{B}$ at $(i, ())$ and k denotes τ. Then $\mathtt{new}\ \mathtt{x} := V;\ M$ at i and k denotes the strategy $\mu(\sigma, \tau)$ where μ is the meta-strategy given by the set of meta-plays awaiting meta-O with

- outer game the P-first game from R_i^{P} to T_k^{P}
- 0-inner game the O-first game from R_i^{P} to $S_{j_0}^{\mathsf{O}}$
- 1-inner game the P-first game from $(R_i \uplus \hat{S})^{\mathsf{P}}$ to T_k^{P}.

such that:

- The initial meta-P-move is a start move for a 1-inner play;
- If a meta-O-move is a P-move in a 1-inner play described as $\overset{\mathsf{source}}{\mathsf{rtmove}}\ \mathsf{inr}\ \mathsf{root}\ \mathsf{write}\ j$—thus passing a Q-token i.e. M assigns to $\mathtt{x}$—then the succeeding meta-P-move is a P-move in that 1-inner play described as $\overset{\mathsf{source}}{\mathsf{rtmove}}\ \mathsf{inl}\ \mathsf{under}(\mathsf{write}\ j, \mathsf{inr}\ \mathsf{done})$—thus passing an A-token i.e. the assignment is completed and control returns to M.
- If a meta-O-move is a P-move m in a 1-inner play described as $\overset{\mathsf{source}}{\mathsf{rtmove}}\ \mathsf{inl}\ \mathsf{root}\ \mathsf{read}$—thus passing a Q-token i.e. M reads $\mathtt{x}$—then:
  - If there is a preceding meta-O-move designating a P-move p in that 1-inner play described as $\overset{\mathsf{source}}{\mathsf{rtmove}}\ \mathsf{inr}\ \mathsf{root}\ \mathsf{write}\ j$ for some j (i.e. M has not written to $\mathtt{x}$ since $\mathtt{x}$ was initialized to V), we write p_{last} for the last such, and j_{last} for the corresponding j. Then the succeeding meta-P-move is an O-move $m+1$ in that 1-inner play described as $\overset{\mathsf{source}}{\mathsf{ptr}\ \tilde{m}}\ \mathsf{inr}\ \mathsf{under}(\mathsf{read}, \mathsf{root}\ j_{\mathsf{last}})$—thus passing an A-token i.e. $\mathtt{x}$ produces the value that it contains. If a subsequent meta-O-move is a P-move in that 1-inner play described as $\overset{\mathsf{source}}{\mathsf{ptr}\ m+1}\ \mathsf{inl}\ \mathsf{under}(\mathsf{read}, \mathsf{under}(j_{\mathsf{last}}, s))$, then the succeeding meta-P-move $n+1$ is an O-move in that 1-inner play described as $\overset{\mathsf{source}}{\mathsf{ptr}\ \tilde{p}_{\mathsf{last}}}\ \mathsf{inr}\ \mathsf{under}(\mathsf{write}\ j_{\mathsf{last}}, \mathsf{inl}\ s)$, beginning copycat along the isomorphism

$$\begin{array}{ccc} R_i \uplus \hat{S} \restriction_{\mathsf{inl}\ \mathsf{under}(\mathsf{read}, \mathsf{under}(j_{\mathsf{last}}, s))} & & \\ \cong \uparrow \mathsf{inl}\ \mathsf{under}(\mathsf{read}, \mathsf{under}(j_{\mathsf{last}}, -)) & & \\ S_{j_{\mathsf{last}}} \restriction_s & \xrightarrow[\mathsf{inr}\ \mathsf{under}(\mathsf{write}\ j_{\mathsf{last}}, \mathsf{inl}\ -)]{\cong} & R_i \uplus \hat{S} \restriction_{\mathsf{inr}\ \mathsf{under}(\mathsf{write}\ j_{\mathsf{last}}, \mathsf{inl}\ s)} \end{array}$$

  - If there is no preceding meta-O-move which is a P-move in that 1-inner play described as $\overset{\mathsf{source}}{\mathsf{rtmove}}\ \mathsf{inr}\ \mathsf{root}\ \mathsf{write}\ j$ for some j (i.e. M has not written to $\mathtt{x}$ since $\mathtt{x}$ was initialized to V), then the succeeding meta-P-move is an O-move $m+1$ in that 1-inner play described as $\overset{\mathsf{source}}{\mathsf{ptr}\ m}\ \mathsf{inr}\ \mathsf{under}(\mathsf{read}, \mathsf{root}\ j_0)$—thus passing an A-token i.e. $\mathtt{x}$ produces the value that it contains. If a subsequent meta-O-move is a P-move in that 1-inner play described as $\overset{\mathsf{source}}{\mathsf{ptr}\ m+1}\ \mathsf{inl}\ \mathsf{under}(\mathsf{read}, \mathsf{under}(j, s))$, then the succeeding meta-P-move is a start-move for a 0-inner play described as $\overset{\mathsf{target}}{\mathsf{rtmove}}\ s$, beginning copycat along the isomorphism

$$R_i \uplus \hat{S} \restriction_{\mathsf{inl}\ \mathsf{under}(\mathsf{read}, \mathsf{under}(j, s))} \xleftarrow[\cong]{\mathsf{inl}\ \mathsf{under}(\mathsf{read}, \mathsf{under}(j, -))} S_j \restriction_s$$

- If a meta-O-move is a P-move in a 0-inner game described as $\overset{\text{target}}{\underset{\text{rtmove}}{}}\ t$ then the succeeding meta-O-move is a start-move in a 1-inner game described as $\overset{\text{target}}{\underset{\text{rtmove}}{}}\ t$, beginning copycat along the identity on $T_k \restriction_t$
- If the meta-O-move m is a P-move in a 0-inner game described as $\overset{\text{source}}{\underset{\text{rtmove}}{}}\ r$, then the succeeding meta-P-move is a P-move in the outer game described as $\overset{\text{source}}{\underset{\text{rtmove}}{}}\ r$, beginning copycat along the identity on $R_i \restriction_r$
- If the meta-O-move m is a P-move in a 1-inner game described as $\overset{\text{source}}{\underset{\text{rtmove}}{}}\ \mathsf{inl}\ r$, then the succeeding meta-P-move is a P-move in the outer game described as $\overset{\text{source}}{\underset{\text{rtmove}}{}}\ r$, beginning copycat along the isomorphism

$$R_i \uplus \hat{S} \restriction_{\mathsf{inl}\ r} \underset{\cong}{\overset{\mathsf{inl}}{\longleftarrow}} R_i \restriction_r$$

B.5.2 Pre-Families Adjunction Model

We give the details omitted in Sect. 14.4.7

The identity from R to R is defined by the set of O-first plays awaiting-O from R^{P} to R^{O} such that:

- If O-move 0 is described as $\overset{\text{target}}{\underset{\text{rtmove}}{}}\ r$ then P-move 1 is described as $\overset{\text{source}}{\underset{\text{rtmove}}{}}\ r$, beginning copycat along the identity on $R \restriction_r$

Given an O-first strategy σ from R to S and an O-first strategy τ from S to T, the composite $\sigma;\tau$ is $\mu(\sigma,\tau)$, where μ is the meta-strategy given by the set of meta-plays awaiting meta-O with

- outer game the O-first game from R^{P} to T^{O}
- 0-inner game the O-first game from R^{P} to S^{O}
- 1-inner game the O-first game from S^{P} to T^{O}

such that:

- If in the initial meta-O-move 0, which is the initial O-move in the outer game, this O-move is described as $\overset{\text{target}}{\underset{\text{rtmove}}{}}\ t$, then the succeeding meta-P-move is an initial O-move in a 1-inner game, described as $\overset{\text{target}}{\underset{\text{rtmove}}{}}\ t$, beginning copycat along the identity on $T \restriction_t$.
- If a meta-O-move is a P-move in a 1-inner game described as $\overset{\text{source}}{\underset{\text{rtmove}}{}}\ s$, then the succeeding meta-P-move is an initial O-move in a 0-inner game described as $\overset{\text{target}}{\underset{\text{rtmove}}{}}\ s$, beginning copycat along the identity on $S \restriction_s$.
- If a meta-O-move is a P-move in a 0-inner game described as $\overset{\text{source}}{\underset{\text{rtmove}}{}}\ r$, then the succeeding meta-P-move is a P-move in the outer game described as $\overset{\text{source}}{\underset{\text{rtmove}}{}}\ r$, beginning copycat along the identity on $R \restriction_r$.

Given an O-first strategy σ from R to S and an O-first strategy σ' from R to S', the O-first strategy (σ,σ') from R to $S \uplus S'$ is the set of plays

$$\{a \text{ replacing } \overset{\text{target}}{}\ s \mapsto \overset{\text{target}}{}\ \mathsf{inl}\ s | a \in \sigma\} \cup \{a \text{ replacing } \overset{\text{target}}{}\ s' \mapsto \overset{\text{target}}{}\ \mathsf{inr}\ s' | a \in \sigma'\}$$

Given an O-first strategy σ from R to $S \uplus S'$, the strategy $\pi\sigma$ from R to S is the set of plays

$$\{a | a \text{ replacing } \mathsf{target}\ s \mapsto \mathsf{target}\ \mathsf{inl}\ s \in \sigma\}$$

Similarly $\pi'\sigma$.

This completes the construction of the cartesian category $\hat{\mathcal{C}}$.

The identity from $\underline{R}$ to $\underline{R}$ over Γ is defined by the set of O-first plays awaiting-O over Γ^{P} from $\underline{R}^{\mathsf{O}}$ to $\underline{R}^{\mathsf{P}}$ such that:

- If initial O-move 0 is described as $\overset{\mathsf{source}}{\mathsf{rtmove}}\ r$ then P-move 1 is described as $\overset{\mathsf{target}}{\mathsf{rtmove}}\ r$, beginning copycat along the identity on $\underline{R}\restriction_r$

Given an O-first strategy σ over Γ from $\underline{R}$ to $\underline{S}$ and an O-first strategy τ over Γ from $\underline{S}$ to $\underline{T}$, the composite $\sigma;\tau$ is $\mu(\sigma,\tau)$ where μ is the meta-strategy given by the set of meta-plays awaiting meta-O with

- 0-inner game the O-first game over Γ^{P} from $\underline{R}^{\mathsf{O}}$ to $\underline{S}^{\mathsf{P}}$
- 1-inner game the O-first game over Γ^{P} from $\underline{S}^{\mathsf{O}}$ to $\underline{T}^{\mathsf{P}}$
- outer game the O-first game over Γ^{P} from $\underline{R}^{\mathsf{O}}$ to $\underline{T}^{\mathsf{P}}$

such that:

- If, in the initial meta-O-move 0, which is an initial O-move in the outer game, this O-move is described as $\overset{\mathsf{source}}{\mathsf{rtmove}}\ r$, then the succeeding meta-P-move is an initial O-move in a 0-game described as $\overset{\mathsf{source}}{\mathsf{rtmove}}\ r$, beginning copycat along the identity on $\underline{R}\restriction_r$.
- If a meta-O-move is a P-move in a 0-inner game described as $\overset{\mathsf{target}}{\mathsf{rtmove}}\ s$ then the succeeding meta-P-move $m+1$ is an initial O-move in a 1-game described as $\overset{\mathsf{source}}{\mathsf{rtmove}}\ s$, beginning copycat along the identity on $\underline{S}\restriction_s$.
- If a meta-O-move is a P-move in a 1-inner game described as $\overset{\mathsf{target}}{\mathsf{rtmove}}\ t$ then the succeeding meta-P-move is a P-move in the outer game described as $\overset{\mathsf{source}}{\mathsf{rtmove}}\ t$, beginning copycat along the identity on $\underline{T}\restriction_t$.
- If a meta-O-move is a P-move in a 0-inner game described as $\overset{\mathsf{context}}{\mathsf{rtmove}}\ u$ then the succeeding meta-P-move is a Pmove in the outer game described as $\overset{\mathsf{source}}{\mathsf{rtmove}}\ u$, beginning copycat along the identity on $\Gamma\restriction_u$.
- If a meta-O-move is a P-move in a 1-inner game described as $\overset{\mathsf{context}}{\mathsf{rtmove}}\ u$ then the succeeding meta-P-move is a Pmove in the outer game described as $\overset{\mathsf{source}}{\mathsf{rtmove}}\ u$, beginning copycat along the identity on $\Gamma\restriction_u$.

The remaining reindexing and composition constructions are similar. The closure on $\hat{\mathcal{D}}$ is defined similarly to the cartesian structure on $\hat{\mathcal{C}}$.

Given, for each $i \in I$, a P-first strategy σ_i from Γ to $\underline{R}_i$, we obtain an O-first strategy from Γ^{P} to $(\mathsf{pt}^{\mathsf{Q}}_{i\in I}\underline{R}_i)^{\mathsf{O}}$ as

$$\bigcup_{i\in I}\{\overset{\mathsf{target}}{\mathsf{rtmove}}\ \mathsf{root}\ i\ .\ a \text{ replacing } \mathsf{target}\ r \mapsto \mathsf{target}\ \mathsf{under}(i,r) | a \in \sigma_i\}$$

Conversely, given an O-first strategy from Γ to $\mathsf{pt}^{\mathsf{Q}}_{i\in I}\underline{R_i}$, we obtain a P-first strategy from Γ^{P} to $\underline{R_i}^{\mathsf{P}}$ as

$$\{a|\ \begin{smallmatrix}\mathsf{target}\\ \mathsf{rtmove}\end{smallmatrix}\ \mathsf{root}\ i\ .\ a \text{ replacing } \ \mathsf{target}\ \ r \mapsto\ \ \mathsf{target}\ \ \mathsf{under}(i,r) \in \sigma\}$$

The isomorphism for F is defined similarly.

B.5.3 JWA Term Constructors

We provide the details omitted in Sect. 9.3.5. Most of the constructs (including the storage constructs) are interpreted as in Sect. B.5.1. We will describe how to interpret the $\neg$ constructs.

Suppose Γ denotes $\{R_i\}_{i\in I}$ and A denotes $\{S_j\}_{j\in J}$. Suppose that, for each $j \in J$, the non-returning command $\Gamma, \mathtt{x} : A \vdash^{\mathsf{n}} M$ denotes at (i,j) a P-first strategy σ_j on $R_i \uplus S_j$. Then $\gamma\mathtt{x}.M$ denotes at i the O-first strategy from R_i^{P} to $(\mathsf{pt}_{j\in J}S_j)^{\mathsf{O}}$ defined by

$$\bigcup_{j\in J}\{\ \begin{smallmatrix}\mathsf{target}\\ \mathsf{rtmove}\end{smallmatrix}\ \mathsf{root}\ j\ .\ a \text{ replacing } \begin{array}{lll}\mathsf{inl}\ r & \mapsto & \mathsf{source}\ \ r\\ \mathsf{inr}\ s & \mapsto & \mathsf{target}\ \ s\end{array} |a \in \sigma_j\}$$

Suppose Γ denotes $\{R_i\}_{i\in I}$ and A denotes $\{S_j\}_{j\in J}$. Suppose that the value $\Gamma \vdash^{\mathsf{v}} V : A$ denotes at i the pair (j,σ) and that the value $\Gamma \vdash^{\mathsf{v}} W : \neg A$ denotes at i the pair $((),\tau)$. Then the non-returning command $V \nearrow W$ at i denotes the strategy $\mu(\sigma,\tau)$ where μ is the meta-strategy defined by the set of meta-plays awaiting meta-O with

- outer game the P-first game on R_i^{P}
- 0-inner game the O-first game from R_i^{P} to S_j^{O}
- 1-inner game the O-first game from R_i^{P} to $(\mathsf{pt}_{j\in J}S_j)^{\mathsf{O}}$.

such that:

- The initial meta-P-move is the initial O-move 0 in a 1-inner game described as $\begin{smallmatrix}\mathsf{target}\\ \mathsf{rtmove}\end{smallmatrix}\ \mathsf{root}\ j$. If a subsequent meta-$\mathsf{O}$-move is a P-move in this 1-inner game described as $\begin{smallmatrix}\mathsf{target}\\ \mathsf{ptr}\ 0\end{smallmatrix}\ \mathsf{under}(j,r)$, then the succeeding meta-P-move is an initial O-move in a 0-inner game described as $\begin{smallmatrix}\mathsf{target}\\ \mathsf{ptr}\ 0\end{smallmatrix}\ r$, beginning copycat along

$$\mathsf{pt}_{j\in J}S_j \restriction_{\mathsf{under}(j,r)} \xleftarrow{\mathsf{under}(j,-)} S_j \restriction_r$$

- If a meta-O-move is a P-move in a 0-inner game described as $\begin{smallmatrix}\mathsf{source}\\ \mathsf{rtmove}\end{smallmatrix}\ u$, then the succeeding meta-P-move is a P-move in the outer game described as $\mathsf{rtmove}\ \ u$, beginning copycat along the identity on $R_i \restriction_u$.
- If a meta-O-move is a P-move in a 1-inner game described as $\begin{smallmatrix}\mathsf{source}\\ \mathsf{rtmove}\end{smallmatrix}\ u$, then the succeeding meta-P-move is a P-move in the outer game described as $\mathsf{rtmove}\ \ u$, beginning copycat along the identity on $R_i \restriction_u$.

B.5.4 Pre-Families Non-Return Model

We give the details omitted in Sect. 8.8.2. The cartesian category $\mathcal{C}$ is constructed as in Sect. B.5.2.

Given an O-first strategy σ from R to S and a P-first strategy τ on S, we define $\sigma;\tau$ to be $\mu(\sigma,\tau)$ where μ is the meta-strategy defined by the set of meta-plays awaiting meta-O with

- outer game the P-first game on R^{P}
- 0-inner game the O-first game from R^{P} to S^{O}
- 1-inner game the P-first game on S^{P}.

such that:

- The initial meta-P-move 0 is a start move for a 1-inner game. If a subsequent meta-O-move designates a P-move for this 1-inner game described as $\mathsf{rtmove}\ s$, then the succeeding meta-P-move is an initial O-move in a 0-inner game described as $\overset{\mathsf{target}}{\mathsf{rtmove}}\ s$, beginning copycat along the identity on $S\restriction_s$.
- If a meta-O-move is an initial O-move for a 0-inner game described as $\overset{\mathsf{source}}{\mathsf{rtmove}}\ r$ then the succeeding meta-P-move is a P-move for the outer game described as $\mathsf{rtmove}\ r$, beginning copycat along the identity on $R\restriction_r$.

The isomorphism for $\neg$ is described similarly to the isomorphism for U in Sect. B.5.2.

B.6 Type Definability Proof

The aim of this section is to construct the isomorphisms (9.1)–(9.2) required in the proof of Prop. 81.

B.6.1 Global Semantics of Types

Before we do this, we describe the semantics of types in a "global" way. We define predicates

Predicate	Intended meaning
$A : i$	i is an index in $[\![A]\!]$
$A : i : r - \mathsf{Q}$	r is a Q-token in $[\![A]\!]i$
$A : i : r - \mathsf{A}$	r is an A-token in $[\![A]\!]i$
$A : i : r - \mathsf{root}$	r is a root of $[\![A]\!]i$
$A : i : r \vdash s$	$r \vdash s$ in $[\![A]\!]i$

inductively, using the following clauses for U

$$\frac{}{U\underline{B} : ()} \qquad \frac{\underline{B} : i}{U\underline{B} : () : \mathsf{root}\ i - \mathsf{Q}}$$

$$\frac{\underline{B} : i : r - \mathsf{Q}}{U\underline{B} : () : \mathsf{under}(i,r) - \mathsf{Q}} \qquad \frac{\underline{B} : i : r - \mathsf{A}}{U\underline{B} : () : \mathsf{under}(i,r) - \mathsf{A}}$$

$$\frac{\underline{B} : i}{U\underline{B} : () : \mathsf{root}\ i - \mathsf{root}}$$

$$\frac{\underline{B} : i : r - \mathsf{root}}{U\underline{B} : () : \mathsf{root}\ i \vdash \mathsf{under}(i,r)} \qquad \frac{\underline{B} : i : r \vdash s}{U\underline{B} : () : \mathsf{under}(i,r) \vdash \mathsf{under}(i,s)}$$

and similar clauses for all the other type constructors. We have to show that this agrees with the semantics of types that we gave in Sect. 9.2.2. (To keep things simple, we will ignore type

recursion, as it is not used in the type canonical forms.) First we prove (both for value types and computation types) that if $A \lhd_{\mathsf{fin}} B$ then

- i is an index of $[\![A]\!]$ implies $B : i$
- r is a Q-token of $[\![A]\!]i$ implies $B : i : r - \mathsf{Q}$

and similarly for the other 3 predicates. This is by induction on $A \lhd_{\mathsf{fin}} B$. Then we prove that

- if $B : i$ then i is an index of $[\![A]\!]$ for some $A \lhd_{\mathsf{fin}} B$
- if $B : i : r - \mathsf{Q}$ then r is a Q-token of $[\![A]\!]i$ for some $A \lhd_{\mathsf{fin}} B$

and similarly for the other 3 predicates. This is by induction on the 5 predicates. Consequently, for any type B,

- i is an index of $[\![B]\!]$ iff $B : i$
- r is a Q-token of $[\![B]\!]i$ iff $B : i : r - \mathsf{Q}$

and similarly for the other 3 predicates. Thus, as required, the two descriptions of the semantics of types agree.

B.6.2 Constructing The Isomorphisms

Now we construct the required isomorphisms (9.1)–(9.2). First we have to give a bijection on indexing sets. It is easy to see that both $[\![\theta_{\mathsf{val}}\{R_i\}_{i\in I}]\!]$ and $[\![\theta_{\mathsf{comp}}\{R_i\}_{i\in I}]\!]$ are indexed by $I \times (1 \times 1)$. So, for both (9.1) and (9.2), the bijection on indexing sets takes $i \in I$ to ${}^\circ i = (i, ((), ()))$. We define two predicates

Predicate	Intended meaning
$\{R_i\}_{i\in I} : \hat{\imath} : r \mapsto_{\mathsf{val}} r'$	r in arena $\hat{\imath}$ corresponds across (9.1) to r' in arena ${}^\circ\hat{\imath}$
$\{R_i\}_{i\in I} : \hat{\imath} : r \mapsto_{\mathsf{comp}} r'$	r in arena $\hat{\imath}$ corresponds across (9.2) to r' in arena ${}^\circ\hat{\imath}$

inductively as follows

$$\frac{j \in \mathsf{Qrt}\ R_{\hat{\imath}}}{\{R_i\}_{i\in I} : \hat{\imath} : j \mapsto_{\mathsf{val}} \mathsf{inl}\ \mathsf{root}\ j} \qquad \frac{\{R_{\hat{\imath}} \restriction_j\}_{j\in \mathsf{Qrt}\ R_i} : \hat{\jmath} : s \mapsto_{\mathsf{comp}} s'}{\{R_i\}_{i\in I} : \hat{\imath} : s \mapsto_{\mathsf{val}} \mathsf{inl}\ \mathsf{under}({}^\circ\hat{\jmath}, s')}$$

$$\frac{j \in \mathsf{Art}\ R_{\hat{\imath}}}{\{R_i\}_{i\in I} : \hat{\imath} : j \mapsto_{\mathsf{val}} \mathsf{inr}\ \mathsf{root}\ j} \qquad \frac{\{R_{\hat{\imath}} \restriction_j\}_{j\in \mathsf{Art}\ R_i} : \hat{\jmath} : s \mapsto_{\mathsf{val}} s'}{\{R_i\}_{i\in I} : \hat{\imath} : s \mapsto_{\mathsf{val}} \mathsf{inr}\ \mathsf{under}({}^\circ\hat{\jmath}, s')}$$

$$\frac{j \in \mathsf{Qrt}\ R_{\hat{\imath}}}{\{R_i\}_{i\in I} : \hat{\imath} : j \mapsto_{\mathsf{comp}} \mathsf{inl}\ \mathsf{root}\ j} \qquad \frac{\{R_{\hat{\imath}} \restriction_j j \in \mathsf{Qrt}\ R_{\hat{\imath}}\} : \hat{\jmath} : s \mapsto_{\mathsf{comp}} s'}{\{R_i\}_{i\in I} : \hat{\imath} : s \mapsto_{\mathsf{comp}} \mathsf{inl}\ \mathsf{under}({}^\circ\hat{\jmath}, s')}$$

$$\frac{j \in \mathsf{Art}\ R_{\hat{\imath}}}{\{R_i\}_{i\in I} : \hat{\imath} : j \mapsto_{\mathsf{comp}} \mathsf{inr}\ \mathsf{root}\ j} \qquad \frac{\{R_{\hat{\imath}} \restriction_j j \in \mathsf{Art}\ R_{\hat{\imath}}\} : \hat{\jmath} : s \mapsto_{\mathsf{val}} s'}{\{R_i\}_{i\in I} : \hat{\imath} : s \mapsto_{\mathsf{comp}} \mathsf{inr}\ \mathsf{under}({}^\circ\hat{\jmath}, s')}$$

We prove that

- if r is a Q-token of $R_{\hat{\imath}}$ then there is a unique r' such that $\{R_i\}_{i\in I} : \hat{\imath} : r \mapsto_{\mathsf{val}} r'$, and $\theta_{\mathsf{val}}\{R_i\}_{i\in I} : {}^\circ\hat{\imath} : r' - \mathsf{Q}$
- similarly for A-tokens
- if r is a root of $R_{\hat{\imath}}$ and $\{R_i\}_{i\in I} : \hat{\imath} : r \mapsto_{\mathsf{val}} r'$ then $\theta_{\mathsf{val}}\{R_i\}_{i\in I} : {}^\circ\hat{\imath} : r' - \mathrm{root}$

- if $r \vdash s$ in $R_{\hat{\imath}}$ and $\{R_i\}_{i\in I} : \hat{\imath} : r \mapsto_{\mathsf{val}} r'$ and $\{R_i\}_{i\in I} : \hat{\imath} : s \mapsto_{\mathsf{val}} s'$ then $\theta_{\mathsf{val}}\{R_i\}_{i\in I} : {}^{\circ}\hat{\imath} : r' \vdash s'$
- if r is a root of $R_{\hat{\imath}}$ and $\{R_i\}_{i\in I} : \hat{\imath} : r \mapsto_{\mathsf{val}} r'$ then
- if r is a Q-token of $R_{\hat{\imath}}$ then there is a unique r' such that $\{R_i\}_{i\in I} : \hat{\imath} : r \mapsto_{\mathsf{comp}} r'$, and $\theta_{\mathsf{comp}}\{R_i\}_{i\in I} : {}^{\circ}\hat{\imath} : r' - \mathsf{Q}$
- similarly for A-tokens
- if r is a root of $R_{\hat{\imath}}$ and $\{R_i\}_{i\in I} : \hat{\imath} : r \mapsto_{\mathsf{comp}} r'$ then $\theta_{\mathsf{comp}}\{R_i\}_{i\in I} : {}^{\circ}\hat{\imath} : r' - \text{root}$
- if $r \vdash s$ in $R_{\hat{\imath}}$ and $\{R_i\}_{i\in I} : \hat{\imath} : r \mapsto_{\mathsf{comp}} r'$ and $\{R_i\}_{i\in I} : \hat{\imath} : s \mapsto_{\mathsf{comp}} s'$ then $\theta_{\mathsf{comp}}\{R_i\}_{i\in I} : {}^{\circ}\hat{\imath} : r' \vdash s'$

simultaneously, by induction on the depth of r. We prove that

- if $A : i' : r' - \mathsf{Q}$ then if $A = \theta_{\mathsf{val}}\{R_i\}_{i\in I}$ then
  - $i' =^{\circ} \hat{\imath}$ for some unique $i \in I$
  - there is a unique r such that $\{R_i\}_{i\in I} : \hat{\imath} : r \mapsto_{\mathsf{val}} r'$
  - this r is a Q-token in $R_{\hat{\imath}}$
- similarly for A-tokens
- if $A : i' : r' - \text{root}$ then if $A = \theta_{\mathsf{val}}\{R_i\}_{i\in I}$ then
  - $i' =^{\circ} \hat{\imath}$ for some unique $i \in I$
  - $\{R_i\}_{i\in I} : \hat{\imath} : r \mapsto_{\mathsf{val}} r'$ implies that r is a root of $R_{\hat{\imath}}$
- if $A : i' : r' \vdash s'$ then if $A = \theta_{\mathsf{val}}\{R_i\}_{i\in I}$ then
  - $i' =^{\circ} \hat{\imath}$ for some unique $i \in I$
  - $\{R_i\}_{i\in I} : \hat{\imath} : r \mapsto_{\mathsf{val}} r'$ and $\{R_i\}_{i\in I} : \hat{\imath} : s \mapsto_{\mathsf{val}} s'$ implies that $r \vdash s$ in $R_{\hat{\imath}}$
- if $A : i' : r' - \mathsf{Q}$ then if $A = \theta_{\mathsf{comp}}\{R_i\}_{i\in I}$ then
  - $i' =^{\circ} \hat{\imath}$ for some unique $i \in I$
  - there is a unique r such that $\{R_i\}_{i\in I} : \hat{\imath} : r \mapsto_{\mathsf{comp}} r'$
  - this r is a Q-token in $R_{\hat{\imath}}$
- similarly for A-tokens
- if $A : i' : r' - \text{root}$ then if $A = \theta_{\mathsf{comp}}\{R_i\}_{i\in I}$ then
  - $i' =^{\circ} \hat{\imath}$ for some unique $i \in I$
  - $\{R_i\}_{i\in I} : \hat{\imath} : r \mapsto_{\mathsf{comp}} r'$ implies that r is a root of $R_{\hat{\imath}}$
- if $A : i' : r' \vdash s'$ then if $A = \theta_{\mathsf{comp}}\{R_i\}_{i\in I}$ then
  - $i' =^{\circ} \hat{\imath}$ for some unique $i \in I$
  - $\{R_i\}_{i\in I} : \hat{\imath} : r \mapsto_{\mathsf{comp}} r'$ and $\{R_i\}_{i\in I} : \hat{\imath} : s \mapsto_{\mathsf{comp}} s'$ implies that $r \vdash s$ in $R_{\hat{\imath}}$

simultaneously by induction. We define

- isomorphism (9.1) by saying that $r \in R_{\hat{\imath}}$ corresponds to $r' \in [\![\theta_{\mathsf{val}}\{R_i\}_{i\in I}]\!]^{\circ}\hat{\imath}$ when $\{R_i\}_{i\in I} : \hat{\imath} : r \mapsto_{\mathsf{val}} r'$
- isomorphism (9.2) by saying that $r \in R_{\hat{\imath}}$ corresponds to $r' \in [\![\theta_{\mathsf{comp}}\{R_i\}_{i\in I}]\!]^{\circ}\hat{\imath}$ when $\{R_i\}_{i\in I} : \hat{\imath} : r \mapsto_{\mathsf{comp}} r'$

and we have shown that these indeed give isomorphisms.

Index

Bibliography

[Abr90] S. Abramsky. The lazy lambda-calculus. In *Research topics in Functional Programming*, pages 65–117. Addison Wesley, 1990. (p 37)

[Abr92] S. Abramsky. Games and full completeness for multiplicative linear logic (extended abstract). In Rudrapatna Shyamasundar, editor, *Proceedings of Foundations of Software Technology and Theoretical Computer Science*, volume 652 of *LNCS*, pages 291–301, Berlin, Germany, December 1992. Springer. (p 139)

[Acz88] P. Aczel. *Non-Well-Founded Sets*. Center for the Study of Language and Information, Stanford University, 1988. CSLI Lecture Notes, Volume 14. (p 79)

[AHM98] S. Abramsky, K. Honda, and G. McCusker. A fully abstract game semantics for general references. Proceedings, Thirteenth Annual IEEE Symposium on Logic in Computer Science, IEEE Computer Society Press, 1998. (p 139)

[AJ89] A. W. Appel and T. Jim. Continuation-passing, closure-passing style. In *Conference Record of the 16th Annual ACM Symposium on Principles of Programming Languages*, pages 293–302. ACM, ACM, 1989. (p 118)

[AJM94] S. Abramsky, R. Jagadeesan, and P. Malacaria. Full abstraction for PCF (extended abstract). In Masami Hagiya and John C. Mitchell, editors, *Theoretical Aspects of Computer Software. International Symposium TACS'94*, volume 789 of *LNCS*, pages 1–15, Sendai, Japan, April 1994. Springer-Verlag. (p 139)

[AM97] S. Abramsky and G. McCusker. Linearity, sharing and state: a fully abstract game semantics for Idealized Algol with active expressions. In P. W. O'Hearn and R. D. Tennent, editors, *Algol-like languages*. Birkhaüser, 1997. (pp 139, 145)

[AM98a] S. Abramsky and G. McCusker. Call-by-value games. In M. Nielsen and W. Thomas, editors, *Computer Science Logic: 11th International Workshop Proceedings*, LNCS. Springer-Verlag, 1998. (pp 17, 70, 74, 132, 140, 215, 278, 281)

[AM98b] S. Abramsky and G. McCusker. Game semantics. Lecture notes to accompany Samson Abramsky's lectures at the 1997 Marktoberdorf summer school. To appear, 1998. (p 154)

[App91] A. W. Appel. *Compiling with Continuations.* CUP, 1991. (p 118)

[Bie98] G. M. Bierman. A computational interpretation of the $\lambda\mu$-calculus. In L. Brim, J. Gruska, and J. Zlatuska, editors, *Mathematical Foundations of Computer Science, Proc. 23rd Int. Symp.*, volume 1450 of *LNCS*, pages 336–345, Brno, Czech Republic, August 1998. Springer-Verlag, Berlin. (pp 46, 138)

[BW96] N. Benton and P. Wadler. Linear logic, monads and the lambda calculus. In *Proceedings, 11$^{\text{th}}$ Annual IEEE Symposium on Logic in Computer Science*, pages 420–431, New Brunswick, 1996. IEEE Computer Society Press. (pp 18, 254)

[CLW93] A. Carboni, S. Lack, and R. F. C. Walters. Introduction to extensive and distributive categories. *Journal of Pure and Applied Algebra*, 84:145–158, 1993. (p 173)

[Coc93] J. R. B. Cockett. Introduction to distributive categories. *Mathematical Structures in Computer Science*, 3(3):277–307, September 1993. (pp 173, 174)

[Cro94] R. L. Crole. *Categories for Types*. Cambridge Mathematical Textbooks. Cambridge, Cambridge, UK, 1994. (p 177)

[Dan92] O. Danvy. Back to direct style. In *ESOP'92, Proc. European Symposium on Programming, Rennes*, volume 582 of *LNCS*, pages 130–150. Springer Verlag, 1992. (pp 118, 119)

[DHM91] B. Duba, R. Harper, and D. MacQueen. Typing first-class continuations in ML. In ACM-SIGPLAN ACM-SIGACT, editor, *Conference Record of the 18th Annual ACM Symposium on Principles of Programming Languages (POPL '91)*, pages 163–173, Orlando, FL, USA, 1991. ACM Press. (p 124)

[DHR96] V. Danos, H. Herbelin, and L. Regnier. Game semantics and abstract machines. In *Proceedings of the Eleventh Annual IEEE Symposium On Logic In Computer Science (LICS'96)*, pages 394–405, New York, 1996. IEEE Computer Society Press. (pp 140, 154)

[FF86] M. Felleisen and D. P. Friedman. Control operators, the SECD-machine, and the λ-calculus. In M. Wirsing, editor, *Formal Description of Programming Concepts*, pages 193–217. North-Holland, 1986. (pp 23, 42, 46)

[Fil96] A. Filinski. *Controlling Effects*. PhD thesis, School of Computer Science, Carnegie Mellon University, Pittsburgh, Pennsylvania, 1996. (pp 18, 19)

[FPD99] M. Fiore, G. D. Plotkin, and D.Turi. Abstract syntax and variable binding. In G. Longo, editor, *Proceedings of the 14th Annual Symposium on Logic in Computer Science (LICS'99)*, pages 193–202, Trento, Italy, July 1999. IEEE Computer Society Press. (p 80)

[Fre91] P. J. Freyd. Algebraically complete categories. In A. Carboni et al., editors, *Proc. 1990 Como Category Theory Conference*, pages 95–104, Berlin, 1991. Springer-Verlag. Lecture Notes in Mathematics Vol. 1488. (p 74)

[Fri78] H. Friedman. Classically and intuitionistically provably recursive functions. In Scott, D. S. and G. H. Muller, editor, *Higher Set Theory*, volume 699 of *Lecture Notes in Mathematics*, pages 21–28. Springer-Verlag, 1978. (pp 119, 137)

[Ghi97] D. R. Ghica. Semantics of dynamic variables in algol-like languages. Master's thesis, Queens' University, Kingston,Ontario, 1997. (pp 24, 103, 114)

[Gir87] J.-Y. Girard. Linear logic. *Theoretical Computer Science*, 50:1–102, 1987. (p 18)

[GLT88] J.-Y. Girard, Y. Lafont, and P. Taylor. *Proofs and Types*. Cambridge Tracts in Theoretical Computer Science 7. Cambridge University Press, 1988. (pp 29, 136)

[Gri90] T. G. Griffin. The formulae-as-types notion of control. In *Conf. Record 17th Annual ACM Symp. on Principles of Programming Languages, POPL'90, San Francisco, CA, USA, 17–19 Jan 1990*, pages 47–57. ACM Press, New York, 1990. (pp 119, 137)

[GTW79] J. Goguen, J. Thatcher, and E. Wagner. An initial algebra approach to the specification, correctness, and implementation of abstract data types. In R. Yeh, editor, *Current Trends in Programming Methodology IV*, pages 80–149. Prentice-Hall, 1979. (p 81)

[Gun95] C. A. Gunter. *Semantics of Programming Languages*. MIT Press, 1995. (p 17)

[HA80] M. C. B. Hennessy and E. A. Ashcroft. A mathematical semantics for a nondeterministic typed lambda -calculus. *Theoretical Computer Science*, 11(3):227–245, July 1980. (p 37)

[HD97] J. Hatcliff and O. Danvy. Thunks and the λ-calculus. *Journal of Functional Programming*, 7(3):303–319, May 1997. (pp 17, 18, 37, 124, 270)

[Hen80] M. C. B. Hennessy. The semantics of call-by-value and call-by-name in a nondeterministic environment. *SIAM Journal on Computing*, 9(1):67–84, February 1980. (p 16)

[HM99] R. Harmer and G. McCusker. A fully abstract game semantics for finite nondeterminism. In *LICS: IEEE Symposium on Logic in Computer Science*, 1999. (p 139)

[HO94] M. Hyland and L. Ong. On full abstraction for PCF. submitted, 1994. (pp 17, 70, 139, 142, 149)

[Hof95] M. Hofmann. Sound and complete axiomatisations of call-by-value control operators. *Mathematical Structures in Computer Science*, 5(4):461–482, December 1995. (p 130)

[HS97] M. Hofmann and Th. Streicher. Continuation models are universal for $\lambda\mu$-calculus. In *Proceedings, Twelth Annual IEEE Symposium on Logic in Computer Science*, pages 387–395, Warsaw, Poland, 1997. IEEE Computer Society Press. (pp 93, 124, 137)

[HY97] K. Honda and N. Yoshida. Game theoretic analysis of call-by-value computation. In P. Degano, R. Gorrieri, and A. Marchetti-Spaccamela, editors, *Automata, Languages and Programming, 24th International Colloquium*, volume 1256 of *LNCS*, pages 225–236, Bologna, Italy, 1997. Springer-Verlag. (pp 17, 140)

[Ing61] P. Z. Ingerman. Thunks: A way of compiling procedure statements with some comments on procedure declarations. *Communications of the ACM*, 4(1):55–58, January 1961. (p 20)

[Jac99] Bart Jacobs. *Categorial Logic and Type Theory*. Elsevier, Amsterdam, 1999. (pp 24, 163, 176)

[Jef99] A. Jeffrey. A fully abstract semantics for a higher-order functional language with nondeterministic computation. *TCS: Theoretical Computer Science*, 228, 1999. (p 168)

[Jun90] Achim Jung. Colimits in DCPO. 3-page manuscript, available by fax, 1990. (p 115)

[Kri85] J.-L. Krivine. Un interpréteur de λ-calcul. Unpublished, 1985. (p 46)

[Lai97] J. Laird. Full abstraction for functional languages with control. In *Proceedings, Twelth Annual IEEE Symposium on Logic in Computer Science*, pages 58–67, Warsaw, Poland, 1997. IEEE Computer Society Press. (pp 37, 139, 141)

[Lai98] J. Laird. *A Semantic Analysis of Control*. PhD thesis, University of Edinburgh, 1998. (p 93)

[Lan64] P. J. Landin. The mechanical evaluation of expressions. *The Computer Journal*, 6(4):308–320, January 1964. (p 46)

[Lev96] P. B. Levy. λ-calculus and cartesian closed categories. Essay for Part III of the Mathematical Tripos, Cambridge University, manuscript, 1996. (p 168)

[Lev99] P. B. Levy. Call-by-push-value: a subsuming paradigm (extended abstract). In J.-Y Girard, editor, *Typed Lambda-Calculi and Applications*, volume 1581 of *LNCS*, pages 228–242, L'Aquila, 1999. Springer. (p 95)

[LP97] J.R. Longley and G.D. Plotkin. Logical full abstraction and PCF. In J. Ginzburg, editor, *Tbilisi Symposium on Language, Logic and Computation*. SiLLI/CSLI, 1997. Also available as LFCS Report ECS-LFCS-97-353. (p 139)

[LRS93] Y. Lafont, B. Reus, and Th. Streicher. Continuation semantics or expressing implication by negation. Technical Report 9321, Ludwig-Maximilians-Universität, München, 1993. (pp 119, 137)

[Mac71] S. MacLane. *Categories for Working Mathematicians*. Springer-Verlag, New York, 1971. (pp 164, 186, 191, 201)

[Mar00] M. Marz. *A Fully Abstract Model for Sequential Computation*. PhD thesis, Technische Universität Darmstadt, 2000. published by Logos-Verlag, Berlin. (p 18)

[McC96] G. McCusker. *Games and Full Abstraction for a Functional Metalanguage with Recursive Types*. PhD thesis, University of London, 1996. (pp 79, 142, 143, 144, 288)

[McC97] G. McCusker. Games and definability for **FPC**. *The Bulletin of Symbolic Logic*, 3(3):347–362, September 1997. (pp 139, 157)

[Mog88] E. Moggi. Computational lambda-calculus and monads. LFCS Report ECS-LFCS-88-66, University of Edinburgh, October 1988. (pp 18, 35)

[Mog91] E. Moggi. Notions of computation and monads. *Information and Computation*, 93:55–92, 1991. (pp 16, 18, 19, 34, 35, 37, 84, 100, 189, 190, 197, 253, 263)

[Mog90] E Moggi. An abstract view of programming languages. Technical Report ECS-LFCS-90-113, Dept. of Computer Science, Edinburgh Univ., 90. (pp 17, 102, 115)

[MRS99] M. Marz, A. Rohr, and Th. Streicher. Full abstraction and universality via realisability. In *LICS: IEEE Symposium on Logic in Computer Science*, 1999. (p 18)

[Mur90] C. Murthy. *Extracting Constructive Content from Classical Proofs*. PhD thesis, Cornell University, Department of Computer Science, 1990. (p 137)

[Nic96] H. Nickau. *Hereditarily Sequential Functionals: A Game-Theoretic Approach to Sequentiality*. Shaker-Verlag, 1996. Dissertation, Universität Gesamthochschule Siegen. (pp 17, 139)

[Ode94] M. Odersky. A functional theory of local names. In ACM, editor, *Proceedings of 21st Annual ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL)*, pages 48–59, New York, NY, USA, January 1994. ACM Press. (p 115)

[O'H93] P. W. O'Hearn. Opaque types in algol-like languages. manuscript, 1993. (pp 85, 100)

[Ole82] F. J. Oles. *A Category-Theoretic Approach to the Semantics of Programming Languages*. Ph. D. dissertation, Syracuse University, August 1982. (pp 17, 102, 103, 114)

[Ong88] C. H. L. Ong. *The Lazy Lambda Calculus: An Investigation into the Foundations of Functional Programming*. PhD thesis, Imperial College of Science and Technology, 1988. (p 37)

[Ong96] C.-H. L. Ong. A semantics view of classical proofs: type-theoretic, categorical, denotational characterizations. In *Proc. 11th IEEE Annual Symposium on Logic in Computer Science*, pages 230–241. IEEE Computer Society Press, 1996. (p 138)

[OT95] P. W. O'Hearn and R. D. Tennent. Parametricity and local variables. *Journal of the ACM*, 42(3):658–709, May 1995. (p 115)

[Par68] D. Park. Some semantics for data structures. In D. Michie, editor, *Machine Intelligence 3*, pages 351–371. American Elsevier, New York, 1968. (p 19)

[Par92] M. Parigot. $\lambda\mu$-calculus: An algorithmic interpretation of classical natural deduction. In A. Voronkov, editor, *Proceedings of the International Conference on Logic Programming and Automated Reasoning (LPAR'92)*, volume 624 of *LNAI*, pages 190–201, St. Petersburg, Russia, 1992. Springer Verlag. (p 138)

[Pit96] A. M. Pitts. Relational properties of domains. *Information and Computation*, 127:66–90, 1996. (A preliminary version of this work appeared as Cambridge Univ. Computer Laboratory Tech. Rept. No. 321, December 1993.). (pp 76, 117)

[Plo76] G. D. Plotkin. Call-by-name, call-by-value and the λ-calculus. *Theoretical Computer Science*, 1(1):125–159, 1976. (pp 18, 37, 124, 126)

[Plo77] G. D. Plotkin. LCF as a programming language. *Theoretical Computer Science*, 5, 1977. (pp 16, 37)

[Plo83] G. D. Plotkin. Domains. 1992 TeXed edition of course notes prepared by Yugo Kashiwagi and Hidetaka Kondoh from notes by Tatsuya Hagino., 1983. (p 70)

[Plo85] G. D. Plotkin. Lectures on predomains and partial functions. Course notes, Center for the Study of Language and Information, Stanford, 1985. (pp 16, 34)

[PR97] A. J. Power and E. P. Robinson. Premonoidal categories and notions of computation. *Math. Struct. in Comp. Sci.*, 7(5):453–468, October 1997. (pp 199, 202)

[PS93] A. M. Pitts and I. D. B. Stark. Observable properties of higher order functions that dynamically create local names, or: What's new? In *Mathematical Foundations of Computer Science, Proc. 18th Int. Symp., Gdańsk, 1993*, volume 711 of *LNCS*, pages 122–141. Springer-Verlag, Berlin, 1993. (p 102)

[PT97] A. J. Power and H. Thielecke. Environments, continuation semantics and indexed categories. In *Proceedings TACS'97*, volume 1281 of *LNCS*, pages 391–414. Springer-Verlag, Berlin, 1997. (p 186)

[PT99] A. J. Power and H. Thielecke. Closed Freyd- and kappa-categories. In *Proc. ICALP '99*, volume 1644 of *LNCS*, pages 625–634. Springer-Verlag, Berlin, 1999. (pp 186, 199, 202)

[Rey81] J. C. Reynolds. The essence of Algol. In J. W. de Bakker and J. C. van Vliet, editors, *Algorithmic Languages*, pages 345–372, Amsterdam, 1981. North-Holland. (pp 102, 145, 157)

[San99] D. Sangiorgi. Interpreting functions as pi-calculus processes: a tutorial. RR 3470, INRIA Sophia-Antipolis, France, February 1999. (pp 17, 18)

[Sco82] D. S. Scott. Domains for denotational semantics. In M. Nielson and E. M. Schmidt, editors, *Automata, Languages and Programming: Proceedings 1982*. Springer-Verlag, Berlin, 1982. LNCS **140**. (p 79)

[SF93] A. Sabry and M. Felleisen. Reasoning about programs in continuation-passing style. *Lisp and Symbolic Computation*, 6(3/4):289–360, November 1993. (pp 118, 119, 133)

[SHLG94] V. Stoltenberg-Hansen, I. Lindstroem, and E. R. Griffor. *Mathematical Theory of Domains*. Cambridge University Press, Cambridge, 1 edition, 1994. (pp 69, 70, 75, 79)

[Sim92] A. K. Simpson. Recursive types in Kleisli categories. Unpublished manuscript, 1992. (p 40)

[SJ98] M. Shields and S. Peyton Jones. Bridging the gulf better. Draft, 1998. (p 18)

[SP82] M. Smyth and G. D. Plotkin. The category-theoretic solution of recursive domain equations. *SIAM J. Computing*, 11, 1982. (pp 73, 74)

[SR98] Th. Streicher and B. Reus. Classical logic, continutation semantics and abstract machines. *Journal of Functional Programming*, 8(6):543–572, November 1998. (pp 17, 46, 85, 93, 100, 130)

[Sta94] I. D. B. Stark. *Names and Higher-Order Functions*. PhD thesis, University of Cambridge, December 1994. Also published as Technical Report 363, University of Cambridge Computer Laboratory. (pp 17, 102, 115)

[Ste78] G. L. Steele. RABBIT: A compiler for SCHEME. Technical Report 474, Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, MA, 1978. MSc Dissertation. (pp 24, 118, 119)

[Tai67] W. W. Tait. Intensional interpretation of functionals of finite type I. *Journal of Symbolic Logic*, 32(2):198–212, June 1967. (pp 44, 71, 113, 127)

[TG00] Robert D. Tennent and Dan R. Ghica. Abstract models of storage. *Higher-Order and Symbolic Computation*, 13(1–2):119–129, April 2000. (p 19)

[Thi97a] H. Thielecke. *Categorical Structure of Continuation Passing Style*. PhD thesis, University of Edinburgh, 1997. (pp 118, 119, 124, 130)

[Thi97b] H. Thielecke. Continuation semantics and self-adjointness. In *Proceedings MFPS XIII*, Electronic Notes in Theoretical Computer Science. Elsevier, 1997. (p 130)

[Wad76] C. P. Wadsworth. The relation between computational and denotational properties for Scott's D_∞-models of the lambda-calculus. *SIAM Journal on Computing*, 5(3):488–521, September 1976. (p 37)