A Semantic analysis of control

James David Laird

Doctor of Philosophy
University of Edinburgh
1998

Abstract

This thesis examines the use of denotational semantics to reason about control flow in sequential, basically functional languages. It extends recent work in game semantics, in which programs are interpreted as strategies for computation by interaction with an environment.

Abramsky has suggested that an intensional hierarchy of computational features such as state, and their fully abstract models, can be captured as violations of the constraints on strategies in the basic functional model. Non-local control flow is shown to fit into this framework as the violation of strong and weak 'bracketing' conditions, related to linear behaviour.

The language μPCF (Parigot's $\lambda\mu$ with constants and recursion) is adopted as a simple basis for higher-type, sequential computation with access to the flow of control. A simple operational semantics for both call-by-name and call-by-value evaluation is described. It is shown that dropping the bracketing condition on games models of PCF yields fully abstract models of μPCF.

The games models of μPCF are instances of a general construction based on a continuations monad on $\mathsf{Fam}(\mathcal{C})$, where $\mathcal{C}$ is a rational cartesian closed category with infinite products. Computational adequacy, definability and full abstraction can then be captured by simple axioms on $\mathcal{C}$.

The fully abstract and universal models of μPCF are shown to have an effective presentation in the category of Berry-Curien sequential algorithms. There is further analysis of observational equivalence, in the form of a context lemma, and a characterization of the unique functor from the (initial) games model, which is an isomorphism on its (fully abstract) quotient. This establishes decidability of observational equivalence for finitary μPCF, contrasting with the undecidability of the analogous relation in pure PCF.

Acknowledgements

First and foremost, I wish to express my gratitude to my supervisor, Samson Abramsky. He has freely given the inspiration, good advice, criticism and encouragement which have enabled me not only to complete this thesis, but to enjoy it. His breadth of knowledge and enthusiasm have been particularly inspiring to me, as a newcomer to computer science.

The financial support for this thesis came from an Engineering and Physical Sciences Research Council studentship. It was typeset using $\LaTeX\,2_\varepsilon$.

The members of the Laboratory for Computer Science at Edinburgh have made it an intellectually stimulating, friendly, and well-supported working environment. Peter Hancock and Conor McBride have nobly tolerated the chaos in our office, but more importantly their company has often provided a welcome diversion from work. Thanks also to Jose Espirito Santo and John Longley for their comments.

Beyond Edinburgh, I am grateful for help, encouragement, and discussion from Pierre-Louis Curien, Hayo Thielecke, Paul Levy, Matthias Felleisen, and the members of the TCOOL project and Russ Harmer. Special thanks are due to Guy McCusker, whose friendship and advice has been very valuable.

The generosity of friends from Oxford and Haslemere has kept me going over some lean times. Most especially my family, — my mother and father and my sisters Becky and Alice have always given me any emotional and material support I have needed.

Declaration

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

(*James David Laird*)

Table of Contents

List of Figures

Chapter 1

Introduction and Background

1.1 Control

This thesis is a study of sequential computation based on the 'flow of control'; essentially the order in which programs are evaluated. This is a pervasive and expressive notion, as complex sequential computations are invariably performed as a series of interdependent processes. Indeed these steps can be so trivial that the *only* substantive information contained in the program is contained in the flow of control.

Data can be represented intensionally using control, a fact which has been known since the definition of the Church numerals. As a minimal example, the element $\mathbf{tt}$ in the domain of booleans, $\mathbf{tt} + \mathbf{ff}$, is represented by passing control to $\mathbf{tt}$, and stopping.

Functional programs A functional program of type $A \Rightarrow B$ can be considered as a way of passing control between the 'argument' A and the 'result' B. Control flow dictates whether this is done in a 'demand driven' or 'data driven' way [73].

Control operators Purely functional programs have a natural order of evaluation; they exhibit only 'local control flow'. Only the most recently scheduled task can be completed and so every subprocess which has been started must be completed (in reverse order) before a result is returned, even if an error has been encountered. This leads to obvious inefficiencies. The solution is to include explicit operators giving programs internal 'access to the flow of control'. These range from the brutal `GOTO`, to more elegant *exception handling* mechanisms.

Proofs of classical logic It was observed by Griffin [32], that Felleisen's con-

> trol operator $\mathcal{C}$ can be typed as a rule of classical logic, and that its computational rules correspond to normalization of classical proofs. Similarly, Parigot's $\lambda\mu$ [69], a term-language for classical proofs, can be used as a calculus of control. That is the context in which $\lambda\mu$ appears in this thesis, but the possibility of using games as a fully complete semantics for constructive classical logic remains a tantalizing prospect (discussed further in Section 6.1).

An important concept used to formalize these ideas is that of *continuation passing*. A continuation is a completion of a program; the evaluation which remains to be done before a result can be returned. Hence it can be represented as a map from the current 'control point' to a conclusion. Computation can be performed by *continuation passing* between parts of the program; in purely functional programs with *local* control flow, this corresponds to performing the next task scheduled by the continuation, and passing it on. However, if continuations are 'first class objects' they can be stored, discarded and substituted, causing jumps in the flow of control. Syntactically, this gives a 'continuation-passing-style' (cps) translation from functional languages with control operators and data, to a much simpler functional language representing only control flow.

The discovery of continuation passing was a seminal event; for the first time control was given its own semantic representation. In fact (as so often), it was not the result of a single inspiration, but a process of enlightenment. In the conclusion of a survey by Reynolds [77]:

> *"... continuations or closely related concepts were first discovered in 1964 by van Wijngaarden, repeatedly discovered in a wide variety of settings — both intellectual and geographical — during 1970-71, and occasionally rediscovered thereafter."*

Control was placed in the context of other computational features by the work of Moggi [62], showing that continuation passing (along with many side effects) could be represented (and reasoned about) as a monad. More recent work by Thielecke [82] and others has further abstracted the notion of continuation passing, giving it an elegant basis in premonoidal categories [74].

The use of first class continuations as 'control points' in functional programs which can be manipulated like ordinary variables can be represented syntactically using the control operator *call-with-current-continuation*, or **call/cc**. This originated in the language Scheme [17], (following other, similar, constructs such as Landin's J-operator [50]). It can be used both for simple 'escapes' from nested computations, and to manipulate the flow of control in more subtle and complex ways; for instance to write coroutines [37]. Subsequent work on formalizing control operators (for instance, by Felleisen and co-workers [25]), has resulted in a range of 'idealized' control constructs. The approach taken here (advocated by Ong and Stewart [67]) is to adopt the $\lambda\mu$-calculus, and a programming language based on it, as a convenient syntax into which other control operators can be translated.

Although control is a useful paradigm for understanding several aspects of computation, it is non-local control flow which requires a specifically control-based analysis. Thus, modelling this behaviour and relating it to the operational semantics of control operators is the main feature of this thesis. However, the intention is to study control within a realistic computational framework so it has been necessary to synthesize the work on the semantics of control calculi and PCF, to show how a few basic notions; cartesian closure, infinite products and recursion, can be the basis of a more structured model of computation. A semantics for a similar language was considered by Sitaram and Felleisen [80], however, it contained parallel features; this thesis is an analysis of feasible, *sequential* control.

1.2 A complete analysis

The investigation is grounded in categorical, syntactic and denotational characterizations of control, via continuation passing models, μPCF, and the bracketing condition of game semantics. None of these can be considered *definitive*, but by finding isomorphisms between syntax and semantics, a coherent analysis emerges, based on *soundness*, and the following *completeness* properties.

Categorical completeness: Purely categorical and syntactic reasoning can be used to show that the representation of functional languages with control operators via continuation passing is complete as well as sound; that the definable part of all models of PCF+**call/cc** arise in this way from a cartesian closed category (so the cps translation is *invertible*). This notion of completeness of models of control was considered by Hofmann, [39] for Felleisen's '$\mathcal{C}$ operator' [25], with subsequent work on $\lambda\mu$ by Hofmann and Streicher [40] and Selinger [79].

Denotational completeness: Completeness results of the above kind can be used to show that one form of syntax (or categorical construction) can be represented by another. A complete semantics of *programming* benefits from a more abstract denotational description of control in order to give a natural interpretation of infinitary features such as recursion. Denotational completeness can also lead to enhanced understanding of the processes at work in computation. But semantics of this kind can still be given formal, categorical characterizations. In particular, there is the possibility of giving an axiomatic description along the lines of Abramsky's 'Axioms for full abstraction' [1].

Full abstraction: This is the natural notion of completeness for models of programming languages, based not on what is representable in the language, but on what is *distinguishable*. The evaluation of programs induces a notion of *observational equivalence* between them.

Definition 1.2.1 *[Full Abstraction [60]] Given a language L with an operational semantics giving a notion of termination $\Downarrow$, an observational equivalence can be defined:*
$M \simeq N$ *if* $(C[M] \Downarrow \iff C[N] \Downarrow)$ *for every program-context* $C[\cdot]$ *which accepts* M *and* N.
A semantics is equationally fully abstract if observable equivalence and denotational equivalence define the same relation, i.e.
$[\![M]\!] = [\![N]\!]$ *if and only if* $M \simeq N$.

To define an operational semantics requires some notion of control behaviour, but observational equivalence 'bootstraps' this to a powerful and precise theory. There are many alternative axiomatizations of control operators such as **call/cc** which are sound with respect to the operational semantics, for instance, but the theory of observational equivalence induced by it is unique. A 'truly denotational' and complete model of this theory should therefore constitute a powerful analysis of control.

It is certainly the case that the search for a fully abstract model for PCF, from which this study borrows much of its motivation, has generated several important insights into the semantics of functional languages, starting from the original observation by Plotkin [71] that full abstraction fails for the Scott continuous functional model of PCF because of the presence of parallel elements. (The problem with the Sitaram and Felleisen model of control [80] is essentially the same, but it also contains sequential as well as parallel 'junk'.) Construction of fully abstract models can be performed solely by syntactic and categorical methods, as shown in the case of PCF by Milner [61]. To move beyond a 'superficial' analysis, therefore, additional conditions can be applied, — that the presentation of the model be denotational, or *syntax-independent*, and (the 'Jung and Stoughton criterion' [44]) that it should also be *effective*. The existence of such a semantics of μPCF is made more significant by the fact that, following a result of Loader [54], there can be no effectively presentable fully abstract model of PCF.

An important precursor to this thesis came from the observation of Cartwright and Felleisen [15] that as well as being important programming features, control operators make the intensional behavour of programs explict. This makes a fully abstract model a realizable aim. They gave a fully abstract semantics to a language called SPCF which extends PCF with control features: errors, and a form of catch operator. Their SPCF semantics was subsequently shown (with Curien) to be equivalent to a presentation using sequential algorithms [16]. Their work is developed here, connecting the sequential algorithms interpreattation of control flow and control operators to the more general notion of continuation-passing, to the less toy-like control operator (call-by-value) **call/cc** and to the local control flow of the games models of PCF.

1.3 Game semantics and the intensional hierarchy

Given the categorical completeness result described above, one could argue that the semantics of control is, in effect, the semantics of cartesian closed categories. Indeed, this proves to be a useful simplification of the proofs of denotational definability and full abstraction. However, this reductionist approach can be compared to that of a naturalist who never leaves the laboratory. He has a profound knowledge of the anatomy and genetics of his chosen species, but no understanding at all of their behaviour and place in the environment. Similarly, a significant part of the analysis of control is contextual, — for instance

- what is local control flow ?
- how does state interact with control ?

There is also a tension between full abstraction and this need for a broader perspective embracing other computational effects: a semantics in which objects are identified up to observational equivalence cannot be naturally extended with more expressive features. (As this would allow observationally equivalent objects to be distinguished.)

The outline of a solution can be found in the 'intensional hierarchy' proposed by Abramsky; a sound, complete and universal semantic characterization of computation. It is grounded in game semantics, in which functional-based computation is modelled by interaction of certain 'typed processes', or *strategies*,

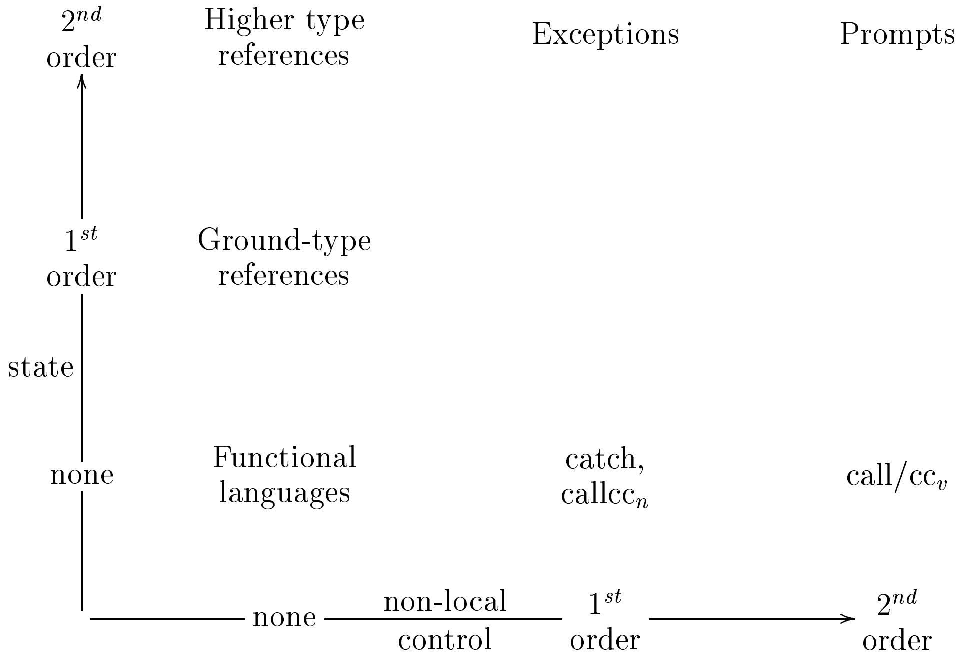

Figure 1.1: The syntactic hierarchy: Expressivity of various programming language features

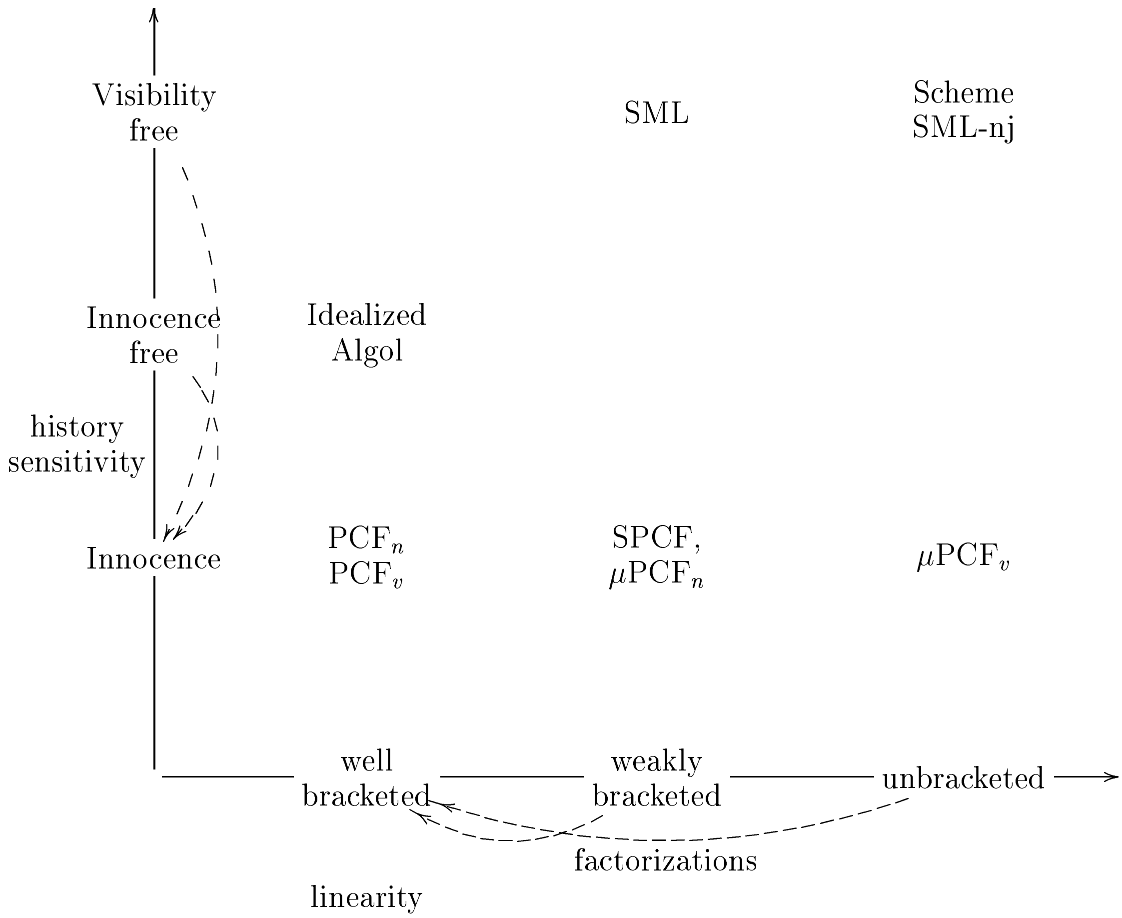

Figure 1.2: The Semantic Hierarchy: Constraints and fully abstract models corresponding to the syntactic hierarchy

for computation. This paradigm can be used to give semantics to a wide variety of computational effects in a coherent way. It can be represented graphically as a partial ordering with syntactic and semantic reflections as in fig. 1.1

vertices correspond semantically to a combination of constraints on the behaviours of strategies, and syntactically, to a 'function (PCF)-based language' which can be given a fully abstract model in the semantic category by a semantic collapse; characterizing this collapse is thus the problem of determining observational equivalence in the language.

In particular, state and control occupy orthogonal 'axes' of the hierarchy, of which state corresponds to restrictions on *sharing* of information — the amount of access of a strategy to the history of the game, and control corresponds to restrictions on *copying* of requests for information. In both cases this basic picture has been refined by showing that there are several 'degrees of freedom' with computational meaning along each axis.

edges are bidirectional:

one direction corresponding to the embedding of the more constrained model in the more general one,

the other corresponding to a 'factorization' of the more liberated strategies. This shows that they can be represented by composition with some characteristic group of unconstrained strategies denoting a new feature such as control operators. It is the key lemma in a full abstraction theorem binding syntax and semantics together, and building on the basic result for PCF.

The intensional hierarchy developed from the models described by Abramsky, Jagadeesan and Malacaria [6], Hyland and Ong [43], and Nickau [65] which appeared, more-or-less simultaneously, as a solution to the full abstraction problem for PCF. The AJM work used ideas from the Geometry of Interaction [30], and the game semantics of Blass [13], whilst the Hyland-Ong approach grew from the work of Lorenzen [57] and subsequent researchers, which used games to study the semantics of logic (Felscher's paper [26] is a useful survey). Subsequent developments include games for recursive types [59], call-by-value [3], and games models of state[2], and higher-type references [4].

A connection between games and control is in any case a natural step; both are offered as paradigms for explaining a variety of computational features, modelled intensionally. More specifically, sequential games are represented as the passing of a token between two participants, this can be thought of as the *system* interacting with the *environment* (or the system interacting with the current continuation).

Such plays are a concrete representation of the history of the control flow in the computation, moreover this is the only information which is carried by plays, confirming the expressive power of control. The games models of PCF showed that it is possible to capture local control by labelling moves as questions and answers, and imposing a 'well-bracketing' constraint. This condition requires that processes must be terminated in the reverse order to which they were initiated. Implicit in the definition of well-bracketing is the notion which is analysed here, that access to the flow of control is gained by dropping this condition. It is a possibility which has also been studied by Herbelin [38], as well as in previous work by the author [48]. Here it is developed, and the relationship with continuation passing studied in detail.

1.4 The contents and contributions of the thesis

Chapter 2 is a more technical introductory chapter, combining material on categorical models of continuation passing, and PCF and its extensions with computational effects. This becomes a basis for a more detailed programme of analysis into the denotational and operational semantics of control, which is summarised in Section 2.6, and pursued in the succeeding three chapters.

Chapter 3 is about game semantics. It describes a novel development of the intensional hierarchy, based on the specification of games by the application of *rules* to arenas. A 'linear decomposition' of this category is described, which allows the construction of a cartesian closed category in a manner consistent with different combinations of rules (allowing, for instance, the modelling of references in the style of [4] in this setting). A generalized axiomatic decomposition which applies to the category of games and *innocent* strategies extends some results of Abramsky by giving both a soundness and a full completeness result for this category as a model of control calculi. The modelling of local and non-local control flow in games via 'bracketing rules' is analyzed. It is shown that the result of dropping this condition on the models of PCF (and imperative extensions) can be represented as a continuation passing semantics. Also, there are strong and weak versions of the bracketing condition, with violation corresponding to 'first-order' use of control operators (escapes and exceptions) and 'second-order' ('upward continuation passing').

Chapter 4 is a study of the operational and denotational semantics of the 'functional language with control', μPCF. A new, simplified, one-step operational semantics of call-by-name and call-by-value μPCF is given, and a general description of computationally adequate models with respect to this semantics is described and verified. Existing continuation passing translations for control *calculi* are extended to μPCF. In particular, this entails giving a new cps-based interpretation of datatypes such as the natural numbers. Because these translations are surjective, the axiomatic definability results for models of the λ-calculus can be applied to the semantics obtained by dropping the bracketing condition on the original games models of PCF. In the case of the call-by-value calculus, a metalanguage based on the computational lambda-calculus is used to show soundness of the model (by showing that the 'deGroote translation' into $\lambda\mathcal{C}$ is sound and complete in this case, *pace* call-by-name).

Chapter 5 is an analysis of the fully abstract model, and its effective presentation in the category of sequential algorithms. The definability result from the previous chapter is used to derive context lemmas for μPCF and these, together with the continuation-passing-style translations already given are shown to have a number of straightforward but significant consequences. For instance, this yields a way to model call-by-value languages using sequential algorithms for the first time. It is also observed that the fact that the fully abstract model is effectively presentable (in contrast to PCF) means that it yields a semantic characterization of the 'minimal model' of the λ-calculus [68], and hence a solution to a part of the higher-order matching problem. Extending full abstraction, *universality* is proved, the first example of a universal model of this kind of language without error terms. The relationship between the two fully abstract models is then investigated. The most significant findings are translations between the two models which capture observational equivalence semantically. One removes repeated queries from innocent strategies, mapping observationally equivalent elements to the unique sequential algorithm denoted by the same lambda terms. The other saturates sequential algorithms with repeated queries, sending each one to a innocent strategy representing a 'normal form' for its observational equivalence class.

Chapter 2

Models of Control

This chapter consists of background on the semantics of functional based languages in general, and continuation passing models of functional control in particular. A semantic basis for the 'intensional hierarchy' is given, embracing call-by-name and call-by-value evaluation, and side-effects via computational monads. Control is, of course, the specific example of a non-functional feature which will be studied semantically. The construction of simple computational 'models of control' is described, together with conditions for full abstraction to hold. In doing so, a series of problems of denotational and operational semantics emerge, and these are distilled to give the outline of an analysis of the semantics of control. Much of the following material is standard, and the proofs are correspondingly sketchy.

2.1 The λ-calculus and its semantics

The simply-typed λ-calculus and its semantics have two distinct rôles in this thesis. One is as the basis of functional programming languages, extended with non-functional features in general, and control operators in particular. Its other rôle (specific to control) is as the target language for continuation-passing-style translations. In other words, the simply typed λ-calculus without values is used as a low-level language into which languages with control can be *compiled*. An important observation showing the suitability of the λ-calculus for this second purpose is that the translations are in several senses complete, — in appropriate circumstances they will be invertible, or at least surjective. There are, therefore, associated isomorphisms between models of these languages. This means that many problems in the semantics of control (such as full abstraction) can be reduced to analogous, but more simply stated problems in the semantics of the λ-calculus. Accordingly, some of these problems and associated notions are reviewed here, although as cartesian closure, the λ-calculus and the connections between them, are now standard, the reader is referred to e.g. Lambek and Scott [49] for background on semantics, and Barendregt [8] for syntax.

The main distinguishing feature of the models of control considered here (as opposed to [39], [79] etc.) is that they include a notion of partiality ab initio, as the intention is to model programming languages in which there is a possibility of non-termination. (Extension to infinitary types and fixpoints can be added later (see Section 2.5) as a natural procedure of chain completion.) Following Braüner [14], and [1], the basic notion used is the $\perp$-map.

Definition 2.1.1 *A category $\mathcal{C}$ has $\perp$-maps if every hom-set $\mathcal{C}(A,B)$ contains a distinguished morphism $\perp_{A,B}$ such that for every $f : C \to A$, $f;\perp_{A,B} = \perp_{C,B}$.*

If $\mathcal{C}$ has a terminal object, then $\perp$-maps can be specified by giving only the morphisms $\perp_A = \perp_{\mathbf{1},A}$ and letting $\perp_{A,B} = t_A;\perp_B$.

In particular, the models of control boil down to a continuation passing construction applied to a cartesian closed category with $\perp$-maps.

Definition 2.1.2 *Define a* pointed cartesian closed category *to be a CCC with $\perp$-maps such that for all objects A, B, $\perp_{A\times B} = \langle \perp_A, \perp_B\rangle$, and $\perp_{A\Rightarrow B} = \Lambda(\perp_{A,B})$.*

$\perp$-maps can be used to define the useful notion of *strictness.*

Definition 2.1.3 *In a category $\mathcal{C}$ with $\perp$-maps, the* strict *maps are morphisms $f : A \to B$ such that $\perp_A; f = \perp_B$. Hence $\mathtt{id}_A$, and $\perp_{A,B}$ are always strict, and the composition of strict maps is strict, so the strict morphisms form a subcategory of $\mathcal{C}$, written $\mathcal{C}_S$ (as in [1]).*

As the pointed CCCs used here will be generated over a set of base type-objects S, $\perp$-maps need be specified only at base type, defining $\perp_{A\Rightarrow B} =$ '$\perp_{A,B}$'. In fact, any CCC gives rise to a pointed CCC of this sort, by adding partiality at 'ground type' by freely adjoining a product of the objects in S to the domain of each morphism in $\mathcal{C}$.

Proposition 2.1.4 *Let $\mathcal{C}$ be a cartesian closed category, with non-terminal objects freely generated from some finite set $S = \{s_1, s_2, \ldots s_n\}$ by $\times, \Rightarrow$. Then there is a pointed cartesian closed category $\mathcal{C}_\perp$, with the same objects as $\mathcal{C}$, and morphisms*
$\mathcal{C}_\perp(A,B) = \mathcal{C}(A \times (s_1 \times \ldots \times s_n), B)$.

PROOF: $\mathcal{C}_\perp$ is the co-Kleisli category of the co-monad $\_ \times (s_1 \times \ldots s_n) : \mathcal{C} \to \mathcal{C}$, which sends $f : A \to B$ to $f \times \mathtt{id}_{(s_1 \times \ldots s_n)}$
with canonical natural transformations $\pi_1^A : A \times (s_1 \times \ldots s_n) \to A$, and
$\mathtt{id}_A \times \langle \mathtt{id}, \mathtt{id} \rangle : A \times (s_1 \times \ldots s_n) \times (s_1 \times \ldots s_n) \to A \times (s_1 \times \ldots s_n)$.
For each base object s_i, there is a $\perp$-map, $\perp_{s_i} = \pi_2; \pi_i$. □

Although partiality will not be modelled in this way in the setting of game semantics (as it proves to be more basic than totality) this will be a useful way of adding distinct 'error elements' to a CCC.

Definition 2.1.5 *For any set S, the free pointed CCC over S is the initial object in the category of pointed CCCs whose objects include those of S, and functors which preserve $\perp$-maps and the universal properties of the terminal object, product and exponentials. As in the case of the free CCC [49], it can be constructed as the category in which objects are freely generated from $\{S\} \cup \{\mathbf{1}\}$, with the constructors $\times$ and $\Rightarrow$, and morphisms are freely generated from $\{\perp_x \mid x \in S\}$, $\{\mathtt{id}_x \mid x \in S\}$, and the terminal morphisms $t : \mathbf{1} \to A$ for each object, by forming products, λ-abstractions and applications, and equated modulo the universal properties for these operations.*

Theorem 2.1.6 *The free pointed CCC over S is well-defined and initial in the category of pointed cartesian closed categories and structure-preserving functors.*

PROOF: is by minor adaptation of the proof given in Lambek and Scott [49] to cover $\perp$-maps. □

The analogous extension of the λ-calculus with constants for non-termination is straightforward.

Definition 2.1.7 *Following Barendregt, [8] define $\Lambda(\Omega)_S$ to be the simply-typed λ-calculus over the base types S, together with constants $\Omega^S : S$ for non-termination at each base type. The equational theory of $\Lambda(\Omega)$ is (the compatible closure of) $\beta\eta$ equality.*

$\Lambda(\Omega)$ has an obvious interpretation in a pointed CCC by fixing the base type-objects, and setting $[\![\Omega^S]\!] = \perp_{[\![S]\!]}$. That pointed CCCs are canonical models of $\Lambda(\Omega)$ can be expressed by observing that the theory of $\Lambda(\Omega)$ is *sound and complete* for pointed CCCs.

Definition 2.1.8 (Sound and complete theory of $\Lambda(\Omega)$) *An equational theory T of $\Lambda(\Omega)$ is:*

- sound *with respect to interpretation in pointed CCCs, if every equation in T is valid in every model, — i.e.*
For all terms s, t of the same type, $s =_T t$ implies that in any pointed CCC, $\mathcal{C}$, $[\![s]\!]_{\mathcal{C}} = [\![t]\!]_{\mathcal{C}}$.

- complete *if every equation which is valid in every pointed CCC is in T, — i.e. if $[\![s]\!]_{\mathcal{C}} = [\![t]\!]_{\mathcal{C}}$ in every model, then $s =_T t$.*

Proposition 2.1.9 *The $\Lambda(\Omega)$-theory $(\beta\eta)$ is sound and complete for pointed CCCs.*

PROOF: is by minor adaptation of the proof of [49]. □

By observing that there is a bijective correspondance between terms-in-context $\Gamma \vdash t$ of $\Lambda(\Omega)$, and terms-in-context of the simply-typed λ-calculus with an additional, non-clashing free-variable, x substituted for Ω in t, it can be shown that:

Proposition 2.1.10 *If $\mathcal{C}$ is (isomorphic to) the free CCC over S, then $\mathcal{C}_\perp$ is isomorphic to the free pointed CCC over S.*

Attention will henceforth be restricted to the λ-calculus with a single base type, ι. Models of this calculus will be given as a pair $(\mathcal{C}, \mathbf{a})$ where $\mathcal{C}$ is a pointed CCC $\mathcal{C}$ and $\mathbf{a}$ is a (non-terminal) object $\mathcal{C}$ interpreting ι.

2.1.1 Full abstraction in CCCs

A second 'canonical' example of a pointed CCC will also prove to be central to the discussion of fully abstract models of control. This is the 'fully abstract' model of $\Lambda(\Omega)$, where the relevant observational equivalence (of Definition 1.2.1) is defined with respect to 'termination' at the function type $\iota \Rightarrow \iota$. (A $\Lambda(\Omega)$ term of this type is said to terminate if it is $\beta\eta$-equivalent to $\lambda x.x$.)

Definition 2.1.11 *Define the following congruence on terms of $\Lambda(\Omega)$:*
$s : T \simeq_T t : T$ if for all contexts $C[\cdot : T] : \iota \Rightarrow \iota$,

$$C[s] =_{\beta\eta} C[t]$$

(And $s \sqsubseteq_T^{OBS} t$ if for all contexts, $C[\cdot : T] : \iota \Rightarrow \iota]$:
$C[s] =_{\beta\eta} \lambda x.x$ implies $C[t] =_{\beta\eta} \lambda x.x$) A model of $\mathcal{M}$ of $\Lambda(\Omega)$ is fully abstract if $[\![s]\!]_{\mathcal{M}} = [\![t]\!]_{\mathcal{M}}$ if and only if $s \simeq t$.

The semantic counterpart to this notion of observational order and equivalence on a pointed cartesian closed category is its *intrinsic preorder*. In a pointed CCC with non-$\perp$ morphisms, the elements in $\mathbf{a} \Rightarrow \mathbf{a}$ which are *definable* (denotations of $\Lambda(\Omega)$-terms) form a 'Sierpinski space' — i.e. a pointed two element set.

Definition 2.1.12 (Intrinsic preorder and equivalence) *Let* $(\mathcal{C}, \mathbf{a})$ *be a model of* $\Lambda(\Omega)$. *Writing* 'f' $: \mathbf{1} \to A \Rightarrow B$ *for the 'name of* $f : A \to B$', *and* $f \downarrow$ *for* $f \neq \perp$, *define the following preorder and equivalence relation on the hom-sets of* $(\mathcal{C}, \mathbf{a})$.

$$f \lesssim_{A \to B} g \Longleftrightarrow \forall h \colon A \Rightarrow B \to (\mathbf{a} \Rightarrow \mathbf{a}) : (\text{'}f\text{'}; h \downarrow) \Rightarrow (\text{'}g\text{'}; h \downarrow).$$

$$f \equiv_{A \to B} g \Longleftrightarrow (f \lesssim_{A \to B} g) \wedge (g \lesssim_{A \to B} f).$$

The intrinsic equivalence and preorder preserves the categorical structure of a model of control.

Proposition 2.1.13 *For any pointed CCC* $\mathcal{C}$, $\equiv$ *is a* congruence *on the hom-sets of* $\mathcal{C}$, — *i.e. the following conditions hold.*

i $f \equiv_{A \to B} f'$ *and* $g \equiv_{B \to C} g'$ *implies* $f; g \equiv_{A \to C} f'; g'$.

ii $f \equiv_{C \to A} f'$ *and* $g \equiv_{C \to B} g'$ *implies* $\langle f, g \rangle \equiv_{C \to A \times B} \langle f', g' \rangle$

iii $f \equiv_{A \times B \to C} f'$ *implies* $\Lambda(f) \equiv_{A \to (B \Rightarrow C)}$

(Similarly, the preorder $\lesssim$ *is a pre-congruence, — a preorder satisfying the above conditions.)*

PROOF: is straightforward (see [6]). $\square$

This 'higher-type' notion of observational equivalence is the strongest non-trivial congruence on a pointed CCC.

Proposition 2.1.14 *If* $\doteqdot$ *is a congruence on terms of* $\Lambda(\Omega)$ *properly extending* $\simeq$, *then it is the trivial equivalence which holds between all terms.*

PROOF: Suppose $\doteqdot$ is a compatible relation on terms properly extending $\simeq$.
Then there are terms $a, b : T$ such that $a \doteqdot b$ but there is a term $f : T \Rightarrow (\iota \Rightarrow \iota)$ such that (w.l.o.g.) $f\ a =_{\beta\eta} \lambda x.x$ and $f\ b =_{\beta\eta} \lambda x.\Omega$.
So $\lambda x.x \doteqdot \lambda x.\Omega$, and given any $t : \overline{S} \Rightarrow \iota$,
$t =_{\beta\eta} \lambda \overline{w}.(\lambda x.x)\ (t\ \overline{w}) \doteqdot \lambda \overline{w}.(\lambda x.\Omega)\ (t\ \overline{w}) =_{\beta\eta} \lambda \overline{w}.\Omega$.
i.e. every term at each type is congruent to Ω. $\square$

A fully abstract model of $\Lambda(\Omega)$ can be derived by the quotient construction from any non-trivial pointed CCC in which every morphism is the denotation of a $\Lambda(\Omega)$-term) by 'collapsing under the intrinsic equivalence'. This is a simplified version of the construction of the fully abstract models of PCF in [6], [43].

Proposition 2.1.15 *Suppose $\mathcal{C}$ is a pointed CCC with a congruence $\equiv$. Then the category $\mathcal{C}/\equiv$ with the objects of $\mathcal{C}$, and the equivalence classes of $\equiv$ as morphisms between them, is a pointed CCC (with $\perp$-maps given by $[\perp_{A,B}]$).*

PROOF: is direct by the fact that $\equiv$ is a congruence. $\square$

Corollary 2.1.16 *Suppose $\mathcal{C}$ is isomorphic to the free pointed CCC over $\mathbf{a}$, then $(\mathcal{C}/\equiv, \mathbf{a})$ is a fully abstract model of $\Lambda(\Omega)$.*

In particular, this means that a syntax-free characterization of the free pointed CCC gives rise automatically to a syntax independent presentation of the fully abstract model of $\Lambda(\Omega)$. However, effectiveness of the presentation is not guaranteed (for a formal presentation of this notion, see [70]). Indeed it is impossible in the analogous case of finitary PCF, as Loader's result shows [54]. As in PCF, the problem boils down to decidability of observational equivalence.

Proposition 2.1.17 *The fully abstract model of $\Lambda(\Omega)$ is effectively presentable if and only if $\simeq$ is decidable.*

PROOF: If the fully abstract model is effectively presentable then $\simeq$ is decidable by comparing denotations of λ-terms. Conversely, suppose $\simeq$ is decidable. The free pointed CCC is certainly effectively presentable as the $\beta\eta$-long normal forms of $\Lambda(\Omega)$. The intrinsic collapse of this model can then be effectively presented by recursively enumerating a series of unique representatives of each equivalence class at each type, using decidability of $\simeq$ to discard equivalent terms. The canonical operations on the CCC, such as application, are clearly also effective by decidability of $\simeq$. $\square$

It will be shown in Chapter 5 that decidability of $\simeq$ is a corollary of decidability of observational equivalence in the λ-calculus with *two* constants (in fact, the problems are equivalent) which has been shown by Padovani [68]. However, the goal of that chapter is a construction of the fully abstract model (and an infinitary extension) which is both syntax-independent *and* effective. These properties can be combined in a purely semantic and 'local' characterization of observational equivalence, through the observation that the unique functor ϕ from the initial to the fully abstract CCC has (by definition) the property that

$$\phi(e_1) = \phi(e_2) \iff e_1 \equiv e_2.$$

Hence a semantic solution of the observational equivalence and effective presentability problems can be found by giving an effective syntax-free characterization of the *functor* between the initial and fully abstract models.

2.2 The $\mathsf{fam}(\mathcal{C})$ construction

In order to interpret datatypes in a programming language, some notion of sums will be required in the semantics. These will be obtained via the action of a strong monad (of continuations) on a category with co-products. The latter are incompatible with pointedness in a CCC. However, it is not necessary to define new categories of 'predomains' concretely, to extend the full completeness and full abstraction results. The process of freely constructing co-products from an arbitrary category (the 'co-product completion') can be described in a syntax-independent way, as a category of indexed families of objects and fibred morphisms of $\mathcal{C}$. Distributivity, and duality of products and co-products, then allows cartesian closure to be extended to the category $\mathsf{fam}(\mathcal{C})$.

The category of indexed families of games was first used to construct a fully abstract semantics of call-by-value languages by Abramsky and McCusker [3]. In this work, it was presented as a general method of obtaining a call-by-value semantics from a call-by-name one, given the basic ingredients of a pointed order enriched cartesian closed category, (from which a bicartesian closed category of 'predomains', $\mathsf{fam}(\mathcal{C})$, can be constructed) and a weak co-product on $\mathcal{C}$ (which yields a strong monad on $\mathsf{fam}(\mathcal{C})$). Their claim, that it is a broadly applicable way to transfer full abstraction results from call-by-name to call-by-value, is supported by a growing body of evidence. It has been applied to well-bracketed games and innocent and history-sensitive strategies in [3], and visibility-free strategies in [4], with fully abstract models of call-by-value PCF (an alternative presentation of Honda and Yoshida's [41]), and function-based languages with references being found as a result.

In this thesis, however, it seems appropriate to take a slightly different perspective. In fact, both call-by-name and call-by-value semantics can be interpreted in $\mathsf{fam}(\mathcal{C})$; the interpretation of sum types in the call-by-name semantics being weak, distributive co-products derived from a strong monad (rather than obtaining a strong monad from a weak co-product). This description seems particularly suited to the semantics of control, in which both call-by-name and call-by-value semantics can be given by continuation-passing-style translations based on a single strong monad on a bicartesian closed category.

Definition 2.2.1 *Given a category $\mathcal{C}$, form the category* $\mathsf{fam}(\mathcal{C})$*; the* free completion of $\mathcal{C}$ with respect to co-products *having*

- *as* Objects*, families of objects of $\mathcal{C}$ indexed over finite sets.*
- *as* Morphisms *from $\{A_i \mid i \in I\}$ to $\{B_j \mid j \in J\}$, a pair consisting of a reindexing function $f : I \to J$ and a family of morphisms $\phi_i : A_i \to B_{fi}$ in $\mathcal{C}$, indexed over the source family:*
  i.e. $\mathsf{fam}(\mathcal{C})(\{A_i \mid i \in I\}, \{B_j \mid j \in J\})$
  $= \{\langle f, \{\phi_i \mid i \in I\}\rangle \mid \phi_i \in \mathcal{C}(A_i, B_{f(i)}), f : I \to J\}$.

Without making any demands of $\mathcal{C}$, it is immediate that $\mathsf{fam}(\mathcal{C})$ has an initial object, the empty family $\{\}$, and more generally:

Proposition 2.2.2 $\mathsf{fam}(\mathcal{C})$ *has all finite co-products.*

PROOF:

$$\{A_i \mid i \in I\} + \{B_j \mid j \in J\} = \{C_k \mid k \in I + J\}$$

where $C_i = A_i$ for $i \in I$, and $C_j = B_j$ for $j \in J$. □

Proposition 2.2.3 *If $\mathcal{C}$ is cartesian closed, then* $\mathsf{fam}(\mathcal{C})$ *is cartesian closed.*

PROOF: **Definition 2.2.4** *Products:*

$$\{A_i \mid i \in I\} \times \{B_j \mid j \in J\} = \{A_i \times B_j \mid \langle i, j\rangle \in I \times J\}.$$

The terminal object is the singleton family $\{\mathbf{1}\}$ containing the terminal object of $\mathcal{C}$.

The operation of exponentiation simply forms a family indexed over functions from I to J, where the object indexed by f is the product of the exponentials $A_i \Rightarrow B_{f_i}$ indexed over $i \in I$.

Definition 2.2.5 *Exponentials:*

$$\{A_i \mid i \in I\} \Rightarrow \{B_j \mid j \in J\} = \{\Pi_{i \in I}(A_i \Rightarrow B_{f(i)}) \mid f \in J^I\}.$$

□

If $\mathcal{C}$ is (continuous) partial-order enriched, then so is $\mathsf{fam}(\mathcal{C})$:
$\langle f, \{\phi_i \mid i \in I\}\rangle \quad \leq \langle g, \{\psi_i \mid i \in I\}\rangle$ if $f = g$ and $\forall i \in I : \phi_i \leq \psi_i$.
Pointedness of the ordering ($\perp$-maps), however, is not preserved.

2.2.1 Strong Monads on $\mathsf{fam}(\mathcal{C})$

Computational monads, as introduced by E. Moggi [62] can now be employed to obtain call-by-name and call-by-value semantics from $\mathsf{fam}(\mathcal{C})$. As this is a well established notion, only a few facts are recalled here; for further details see [64].

Definition 2.2.6 *A strong monad on a category $\mathcal{C}$ with cartesian products and $\perp$-maps is specified by a quadruple $\mathbf{T}, \eta, \mu, t$ (Kleisli triple with tensorial strength t) where:*

$$\begin{tikzcd}
T^3A \arrow[r, "\mu_{TA}"] \arrow[d, "T\mu_A"'] & T^2A \arrow[d, "\mu_A"] \\
T^2A \arrow[r, "\mu_A"'] & TA
\end{tikzcd}
\qquad
\begin{tikzcd}
TA \arrow[r, "\eta_{TA}"] \arrow[dr, "id_{TA}"'] & T^2A \arrow[d, "\mu_A"] & TA \arrow[l, "T\eta_A"'] \arrow[dl, "id_{TA}"] \\
& TA &
\end{tikzcd}$$

Figure 2.1: Defining equations of a monad

- $\mathbf{T}\colon \mathcal{C} \to \mathcal{C}$ *is a functor giving the appropriate structure for the 'notion of computation',*
- $\eta\colon \mathbf{1} \to \mathbf{T}$ *is a natural transformation giving the inclusion of elements of $\mathcal{C}$ as values (i.e. fully computed elements) which are trivial computations,*
- $\mu\colon \mathbf{T}^2 \to \mathbf{T}$ *is a natural transformation which 'computes' the computation of a computation,*
- $t_{A,B}\colon A \times \mathbf{T}B \to (\mathbf{T}A \times \mathbf{T}B)$ *is a tensorial strength. (A natural transformation such that the following diagrams commute):*

$$\begin{tikzcd}
A \times B \arrow[rr, "id_{A\times B}"] \arrow[d, "id_A \times \eta_B"'] & & A \times B \arrow[d, "\eta_{A\times B}"] \\
A \times TB \arrow[rr, "t_{A,B}"] & & T(A \times B) \\
A \times T^2B \arrow[u, "id_A \times \mu_B"] \arrow[r, "t_{A,TB}"'] & T(A \times TB) \arrow[r, "Tt_{A,B}"'] & T^2(A \times B) \arrow[u, "\mu_{A\times B}"']
\end{tikzcd}
\qquad
\begin{tikzcd}
\mathbf{1} \times \mathbf{T}A \arrow[r, "t_{\mathbf{1},A}"] \arrow[dr, "u_{\mathbf{T}A}"'] & \mathbf{T}(\mathbf{1} \times A) \arrow[d, "\mathbf{T}u_A"] \\
& \mathbf{T}A
\end{tikzcd}$$

$$\begin{tikzcd}
(A \times B) \times \mathbf{T}C \arrow[rr, "t_{A\times B,C}"] \arrow[d, "ass_{A,B,\mathbf{T}C}"'] & & \mathbf{T}((A \times B) \times C) \arrow[d, "\mathbf{T}ass_{A,B,C}"] \\
A \times (B \times \mathbf{T}C) \arrow[r, "\mathtt{id}_A \times t_{B,C}"] & A \times \mathbf{T}(B \times C) \arrow[r, "t_{A,B\times C}"] & \mathbf{T}(A \times (B \times C))
\end{tikzcd}$$

The tensorial strength allows a pair of computations to be constructed from the computation of a pair, in *two* ways, which, in the general case in which $\mathbf{T}$ is non-commutative, will be distinct. This corresponds in a sequential language to the choice of which component to evaluate first. Taking the option of evaluating left-first:

Definition 2.2.7 *Let*

$$\psi_{A,B} \colon (\mathbf{T}A \times \mathbf{T}B) \to \mathbf{T}(A \times B) = c_{TA,TB}; t_{TB,A}; T(c_{TB,A}; t_{A,B}); \mu_{A\times B}$$

where $c_{A,B} \colon A \times B \to B \times A$ *is the 'twist' map.*

$$\begin{tikzcd}
TA \times TB \arrow[r, "t_{TA,B}"] \arrow[d, "\psi_{A,B}"'] & T(TA \times B) \arrow[r, "Tc_{TA,B}"] & T(B \times TA) \arrow[d, "Tt_{B,A}"] \\
T(A \times B) & T(B \times A) \arrow[l, "Tc_{B,A}"] & T^2(B \times A) \arrow[l, "\mu_{B\times A}"]
\end{tikzcd}$$

Figure 2.2: Definition of $\psi_{A,B}$

Definition 2.2.8 *A monad* $\mathbf{T} : \mathcal{C} \to \mathcal{C}$ *is pointed (w.r.t. the order enrichment on* $\mathcal{C}$*):*
if for each object there is a $\perp$*-map in* $\mathcal{C}$*,* $\perp_{\mathbf{T}A} : \mathbf{1} \to \mathbf{T}A$*, and* $\perp_{A,\mathbf{T}A} \neq \eta_A$*.*
In $\mathsf{fam}(\mathcal{C})$*, (*$\mathcal{C}$ *pointed),* $\perp$*-maps exist only for singleton families, so if* $\mathbf{T}$ *is a pointed monad on* $\mathsf{fam}(\mathcal{C})$*, the image of* $\mathbf{T}$ *consists of singleton families of objects of* C*, i.e* $\mathbf{T}$ *resolves as an adjunction between a functor from* $\mathsf{fam}(\mathcal{C})$ *to* $\mathcal{C}$*, and the inclusion* $\{\_\} : \mathcal{C} \to \mathsf{Fam}(\mathcal{C})$*.*

Compare this with the notion of a weak co-product on a pointed category $\mathcal{C}$.

Definition 2.2.9 $\mathcal{C}$ *has weak (distributive, pointed) co-products if*
for each finite family $\{A_i \mid i \in I\}$ *of objects of* $\mathcal{C}$*, there is an object* $\Sigma_{i\in I} A_i$ *such that there are:*

- *injections* $\mathtt{in}_i : A_i \to \Sigma_{i\in I} A_i$*, for each* $i \in I$*, and*
- *for any family* $\{f_i : A_i \to B \mid i \in I\}$*, there is a 'co-pairing':*
  A unique strict map $[f_i \mid i \in I] \colon \Sigma_{i\in I} A_i \to B$*, such that*
  $\mathtt{in}_i; [f_i \mid i \in I] = f_i$.
- *A distribution map for each object* B*,*
  $\mathtt{dist}_B : B \times \Sigma_{i\in I} A_i \to \Sigma_{i\in I}(B \times A_i)$ *such that*
  $\langle f, [g_i \mid i \in I]\rangle; \mathtt{dist} = [\langle f, g_i\rangle \mid i \in I]$.

- *a $\perp$-map $\perp_{\Sigma_{i\in I}A_i} : \mathbf{1} \to \Sigma_{i\in I}A_i$, such that $\perp_{A_i,\Sigma_{i\in I}A_i} \neq \mathtt{in}_i$.*

Abramsky and McCusker [3] have observed the following correspondence between weak co-products and strong monads on $\mathsf{fam}(C)$.

Definition 2.2.10 *Suppose $\mathcal{C}$ is a category with a notion of pointed, distributive weak co-products, Σ, satisfying the following additional condition:*
for $f_i : A_i \to B,\quad i \in I$, and $g : B \to C$, $[f_i \mid i \in I]; g = [f_i; g \mid i \in I]$.
Then define the following strong monad:
$\mathbf{T}_\Sigma : \mathsf{fam}(\mathcal{C}) \to \mathsf{fam}(\mathcal{C})$
$\mathbf{T}_\Sigma(\{A_i \mid i \in I\} = \{\Sigma_{i\in I}A_i\}$
$\mathbf{T}_\Sigma\langle f : I \to J, \{\phi_i : A_i \to B_{fi} \mid i \in I\}\rangle = \{[\phi_i; \mathtt{in}_{fi} \mid i \in I]\}$
with natural transformations

- $\eta_{\{A_i \mid i\in I\}} : \{A_i \mid i \in I\} \to \{\Sigma_{i\in I}A_i\} = \{\mathtt{in}_i \mid i \in I\}$
- $\mu_{\{A_i \mid i\in I\}} : \{\Sigma(\Sigma_{i\in I}A_i)\} \to \{\Sigma_{i\in I}A_i\} = \{[\mathtt{id}_{\Sigma_{i\in I}A_i}]\}$
- $t_{\{A_i \mid i\in I\},\{B_j \mid j\in J\}} : \{A_i \times \Sigma_{j\in J}B_j \mid i \in I\} \to \{\Sigma_{\langle i,j\rangle\in I\times J}A_i \times B_j\}$
  $= \{\mathtt{dist}_{A_i} \mid i \in I\}\}$

Proposition 2.2.11 $\mathbf{T}_\Sigma$ *is a strong pointed monad.*

Proof: It is straightforward to show that the families of morphisms so defined are natural transformations, and that they satisfy the equations given above. $\square$

2.2.2 Call-by-value and call-by-name semantics in $\mathsf{fam}(\mathcal{C})$

Moggi ([62], etc.) describes the semantics of the call-by-value computational λ-calculus in the Kleisli category of a strong monad. A point made effectively in this work is that the order of evaluation (or flow of control), which is relevant to all forms of side-effects, can be studied using the computational λ-calculus and its theory, which is complete for these models. As will become apparent in Chapter 4, the case of non-local control itself is special, in that the μ-transformation of the monad is the denotation of a control operator. It will accordingly prove useful to define a monadic metalanguage along the lines of the computational λ-calculus; more detail is contained in [62] [64]. This language is exploited in Chapter 4 by relating it directly to calculi with control operators.
Models of $\lambda_c(\Omega)$ (the computational λ-calculus with constants for non-termination) are given by a strong, pointed monad $\mathbf{T}$ on a cartesian closed category $\mathcal{C}$ which has all exponentials over the image of $\mathbf{T}$ in the following sense.

Definition 2.2.12 *A cartesian category $\mathcal{C}$ has all exponentials of (the $\mathcal{C}$-object) B if there is a contravariant functor $\_ \Rightarrow B : \mathcal{C}^{OP} \to \mathcal{C}$, with an evaluation map $\mathrm{App}_A : A \Rightarrow B \times A \to B$ at each A, with the standard universal property that for every $f : C \times A \to B$, there is a unique map $\Lambda(f) : C \to A \Rightarrow B$ such that $\Lambda(f \times \mathtt{id}_A; \mathrm{App}_A) = f$.*

The types of λ_c are formed from a non-empty set of base types, $\mathcal{B}$ by making the action of the monad in the Kleisli category explicit.

$$A ::= A \in \mathcal{B} \mid \mathbf{T}A \mid A \Rightarrow \mathbf{T}B$$

The core typing system can be extended with product and sum types.
Terms are formed as shown in Figure 2.3.

- The operations of λ-abstraction and application, which are interpreted using the monad strength, and exponentials of the base category. An undefined term denoting $\perp$-elements has also been added.

- A `let` constructor, which is interpreted using composition in the Kleisli category ($\mathtt{let}\ x = s\ \mathtt{in}\ t$ is computed by evaluating s to a value, substituting it for x in t, and evaluating). This is a useful programming feature in its own right, and it also allows the equational theory to be extended beyond $\beta\eta$-equality, to reflect the order of evaluation. Note, however, that uses of `let` can be eliminated by replacing $\mathtt{let}\ x = s\ \mathtt{in}\ t$ with the equivalent $(\lambda x.t)\ s$.

- Operations $[\_]$ and μ, corresponding directly to composition with the natural transformations of lifting (η), and flattening, μ.
  These maps define an isomorphism between terms of type $\mathbf{T}A$, and values of type $\mathbf{T}^2 A$, (a 'monadic reflection' [27]) which in the case of control defines an 'idealized' control operator.

The *call-by-name* interpretation of the simply-typed λ-calculus in the cartesian closed category $\mathsf{fam}(\mathcal{C})$ is completely standard. The problems come with the addition of sum types due to the inconsistency of pointed order enrichment and cartesian closure with true co-products. One solution, adopted in [14], and [59], is to take weak co-products, giving separated sums. (Hence the claim that the $\mathsf{fam}(\mathcal{C})$ construction represents a way to generalise call-by-name semantics to call-by-value.) But one can also use co-products and a strong monad to define a call-by-name semantics of sums.

$$\frac{}{[\![\Gamma,x_n{:}A\vdash x_n{:}\mathbf{T}A]\!]=\pi_n;\eta_{[\![A]\!]}} \qquad \frac{[\![\Gamma\vdash t{:}\mathbf{T}A]\!]=f \quad [\![\Gamma,x{:}A\vdash s{:}\mathbf{T}B]\!]}{[\![\Gamma\vdash \mathtt{let}\ x{=}t\ \mathtt{in}\ s{:}\mathbf{T}B]\!]=\langle \mathtt{id}_{[\![\Gamma]\!]},f\rangle;t_{[\![\Gamma]\!],[\![A]\!]};\mathbf{T}g;\mu_{[\![B]\!]}}$$

$$\frac{}{[\![\Gamma\vdash\Omega{:}\mathbf{T}A]\!]=\bot_{\mathbf{T}[\![A]\!]}}$$

$$\frac{[\![\Gamma,x{:}A\vdash t{:}\mathbf{T}B]\!]=f}{[\![\Gamma\vdash\lambda x.t{:}\mathbf{T}(A\Rightarrow B)]\!]=\Lambda(f);\eta_{A\Rightarrow\mathbf{T}B}} \qquad \frac{[\![\Gamma\vdash t{:}\mathbf{T}A]\!]=f \quad [\![\Gamma\vdash s{:}\mathbf{T}(A\Rightarrow\mathbf{T}B)]\!]=g}{[\![s\ \ t{:}\mathbf{T}B]\!]=\langle g,f\rangle;\psi_{[\![A]\!]\Rightarrow\mathbf{T}[\![B]\!],[\![A]\!]};\mathbf{T}\mathrm{App}_{[\![A]\!],\mathbf{T}[\![B]\!]};\mu_{[\![B]\!]}}$$

$$\frac{[\![\Gamma\vdash t{:}A]\!]=f}{[\![\Gamma\vdash[t]{:}TA]\!]=f;\eta_{[\![A]\!]}} \qquad \frac{[\![\Gamma\vdash t{:}\mathbf{T}^2A]\!]=f}{[\![\Gamma\vdash\mu(t){:}\mathbf{T}A]\!]=f;\mu_{[\![A]\!]}}$$

Figure 2.3: Term formation and semantics for $\lambda_c(\Omega)$

Proposition 2.2.13 *Suppose* $\mathbf{T}$ *is a strong monad on* $\mathsf{Fam}(\mathcal{C})$*, together with a natural transformation* $\varrho : T \to \mathbf{Id}$*, such that* $\eta;\varrho = \mathtt{id}$*, and for any* strict *map* $f : \mathbf{T}A \to B$*, if* $\varrho_A;\eta_A;f = g$*, then* $f = g$*. Then* $\mathbf{T}(A+B)$ *is a weak co-product of* A *and* B *in* $\mathcal{C}$*.*

Proof: Injections into $T(A+B)$ are given by $\mathtt{in_l};\eta_{A+B}$ and $\mathtt{in_r};\eta_{A+B}$, where $\mathtt{in_l}, \mathtt{in_r}$ are the injections into the true co-procuct.
The co-pairing of $f\colon A \to C, g\colon B \to C$ is $\varrho_{A+B};[f,g] : T(A+B) \to C$, so that this is a weak co-product: $(\mathtt{in_r};\eta_{A+B});\varrho_{A+B};[f,g] = \mathtt{in_r};[f,g] = f$, and $\varrho;[f,g]$ is unique as if $\{f,g\} : T(A+B) \to C$ is a strict map such that $\mathtt{in_l};\eta_{A+B};\{f,g\} = f$, and $\mathtt{in_r};\eta_{A+B};\{f,g\} = g$, then $\eta_{A+B};\{f,g\} = [f,g]$. Hence by uniquenes of $[f,g]$, $\varrho_{A+B};\eta_{A+B};\{f,g\} = \varrho_{A+B};[f,g]$ and hence $\{f,g\} = \varrho_{A+B};[f,g]$ as required.
The distributivity transformation is derived from the monadic strength, together with distributivity for the true co-product:
$t_{A,\mathbf{T}(B+C)};\mathtt{Tdist}_{A,B+C} : A \times \mathbf{T}(B+C) \to \mathbf{T}(A \times B + A \times C)$. $\square$

2.3 Models of continuation-passing

The semantics of the specific computational feature of access to the flow of control can now be described, based on the key notion of continuation passing style (cps) semantics. But it is necessary to make a choice. There are several possible semantic frameworks for continuation passing, although (see Section 2.3.2 below) there is a sense in which each is representable in terms of the other. In any case, the more concrete models of simple programming languages considered here can be given in terms of simple cps translations to which the differences between styles of continuation-passing semantics seem less relevant. In fact, an observation made in Chapter four (see Example 4.1.1) is that the *finite* parts of the models of PCF with control are given by a continuation-passing interpretation over a single, empty type. Since definability of this finite fragment is the key to full abstraction, attention will be focussed on this simple and specific notion of model.

Say that an object $\mathbf{a}$ of a cartesian category $\mathcal{C}$ is an *answer object* of $\mathcal{C}$ if $\mathcal{C}$ has all exponentials of $\mathbf{a}$.

Proposition 2.3.1 *For any answer object* $\mathbf{a}$*,*
the contravariant functor $\_ \Rightarrow \mathbf{a} : \mathcal{C}^{OP} \to \mathcal{C}$ *is self-adjoint.*

Cartesian closure of $\mathcal{C}$ is not necessary, although as shown by Hofmann [39] any Kleisli category of a continuations monad can be embedded in one constructed from a cartesian closed category (using the Yoneda lemma). It is also the case that if $\mathcal{C}$ is bicartesian closed, there is a cartesian closed full subcategory of $\mathcal{C}$ with objects freely generated from $\mathbf{a}$ by the operations of $\times, \Rightarrow$.
Write $\mathbf{a}^{\mathcal{C}}$ for the category of exponentials of $\mathbf{a}$, with $\mathcal{C}$-morphisms between them ('the category of $\mathbf{a}$-continuations').

Proposition 2.3.2 *If* $\mathcal{C}$ *has co-products and all exponentials of* $\mathbf{a}$*, then* $\mathbf{a}^{\mathcal{C}}$ *is cartesian closed, and so if* $\mathcal{C}$ *is a cartesian closed category constructed from a single base object* $\mathbf{a}$*, then* $\{\mathbf{a}\}^{\mathsf{fam}(\mathcal{C})} \cong \mathcal{C}$

PROOF: For the first part, define:
$(A \Rightarrow \mathbf{a}) \times (B \Rightarrow \mathbf{a}) = (A + B) \Rightarrow \mathbf{a}$,
$(A \Rightarrow \mathbf{a}) \Rightarrow (B \Rightarrow \mathbf{a}) = (A \Rightarrow \mathbf{a} \times B) \Rightarrow \mathbf{a}$.
The objects of $\{\mathbf{a}\}^{\mathsf{fam}(\mathcal{C})}$ are the singletons $\{((B_1 \Rightarrow \mathbf{a}) \times \ldots \times (B_n \Rightarrow \mathbf{a})\}$, where each B_i is either terminal, or an object of $\mathcal{C}$. $\square$

Hence the call-by-name λ-calculus with control over a single, empty type can be interpreted in a category of continuations. This is a degenerate example of the more general call-by-name cps models described by Hoffman and Streicher [40] (see Remark 4.5.10).

2.3.1 The continuations monad

It was observed by Moggi [63] that continuation-passing is an example of a 'notion of computation' which can be represented as a strong monad, and has since been studied widely using this paradigm, for instance by Filinski, in his thesis [27]. As $\_ \Rightarrow \mathbf{a}$ is a self-adjoint functor, it resolves a monad, which can be described as follows.

Definition 2.3.3 (Continuations Monad) *For a category* $\mathcal{C}$ *and answer object* $\mathbf{a}$ *define the following 'monad of* $\mathbf{a}$*-continuations':*

$\mathbf{T}X = (X \Rightarrow \mathbf{a}) \Rightarrow \mathbf{a}$
$\mathbf{T}(f\colon A \to B) = [\![x\colon (A \Rightarrow \mathbf{a}) \Rightarrow \mathbf{a} \vdash \lambda y\colon B \Rightarrow \mathbf{a}.x\ (\lambda z\colon A.y\ (f(z)))]\!]$
$\eta_A : A \to \mathbf{T}A = [\![x\colon A \vdash \lambda y\colon A \Rightarrow \mathbf{a}.(y\ x)]\!]$
$\mu_A : \mathbf{T}^2 A \to \mathbf{T}A$
$= [\![x\colon ((((A \Rightarrow \mathbf{a}) \Rightarrow \mathbf{a}) \Rightarrow \mathbf{a}) \Rightarrow \mathbf{a}) \vdash \lambda y\colon (A \Rightarrow \mathbf{a}).x\ (\lambda z.((A \Rightarrow \mathbf{a}) \Rightarrow \mathbf{a}).z\ y)]\!]$
and monad strength:
$t_{A,B} : A \times \mathbf{T}B \to \mathbf{T}(A \times B)$
$= [\![x\colon (A \times ((B \Rightarrow \mathbf{a}) \Rightarrow \mathbf{a})) \vdash \lambda y\colon ((A \times B) \Rightarrow \mathbf{a}).(\pi_2(x)\ (\lambda z\colon B.y\ \langle \pi_1(x), z\rangle))]\!]$

If $\mathcal{C}$ has all exponentials over $\mathbf{a}$, it has all exponentials over $(A \Rightarrow \mathbf{a}) \Rightarrow \mathbf{a}$. (Any exponential $B \Rightarrow (A \Rightarrow \mathbf{a}) \Rightarrow \mathbf{a}$ is isomorphic to an exponential $((B \times (A \Rightarrow \mathbf{a})) \Rightarrow \mathbf{a})$ over $\mathbf{a}$.) Hence the Kleisli category of continuations on $\mathcal{C}$ will be a $\lambda_c(\Omega)$-model.

The requirement that the continuations monad is pointed restricts the possible answer objectss in $\mathsf{fam}(\mathcal{C})$ to singletons of non-terminal objects.

Proposition 2.3.4 *For any pointed CCC, $\mathcal{C}$, the monad of O-continuations on $\mathsf{fam}(\mathcal{C})$ is pointed (with the $\perp$-maps of $\mathsf{fam}(\mathcal{C})$) if and only if $O = \{\mathbf{a}\}$ for some non-terminal object $\mathbf{a}$ of $\mathcal{C}$.*

PROOF: If $\{A_i \mid i \in I\}$ is a non-singleton, non-empty family, then $(\{A_i \mid i \in I\} \Rightarrow O) \Rightarrow O$ is a singleton (and hence pointed), if and only if O is a singleton. $\square$

Remark 2.3.5 *The continuations monad (by Proposition 2.2.13 above)* can *be described as a weak co-product on an appropriate category; it is the sum which is definable in the CCC of type-objects freely constructed from* $\mathbf{a}$ *as the weak dual of the product:*

$$A + B = ((A \Rightarrow \mathbf{a}) \times (B \Rightarrow \mathbf{a})) \Rightarrow \mathbf{a}.$$

The categorical models of control which will be used throughout this thesis can now be defined. The interpretation of a functional basis ($\lambda^+(\Omega)$: the simply-typed λ-calculus with sums and constants for non-termination) is described here, the extension with control operators described in Chapter 4.

Definition 2.3.6 (A model of control) *is designated by a pair of a category with $\perp$-maps, $\mathcal{C}$, and an object* $\mathbf{a}$ *such that $\mathcal{C}$ has all exponentials of* $\mathbf{a}$*. The full subcategory of $\mathcal{C}$ consisting of objects freely constructed from* $\mathbf{a}$ *with $\times$ and $\Rightarrow$ yields interpretations of $\lambda^+(\Omega)$ (via the* $\mathsf{fam}(\mathcal{C})$ *construction and the monad of* $\{\mathbf{a}\}$ *continuations) as detailed in Section 2.2.2. The call-by-name and call-by-value interpretations will be respectively called call-by-name and call-by-value models of* *control.*

The denotations of types in these models is therefore as follows:

The call-by-value control model *is the Kleisli category of the* $\mathbf{a}$*-continuations monad on* $\mathsf{fam}(\mathcal{C})$ *giving the following interpretation of the* $\mathbf{0}, \times, \Rightarrow, +$ *types.*

- $[\![\mathbf{0}]\!]_v = \{\}$ *(so* $\mathbf{T}[\![\mathbf{0}]\!]_v \cong \{\mathbf{a}\}$*)*
- $[\![S \Rightarrow T]\!]_v = [\![S]\!]_v \Rightarrow \mathbf{T}[\![T]\!]_v$
- $[\![S \times T]\!]_v = \mathbf{T}[\![S]\!]_v \times \mathbf{T}[\![B]\!]_v$
- $[\![S + T]\!]_v = [\![S]\!]_v + [\![T]\!]_v$

For each finite set X*, a 'ground type'* $\widetilde{X}$ *corresponding to* $\{\mathbf{1}_x \mid x \in X\}$*, can be defined from the empty type.*
$[\![\mathbf{1}]\!]_v = [\![\mathbf{0} \Rightarrow \mathbf{0}]\!]_v$*, and* $\widetilde{X} = \coprod_{x \in X} \mathbf{1}$*.*

The call-by-name control model *is the pointed CCC with weak co-products given by* $\mathbf{T}(A + B)$*, giving the following interpretation of the* $\mathbf{0}, \times, \Rightarrow, +$ *types.*

- $[\![\mathbf{0}]\!]_n = \{\mathbf{a}\}$
- $[\![S \Rightarrow T]\!]_n = [\![S]\!]_n \Rightarrow [\![T]\!]_n$
- $[\![S \times T]\!]_n = [\![S]\!]_n \times [\![T]\!]_n$
- $[\![S + T]\!]_n = \mathbf{T}([\![S]\!]_n + [\![T]\!]_n) \cong (([\![S]\!]_n \Rightarrow \{\mathbf{a}\}) \times ([\![T]\!]_n \Rightarrow \mathbf{a})) \Rightarrow \mathbf{a}$

Other ground types (flat domains) can be constructed from $\mathbf{0}$ *as weak co-products:* $\widetilde{X_*} = \Sigma_{x \in X} \mathbf{1} = (\Pi_{x \in X} \mathbf{a}) \Rightarrow \mathbf{a}$*.*

Proposition 2.3.7 *The full subcategory of type-objects in a model of control specified by* $(\mathcal{C}, \mathbf{a})$ *(call-by-name or call-by-value) is isomorphic to the full subcategory of* $\mathcal{C}$ *with objects freely constructed from* $\mathbf{a}$ *with* $\Rightarrow$*.*

PROOF: This is trivial for the call-by-name model. In the call-by-value model it is necessary to show that for every non-terminal object A in $(\mathcal{C}, \mathbf{a})$ there is a type S such that $\mathbf{T}[\![S]\!] = \{A\}$

- $\{\mathbf{a}\} = \mathbf{T}\{0\} = \mathbf{T}[\![\mathbf{0}]\!]_v$
- suppose $\{A\} = \mathbf{T}[\![S]\!]_v$, and $\{B\} = \mathbf{T}[\![T]\!]_v$.
  Then $[\![(S \Rightarrow \mathbf{0}) + T]\!] = \mathbf{T}(([\![S]\!]_v \Rightarrow \{\mathbf{a}\}) + [\![T]\!]_v)$
  $\cong ((([\![S]\!]_v \Rightarrow \{\mathbf{a}\}) \Rightarrow \{\mathbf{a}\}) \times ([\![T]\!]_v \Rightarrow \{\mathbf{a}\})) \Rightarrow \{\mathbf{a}\} = \{A \Rightarrow B\}$

$\square$

2.3.2 The continuations monad and other control models

A natural question which arises is whether the semantics of control based on a models of continuations is the simplest and most general choice. The semantics of continuation-passing via translation uses a notion, cartesian closure, which is not, on the face of it, specifically relevant to control, as the operation of continuation forming need not be described in terms of exponentiation by an answer object. An attempt to describe the semantics of control at a more fundamental level has recently been developed, based on the notion of premonoidal category. Initial work by Thielecke [82] took a self-adjoint, contravariant 'not-functor' as primitive, rather than any notion of exponentiation, an approach which has been developed by Führmann [28].

Particularly relevant in this context is a recent paper by Selinger [79]. This describes 'control categories', certain premonoidal categories, in which the call-by-name $\lambda\mu$-calculus can be soundly interpreted. Their duals, co-control categories, are (essentially) tensor-not categories, which are models of the call-by-value $\lambda\mu$ calculus. These models have strong completeness properties.

The key to a comparison between the monadic and premonoidal approaches is the 'not-functor' $\_ \Rightarrow \mathbf{a}$ which resolves the continuation monad, and its image, the (premonoidal) category of continuations. A *premonoid* [74] is a binary operation on a category which is functorial in its arguments individually, but not jointly. For instance, in the Kleisli category of a monad on a cartesian category, the cartesian product from the base category is not cartesian, but it is a premonoid $\otimes$. A tensor-not category is a premonoidal category with a not-functor, $\neg$, as a primitive, so that the call-by-value function-space $A \Rightarrow B$ can be represented as $\neg(A \otimes \neg B)$.

Working in the dual setting of categories of continuations and call-by-name cps gives an example of a premonoidal 'disjunction'.

Proposition 2.3.8 *If $\mathbf{a}$ is an answer object for a cartesian category $\mathcal{C}$, then the category of $\mathbf{a}$ continuations on $\mathcal{C}$ is a symmetric premonoidal category, with the premonoid $\oplus$ defined by:* $(A \Rightarrow \mathbf{a}) \oplus (B \Rightarrow \mathbf{a}) = A \times B \Rightarrow \mathbf{a}$

The call-by-name function-space $A \Rightarrow B$ can then be represented as $\neg A \oplus B$. Thus each category of continuations is a control category, and its dual is a tensor-not category (there are further requirements to satisfy, see [82] and [79]). The question posed by Führmann [28], (for tensor not) and Selinger [79], is: which control categories arise in this way? The answer is that they all do.

Theorem 2.3.9 (Structure theorem for control categories [79]) *Every control category is equivalent to a category of continuations $R^{\mathcal{C}}$.*

Hence generality is not gained or lost by choosing the premonoidal presentation of control models. This means that the simply-typed λ-calculus is a completely expressive language for expressing control (notwithstanding its other roles as the basis of functional languages). Given the similarity in motivation behind the introduction of monads and premonoidal categories, as corresponding to a general notion of computation with side effects, could there be other results of this kind?

The deciding factors in the choice of presentation of the semantics of control are therefore philosophical and pragmatic, depending on the way in which control is to be analysed. Continuation-passing creates complex high-level objects from a simple cartesian closed category; for the 'reductive' analysis of denotational models of control which form the major part of the technical contribution of this thesis (for instance the axiomatic definability result in Section 3.7), it is the simplicity of this basis which is important. For understanding behaviour in the model, reductionism is not appropriate, but structuring tools are required. Monads and premonoidal categories are both examples of such tools; of the two, control categories offer perhaps a more direct and elegant presentation of the semantics of control. *In the context* of other computational behaviour, however, understood via the intensional hierarchy of game semantics, the continuations monad is more directly relevant (see, for instance the work of Abramsky and McCusker on structuring games for purely functional languages and state with monads [3], [4]). An interesting possibility is a similarly general investigation of game semantics via premonoidal categories.

2.3.3 Complete languages and models of control

Having fixed on a notion of model, it will also be necessary to find an extension of the λ-calculus with control operators to interpret in it. As in the case of the semantics, there are a number of possible (and broadly equivalent) alternatives. A natural property to demand is that the chosen language also has a theory which is sound and complete for an interpretation in models of control, in the sense of Definition 2.1.8. (A related requirement (see e.g. [79]) is for a λ-calculus extension which is an *internal language* of models of control.) Instead, this requirement can be expressed in terms of syntax, by asking for a relationship with the language $\Lambda(\Omega)$ which is analogous to the semantic relationship between models of control and pointed CCCs.

Definition 2.3.10 *A language $\mathcal{L}$ of typed terms-in-contexts is an* extension *of $\lambda^+(\Omega)$ if the types, terms and contexts of $\lambda^+(\Omega)$ are included in the types, terms and contexts of $\mathcal{L}$, and the operations of application, λ-abstraction, and their associated typing judgements, are also included in $\mathcal{L}$.*

Definition 2.3.11 (A complete syntax of control) *ia a language $\mathcal{L}$, extending the $\lambda^+(\Omega)$-calculus, such that there are translations $(\_)^*$ from $\mathcal{L}$ into $\Lambda(\Omega)$. These are required to be surjective on $\beta\eta$-equivalence classes, and have the property that for any pointed CCC, $\mathcal{C}$, and associated model of control $\mathcal{M}$, interpretation of $\lambda^+(\Omega)$-terms in $\mathcal{M}$ coincides with translation into $\Lambda(\Omega)$, and interpretation in $\mathcal{C}$, i.e. for all t:*

$$[\![t]\!]_{\mathcal{M}} = [\![t^*]\!]_{\mathcal{C}}$$

A principal aim of this thesis is to describe *denotational* models of control with completeness properties, which will form the finite part of a fully abstract model. (It is not straightforward, for instance, to obtain a continuous order-enrichment on the free constructions). The notion of *full and faithful completeness* (originating in the semantics of proofs [5]) captures the property which is sought, — that elements of the model are all denotations of terms of the language which are unique up to equality in its theory.

Definition 2.3.12 *Let $\mathcal{L}$ be a language extending the $\lambda^+(\Omega)$-calculus and $\mathbf{T}$ an equational $\mathcal{L}$-theory. A model of $\mathcal{L}$ is fully complete if for every map between type-objects, $f : [\![A]\!] \to [\![B]\!]$, there is an $\mathcal{L}$-term-in-context $x : A \vdash t_f : B$ such that $f = [\![x : A \vdash t_f : B]\!]$. It is fully and faithfully complete if each t_f is unique up to equality in T.*

Although denotationally fully complete models will be the main objective , categorical completeness has a part to play. As they are constructed by continuation-passing construction, finding such models will boil down to finding a fully complete model of $\Lambda(\Omega)$, and a complete syntax of control.

Proposition 2.3.13 *If $\mathcal{L}$ is a complete syntax of control, and $\mathcal{C}$ is isomorphic to the free CCC over a single object, then the models of control defined from $(\mathcal{C}, \mathbf{a})$ are fully and faithfully complete models of $\mathcal{L}$.*

2.4 Models of programming languages

The simply-typed λ-calculus with control operators is not sufficiently expressive for the analysis of functional control in a programming context, which requires (at least) the natural numbers and recursion. PCF [78], which includes these features, can be considered not only as *the* prototypical functional language, but also as a basis for languages which combine the λ-calculus with non-functional features like state, concurrency or access to the flow of control. This choice also allows the semantic techniques which have been developed in the study of PCF, for modelling recursion and proving full abstraction, to be adopted (with minor adaptations) in the case of control. They are described succinctly here, reverting to the general framework of λ-calculus based languages interpreted via a strong, pointed monad.

To give a semantics of recursion requires a fixpoint operator. The standard solution for PCF [10] is to assume a continuous order-enrichment on the model, so that least upper bounds of all directed sets exist. Definability of *compact* elements is then sufficient to construct a fully abstract model from an algebraic category. In practice, this is a good syntax-independent way to present a model. However, the 'collapsed' categories on which the fully abstract PCF semantics is based may not have continuous order-enrichment (although, as will be shown in Chapter 5, the fully abstract models of PCF with control *can* be cpo-enriched). A minimal solution is to use the implicit order-structure on *definable* elements defined by pointedness and monotonicity of application and λ-abstraction in a CCC. Instead of general chain-completeness, recursion can be based on the existence of least upper bounds of chains of endomorphisms (following AJM [6]).

Definition 2.4.1 *In a category with $\perp$-maps, for each endomorphism $f : A \to A$, define the chain*
$\{f^i : \mathbf{1} \to A \mid i \in \omega\}$ *(such that $f^i \sqsubseteq f^{i+1}$)*
$f^0 = \perp_A$,
$f^{i+1} = f^i ; f$.
A category is rational *with respect to some order-enrichment $\sqsubseteq$ of its hom-sets if all ω-chains of the above form have least upper bounds,*

$$f^\nabla = \bigsqcup_{i \in \omega} f^i$$

and for every $g : A \to B$, $f^\nabla ; g = \bigsqcup_{i\in\omega}(f^i ; g)$.

It is also important to have access to recursively defined datatypes, such as the natural numbers. Infinite (small) sum types can be interpreted in $\mathsf{FAM}(\mathcal{C})$, the category of families of objects of $\mathcal{C}$ indexed over *infinite* sets, defined as for $\mathsf{fam}(\mathcal{C})$.

Proposition 2.4.2 *If $\mathcal{C}$ has all small products, then $\mathsf{FAM}(\mathcal{C})$ is cartesian closed.*

PROOF: is by extension of the definition of exponential for $\mathsf{fam}(\mathcal{C})$ to infinite families and products, recalling that

$$\{A_i \mid i \in I\} \Rightarrow \{B_j \mid j \in J\} = \{\Pi_{i\in I}(A_i \Rightarrow B_{f(i)}) \mid f \in J^I\}.$$

$\square$

In particular, there is a natural numbers object in $\mathsf{FAM}(\mathcal{C})$

Definition 2.4.3 *Let $\mathbb{N} = \{\mathbf{1}_i \mid i \in \omega\}$. This is a natural numbers object with* $\mathtt{zero} = \langle \mathtt{id}_{\mathbf{1}}, 0\rangle$, *and* $\mathtt{successor} = \langle \mathtt{id}_{\mathbf{1}}, \mathtt{succ}\ \rangle$.

The weak ω-indexed co-product $\mathbf{T}\mathbb{N}$ is a flat domain of natural numbers, or 'object of numerals' in the terminology of [43]. By the assumption that $\perp_{\mathbf{T}\mathbb{N}} \neq \mathtt{in}_i$ for any i, it is also a *standard datatype*, — numerals correspond to distinct non-$\perp$ elements.

2.4.1 PCF and its semantics

PCF was introduced as a programming language by Plotkin [71]. It extends the λ-calculus with arithmetical constants, conditionals, and a recursion combinator (to which sum and products can be added).

Definition 2.4.4 (Call-by-name semantics of PCF) Types *are freely generated from the ground type* $\mathtt{nat}$ *by an arrow constructor* $\Rightarrow$, *and interpreted as follows:*
$[\![\mathtt{nat}]\!] = \mathbf{T}\mathbb{N} = \Sigma_{i\in\omega}\mathbf{1}_i$, *— the flat domain of natural numbers or ω-indexed separated sum of empty domains,*
$[\![T_1 \Rightarrow T_2]\!] = [\![T_1]\!] \Rightarrow [\![T_2]\!]$.
The original presentation of PCF *also contained a ground type of booleans, with relevant constants, which is omitted here.*
The interpretation of terms is described in Figure 2.4.4.

Numerals *are interpreted as the relevant injection $\overline{n} : 1 \to \widetilde{\mathbb{N}_*} = \mathtt{in}_n$ into the object of numerals.*

Conditionals *at type T are interpreted by the composition (in context) of the tested term, which is an injection into $\Sigma_{i\in\omega}\mathbf{1}_i$, with the co-pairing of the 'zero case', (as a map into $[\![T]\!]$) with (inifinitely many instances of) the non-zero case.*

The Y combinator *can be interpreted in a rational category as the least fixed-point of $F_A\colon ((A \Rightarrow A) \Rightarrow A) \to ((A \Rightarrow A) \Rightarrow A)$*
$= \Lambda((\mathtt{id}_{(A\Rightarrow A)\Rightarrow A} \ \times id_{A\to A}); \mathrm{App}_{(A\Rightarrow A)\Rightarrow A, A\Rightarrow A} \times \mathtt{id}_{A\Rightarrow A}; \mathtt{con}_{\mathtt{A\Rightarrow A}})$
So for any endomorphism, $f\colon A \to A$, $f; [\![\mathbf{Y}]\!] = f^{\nabla}$
For any type T, there is an 'undefined term' of type T, $\Omega^T = \mathbf{Y}(\lambda x\colon T.x)$.

Note also that a series of case-splitting constructs can be defined along the lines of IF0 with the intended meaning that $\mathtt{case}_k t|_{i\leq k} s_i$ evaluates to s_i if t evaluates to i. These have proved useful for definability results, but full abstraction is not affected, as they can be *simulated* up to observable equivalence using IF0 as follows:

$$\mathtt{case}_0\ M \mid N_0 \mid N' = \text{IF0}\ M \ \text{ then } N_0 \text{ else } N',$$

$$\mathtt{case}_{n+1} M \mid N_0 \mid \ \ldots \ \mid N_n \mid N_{n+1} \mid N' =$$
$$\mathtt{case}_n M \mid N_0 \mid \ \ldots \ \mid N_n \mid \text{IF0}\ (\mathtt{pred}^{n+1}\ M) \ \text{ then } \ N_{n+1} \mid N'.$$

The definability results for call-by-name PCF were given with respect to PCF$_c$, which extends PCF with these constructs, but the related full abstraction results hold for PCF itself because of this simulation.

$$[\![\Gamma, x\colon T \vdash x\colon T]\!] = \pi_r\colon [\![\Gamma]\!] \times [\![T]\!] \to [\![T]\!]$$

$$[\![\vdash 0\colon \mathtt{nat}]\!] = \mathtt{in}_0 : \mathbf{1} \to \mathbf{T}\mathbb{N}$$

$$[\![\Gamma \vdash \mathtt{succ} \ \ M\colon \mathtt{nat}]\!] = [\![\Gamma \vdash M\colon \mathtt{nat}]\!]; plus1$$

$$[\![\Gamma \vdash \mathtt{pred} \ \ M\colon \mathtt{nat}]\!] = [\![\Gamma \vdash M\colon \mathtt{nat}]\!]; minus1$$

$$[\![\Gamma \vdash \lambda x.M\colon T \Rightarrow U]\!] = \Lambda([\![\Gamma, x\colon T \vdash M\colon U]\!])$$

$$[\![\Gamma \vdash M \ \ N\colon U]\!] = \langle [\![\Gamma \vdash N\colon T]\!], [\![\Gamma \vdash M\colon T \Rightarrow U]\!]); \mathrm{App}$$

$$[\![\Gamma \vdash (\text{IF0} \ \ L \text{ then } M \text{ else } N)\colon \mathtt{nat}]\!] = ([\![\Gamma \vdash L\colon \mathtt{nat}]\!]; [[\![\Gamma \vdash M\colon \mathtt{nat}]\!], \overline{[\![\Gamma \vdash N\colon \mathtt{nat}]\!]}]$$

$$[\![\Gamma \vdash \mathbf{Y}^T M\colon T]\!] = [\![\Gamma \vdash M\colon (T \Rightarrow T)]\!]; F^{\nabla}$$

Figure 2.4: Terms of PCF in contexts, with a call-by-name interpretation

Definition 2.4.5 (Call-by-value semantics of PCF**)** *The interpretation of call-by-value* PCF *in* $\mathsf{Fam}(\mathcal{C})_{\mathsf{T}}$ *is similar to that described above, with the following modifications.*

- *λ-abstraction and application are interpreted in the Kleisli category of a strong monad on* $\mathsf{Fam}(\mathcal{C})$ *(as described in Section 2.2.1).*

- *The type* `nat` *denotes the ω-indexed co-product $\{\mathbf{1}_i \mid i \in \omega\}$, with the same interpretation of numerals as injections, and conditionals as co-pairing.*

- *Recursion behaves quite differently in call-by-value to call-by-name. As every endomorphism $f : A \to A$ in the Kleisli category is strict,all least fixpoints are trivial, so one cannot add recursion by taking $f^{\nabla} = \bigsqcup_{n\in\omega} f^n$. The solution generally adopted is to take 'η-expansions' at each iteration, — based on the following conversion for the $\mathbf{Y}$ combinator:*

$$\mathbf{Y}M \longrightarrow M\ (\lambda x.((\mathbf{Y}M)\ x)).$$

So $\mathbf{Y}M$ is approximated by $\Omega, M(\lambda x.\Omega x), M\ (\lambda y.(M(\lambda x.\Omega x)y)), \ldots$
Provided $\mathcal{C}$ is rational, these fixpoints can be modelled in the Kleisli category.

Definition 2.4.6 *For objects A, B in the Kleisli category, define*
$\chi_{A,B} \colon \mathbf{T}(A \Rightarrow \mathbf{T}B) \to \mathbf{T}(A \Rightarrow \mathbf{T}B) =$
$\Lambda(t_{A,A\Rightarrow\mathbf{T}B}; \mathbf{T}(\mathrm{App}_{A,A\Rightarrow\mathbf{T}B}); \mu_B); \eta_{A\Rightarrow\mathbf{T}B}$.
(So for any $t : \mathbf{T}_A \Rightarrow \mathbf{T}_B$, $[\![\lambda x.(tx)]\!] = \chi_{A,B}; [\![t]\!]$.)
Now define a fixpoint operator $(\_)^{\Delta}$ for the Kleisli category:
If $f \colon \mathbf{T}((A \Rightarrow \mathbf{T}B) \to \mathbf{T}(A \Rightarrow \mathbf{T}B))$

$$f^{\Delta} = (\chi_{A\to\mathbf{T}B}; f; \mu_{A\to\mathbf{T}B})^{\nabla} = \bigsqcup_{n\in\omega} (\chi_{A\to\mathbf{T}B}; t; \mu_{A\to\mathbf{T}B})^n.$$

Hence the call-by-value $\mathbf{Y}$-combinator can be interpreted
$[\![\Gamma \vdash \mathbf{Y}M]\!]_v = [\![\Gamma \vdash M]\!]_v; F^{\Delta}$.

An operational semantics for PCF was defined in [71]. By soundness, rationality, and standardness of datatypes, the crucial property of *computational adequacy* (a program denotes a value if and only if it reduces to that value) of the denotational model with respect to this semantics can be established. (See Section 4.4 and [14] for more details.)

2.4.2 Fully abstract models of PCF-based languages

The notion of observational equivalence defined in Chapter 1 (Definition1.2.1) can be generalised to an observational preorder on terms.

Definition 2.4.7 *For a language L extending* PCF *(in the sense of Definition 2.3.10), define the observational preorder between terms in (the same) context, $\Gamma \vdash M$ and $\Gamma \vdash N$*
$M \sqsubseteq^{OBS} N$ if for every compatible context $C[\cdot] : \mathtt{nat}$, $C[M] \Downarrow$ implies $C[N] \Downarrow$.
Full abstraction for partial-order-enriched models is equivalent to demanding that $M \sqsubseteq^{OBS} N$ if and only if $[\![M]\!] \sqsubseteq [\![N]\!]$.

The key properties for full abstraction for PCF and extensions are computational adequacy and finite definability, although these can be combined in a variety of ways.

Proposition 2.4.8 *Let $\mathcal{M}$ be a cpo-enriched denotational semantics of a language L (extending* PCF*) which is computationally adequate with respect to the operational semantics of L. Suppose that every morphism between type-objects of $\mathcal{M}$ is the least upper bound of a chain of L-definable elements, then for any closed terms (values in the case of call-by-value) $M, N : T$,*

$$M \sqsubseteq^{OBS} N \Longleftrightarrow \forall h \text{ (definable)}: A \to \mathbf{T}(\mathbb{N}) : \; f; h \downarrow \Longrightarrow g; h \downarrow .$$

PROOF: Suppose $M \not\sqsubseteq^{OBS} N$, then there is some L-context $C[\cdot : T] : \mathtt{nat}$ such that $C[M] \Downarrow$ and $C[N] \not\Downarrow$, and hence by adequacy, $[\![C[M]]\!] \downarrow$, and $[\![C[N]]\!] \uparrow$, hence $[\![M]\!]; [\![\lambda x.C[x]]\!] \downarrow$ and $[\![N]\!]; [\![\lambda x.C[x]]\!] \uparrow$ as required.

In the other direction, suppose there is some h such that $[\![M]\!]; h \downarrow$ and $[\![N]\!]; h = \bot$, then $h = \bigsqcup_{i \in \omega} h_i$ for some chain of definable h_i, so $\bigsqcup_{i \in \omega} [\![M]\!]; h_i \downarrow$, so $[\![M]\!]; h_i \downarrow$ for some i such that $[\![N]\!]; h_i \uparrow$.
By the assumption of finite definability, $h_i = [\![L]\!]$ for some term $L : T \Rightarrow \mathtt{nat}$ such that $[\![L\ M]\!] \downarrow$ and $[\![L\ N]\!] \uparrow$, so by computational adequacy $L\ M \Downarrow$, and $L\ N \not\Downarrow$, so $M \not\sqsubseteq^{OBS} N$. □

(This is shown to extend to terms with free variables in Lemma 4.6.8.)

Corollary 2.4.9 *Suppose $\mathcal{M}$ is an adequate, cpo-enriched model of L such that all elements are suprema of chains of definable elements,*
and *for all elements $f, g : A$, $\forall h : A \to [\![\mathtt{nat}]\!], f; h \neq \bot \implies g; h \neq \bot$ implies $f \sqsubseteq g$, then $\mathcal{M}$ is fully abstract.*

Alternatively, the 'intrinsic preorder collapse' of a cpo-enriched model satisfying definability will be fully abstract.

Definition 2.4.10 *Let $\mathcal{M}$ be a model of L (a* PCF *extension) in a (cartesian closed) category $\mathcal{C}$, and define the intrinsic preorder*

$$f \lesssim_{A \to B} g \iff \forall h : A \Rightarrow B \to \mathbf{T}\mathbb{N}\ \text{'}f\text{'}; h \downarrow \implies (\text{'}g\text{'}; h \downarrow),$$

and in the call-by-value model

$$f \lesssim_{A} g \iff \forall h : A \to \mathbf{T}\mathbb{N}\ \text{'}f; \eta_B\text{'}; h \downarrow \implies \text{'}g; \eta_B\text{'}; h \downarrow).$$

Define the collapse of $\mathcal{M}/\lesssim$ as in Definition 2.1.15, — type-objects are as in $\mathcal{M}$, morphisms are equivalence classes of $\mathcal{M}$-morphisms partially-ordered by $\lesssim$.

Proposition 2.4.11 *Supppose $\mathcal{M}$ is a cpo-enriched model of L such that every element is a supremum of a chain of definable elements, and that its collapse $\mathcal{M}/\lesssim$ is an adequate model of L. Then $\mathcal{M}/\lesssim$ is fully abstract.*

PROOF: By Proposition 2.4.8, for all terms $M, N : T$, $M \sqsubseteq_T N$ if and only if for all definable $h : [\![T]\!] \to \mathbf{T}\mathbb{N}$ in $\mathcal{M}$, $[\![M]\!]_{\mathcal{M}}; h \downarrow$ implies $[\![N]\!]_{\mathcal{M}}; h \downarrow$. As every element of $\mathcal{M}$, is approximated by definable elements, $M \sqsubseteq_T N$ if and only $[\![M]\!]_{\mathcal{M}} \lesssim_{[\![T]\!]} [\![N]\!]_{\mathcal{M}}$, as required. □

Collapse under the intrinsic preorder preserves the key features of PCF adequacy at least, — cartesian closure, as in Proposition 2.1.13, and rationality.

Proposition 2.4.12 *The collapse of a cpo-enriched CCC under its intrinsic preorder is rational.*

PROOF: Given $[f] : A \to A$, define $[f]^\nabla = [f^\nabla]$, — to show that this is consistent and rational it is necessary to show that for any $g : A \to B$, $f^\nabla; g$ is a least upper bound (with respect to $\lesssim$) for $\{f^i; g \mid i \in \omega\}$. (and if $g \equiv f$, then $f^\nabla \equiv g^\nabla$). So it suffices to show that if $f^i; g \lesssim h$ for all $i \in \omega$, then $f^\nabla; g \lesssim h$.
Suppose $f^\nabla; g \not\lesssim_B h$, then there is some $k : B \to \texttt{Tnat}$ such that $h; k = \perp$ $(\bigsqcup_{i \in \omega} f^i; g); k = \bigsqcup_{i \in \omega} f^i; g; k \neq \perp$, hence for some i, $f^i; g; k \neq \perp$, so $f^i; g \not\lesssim_B h$ as required. □

2.5 Models of PCF with control

The language μPCF will be obtained by *extending* PCF with certain operations from the control calculus $\lambda\mu$. Proceeding from the basis of finitary models of control on the other hand, it is relatively straightforward to describe the passage to 'computational models' of control with recursion as a process of completion which is in some respects simpler than the general notion of PCF-model.

Definition 2.5.1 *Computational models of control (call-by-name and call-by-value) are specified by a pair $(\mathcal{C}, \mathbf{a})$ of a* rational *pointed CCC with ω-indexed products and* non-terminal *answer object $\mathbf{a}$.*

Definition 2.5.2 *A category $\mathcal{C}$ has ω-indexed products, if for every object A, there is an object A^ω, with projection maps*
$head : A^\omega \to A$ and $tail : A^\omega \to A^\omega$ with the following universal property:
for every pair of maps $f : B \to A, g : B \to A^\omega$, there is a unique map
$f{::}g : B \to A^\omega$ such that $f{::}g; head = f$ and $f{::}g; tail = g$.

(Informally), A^ω is the least solution to $A = A \times A^\omega$.

Definition 2.5.3 *Let $A^0 = \mathbf{1}$, $A^{i+1} = A \times A^i$. Then a series of embeddings and projections $e_i : A^i \to A^\omega$ and $p_i : A^\omega \to A^i$ can be defined such that $e_i; p_i = \mathtt{id}_{A_i}$, and $p_i; e_i \sqsubseteq \mathtt{id}_{A^\omega}$, viz:*
$e_0 = \perp_{A^\omega}, \quad e_{i+1} = \pi_l{::}e_i$
$p_0 = \mathbf{1}_{A^\omega}, \quad p_{i+1} = \langle head, tail; p_i \rangle$

These embeddings and projections can be used to define embeddings and projections between type-objects constructed from finite products, and from ω-indexed products.

Definition 2.5.4 *Define the $\Lambda(\Omega)^\omega$-types as follows: $T ::= \iota \mid T^\omega \mid S \Rightarrow T$ and interpret them in a pointed CCC $\mathcal{C}$ with ω-indexed products and answer object $\mathbf{a}$:*
$[\![\iota]\!]_{(\mathcal{C},\mathbf{a})} = \mathbf{a}$, $[\![T^\omega]\!]_{(\mathcal{C},\mathbf{a})} = [\![T]\!]^\omega_{(\mathcal{C},\mathbf{a})}$, $[\![S \Rightarrow T]\!]_{(\mathcal{C},\mathbf{a})} = [\![S]\!]_{(\mathcal{C},\mathbf{a})} \Rightarrow [\![T]\!]_{(\mathcal{C},\mathbf{a})}$.
For each $\Lambda(\Omega)^\omega$-type T, form T_i by 'truncating all ω-indexed products at i'
i.e. $\iota_i = \mathbf{a}$, $(T^\omega)_i = (T_i)^i$, $(T \Rightarrow S)_i = T_i \Rightarrow S_i$,
Then there is an embedding/projection pair $[\![T]\!]_{(\mathcal{C},\mathbf{a})} \to_{p_i^T} [\![T_i]\!]_{(\mathcal{C},\mathbf{a})} \to_{e_i^T} [\![T]\!]_{(\mathcal{C},\mathbf{a})}$

- $e_i^{S \Rightarrow T} = \Lambda(\mathtt{id}_{[\![S_i \Rightarrow T_i} \times p_i^S; \mathrm{App}; e_i^T)$,
  $p_i^{S \Rightarrow T} = \Lambda(\mathtt{id}_{S \Rightarrow T} \times e_i^S; \mathrm{App}; p_i^T)$,

- $e_i^{S^\omega} = \Pi_{j\leq i} e_i^S; e_i$,
  $p_i^{S^\omega} = p_i; \Pi_{j\leq i} e_i^S$.

Proposition 2.5.5 *For every $\Lambda(\Omega)^\omega$-type T, and $i \in \omega$, $e_i^T; p_i^T = \mathtt{id}_{T_i}$.*

Proof: is by induction on the structure of T, using the definition of e_i, p_i and the universal properties of a CCC. □

Although all set-indexed products are required for cartesian closure of $\mathsf{FAM}(\mathcal{C})$ (and the specific models considered will have such products), the semantics will require only countable co-products. So attention will be restricted to $\mathsf{Fam}(\mathcal{C})$, — the category of *countably indexed* families of objects of $\mathcal{C}$. $\mathsf{Fam}(\mathcal{C})$ is clearly not cartesian closed. However, the observation that a continuations model requires only limited exponentials can be used here, as provided $\mathcal{C}$ has ω-indexed products, $\mathsf{Fam}(\mathcal{C})$ has all exponentials of $\{\mathbf{a}\}$.

Proposition 2.5.6 *If $\mathcal{C}$ has ω-indexed products, then the monad of $\mathbf{a}$-continuations on $\mathcal{C}$ is well-defined, and for any object A, $\mathcal{C}$ has all exponentials of $\mathbf{T}A$.*

Note that it is still not necessary to introduce ground type values into control models, — data is modelled intensionally. For instance, in the domain corresponding to $\mathtt{nat}$, $\mathbf{T}\mathbb{N} = (\mathbb{N} \Rightarrow \{\mathbf{a}\}) \Rightarrow \{\mathbf{a}\} \cong \{\mathbf{a}^\omega \Rightarrow \mathbf{a}\}$, each numeral corresponds to a projection from $\mathbf{a}^\omega$, 'the countable list of $\mathbf{a}$s'. This is the basis for fully abstract translations of PCF with control into the simply-typed λ-calculus with fixpoint combinators and infinite lists. This is described, with the call-by-name translation, in Section 4.5.

Because the initial model of control is constructed by a process of completion from a simple cartesian closed category, there is a direct relationship between the CCCs with the associated properties, — the fully complete and fully abstract models of $\Lambda(\Omega)$ described in Section 2.1, — and the fully complete and fully abstract models of control.

In order to give a full abstraction result based on finitary definability, a continuous order-enrichment is assumed (although rationality, together with continuity of composition is all that is used).

Definition 2.5.7 *A continuous model of control is specified by a pointed cpo-enriched CCC, with ω-indexed products such that for every embedding/projection pair $e_i : A^i \to A^\omega, p_i : A^\omega \to A^i$,*
$\bigsqcup p_i; e_i = \mathtt{id}_A$.

Lemma 2.5.8 *In a continuous control model, the above property extends to all of the defined embeddings and projections* $e_i^T : [\![T_i]\!] \to [\![T^\omega]\!], p_i^T : [\![T^\omega]\!] \to [\![T_i]\!]$, *i.e.* $\bigsqcup_{i\in\omega} p_i; e_i = \mathtt{id}_{[\![T]\!]}$.

PROOF: is by continuity of composition, and the universal properties of pairing and λ-abstraction. □

Say that a morphism of a computational control model (between $\Lambda(\Omega)^\omega$ type-objects) is *definable* if it can be defined via λ-abstraction, application, the operations ($head, tail, ::$) of the countable product, and taking least fixpoints, $(\_)^\nabla$. The following proposition shows, in essence, that the definable part of a continuous model of control is ω-algebraic, with a finite basis of finitary definable elements embedded from the definable elements of a model of control.

Proposition 2.5.9 *If* $(\mathcal{C}, \mathbf{a})$ *specifies a continuous computational model of control, then every definable morphism is the least upper bound of a chain of morphisms definable in the model of control specified by* $(\mathcal{C}, \mathbf{a})$,
i.e. if $f : [\![S]\!] \to [\![T]\!]$ *is definable, there is a series of morphims* $f_i : [\![S_i]\!] \to [\![T_i]\!]$ *such that* $\{\ p_i^S; f_i; e_i^T : [\![S]\!] \to [\![T]\!] \mid i \in \omega\}$ *is an* ω*-chain, and*

$$f = \bigsqcup_{i\in\omega}(p_i^S; f_i; e_i^T).$$

PROOF: is by induction on the possible definitions of morphisms in the computational model of control. By the Lemma 2.5.8 above, $\mathtt{id}_{[\![S]\!]} = \bigsqcup_{i\in\omega} p_i^S; \mathtt{id}_{S_i}; e_i^S$
If $f = \Lambda(f')$ for some f', then the hypothesis extends to f by the universal property.
Suppose $f = \langle g, h\rangle; \mathrm{App}$, then by hypothesis, there are chains of finitary elements such that $g = \bigsqcup_{i\in\omega} p_i^S; g_i; e_i^{T\Rightarrow R}$ and $h = \bigsqcup_{i\in\omega} p_i^S; h_i; e_i^T$.
Thus $g = \bigsqcup_{i\in\omega} p_i^S; g_i \times p_i^T; \mathrm{App}; e_i^R$, and so by continuity of composition,
$h = \langle id_{[\![S]\!]}, g\rangle; f \times \mathtt{id}_{[\![S]\!]}; \mathrm{App}$
$= \bigsqcup_{i\in\omega}(\langle \mathtt{id}_{[\![S]\!]}, p_i^S; g_i; e_i\rangle; p_i^S; f \times p_i^T; \mathrm{App}; e_i^R$
$= \bigsqcup_{i\in\omega}(p_i^S\langle \mathtt{id}_{[\![S_i]\!]}, g_i\rangle; f \times \mathtt{id}_{[\![T_i]\!]}; \mathrm{App}; e_i^R$ as required.
If $f = g^\nabla$ for some $g : A \to A$, then by hypothesis, $g = \bigsqcup p_i^S; g_i; e_i^S$, for some series of morphisms $\{g_i : [\![S_i]\!] \to [\![S_i]\!] \mid i \in \omega\}$.
Let $f_i = g_i^i$, then $f_i; e_i = g_i^i; e_i \sqsubseteq g_{i+1}^{i+1}; e_{i+1} = f_{i+1}; e_{i+1}$. □

Recall that a complete syntax of control is an extension of the λ-calculus (including sums) with control operators, such that there are surjective translations into the simply-typed λ-calculus.

Definition 2.5.10 *A complete* computational *syntax of control is an extension of* PCF *(with the empty type), such that there is a translation* $(\_)^*$ *into the simply-typed* λ*-calculus with lists,* $\Lambda(\Omega)^\omega$ *(see Section 4.5) which is surjective on* $\Lambda(\Omega)^\omega$*-equivalence classes at translated types* T^**, and such that for all* PCF *terms* M*, interpreted in a PCF model* $\mathcal{M}$*, constructed from a computational model of control* $(\mathcal{C}, \mathbf{a})$*:*

$$[\![M]\!]_{\mathcal{M}} = [\![M^*]\!]_{\Lambda(\Omega)^\omega}$$

Corollary 2.5.11 *If* $\mathcal{M}$*, specified by* $(\mathcal{C}, \mathbf{a})$ *is a continuous computational model of control, and* L *is a complete computational syntax of control then every morphism between type-objects of* $\mathcal{M}$ *is a least upper bound of a chain of* L*-definable elements, in the form* $\bigsqcup_{i \in \omega}(p_i^A; f_i; e_i^B)$*.*

Similarly, the intrinsic preorder on the finitary part of a computational control model characterizes the preorder on the whole model.

Proposition 2.5.12 *Suppose* $(\mathcal{C}, \mathbf{a})$ *specifies continuous models of control via the monad of* $\mathbf{a}$*-continuations on* $\mathsf{Fam}(\mathcal{C})$*, such that every element is a least upper bound of definable elements. Then the intrinsic preorder on the call-by-name and call-by-value* PCF *models in* $\mathsf{Fam}(\mathcal{C})$ *given by Definition 2.4.10 is equivalent to the preorder induced by by taking observations on* $\mathbf{a} \Rightarrow \mathbf{a}$ *in the CCC generated from* $\mathbf{a}$*. i.e. if* $f : A \to B = \bigsqcup_{i \in \omega} p_i^A; f_i; e_i^B$*, and* $g = \bigsqcup_{i \in \omega} p_i^A; g_i; e_i^B$*, then* $f \lesssim_{A \to B} g$ *if and only if for every* $i \in \omega$*, there exists* $j > i$ *such that* $f_j \lesssim_{A_i \to B_i} g_j$*.*

Proof: is by Proposition 2.5.9 together with continuity. □

Together with Proposition 2.1.14, this yields the following corollary.

Corollary 2.5.13 *The observational equivalence* $\equiv$ *is the strongest non-trivial congruence on computational models of control.*

Note that this is not an extensional equivalence on models of control (pace, for instance PCF). This is simply the semantic counterpart of the non-extensionality of observational equivalence in the presence of control operators (for further discussion see Chapter 5).

Thus if $(\mathcal{C}, \mathbf{a})$ defines a computational model of control, $(\mathcal{C}/\lesssim, \mathbf{a})$ defines a model of control such that $[\![M]\!]_{\mathcal{C}/\lesssim} = [[\![M]\!]_{\mathcal{C}}]$. So there are two routes to a syntax-free presentation of the fully abstract model of control; take the quotient of a continuous model of control for which the restriction to finite types is isomorphic to the free CCC, or more directly, give a continuous model for which the restriction to finite types is the fully abstract pointed CCC. The first (based on Hyland-Ong games) is taken in Chapter 4, the second (based on sequential algorithms) in Chapter 5, where they are compared.

Corollary 2.5.14 *The fully abstract computational model of control is effectively presentable if and only if the observational equivalence on $\Lambda(\Omega)$ is decidble.*

PROOF: By Proposition 2.5.9 every element of the fully abstract model is a least upper bound of (embedded) elements of the fully abstract pointed CCC , and by Proposition 2.1.17, this is effectively presentable if and only if observational equivalence on $\Lambda(\Omega)$ (or the simply-typed λ-calculus itself) is decidable. $\square$

2.5.1 Universality

A good solution to the full abstraction problem will be a cpo-enriched and fully abstract model of control satisfying the following conditions:

The Jung and Stoughton criterion (as posed for PCF in [44]) The (finitary) fully abstract model of control is effectively presentable.

Universality All recursive elements of the model are definable.

The requirement that these conditions be satisfied together is significant, because it suggests that as well as being effective, such a presentation will be syntax independent. For instance, one could give a syntactic construction of a fully abstract model of μPCF along the lines of Milner's construction of a fully abstract model of PCF [61], and even make it effective by giving a decision procedure for finitary observational equivalence. To give a universal presentation, however, requires some identification of the computable infinitary elements of the model, as in the AJM [6] and HO [43] game-based universal models of PCF. But there is no obvious notion of computability for syntax-based models which leads to universality.

Universality is also a highly relevant condition because it allows an equational full abstraction result to be strengthened to 'logical full abstraction' (described by Longley and Plotkin in [56]). This extends full abstraction from the single test of program-context equality, or observational equivalence, to formulas of a first-order logic for reasoning about programs.

Definition 2.5.15 *For an extension of* PCF, *L, let J_L be the language of first-order logic with equality which has the* terms *(with arities) given by the signature of L, and the single predicate $\Downarrow$ for termination. A J_L-structure is thus defined* *by the assignment of meanings to the terms, and truth values to the predicates of J_L in the standard way, with the following natural examples.*

- *An operational semantics of L based on evaluation to ground type defines a J_L structure over the observational equivalence classes of terms of J_L (with constants interpreted as their own equivalence class, and $\Downarrow$ as operational convergence).*
- *A denotational semantics $\mathcal{M}$ for L defines a J_L structure over $\mathcal{M}$ in which valuations can range over the whole domain, and constants receive the interpretation given to them by $\mathcal{M}$, with the predicate $\Downarrow$ corresponding to some set of non-$\perp$ elements. (Under certain 'standardness assumptions', see [56], — it is certainly sufficient to require that $\mathcal{M}$ is given by a strong pointed monad and a rational CCC, as in Definition 2.4.4.)*

This allows the following definition to be made.

Definition 2.5.16 *A model of L is logically fully abstract with respect to an operational semantics if the formulas of J_L valid in the corresponding denotational structure coincide with those which are valid in the operational structure.*

The following theorem captures the intuition that logical full abstraction requires the *effective* elements of a fully abstract model to be identifiable.

Theorem 2.5.17 (Longley and Plotkin [56]) *Let $\mathcal{I}$ be a standard model of L. Then it is logically fully abstract for J_L if and only if it is both universal and fully abstract.*

(The implication from right to left is in fact quite simple to prove: equational full abstraction implies that *atomic* formulae are valid in the observational structure if and only if they are valid in the denotational structure, and this extends naturally to quantifier-free formulae. A universally quantified formula is valid in the observational structure if there is no valuation of *terms* which gives a counter-example, it is valid in the denotational structure if there is no valuation of *elements* yielding a counter-example. Hence if all elements are definable, the valid first-order formulas of the structures coincide.)

2.6 Summary

A programme for the construction and analysis of an effectively presented fully abstract syntax-independent model of control has been established. It is implemented in Chapters 3, 4 and 5, as follows.

Chapter 3 Gives denotational and axiomatic characterizations of the free pointed CCC, and hence the initial models of control. This falls naturally into three parts.

- A presentation of the intensional hierarchy of game semantics based on modelling types through some invariant structure, and computational effects through constraints on the copying and sharing of information.
- An axiomatic proof that the category of 'unbracketed' games and innocent strategies is isomorphic to the free (pointed) CCC.
- An examination of control flow within the control model, characterizing 'upwards' and 'downwards' continuation passing and local control flow, using linear moves and the 'bracketing condition'.

Chapter 4 Describes a 'syntax of control', $\lambda\mu(\Omega)$, and a PCF extension μPCF, showing that its equational theory and operational semantics are sound and complete for models of control. This is proved using invertible cps translations into the simply-typed λ-calculus, and a monadic metalanguage for reasoning about computation with continuation passing.

Chapter 5 Gives an effective presentation of the fully abstract, universal model of control (in the category of sequential algorithms).
Characterizes the observational preorder semantically, and compare the fully abstract models using a syntax independent description of the functor from the initial games model.

Chapter 3

Game semantics of control

3.1 Introduction

The primary object of this chapter is to develop a game semantics from which the continuation passing models described in the previous section can be constructed. The basic requirements are thus quite simple: pointed cartesian closed categories of games, and in particular, models of $\Lambda(\Omega)$ with the definability property. An axiomatic characterization of the free pointed CCC over one object is also presented, adapting existing results to models of control.

But a second aim is to study the semantics of the intensional hierarchy. A development of existing work is the formalization of a notion of *rule*, which is used to build up a general symmetric monoidal closed category of games with finer structure created by the imposition of rules. The focus is naturally on the 'control-axis' of the hierarchy. It is shown that constraints on copying information (or demands for information) by repeating moves can be expressed as such global rules. This can be used to give a semantics for classical, intuitionistic, and linear fragments of the category of games, such that relaxing the rule corresponds locally to a translation (such as double negation translation or 'linear decoration').

This sheds some light on the the 'bracketing condition' (used in the models of PCF in [6], [43]) as corresponding to local control flow and an intuitionistic typing discipline, showing that this rule has two (separable) roles, each with a clear computational meaning. On this analysis, it is essentially a linearity condition, requiring that 'answers' to demands for information be given *at most once*, preventing repeated invocation of continuations (see [82]), and *at least once*, preventing discarding of continuations, and hence escapes or jumps. (Similarly, there are two levels at which classical principles such as Peirce's law can be introduced into deduction systems; as 'stability rules' for atomic formulas, or at all levels, to reach full classical logic.)

3.2 A basic framework for games

The game semantics used is based on the Hyland and Ong approach [43] (which is similar to that of Nickau [65], and descended from the work of Lorenzen [57]). 'Dialogue games' are defined using *justified* sequences of moves. This choice was based on the following considerations.

- A neat division of labour between the apparatus of justification, which defines the possible 'logical' connectives for games, and rules, which enrich this repertoire by controlling the order in which moves can be played. The basic structural operations on arenas together with a few simple rules suffice to generate all of the games which will be required.

- A second concept introduced by Hyland and Ong, based on justified sequences, is the notion of the *view* as the relevant history of the play. The definability results rely on the restriction to innocent strategies, which have access only to the information in the view. But views have a useful secondary role in the intensional hierarchy; by hiding Opponent's ill behaviour, they allow all strategies with the same 'innocent function' to be identified. This means, for instance, that there is a faithful embedding of strategies on well-bracketed games into the wilder world via their reponses to views.

A significant development of the Hyland-Ong framework was described in McCusker's thesis [59]. This included a 'linear decomposition' of the cartesian closed structure, which is used in a modified form here. The clarity of exposition in this work also make it valuable background reading.

However, McCusker defines each game as being specified by a particular arena *and* a set of plays in the style of the AJ and AJM approach [5],[6]. These are subsets of the justified sequences over an arena which satisfy both the global rules (of visibility and bracketing) introduced by Hyland and Ong, — the 'legal sequences', but are also subject to some local characterizations. This is incompatible with the perspective advocated here, — that the only distinctions between games on the same arena are to be found in the rules. (And, more importantly, the 'projection conditions' on which his constructions are based simply fail in the games used to model computation with higher-type references [4].) Instead, games here are simply sets of justified sequences, which are specified (but not uniquely) by an arena, and a set of rules. Thus, for instance, removing the question/answer tagging from an arena specifies exactly the same game as retaining the labelling, but dropping the rule (bracketing condition) which refers to the tags.

3.2.1 Arenas and justified sequences

Lorenzen [57] introduced 'dialogue games' to semantics, and the analogy with a debate is a useful one. Debates can be categorized under two headings; their *subject*, and the *rules* under which they are held. Arenas specify the subject; the structure of the information relevant to the debate. This information is carried by a set of tokens representing actions, historically called moves, (although this almost immediately engenders a confusion between the token itself, and a specific instance of its use in the game). Moves are statements relevant to the subject, possibly distinguished by additional labels saying which side may utter them, in which order, whether they can be repeated, and whether they are incompatible with other statements.

The analogy also lends itself to the notion of *justification*. In the midst of a debate, for instance, one has a basic choice between opening an entirely new line of argument or attacking the assumptions underlying one of the opponent's more recent assertions. Attacks of the latter sort, which refer directly to a single earlier utterance are *justified* by it, according to this terminology. To determine when such a contingent attack can be made, there is an *enabling* relation between moves.

Definition 3.2.1 (Arenas) *An arena A is a quadruple:*
$\langle M_A, \vdash_A \subseteq (M_A)_* \times M_A, L, \lambda_A : M_A \to \mathcal{P}(L)\rangle$ *which specifies a tree-like structure with labelled nodes, as follows:*
The objects of M_A, are called moves,
$\vdash_A \subseteq (M_A)_* \times M_A$ *is a relation called* enabling*, such that the transitive closure of $\vdash$,* hereditary enabling*, is an irreflexive partial order. Moves enabled by the additional node $*$ added to M_A are said to be* initial.
L is a set of labels, and the labelling function $\lambda_A : M_A \to \mathcal{P}(L)$ specifies the set of labels attached to each move.

In general $l, m, n \ldots$ will be used to denote the moves of an arena, and $a, b, c, \ldots$ to denote occurrences of these moves in *sequences* r, s, t.

Definition 3.2.2 *For any sequences s, t,* $\mathrm{Occ(t)}$ *is the set of occurrences of moves in t, and $s \sqsubseteq t$ means 's is a prefix of t',*
sa denotes the sequence s extended by the move a, and $s \cdot t$ the sequence s extended by the sequence t.

Attention will be restricted to finite sequences here, although transfinite plays remain an interesting possibility.

Definition 3.2.3 *A justified sequence in an arena A, is a sequence of (occurrences of) moves s, together with a function $\phi : \mathrm{Occ(s)} \to \mathrm{Occ(s)}_*$ such that for every occurrence a of a non-initial move, $\phi(a)$ is an occurrence of a move preceding a, which enables it. (And if a is initial then $\phi(a) = *$.) This relationship between occurrences of non-initial moves and a unique, preceding, enabling move, is called justification, and the function ϕ is said to define 'justification pointers' for each non-initial move. The set of justified sequences of an arena A is denoted J_A.*

The prefix-ordering extends straightforwardly to justified sequences. The transitive closure of the justification relation on occurrences is referred to as hereditary justification.

3.2.2 Constructions on arenas

Arenas can be connected in a few simple ways.

Definition 3.2.4 Products *A product of arenas is formed by placing its components 'side by side'. For any set-indexed family of arenas $\{A_i \mid i \in I\}$, form the product $A = \Pi_{i\in I} A_i$ as follows:*

- $M_{\Pi_{i\in I} A_i} = \coprod_{i\in I} M_{A_i}$,
- $\langle m, i\rangle \vdash_{\Pi_{i\in I} A_i} \langle n, j\rangle$ *if* $i = j$ *and* $m \vdash_{A_i} n$, *and* $* \vdash_{\Pi_{i\in I} A_i} \langle n, j\rangle$ *if* $* \vdash_{A_j} n$,
- $L_{\Pi_{i\in I} A_i} = \bigcup_{i\in I} L_{A_i}$,
- $\lambda_{\Pi_{i\in I} A_i}(\langle m, i\rangle) = \lambda_{A_i}(m)$.

The empty arena, containing no moves (the empty product of arenas) is denoted **1**.

Function Space *This is an asymmetric version of the product arena; the set of moves is again a disjoint union with labelling inherited from the components, but the (formerly) initial moves of the arena which occurs negatively are enabled by the initial moves of the arena occurring positively.*

- $M_{A\multimap B} = M_A + M_B$,
- $\langle m, i\rangle \vdash_{A\multimap B} \langle n, j\rangle$ *if* $i = j$ *and* $m \vdash n$
  or $m \in M_B$, $n \in M_A$ *and* $* \vdash_B m$ *and* $* \vdash_A n$,
  $* \vdash \langle m, i\rangle$ *if* $m \in M_B$ *and* $* \vdash_B m$.

For $m \in A \times B, A \multimap B$, write $m \sqsubseteq A$ if m is a move from the A component, etcetera.

Justified sequences over these constructions can be projected into their constituent components.

Definition 3.2.5 *For a justified sequence s over an arena A, formed from subarenas $B, C, \ldots$ by the above constructions, the restriction $s\restriction(B, C, \ldots)$ is a justified sequence defined as follows:*
$\varepsilon\restriction(B, C, \ldots) = \varepsilon$
$sa\restriction(B, C, \ldots) = s$ *if* $a \not\sqsubseteq B, C, \ldots$
$sa\restriction(B, C, \ldots) = (s\restriction(B, C, \ldots))a$ *if* $a \sqsubseteq B, C, \ldots$.
The justifier of a in $sa\restriction(B, C, \ldots)$ is the most recently played move from $B, C, \ldots$ which hereditarily justifies a. *(If there is no such move, then a is initial.)*

Similarly, a sequence can be restricted to the occurrences of moves hereditarily justified by some single occurrence.

Definition 3.2.6 *Let s be a justified sequence in an arena A, and a an occurrence of a move of A in s. Then the 'thread of a in s', written $s\restriction a$, consists of the moves of s hereditarily justified by a, in the order in which they have occurred, with justification pointers from s.*

Note that if s is a justified sequence such that $s \in J_{A\times B}$ or $s \in J_{A\multimap B}$, then $s\restriction A \in J_A$ and $s\restriction B \in J_B$. If a is initial, and $s \in J_A$ then $s\restriction a \in J_A$. Even with this minimal amount of structure, a basic category of processes and agents (games and 'pre-strategies') can be defined, leaving scope for obtaining particular effects by applying different constraints.

Definition 3.2.7 *The category $\mathcal{A}$ has arenas as objects:*
morphisms $f : A \to B$ are non-empty, prefix-closed sets of justified sequences (pre-strategies) over the function-space $A \multimap B$.
Following Abramsky and Jagadeesan [5], composition of strategies can be described as 'parallel composition with hiding':
Given $f : A \to B$ and $g : B \to C$
$f; g = (f\|g)/B = \{s\restriction A, C \mid s \in f\|g\}$, *where $f\|g$ is the* uncovering *of f played against g:*
$f\|g = \{s \in J_{((A\multimap B)\multimap C)} \mid s\restriction(A, B) \in f \wedge s\restriction(B, C) \in g\}$.
There is a difference here with previous definitions of composition in Hyland-Ong style games — the use of the restriction operation on justified sequences to 'mend' justification pointers following the 'hiding' of moves in B.

For any arena A, the identity morphism $\mathtt{id}_A : A \to A$ is the 'copycat' which copies moves from the left hand component into the right hand and vice-versa.

Canonical morphisms are in general copycat strategies. They can be formalised (similarly to [6]) as follows.

Definition 3.2.8 *Suppose $F\colon A \to B, G\colon B \to A$ are order-preserving maps on arenas:*
i.e. partial functions from M_A to M_B such that:
$F(*_A) = *_B$ and $a \vdash b$ if and only if $F(b)$ defined implies $F(a)$ defined and $F(a) \vdash F(b)$.
Then let $\mathtt{copy}_{F,G} : A \to B$ be the least set of justified sequences closed under the following definition:

- $\varepsilon \in \mathtt{copy}_{F,G}$
- s (even − length) $\in \mathtt{copy}_{F,G}$, *and* $a \sqsubseteq A$ *implies* $saF(a) \in \mathtt{copy}_{F,G}$
- s (even − length) $\in \mathtt{copy}_{F,G}$, *and* $b \sqsubseteq B$ *implies* $sbG(b) \in \mathtt{copy}_{F,G}$

where the justifier of $F(a)$ (or $G(b)$) is the move succeeding the justifier of a (or b).

Thus, for each arena A, if ID(A) is the identity map from A to itself,

$$\mathtt{id}_A = \mathtt{copy}_{ID(A),ID(A)}.$$

Definition 3.2.9 *Suppose $F : B \to C$ is an order-preserving map of arenas. This naturally extends to maps F_r^A from justified sequences over $A \multimap B$, to justified sequences over $A \multimap C$, and F_l^A from $J_{C\multimap D}$ to $J_{B\multimap D}$.*
$F_r^A(\varepsilon) = \varepsilon$
$F_r^A(sa) = F_r^A(s)a$, *if* $a \sqsubseteq A$
$F_r^A(sb) = F_r^A(s)F_r^A(b)$, *if* $b \sqsubseteq B$
where these are defined, with $\phi(F(a)) = F(\phi(a))$ for non-initial moves in B.
So F_r^A sends morphisms on $A \to B$ to morphisms on $A \to C$ and
$F_l^A : \mathcal{A}(C, D) \to \mathcal{A}(B, D)$.

Proposition 3.2.10 *If $F : B \to C$ is an order-isomorphism of arenas, then for any pre-strategies $f : A \to B$, and $g : C \to D$*
$f; \mathtt{copy}_{F,F^{-1}} = F_l^A(f)$, *and* $\mathtt{copy}_{F,F^{-1}}; g = F_r^A(g)$.

Proof: Note that
$s \in \mathtt{copy}_{F,F^{-1}}$ if and only if for all even prefixes $t \sqsubseteq s$, $F(t{\restriction}B) = t{\restriction}C$.
Hence by definition of parallel composition of strategies, $sa \in f\|\mathtt{copy}_{F,F^{-1}}$, if and only if $sa{\restriction}A, B \in f$ and $F_l(t{\restriction}A, B) = F_l^{-1}(t{\restriction}A, C)$ for all prefixes t of s such that $t{\restriction}A, B$ is even-length.
Hence $s \in f; \mathtt{copy}_{F,F^{-1}}$ if and only if $s = F_l(s')$ for some $s' \in f$; i.e. $s \in F_l(f)$. $\square$

For any pre-strategy $f : A \to A$, $ID(A)_l(f) = ID(A)_r(f) = f$, so $\mathtt{id}_A$ is a well-defined identity.

Proposition 3.2.11 ($\mathcal{A}$ is a well-defined category)

PROOF: It remains to show associativity of composition, for which it is sufficient to show that for pre-strategies $f : A \to B$, $g : B \to C$, $h : C \to D$,
$f; g; h = (f; g); h = f; (g; h)$ (for which the cases are similar)
where $f; g; h = (f\|g\|h)\restriction(A, D)$, and $f\|g\|h =$

$$\{s\restriction A \multimap D \mid s \in ((A \multimap B) \multimap C) \multimap D) \wedge s\restriction A, B \in f \wedge s\restriction B, C \in g \wedge s\restriction C, D \in h\}$$

If $s \in f; g; h$, then there is some $t \in ((A \multimap B) \multimap C) \multimap D$ such that $t\restriction A, D = s$, and $t\restriction A, B \in f$, $t\restriction A, B, C \in f\|g$, hence $t\restriction A, B, D \in (f; g)\|h$, and
$t\restriction A, D = s \in (f; g); h$.
To prove the converse:
Suppose $s \in (f; g)\|h$ then by definition of composition, there is some $t \in f\|g$ such that $s\restriction(A, C) = t\restriction(A, C)$. It is sufficient to show by induction on length of s that there exists $r \in f\|g\|h$ such that $r\restriction A, D = s\restriction A, D$ and $r\restriction A, B, C = t$, and hence $s\restriction A, D \in f; g; h$.
So given $sa \in (f; g)\|h$, and $t \cdot u \in f\|g$, such that $sa\restriction(A, C) = t \cdot u\restriction(A, C)$ and $s\restriction(A, C) = t\restriction(A, C)$, applying the inductive hypothesis to s yields $r \in f\|g\|h$ such that $r\restriction A, B, C = t$, and $r\restriction A, D = s\restriction A, D$. Then either a is a move in D (so $u = \varepsilon$), and let $r' = ra$, or a is a move in A, B, C, — let $r' = r \cdot u$. Then $r' \in f\|g\|h$, and $r'\restriction A, B, C = t \cdot u$, and $r'\restriction A, D = s\restriction A, D$ as required. □

Proposition 3.2.12 *The empty arena* $\mathbf{1}$ *is a terminal object in* $\mathcal{A}$*, as* $A \multimap \mathbf{1} = \mathbf{1}$ *for any arena* A*.*

Hence $A \multimap B \cong \mathbf{1} \multimap (A \multimap B)$ for any A, B, and so each morphism in $\mathcal{A}(A, B)$ has a unique 'name' in $\mathcal{A}(\mathbf{1}, A \multimap B)$.

Proposition 3.2.13 (Symmetric monoidal closure) $\mathcal{A}, \otimes, \mathbf{1}, \multimap$, *is a symmetric monoidal closed category, where* $A \otimes B$ *is the product of arenas* A *and* B*, the unit* $\mathbf{1}$ *is the empty arena, and* $A \multimap B$ *is the 'function space' arena defined above.*

PROOF: The isomorphisms of arenas $Assoc : (A \times B) \times C \to A \times (B \times C), I : A \times \mathbf{1} \to A$, and $twist : A \times B \to B \times A$ define copycat strategies which are the canonical morphisms for the monoid, and satisfy the coherence conditions for a

symmetric monoid.
For $f : A \to C$ and $g : B \to D$

$$f \otimes g = \{s \in J_{(A \otimes B) \multimap (C \otimes D)} \mid s{\restriction}A, C \in f \wedge s{\restriction}B, D, \in g\}$$

Bifunctoriality of $\otimes$ (i.e. $(f \otimes g);(h \otimes k) = (f;h) \otimes (g;k)$ for $f : A \to C, g : B \to D, h : C \to E, k : D \to F$) is shown as follows.
Suppose $s \in (f \otimes g);(h \otimes k)$, then there exists $t \in (f \otimes g)\|(h \otimes k)$ such that $t{\restriction}A, C = f$, $t{\restriction}D, E \in g$, $t{\restriction}C, E \in h$, $t{\restriction}D, F \in k$. Then $t{\restriction}A, C, E \in f\|h$, and $t{\restriction}B, D, F \in g\|k$, so $s = t{\restriction}A, B, E, F \in f;h \otimes g;k$.
To show that $f;h \otimes g;k \subseteq f \otimes g;h \otimes k$:
If $s \in f;h \otimes g;k$, then there exists $r \in f\|h$ such that $s{\restriction}A, E = r{\restriction}A.E$, and $t \in g\|k$ such that $s{\restriction}B, F = t{\restriction}B, F$. It is sufficient to show by induction on length of s that there exists $u \in f \otimes g\|h \otimes k$ such that $s = u{\restriction}A, B, E, F$, and $r = u{\restriction}A, C, E$, and $t = u{\restriction}B, D, F$, and hence $s \in f \otimes g;h \otimes k$ as required.
Suppose $sa \in f;h \otimes g;k$, $t \in f\|h$, and $r \in g\|k$ such that $sa{\restriction}A, E = t \cdot v{\restriction}A, E$, and $sa{\restriction}B, F = r \cdot w{\restriction}B, F$, and $s{\restriction}A, E = t{\restriction}A, E$, and $s{\restriction}B, F = r{\restriction}B, F$. Then there is some $u \in f \otimes g\|h \otimes k$, satisfying the inductive hypothesis with respect to s, r, t. Form $u' = u \cdot v \cdot w$, satisfying the inductive hypothesis with respect to $sa, r \cdot v, t \cdot w$.
Finally, there is an order isomorphism $\mathtt{LAMBDA}_{\mathtt{A,B,C}} : \mathtt{A} \otimes \mathtt{B} \multimap \mathtt{C} \cong \mathtt{A} \multimap (\mathtt{B} \multimap \mathtt{C})$, which defines an adjunction between $\_ \otimes A$ and $A \multimap \_$.
Writing, $'f' : A \to B$ for the inverse to the naming operation:
$\Lambda(f : A \otimes B \to C) = '\Lambda_r(`f')'$.
The copycat strategy
$\mathrm{App}_{\mathrm{A,B}} : (\mathrm{A} \multimap \mathrm{B}) \otimes \mathrm{A} \to \mathrm{B} = \mathtt{LAMBDA}^{-1}_{\mathtt{A,B,A} \multimap \mathtt{B}}(`\mathtt{id}_{\mathtt{A} \multimap \mathtt{B}}')$ is the co-unit. $\square$

3.3 Rules and Games

If arenas give a general notion of subject for debate, or type, games correspond to a more specific, and more extensional notion; the interactions possible under a given set of rules.
To define games, the labelling on the moves is used, along with *rules* which govern when they can be played.

Definition 3.3.1 *A rule consists of a pair $\langle R, L_R \rangle$ of a predicate R on justified sequences, with a set L_R of 'R-relevant' moves. So the labelling function λ_A can be 'projected' to a function $\lambda_A^R : M_A \to L_R$, where*
$\lambda_A^R(m) = \lambda_A(m) \cap L_R$.In cases where λ_A^R is always either empty or a singleton,

λ_A^R may be given as a partial function from M_A to L_R.
The predicate R must satisfy the following requirements:

Prefix Closure *$s \sqsubseteq t$ implies $R(t) \Longrightarrow R(s)$*

R-relevance $R(s) \wedge \lambda^R(a) = \varnothing \quad \Longrightarrow R(sa)$.

Note that because equality of justified sequences is defined only in terms of having the same justification pointers, and labellings on moves, a rule can only 'observe' these properties, and not, for instance, whether a sequence was generated from a particular arena. The rules discussed in this thesis can be easily seen to have these properties, as they can be given as inductive definitions of the form:
'$R(sa)$ if and only if $R(s)$ and $\lambda^R(a) \neq \varnothing \Longrightarrow \Phi_R(s, a)$',
where Φ_R is a formula in a semi-formal 'language of justified sequences' containing predicates such as "s is a prefix of t", "a is the justifier of b in t" and so on.

For a finite set of rules $R = \{\langle r_1, L_1\rangle, \ldots \langle r_n, L_n\rangle\}$, R will also be used, by abuse of notation, to denote their conjunction, $\langle r_1 \wedge \ldots \wedge r_n, L_1 \cup \ldots \cup L_n\rangle$

Proposition 3.3.2 *The conjunction of of well-defined rules is a well defined rule.*

Since each rule is directly relevant to a labelling $\lambda_A^R : M_A \to \mathcal{P}(L_R)$, and applies to moves only on which this labelling is defined, rules can be relaxed over an arena simply by removing labels.

Definition 3.3.3 *Given an arena A and a set of moves $M \subseteq M_A$, the rule R is* relaxed *on M in A to give a new arena A' by removing all of the relevant tags from the moves, — i.e. defining $A' = \langle M_A, \vdash_A, L, \lambda_{A'}\rangle$ where*
$\lambda_{A'}(m) = \lambda_A(m)/L_R$ if $m \in M$, and $\lambda_{A'}(m) = \lambda_A(m)$ otherwise.
If a rule is relaxed over the entire arena ($M = M_A$) to form A/R then it is said to have been abandoned.

Definition 3.3.4 *A game is a set of justified sequences of moves which is specified (up to equality of justified sequences) by* some *arena and rule.*
For a rule set R, and arena A, $R(A)$ will denote the game with arena A, and plays subject to the rules R, i.e.

$$R(A) = \{s \in J_A \mid R(s)\}$$

Thus games are in some sense more extensional objects than the arenas which 'realize' them, although this should not be confused with the (more significant) point that strategies are intensional.

Proposition 3.3.5 *For any rule-set S containing a rule R, and arena A, the game specified by abandoning R on A and applying S, is equivalent to applying S without R to A $(S/\{R\})(A) = S(A/R)$.*

PROOF: is direct by definition. $\square$

A strategy on a game G is a prefix-closed set of even-length plays in G.

Definition 3.3.6 *For any rule R, form the 'category' $\mathcal{G}_R$ with*
objects: games $R(A)$ for some arena A
morphisms from A to B: pre-strategies on $R(A \multimap B)$, with composition defined using uncovering:

$$\sigma;\tau = \{s{\restriction}A, B \mid s \in \sigma\|\tau \wedge R(s{\restriction}A, B)\}.$$

In order for this to define a SMCC [51], it is necessary to make the following further requirements of rules:

Copycats: *If $F : A \to B$ is an isomorphism of arenas, then $R(\mathtt{copy}_{F,F^{-1}})$: $R(A \multimap B) = \{s \in \mathtt{copy}_{F,F^{-1}} \mid R(s)\}$ is an isomorphism. So, in particular, for each A, $R(\mathtt{copy}_{ID(A),ID(A)})$ is an identity on $R(A)$.*

Associativity: *If $s \in ((A \multimap B) \multimap C) \multimap D$ is such that $R(s{\restriction}A, B)$, $R(s{\restriction}(B, C)), R(s{\restriction}(C, D))$, and $R(s{\restriction}(A, D))$ then $R(s{\restriction}(A, C))$ if and only if $R(s{\restriction}(B, D))$.*
To show that this means that composition is associative, suppose R is associative, and $\sigma : A \to B$, $\tau : B \to C$ and $\rho : C \to D$. Then if $s \in (\sigma;\tau);\rho$, by Proposition 3.2.11 there exists $t \in ((A \multimap B) \multimap C) \multimap D$ such that $t{\restriction}A, B \in \sigma$, $t{\restriction}(B, C) \in \tau$, $t{\restriction}(C, D) \in \rho$ and $t{\restriction}A, D = s$, and R holds of all these subsequences. So by assumption, $R(s{\restriction}(A, C))$ implies $R(s{\restriction}(B, D))$, so $s \in \sigma;(\tau;\rho)$. The converse is entirely similar, so $(\sigma;\tau);\rho = \sigma;(\tau;\rho)$ as required.

Proposition 3.3.7 *Each category $\mathcal{G}_R$ defined from such a rule R is symmetric monoidal closed.*

PROOF: For games $R(A), R(B)$, define the tensor product and exponential, $R(A) \otimes R(B) = R(A \otimes B)$, and $R(A) \multimap R(B) = R(A \multimap B)$.
The isomorphisms of arenas between $A \times B$ and $B \times A$, $(A \times B) \times C$ and $A \times (B \times C)$ and $(A \times B) \multimap C$ and $A \multimap (B \multimap C)$ define isomorphisms in $\mathcal{G}_R$ by Definition 3.3.6, so the proof of symmetric monoidal closure goes through just as in Proposition 3.2.13. $\square$

A result which depends on using rules and arenas to *specify* games is the following.

Proposition 3.3.8 *Every game is specified by an arena which is a forest. (i.e. one in which every non-initial move is enabled by a unique move).*

PROOF: is for finite arenas, but can be extended to arenas of any size. Call a move a 'branching node' if it is enabled by two or more different moves. Then a finite arena is a tree (with respect to hereditary enabling) if it contains no branching nodes. So the proof is by induction on the number of branching nodes, that every arena is play-equivalent to a tree. Assuming A is not a tree, choose $m \in M_A$ such that $n_i \vdash_A m : i \leq k$ for some distinct $n_1, n_2, \ldots n_k \in M_A$, but there is no branching node hereditarily enabled by m.
Let $A \restriction m$ be the arena of moves hereditarily enabled by m (including m) with enablings as in A. Define the arena A' to be the same as A, except that it has a separate copy of the subtree $A \restriction m$ attached below each n_i. Then A' has fewer branching nodes than A, and $J_A = J_{A'}$, as in any justified sequence over A, each occurrence of m is justified by a unique occurrence of some n_i, which defines an obvious isomorphism of justified sequences (which preserves the labelling on occurrences of moves). Hence $R(A) = R(A')$ for any rule R. $\square$

Note that the one-move arena, o, with a single move (appropriately labelled), is sufficient to define all finite arenas using the function space and product constructions.

3.4 'Linear' Games

This section describes a cartesian closed category and associated 'linear decomposition' of a category of games, based on a single rule. The notion of linear decomposition is borrowed from (intuitionistic) linear logic [29] and its categorical models [11], but the purpose is not to define a full linear type structure. Instead linearity is used to analyze how restricting repetition of moves can define sound and complete models of control (i.e. pointed CCCs) and identify strategies which correspond to functional programs with local control. This is organised along the lines of [6] and [59]; in addition to the SMCC, a cartesian product and co-monad ! such that $!(A \times B) \cong !A \otimes !B$ are described, so that the co-Kleisli category of the ! is cartesian closed. So the 'intuitionistic' function type $A \Rightarrow B$ is modelled as $!A \multimap B$.

The novel feature is that this is done using a notion of 'linear' move. The role of the ! can then be explained in terms of relaxing linearity. In addition, using distinguished moves to define linear connectives will allow the 'bracketing condition', describing local control flow, to be presented and analyzed using the decomposition of classical and intuitionistic types into linear logic.

Definition 3.4.1 (Linearity rule) *Any two distinct linear moves (those labelled 'L') occurring in the same justified sequence have different justifiers.*
More formally:
$L(sa)$ *if* $\lambda^L(a) = L$ *implies that for all* $b \in \mathrm{Occ(s)}$, $\lambda^L(b) = L$ *implies* $\phi(a) \neq \phi(b)$.
(Note that non-initial linear moves can *occur more than once, so long as the different occurrences have different justifiers.)*

However, there is a problem with this rule: — associativity of composition fails. (This is an old problem, going back to Blass' game semantics of linear logic ([13], see also [5]).

Proposition 3.4.2 *Composition of strategies on linear games is not associative.*

PROOF: Consider the following games and strategies $\rho : A \multimap B$ and $\sigma : B \multimap A$, where the initial move of B is linear, but the initial move of A is not (otherwise they are the same)
ρ and σ just play copycat, which are valid strategies, as if B repeats its initial move, the repeated linear move made in A is justified by the new move. Hence $\rho;\sigma = \mathtt{id}_A$, and so $(\rho;\sigma);\tau = \tau$ for any $\tau : A \multimap C$.
But suppose τ repeats the initial move in A, justified by the same initial move of C (again legitimate, as this is not a linear move). But this cannot be the restriction to $A \multimap C$ of a sequence in $\sigma\|\tau$, however, as such a sequence would also contain the initial move in B twice, justified by the same move, violating linearity. Hence this play is not in $\rho;(\sigma;\tau)$ either. Thus $\rho;(\sigma;\tau) \neq \tau$. $\square$

This problem is caused by the possibility of forming arbitrary constructions with linear moves. The solution adopted here is to restrict some of these possibilities.

Definition 3.4.3 *An arena is* well-opened *if all of its initial moves are linear.*
An arena is freely-opened *if none of its initial moves are linear.*

Compare this with McCusker's definition of well-openedness for *games* [59] (Hyland-Ong games are implicitly defined to be well-opened):

Definition 3.4.4 *A game G is well-opened if $sa \in G$ and a is initial implies $s = \varepsilon$.*

So under the linearity rule, the well-opened games are precisely those specified by a well-opened arena.
Well and freely opened games can also be combined.

Definition 3.4.5 *The category of well-opened games $\mathcal{W}$, has well-opened games as objects and pre-strategies on $A \multimap B$ as morphisms.*
The category of freely-opened games $\mathcal{F}$, has freely-opened games as objects and pre-strategies on $A \multimap B$ as morphisms.
The well and freely opened games form a category $\mathcal{O}$ in which
Objects are the well-opened and the freely opened games.
Morphisms from A to B are pre-strategies on $A \multimap B$, provided *that A is well-opened implies B is well-opened (if A is freely opened and B is well-opened then $\mathcal{O}(A, B)$ is defined to be empty).*

In none of these cases can the pathology described above arise, and without this possibility, associativity of composition is implied by the stronger property of *compositionality.*

Definition 3.4.6 *A rule is* compositional *if for all $s \in (A \multimap B) \multimap C$, $R(s \upharpoonright A, B)$ and $R(s \upharpoonright B, C)$ implies $R(s \upharpoonright A, C)$, so that if $\sigma : R(A \multimap B)$ and $\tau : R(B \multimap C)$ then $\sigma; \tau$ is a set of plays in $R(B \multimap C)$.*

Proposition 3.4.7 *Linearity of moves is compositional, and hence the categories of well and freely-opened games and its subcategories of well-opened, and freely opened games are well-defined.*

Proof: Suppose $s \in (A \multimap B) \multimap C$, and $L(s{\upharpoonright}A, B), L(s{\upharpoonright}B, C)$, so $L(s{\upharpoonright}A)$ and $L(s{\upharpoonright}C)$. Thus the only possible violation of linearity in $L(s{\upharpoonright}A, C)$ is if some initial move in A is linear, but an initial move in C is not. But if C is not well-opened, then A is freely opened, so this impossible.
The copycat strategy $L(\mathtt{id}_A)$ is an identity in the category, as moves in $s \in \mathtt{id}_A \| f : A \multimap B$ are repeated with the same justifier only if they are repeated in $s{\upharpoonright}A, B \in f$ with the same justifier, and are therefore not linear. $\square$

The product of a well-opened and a freely opened arena is neither well, nor freely opened, so this will not in general specify a well-defined operation in $\mathcal{O}$. However, it can be applied in both of its subcategories, with different results.

Proposition 3.4.8 *The product of arenas (Definition 3.2.4) defines a cartesian product on the category of well-opened linear games.*

PROOF: For well-opened games $L(A)$ and $L(B)$, define

$$L(A)\&L(B) = L(A \times B)$$

which is also well-opened.
Then $L(A)\&L(B) = \{s \in J_{A\times B} \mid s \in L(A) \vee s \in L(B)\}$, — since the initial move is unique, it must represent a choice to play in one of the two components for the rest of the game.
(Note that this is the *definition* of cartesian product of AJ games, and in [59] etcetera.)
So for maps $f_1 : L(A) \to L(B), f_2 : L(A) \to L(C)$ there is a *unique* pairing

$$\langle f, g\rangle = \{s \in L(A) \multimap (L(B)\&L(C)) \mid s \sqsubseteq f \vee s \sqsubseteq g\}$$

such that $\langle f_1, f_2\rangle; \pi_i = f_i$.
(The pairing operation for the tensor exists, but is not unique.) □

Not only is the subcategory of freely opened games a symmetric monoidal closed category with respect to $\otimes, \multimap$, but the closure extends over the combined category in the following sense.

Proposition 3.4.9 *For any freely opened game B, the map* $B \multimap \_ : \mathcal{O} \to \mathcal{O}$ *is functorial, and the co-unit of the adjunction between* $(B \multimap \_)\restriction\mathcal{F}$ *and* $\_ \otimes B$

$$\mathrm{App} : ((\mathrm{B} \multimap \mathrm{C}) \otimes \mathrm{B}) \to \mathrm{C}$$

extends (with its co-universal property) to all objects C of $\mathcal{O}$.
i.e. $\mathcal{O}(A \otimes B, C) \cong \mathcal{O}(A, B \multimap C)$ *for any C.*

PROOF: is by extension of Proposition 3.2.13. □

3.4.1 The linear exponentials

The linear exponentials, ! and its dual ?, are introduced to linear logic to restore the possibility of performing structural rules, with some measure of restraint. The various translations of classical and intuitionistic types and proofs into linear logic [29], [22], [31] are based upon the 'decoration' of linear formulas with sufficient exponentials to permit the translation of all of the structural rules in the proof. Such linear decomposition of the Hyland-Ong games has already been given, in a different form, by McCusker. The main difference here is that the ! is modelled as a local relaxation of the linearity conditions. Thus there is a direct connection between the linear decomposition, and decoration, and the global relaxation of the linearity conditions. In the presence of the global rules applied in [59] (visibility and well-bracketing) the interpretation of the relevant parts of the negative fragment $(!, \otimes, \&, \multimap)$ proves to be play-equivalent to McCusker's, so [59] is an important source of detail. Both constructions are somewhat ad hoc, because the Hyland-Ong games lack the structure to support the full range of linear types satisfactorily. (The problem with the ! in [59] is that it is not a co-monad, as there is in general no interpretation of the 'dereliction map' $!A \multimap A$. McCusker goes on to show, however, that a true co-monad can be defined (an infinite tensor product with an equivalence on strategies) which yields an equivalent CCC as its co-Kliesli category.) Combining the well and freely opened games in a single category here allows the ! to be described as a co-monad (at the cost of some restriction of symmetric monoidal closure).

In order to define a co-monad, attention is restricted to a certain subset of the pre-strategies, essentially those identified in [4] as being *thread independent.*

Definition 3.4.10 *A (pre-)strategy $\sigma : A$ is* thread-independent *if $sa, t \in \sigma$ and $(s\upharpoonright c) = t\upharpoonright c$ (where c is the initial move hereditarily justifying a) then $ta \in \sigma$.*

So thread-independent strategies are defined by their sets of threads. They are employed (with a notion of polarity and determinacy) by Abramsky, McCusker and Honda to give a semantics of general references [4].

Proposition 3.4.11 *The thread-independent pre-strategies on linear games form a well-defined category.*

Proof: The identity is thread-independent, since $s \in \mathtt{id}_A$ if and only if $r\upharpoonright c \in id_A$ (c initial) for every prefix $r \sqsubseteq s$. Hence if $sa, t \in \mathtt{id}_A$ and $s\upharpoonright c = t\upharpoonright c$, then $sa\upharpoonright c \in \mathtt{id}_A$, so $ta\upharpoonright c \in \mathtt{id}_A$, so $ta \in \mathtt{id}_A$.
The composition of thread-independent strategies $\sigma : A \to B$ and $\tau : B \to C$ is thread-independent.
First observe that the uncovering $\sigma\|\tau$ is itself 'thread independent'. Suppose $sa\upharpoonright b \in \sigma\|\tau$, and $t \in \sigma\|\tau$. Then by thread independence of σ, τ, $ta\upharpoonright A, B \in \sigma$, and $ta\upharpoonright B, C \in \tau$, hence $ta \in \sigma\|\tau$.
The difficulty for going on to prove thread-independence for $\sigma\|\tau\upharpoonright B, C$ is that $t\upharpoonright A, C\upharpoonright a = s\upharpoonright A, C\upharpoonright a$ does not necessarily imply that $t\upharpoonright a = s\upharpoonright a$; the parts of the thread which are hidden may be different. However, threads are in some sense interchangeable, as shown below.

Lemma 3.4.12 *Suppose $s, t \in \sigma\|\tau$, where s is single threaded (well-opened), and there is an initial move a in t such that $t\upharpoonright a\upharpoonright A, C = s$ (but this is not necessarily* *the current thread). Then there exists* $t' \in \sigma \| \tau$ *such that* $t' \restriction A, C = t \restriction A, C$, *and* $t'/(t' \restriction a) = t/(t \restriction a)$ *and* $t' \restriction a = s$.

PROOF: is by induction on the length of t:
Suppose $s, tb \in \sigma \| \tau$, and $tb \restriction a \restriction A, C = s$.
Suppose b is not hereditarily justified by a, then by inductive hypothesis there exists $t' \in \sigma \| \tau$ such that $t' \restriction A, C = t \restriction A, C$, and $t'/(t' \restriction a) = t/(t \restriction a)$. Thus in particular $t' \restriction c = t \restriction c$, where c hereditarily justifies b. So by thread independence of $\sigma \| \tau$, $t'b \in \sigma \| \tau$, and $t'b$ satisfies the above conditions.
Suppose b is hereditarily justified by a, then *either* $b \sqsubseteq B$ — in which case the t' satisfying the inductive hypothesis for t will also satisfy it for tb,
or $b \sqsubseteq A, C$, so $s = s'b \cdot u$ for some suffix u of moves wholly in B, and justified by a. Then by inductive hypothesis there exists $t' \in \sigma \| \tau$ such that $t' \restriction A, C = t \restriction A, C$, and $t'/(t' \restriction a) = t/(t \restriction a)$, and $t' \restriction a = s'$. By repeated applications of thread independence, $t'b \cdot u \in \sigma \| \tau$, (and $t'b \cdot u$ satisfies all of the other conditions as well). $\square$

So to show thread independence of $\sigma ; \tau$, suppose $sa, t \in \sigma ; \tau$, and $sa \restriction b = ta \restriction b$. Then by the lemma there exists $t'a \in \sigma \| \tau$ such that $t' \restriction A, C = t$, and $t \restriction A, C \restriction b = sa \restriction b$, and hence $t'c \restriction A, C = ta \in \sigma ; \tau$.

Definition 3.4.13 *For any arena, A, obtain !A by relaxing the linearity condition on the initial moves of A. For any game* $G = L(A)$,

$$!G = L(!A).$$

For thread-independent $\sigma : A \rightarrow B$, *define the pre-strategy* $!\sigma$ *to be the least prefix-closed set of sequences of* $L(!A \multimap !B)$ *such that*
if $s \in !\sigma$, *and* $sa \restriction b \in \sigma$, *for some b hereditarily justifying a, then* $sa \in !\sigma$.

The following facts are immediate:

- the image of ! is the category of freely opened games and thread-independent strategies,
- its restriction to that category is the identity,
- hence it is idempotent.

Proposition 3.4.14 ! *is a co-monad on the category of well and freely opened games and thread-independent strategies.*

Proof: ! is an endofunctor on $\mathcal{O}$:
For $\sigma : A \to B, \tau : B \to C$, $!(\sigma;\tau) = !\sigma; !\tau$: — proof is by induction on sequence length.
Suppose $sa \in !(\sigma;\tau)$, so that $s \in !\sigma; !\tau$ by hypothesis, and $sa\restriction b \in \sigma;\tau$, and hence $sa\restriction b \in !\sigma; !\tau$ so by thread-independence of $!\sigma; !\tau$, $sa \in !\sigma; !\tau$ as required.
If $sa \in !\sigma; !\tau$, then $s \in !(\sigma;\tau)$, and there exists $t \in !\sigma || !\tau$ such that $t\restriction A, C = sa$.
$t\restriction b\restriction A, B \in \sigma$, and $t\restriction b\restriction B, C \in \tau$, by thread independence,
hence $t\restriction A, C\restriction b = sa\restriction b \in \sigma;\tau$, and so $sa \in !(\sigma;\tau)$, by definition of !.
There are natural transformations, — dereliction, $\mathtt{der} : ! \to \mathrm{Id}$, and promotion, $\mathtt{prom} : !! \to !$. But these are both trivial copycats as $! = !!$, and $\mathtt{der}_A : !A \to A$ is just given by the same function from threads to moves as the identity.
For $A, B \in \mathcal{O}$ which are both well-opened or freely opened

$$!(L(A)\&L(B) = !L(A \times B) = !L(A) \otimes !L(B)$$

so in particular, there is a contraction map $\mathtt{con_A} : \mathtt{!A} \multimap \mathtt{!A} \otimes \mathtt{!A}$ which is the strategy $!\delta_A$, where $\delta : A \to A \times A$ is the diagonal map for $\times$.
The following naturality property for contraction is then reducible to functoriality of ! and $\times$.

$$\begin{tikzcd}
!A \arrow[r, "\mathtt{con_A}"] \arrow[d, "!\sigma"'] & !A \otimes !A \arrow[d, "!\sigma \otimes !\sigma"] \\
!B \arrow[r, "\mathtt{con_B}"'] & !B \otimes !B
\end{tikzcd}$$

□

Since ! is a co-monad, it has a co-Kleisli category, $\mathcal{O}_!$, which has the same objects as $\mathcal{O}$, and as morphisms from A to B, thread-independent strategies on $!A \multimap B$.

Proposition 3.4.15 *$\mathcal{O}_!$ is cartesian closed.*

Proof: The generalised product has been shown to be cartesian on well-opened games; it remains to show cartesian closure.
$\mathcal{O}_!(A \times B, C) = \mathcal{O}(!(A \times B), C) = \mathcal{O}(!A \otimes !B, C)$.
By the extended symmetric monoidal closure (Proposition 3.4.9)
$\mathcal{O}(!A \times !B, C) \cong \mathcal{O}(!A, !B \multimap C)$
$= \mathcal{O}_!(A, B \Rightarrow C)$ as required. □

Thus a *global* relaxation of the linearity rule yields a CCC.

Corollary 3.4.16 *The category of arenas and thread-independent pre-strategies (the image of the well-opened games under !) is cartesian closed.*

3.5 Two-player arenas

A fundamental and distinctive aspect of game semantics is that interaction is modelled as a dialogue between different agents. The simplest and most important case simply distinguishes 'Player', — the system, from the rest of the world, — the environment ('Opponent'). Games need not be seen as competitive, however, but as co-operative efforts between program and input, to produce a useful result.

Definition 3.5.1 *An* alternating *arena,* $\langle M, \vdash, \{O, P\} \subseteq L, \lambda\rangle$ *is an arena in which the set of moves is partitioned between* Opponent *'O-moves' and* Player *'P-moves'. In other words, the projection of the labelling function as* $\lambda^{OP} : M \to \{O, P\}$ *is total. (Sometimes the complementarity* $\overline{O} = P$*,* $\overline{P} = O$ *will be used.) It is further required that*

- *All initial moves are Opponent's:* $* \vdash m$ *implies* $\lambda^{OP}(m) = O$*,*
- *Enabling alternates between Opponent and Player:* $m \vdash n$ *implies* $\lambda^{OP}(m) = \overline{\lambda^{OP}(n)}$*.*

Definition 3.5.2 *A justified sequence satisfies the alternation rule if its moves are made alternately by Player and Opponent:*
$A(sab)$ *if and only if* $A(sa)$ *and* $\lambda^{OP}(b) = \overline{\lambda^{OP}(a)}$*.*

Now that there is a notion of polarity, pre-strategies can be refined to strategies.

Definition 3.5.3 *A strategy on game* A *is a non-empty* even-*prefix-closed set of alternating sequences in* A*.*

The adjustment from pre-strategies as prefix-closed sets, and strategies as even-prefix closed does not radically alter the structure which has been built up so far. A pre-strategy over an alternating game can be transformed to a strategy by erasing all of its odd-length sequences. Odd-length sequences are re-introduced to strategies in a limited way, to represent escapes, in Chapter 5, Section 5.3.2.

In fact, it is necessary to modify the definition of the 'function space' construction $\multimap$ slightly in order to define an alternating arena, as the initial moves of A in $A \multimap B$ were originally Opponent moves, but are justified by the initial (Opponent) moves of B. (Suggesting that alternation is a rather more fundamental kind of rule to the others considered here.) The solution is to *invert* the Player/Opponent labelling on moves in A, in accordance with the fact that A appears negatively.

Definition 3.5.4 *Form the alternating function-space arena $A \multimap B$ as in Definition 3.2.4, except that*
$\lambda^{OP}_{A\multimap B}(m) = \lambda^{OP}_{B}(m)$, *if* $m \sqsubseteq M_B$,
$\lambda^{OP}_{A\multimap B}(m) = \overline{\lambda^{OP}_{A}(m)}$, *if* $m \sqsubseteq M_A$.

However, the 'switching condition', (of [5], for instance) which says that Opponent must play in the same component as the previous move in a game $A \multimap B$ (and similarly Player must do so in $A \otimes B$) is neither implicitly nor explicitly included in this definition (it does not hold in the games model of references [4], for instance).

Composition of strategies over alternating arenas is defined as for pre-strategies, except that the Player/Opponent labellings on arenas of the form $(A \multimap B) \multimap C$ are ignored (necessarily, as there is no coherent way to label the moves in the B component). The labelling is restored by inference from the 'alternation of justification': i.e. initial moves of $s{\restriction}A, C$ are Opponent moves, and the polarity of any other move is the complement of its justifier.

Proposition 3.5.5 *Alternation of even-length sequences is a well defined rule, and the category of alternating games and strategies is well-defined.*

Proof: Prefix closure and invariance under isomorphism is obvious. For any alternating arena A, every justified sequence in $\mathtt{id}_A$ is alternating, as each odd (Opponent) move is followed by a copy from the other component, which will have complementary polarity by the Player/Opponent inversion described above. To show associativity, first note that a sequence s of even-length is alternating if and only if for every even segment t (continuous subsequence) of s, the number of Opponent and Player moves is equal (write $O(t) = P(t)$).
Suppose $s \in ((A \multimap B) \multimap C) \multimap D$ is such that all of
$(s{\restriction}A \multimap B), (s{\restriction}B \multimap C), (s{\restriction}C \multimap D), (s{\restriction}A \multimap D)$ are alternating and even length. Then it is necessary to show that $s{\restriction}A \multimap C$ is alternating and even-length if and only if $s{\restriction}B \multimap D$ is (for which the two cases are entirely similar).
Proof is by induction on sequence length, so suppose every even segment of $s{\restriction}A \multimap C$ is alternating and even-length. Then by induction, every proper even segment of $s{\restriction}B \multimap D$ is alternating. So it is sufficient to show that
$O(s{\restriction}A, C) = P(s{\restriction}A, C)$ implies $O(s{\restriction}B \multimap D) = P(s{\restriction}B \multimap D)$.
But $O(s{\restriction}B \multimap D) = O(s{\restriction}A \multimap D) + O(s{\restriction}B \multimap C) - O(s{\restriction}A \multimap C)$
$= P(s{\restriction}A \multimap D) + P(s{\restriction}B \multimap C) - P(s{\restriction}A \multimap C) = P(s{\restriction}B \multimap D)$ as required. □

Note that the alternating version of the one-move arena, o, specifies the one move game, as there are no Player responses. This game will also be written o.

Strategies are further required to be *deterministic*, although there is clearly scope to study the semantics of concurrent languages by allowing non-determinism (this 'axis of the intensional hierarchy' is currently being explored by Harmer [34]).

Definition 3.5.6 (Deterministic Strategies) *A strategy is deterministic if*

$$sa, sb \in \sigma \Rightarrow a = b.$$

Thus a deterministic strategy on G is determined by a partial function from odd-length sequences of moves in G (situations with player to move) to justified moves. As expected, the identity strategy (and all other copycats), are deterministic strategies, and determinacy is preserved by composition.

It has now been established that the category of of alternating games and thread-independent deterministic strategies is a pointed CCC (with the one-move game as answer object), and so this specifies call-by-name and call-by-value models of control, via the families construction and the monad of o-continuations.

Proposition 3.5.7 (G_{alt}, o) *specifies a continuous computational model of control.*

PROOF: $\mathcal{G}_{alt}$ is cpo-enriched with the inclusion ordering on strategies as sets of plays, which is complete, as for any chain $\{\sigma_i : A \mid i \in \omega\}$ (or directed set), $\bigcup_{i \in \omega} \sigma_i$ is a thread-independent strategy. Moreover, composition in the cartesian closed category is continuous with respect to this ordering, because:

- composition by 'parallel composition with hiding' in the symmetric monoidal category is clearly continuous,

- the promotion operation for the co-monad ! preserves least upper bounds: $\sigma^\dagger : !A \multimap !B$ is defined by the same function from single threads to moves as $\sigma : !A \multimap B$ and hence for any chain $\bigcup_{i \in \omega} \tau_i^\dagger = (\bigcup_{i \in \omega} \tau_i)^\dagger$.

McCusker shows in [59] that recursive domain equations can be solved in categories of games such as $\mathcal{G}_{alt}$. In particular, $\mathcal{G}$ has ω-indexed products as a particular case of the generalized product (Definition 3.2.4).

Definition 3.5.8 *Given any arena A, define the arena $A^\omega = \Pi_{i \in \omega} A$.*
For any i, there are obvious (partial) structure-preserving maps
$E_i : A^i \to A^\omega, P_i : A^\omega \to A^i$ *such that* $E_i; P_i = ID_{A^i}$ *and* $\bigsqcup_{i \in \omega} P_i; E_i = ID_{A^\omega}$.
So the embedding and projection pairs for the countable product are given by $e_i : A = \mathtt{copy}_{F,G}$, *and* $p_i = \mathtt{copy}_{G,F}$.

□

3.6 Views and innocence

Following Hyland and Ong, constraints on games and on strategies which reflect functional behaviour by restricting access to the history of the computation can now be described. The Player and Opponent *views* of a justified play give a notion of 'relevant history' which proves important for understanding several aspects of the intensional hierarchy. They are related to the restriction operation which projects the current 'thread' of play, and defines the thread-independent strategies, but in quite a subtle way.

Definition 3.6.1 (Player and Opponent views.)

The Player view of a finite, justified alternating sequence s, is written $\ulcorner s \urcorner$ and defined inductively on the length of s, as follows

$\ulcorner \varepsilon \urcorner = \varepsilon$.

$\ulcorner sa \urcorner = \ulcorner s \urcorner a$, if a is a P-move.

$\ulcorner sa \urcorner = a$, if $* \vdash a$.

$\ulcorner sa \cdot tb \urcorner = \ulcorner s \urcorner ab$, if b is an O-move justified by a.

There is a dual notion of Opponent view, $\llcorner s \lrcorner$:

$\llcorner \varepsilon \lrcorner = \varepsilon$.

$\llcorner sa \lrcorner = \llcorner s \lrcorner a$, if a is an O-move.

$\llcorner sa \cdot tb \lrcorner = \llcorner s \lrcorner ab$, if b is a P-move justified by a.

In order for the (Player or Opponent) view of a justified sequence to be a justified sequence itself, he must respect the constraint of *visibility*.

Definition 3.6.2 (Visibility) *A justified sequence sa obeys the visibility rule V, if for every occurrence a in s, the unique move in s justifying a is in the view (of whoever is making the move a).*

$V(sa) \iff V(s) \wedge sa \text{ even} \implies \phi_s(a) \in \mathrm{Occ}(\ulcorner \mathrm{s} \urcorner) \wedge \mathrm{sa\ odd} \implies \phi_\mathrm{s}(\mathrm{a}) \in \mathrm{Occ}(\llcorner \mathrm{s} \lrcorner)$.

As noted in the previous section, games without the visibility condition, and thread-independent strategies, form a cpo-enriched CCC, and hence a basis for models of PCF. Moreover, Abramsky, Honda and McCusker showed in [4] (in essence) that as well as PCF, higher-type references can be interpreted in this category. This built on the work of [2] which gave semantics of ground-type assignments by dropping the condition of innocence on strategies (see Section 3.6.1). As visibility can be considered as a rule applying only to Player strategies (which is sufficient for Player views to be justified sequences) the innocent strategies

satisfying this rule form the purely functional core of this model, which can be shown to be finitely definable using the results for PCF on its own. There are *factorization* results showing that all thread-independent strategies can be written as the composition of one obeying visibility (and innocence), and some canonical representation of storage cells. These results were obtained in the presence of an additional rule, — the 'bracketing condition', but they extend naturally to control models as well. Anticipating the definability result for the innocent games model of μPCF, the following working hypothesis can be given:
the models of control generated from the CCC of thread-independent strategies, and the one move game are fully abstract for μPCF with higher-order references. The work that remains to be done to prove this concerns soundness and adequacy. The model without visibility is not yet well understood; the wild behaviour of strategies can be counterintuitive. With the the addition of control operators, things become even more complicated, one can use references and **call/cc** to store continuations and invoke them in unusual places.

The visibility rule is assumed to apply to all plays of the games considered henceforth (these are the 'legal' sequences, as in [43] etc.). This, however, is the only rule strictly required for the initial games model of control.

Definition 3.6.3 *An 'unbracketed' game is one generated from an alternating arena by the rule visibility alone. (So called because the major difference from the games of Hyland-Ong and McCusker is the absence of a bracketing condition.)*

The following results about visibility appeared in [43].

Proposition 3.6.4 *Every Player and Opponent view of a sequence satisfying visibility is a well-formed justified sequence.*

PROOF: is straightforward by induction on sequence length. $\square$

Closure under views can thus be added retrospectively as a well-definedness criterion for rules (it is satisfied by linearity and alternation). The following properties of plays satisfying visibility appear previous work such as [5],[6], [59] (often in *definitions* of games), going some way to explaining the significance of this rule as a measure of 'good behaviour'.

Proposition 3.6.5 (Switching condition) *if $sab \in V(A \otimes B)$ is an even-length sequence then $a \sqsubseteq M_A$ implies $b \sqsubseteq M_A$ and $a \sqsubseteq M_B$ implies $b \sqsubseteq M_B$. (Similarly if $sab \in V(A \multimap B)$ is an odd-length sequence such that b is in the same thread as a (or B is well-opened), then $a \sqsubseteq M_A$ implies $b \sqsubseteq M_A$ and $a \sqsubseteq M_B$ implies $b \sqsubseteq M_B$.)*

PROOF: is by induction on sequence length, showing that if Opponent's last move in $s \in V(A \otimes B)$ is in A, then $\ulcorner s \urcorner$ is wholly in A, so Player's next move must be in A, and if Opponent's last move is in B then $\ulcorner s \urcorner$ is wholly in B. $\Box$

Proposition 3.6.6 (Projection condition) *For unbracketed games* A, B
$A \otimes B = \{s \in J_{A\otimes B} \mid s{\restriction}A \in A \wedge s{\restriction}B \in B\}$,
$A \multimap B = \{s \in J_{A\multimap B} \mid s{\restriction}A \in A \wedge s{\restriction}B \in B\}$,
$!A = \{s \in J_{!A} \mid \forall a \ (initial) \ s{\restriction}a \in A\}$.

PROOF: is in [43] (in essence). It is also a corollary of Lemma 5.4.9, in which it is shown that the projection condition holds for plays which are *innocent*. $\Box$

Each additional rule R preserves the projection condition provided that for $s \in V(A \multimap B)$, $R(s)$ implies $R(s{\restriction}A)$ and $R(s{\restriction}B)$.

Proposition 3.6.7 *The unbracketed games form a category.*

PROOF: The identity strategy always obeys the visibility condition with respect to Player moves:
If $sab \in \mathtt{id}_A$ then b is a copy of a, and is justified by the move preceding the justifier of a, (of which the latter is a copy). This is the penultimate Opponent move in the view. Associativity is a consequence of the fact that the visibility rule is compositional (i.e. the composition of strategies satisfying the visibility condition satisfies the visibility condition.) The is proved in depth in [59] following [43]. It is a corollary of the fact that innocent strategies compose, which is established in there and indirectly in Section 3.7. $\Box$

3.6.1 Innocent functions and strategies

The cartesian closed category of well-opened games and strategies was constructed by placing a linearity constraint on (initial) Opponent moves. Composition of well-opened strategies was defined using a *projection* from games without this constraint (restriction to the current thread). The projection also defined (as its image) a cartesian closed subcategory of the well-opened games, — the thread-independent strategies.

In order to identify the *functionally definable* part of this CCC, corresponding constraints and projections are applied to *all* of the Opponent moves and threads in a game. This defines the innocent functions, — strategies which have responses only to sequences in which Opponent plays linearly, using the Player *view* to project less restricted sequences into these games. The image of this projection consists of the *innocent strategies* [43].

Definition 3.6.8 *An innocent function on an arena A is an even-prefix-closed and evenly branching set of Player views of plays in A. This definition can be expressed in another way, by saying that an innocent function is a deterministic strategy on the game obtained by labelling all of the* Opponent *moves in A as linear. The views act as a projection from sequences in A to sequences in the 'Opponent-linear' game with the same arena.*

Each innocent function $\chi : A$ defines a strategy on the unbracketed game A by composition with the view-function from sequences to views. Or this can be written as a set of traces.

Definition 3.6.9 *For an innocent function χ, define the strategy σ_χ as the inductive closure of the following definition (assuming sequences are alternating and justified):*
$\varepsilon \in \sigma_\chi$,
$s \in \sigma_\chi \wedge \ulcorner sab \urcorner \in \chi \implies sab \in \sigma_\chi$
(So σ_χ satisfies visibility.)

The image of this embedding consists of the 'innocent strategies'.

Definition 3.6.10 *A strategy is innocent if its moves are determined only by the view of the game so far: i.e.*

$$sab, t \in \sigma \wedge \ulcorner sa \urcorner = \ulcorner ta \urcorner \Rightarrow tab \in \sigma.$$

Note that this includes a certain 'liveness' condition which is redundant in the setting of unbracketed games; not only does a strategy have the same response to every view, if it ever has a response to a given view, that response is always valid, hence.

Proposition 3.6.11 *An innocent function χ specifies an innocent strategy on a game G if $s \cap G$ is even-length for all $s \in \sigma_\chi$.*

Definition 3.6.12 *For any strategy $\sigma : A$, let $\ulcorner \sigma \urcorner = \{\ulcorner s \urcorner \mid s \in \sigma\}$, so that $\ulcorner \sigma \urcorner$ is an innocent function on A if σ is an innocent strategy on $R(A)$.*
Define the composition of innocent functions: $\chi : A \to B$ and $\psi : B \to C$:
$\chi; \psi = \ulcorner \sigma_\chi; \sigma_\psi \urcorner$.

For this definition to make sense, it is to show that the composition of innocent strategies is innocent. This is a non-trivial fact which is verified (for well-opened, well-bracketed games) in [43], and [59], and, less directly, in the proof of full and faithful completeness in the next section. For the moment, assume that the composition of innocent, unbracketed strategies is innocent. So the unbracketed games and innocent strategies form a category, which will be written $\mathcal{G}$. The following proposition is a formalization of the remark made above, that views/innocence are a generalization of thread projection/thread-independence. This an important part of the proof of definability.

Proposition 3.6.13 (Bang lemma) *For any innocent strategy $\sigma : A$,*

$$\chi_{!\sigma} = \chi_{\sigma}.$$

Proof: is by noting that only (part of) the current thread is visible to Player, i.e. the Player view of $s \in P_{!A \multimap !B}$ is the same as the Player view of $s \upharpoonright a \in P_{A \multimap B}$, where a is the occurrence of an initial move in B hereditarily justifying the last move in s. □

It is a useful additional property of views that by concealing Opponent's bad behaviour, they can be used to project strategies over games governed by rules into unbracketed ones, thus embedding models of purely functional languages, for instance, into models with non-local control. (This remains a problem for knowing (i.e. non-innocent) strategies: one can, for instance, use state to define a strategy which detects manipulation of the control flow by the Opponent, but without using local control flow itself.)

Definition 3.6.14 *A rule is* embeddable *if it obeys the projection condition, and for every justified sequence on $A \multimap B$ such that $R(s \upharpoonright A)$ and $R(s \upharpoonright B)$, but s itself violates R, $s \cap R$ is odd-length. In other words, if a play on $A \multimap B$ violates R, but its restriction to each component does not, then the fault is Player's.*

(So linearity, for instance, is embeddable.)

Proposition 3.6.15 *For a category of (visibility satisfying) games and innocent strategies $\mathcal{G}_R$, based on a compositional, embeddable rule R,*

$$R(A) \longrightarrow V(A).$$

$$\tau \Longrightarrow \sigma_{\ulcorner \tau \urcorner}$$

defines a faithful functor into the category of unbracketed games and innocent functions.

PROOF: Given $\rho : R(A) \to R(B), \tau : R(B) \to R(C)$, it is necessary to show that $\sigma_{\ulcorner\rho\urcorner}; \sigma_{\ulcorner\tau\urcorner} = \sigma_{\ulcorner\sigma;\tau\urcorner}$.
The inclusion from right to left is straightforward.
From left to right it is sufficient to show that as long as Opponent adheres to the rule R on $\sigma_{\ulcorner\rho\urcorner}; \sigma_{\ulcorner\tau\urcorner}$, then so does Player. So suppose $sab \in \sigma_{\ulcorner\rho\urcorner} \| \sigma_{\ulcorner\tau\urcorner}$ is such that b is a Player move in A or C, and $sa{\upharpoonright}A, C$ obeys R. It is sufficient to show that this entails that $R(sa{\upharpoonright}A, B)$ and $R(sa{\upharpoonright}B, C)$. Then $sa{\upharpoonright}A, B \in \sigma$ and $sa{\upharpoonright}B, C \in \tau$ and so $sab{\upharpoonright}A, C \in \sigma; \tau$ as required.
By the projection condition, $R(sa \upharpoonright A, C)$ implies that $R(sa \upharpoonright A)$ and $R(sa \upharpoonright C)$. Hence also $R(sa{\upharpoonright}B)$, as otherwise σ or τ violate R. By definition of an embeddable rule, if $sa{\upharpoonright}A, B$ violates R, then the first move to do so is a Player move, which is not possible, as σ obeys R, and similarly, $sa{\upharpoonright}B, C$ obeys R. □

Remark 3.6.16 *A consequence of the above observation is that the set of views of an innocent strategy is sufficient to determine whether it originates from a game satisfying the rule R. A natural question follows: is there a simple* characterization *of innocent R-strategies in terms of their view-functions, allowing the relevant subcategory to be identified. In the cases of visibility (Player views are justified sequences) and the well-bracketing rule which will be introduced (Player views are well-bracketed), the answer is yes.*

3.7 The pointed CCC of games and innocent strategies is initial

Rather than showing separately that unbracketed games and finite innocent strategies form a category, that it is cartesian closed, and finally that it is a full and faithfully complete model of $\Lambda(\Omega)$, these facts are established together, by identifying it as the image of the functor from the free partial cartesian closed category over a single object. This is an economy of space; the direct proof that innocent strategies compose, which can be found in [43], and [59] is detailed but not especially illuminating. It does not seem inappropriate to establish soundness and completeness jointly; the innocent strategies can reasonably be defined as the image of the initial functor, and their composition can be described accordingly. (The fact that there is a well-defined notion of composition on Böhm trees, in a sense the syntactic counterpart of innocent functions, has been shown by Curien and Herbelin [20].) In addition, the isomorphism of the unbracketed model to the free CCC is quite a general result in that it characterizes innocent composition in the whole category of finite games and strategies.

Proposition 3.7.1 *The category of finite unbracketed games is isomorphic to the (pointed) cartesian closed category of games freely generated from the one-move game o. (With $\perp$-maps given by the empty strategy in each case).*

Proof: Proof is by induction on depth of the game tree; i.e. the longest justification sequence. Suppose G is a finite, non-empty unbracketed game. Then by Lemma 3.3.8, G is generated by an arena which is a forest. So G is generated by some arena $A_1 \times A_2 \times \ldots A_n$, where $A_1, A_2, \ldots A_n$ are trees, and so each A_i is equal to $B_i \Rightarrow o$, for some forest-arena B_i of strictly smaller depth, and so by hypothesis, B_i can be constructed form o with $\times$ and $\Rightarrow$, hence A can as well. $\Box$

Hence the unique functor from the free pointed CCC over a single object to the pointed CCC of finite games and strategies is surjective on objects. The image of the functor is identified by proving the following theorem.

Theorem 3.7.2 *The image of the unique functor from the free partial CCC to the category of games and strategies consists of precisely the finite, innocent functions.*

Corollary 3.7.3 *The innocent functions form a pointed CCC, of which the finitary part is a fully and faithfully complete model of $\Lambda(\Omega)$.*

3.7.1 Characterizing the initial model

The fact that the free pointed CCC is a fully complete model of the simply-typed λ-calculus with non-termination defined in the previous chapter allows the functor from it into the games model to be defined by inductive decomposition as follows.

Definition 3.7.4 (Evaluation trees of $\Lambda(\Omega)$) *The 'evaluation trees' of $\Lambda(\Omega)$ are familiar as $\beta\eta$-long normal forms. Writing $t \in N(\Gamma;T)$ for 't is an evaluation tree of type T over context Γ', they can be defined as the closure of the following inductive definition (products and the terminal type are omitted):*

$$\frac{}{\Omega \in N(\Gamma;\iota)}$$

$$\frac{t_i \in N(\Gamma, y : \overline{T} \Rightarrow \iota; T_i) : i \leq n}{(((y\ t_1)\ t_2)\ \ldots)\ t_n \in N(\Gamma, y : \overline{T} \Rightarrow \iota; \iota)}$$

$$\frac{t \in N(\Gamma, x : S; T)}{\lambda x.t \in N(\Gamma; S \Rightarrow T)}$$

Proposition 3.7.5 *Every finitary term of $\Lambda(\Omega)$ is $\beta\eta$-equivalent to a finite evaluation tree.*

PROOF: for the simply-typed λ-calculus without constants is in [8]; this extends readily to $\Lambda(\Omega)$ by considering Ω as an unbound, ground-type variable. $\Box$

Corollary 3.7.6 *A model $(\mathcal{C}, \mathbf{a})$ of $\Lambda(\Omega)$ is fully and faithfully complete if and only if every map $f : \overline{A} \rightarrow \mathbf{a}$ has a unique decomposition in the following sense: either $f = \perp_{\overline{A},\mathbf{a}}$ or*
$f = (\langle \pi_i, \langle [\![M_1]\!]_{\mathcal{M}}, [\![M_2]\!]_{\mathcal{M}}, \ldots [\![M_n]\!]_{\mathcal{M}} \rangle \rangle; \mathrm{App})$
for some evaluation trees $M_i \in E(\Gamma; A_i) : i \leq n$.

Definition 3.7.7 *The unique functor from the initial model to $(\mathcal{C}, \mathbf{a})$ can therefore be defined on objects as $F([\![\iota]\!]) = \mathbf{a}$, and $F([\![S \Rightarrow T]\!]) = F([\![A]\!]) \Rightarrow F([\![B]\!])$, and on morphisms:*
$F(\perp_{[\![\iota]\!]}) = \perp_{\mathbf{a}}$,
$F(\langle \pi_i, \langle g_1, g_2, \ldots g_n \rangle \rangle; \mathrm{App}) = \langle \pi_i, \langle F(g_1), F(g_2), \ldots F(g_n) \rangle \rangle; \mathrm{App}.$
$F(\Lambda(g)) = \Lambda(F(g))$.

Thus to identify the fully complete model inside $\mathcal{C}$ it is sufficient to find a set of morphisms $\mathcal{I}(A, B) \subseteq \mathcal{C}(A, B)$ at each A, B such that for each
$A = \Pi_{i \leq n}(\Sigma_{j \leq m_i} B_{i,j})$

$$\mathcal{I}(A, \mathbf{a}) \cong (\coprod_{i \in I} \Pi_{j \in J} \mathcal{C}(A \times B_{i,j}, \mathbf{a}))_*$$

(writing $\Sigma_{j \leq m_i} B_{i,j}$ for the weak co-product $(\Pi_{i \leq m_i}(B_{ij} \Rightarrow \mathbf{a})) \Rightarrow \mathbf{a}$ and $\coprod$ for disjoint union of sets).
The isomorphism is given by

$$\langle f, i \rangle \longrightarrow \langle \pi_i, \langle g_{i1}, g_2, \ldots g_{in_i} \rangle \rangle; \mathrm{App}.$$

Moreover this must be a *least* solution to the above recursive equation so (following Abramsky [1]) a norm function $\sharp_{A,B} : \mathcal{I}(A, B) \rightarrow \mathbb{N}$ is required such that

$$\sharp(\langle \pi_i, \langle g_1, g_2, \ldots g_n \rangle \rangle; \mathrm{App}) > \sharp(g_i) \quad i \leq n.$$

As this identifies $\mathcal{I}$ as the image of a structure preserving functor into $\mathcal{C}$, soundness is automatic.

Proposition 3.7.8 *If $\mathcal{C}$ is a pointed CCC, with a normed set of morphisms $\mathcal{I}$ satisfying the above criteria, then the subcategory of $\mathcal{C}$ with objects freely generated from $\mathbf{a}$, and morphisms in $\mathcal{I}$ is isomorphic to the free pointed CCC.*

PROOF: is via Corollary 3.7.6 above, by induction on norms;
if $f \in \mathcal{I}(\overline{A}, \mathbf{a}) \neq \perp$, then $f = \langle \pi_i, \langle g_1, g_2, \ldots g_n \rangle \rangle; \mathrm{App}$, for some i, and $g_1, \ldots g_n$ such that $\sharp(f) > g_j$ for each j, so by induction each $g_j = [\![M_j]\!]$ for some evaluation tree M_j. $\Box$

3.7.2 Axioms for denotational completeness in models of control

It is now possible to present an axiomatic characterization of denotational models of $\Lambda(\Omega)$ which are fully and faithfully complete, being (least) solutions to a recursive equation on hom-sets. This gives a relatively elegant and economical proof of definability for the games model, and by identifying precisely the assumptions on which it is based, could allow the results achieved in this thesis to be generalised to new models of control (using games, or some other presentation). Well chosen axioms may even help to identify more elegant constructions as they contain implicitly an abstract characterization of *the* initial model of control.

This approach was initiated by Abramsky in 'Axioms for full abstraction and full completeness' [1]. This axiomatised definability results for models of (call-by-name) PCF, and full (and faithful) completeness for models of the pure simply typed λ-calculus over a single base type (i.e. the free CCC over a single object) ('sequential' and 'pure sequential' categories, in his terminology). The axioms given here, and the resulting definition, are somewhere between the two, incorporating the partiality structure of one with the simple structure at ground type (no values) of the other. In fact this allows the axiomatization to be simplified slightly. The major extension in their scope, however, is that via the (fully complete) cps translations, they can be regarded as 'axioms for definability in models of control'. Notwithstanding the indirectness of the interpretation, the axioms themselves are directly relevant to models of control; as will be shown, they characterize the answer object as minimal, continuations as π-atomic, and the underlying category as sequential.

It is also necessary to state the axioms differently, as soundness of the models was assumed in 'Axioms', whilst here it is simply assumed that the axiomatized maps form a subset of each hom-set of a pointed CCC. Refining the axioms in this way makes sense in the context of the intensional hierarchy, as it is consistent with the aim of identifying fully complete models of functional languages as subcategories of more general categories with fewer constraints.

The axioms assume the following structure on a pointed cartesian closed category $\mathcal{C}$:

- a linear decomposition into a symmetric monoidal closed category $\mathcal{L}$ with a co-monad ! such that $\mathcal{C}(A, B) = \mathcal{L}(!A, B)$, and $A \Rightarrow B = !A \multimap B$.

- proposed *definable morphisms* $\mathcal{I}$ in $\mathcal{L}$. These are given as a specified, normed subset $\mathcal{I}(A, B)$ of each hom-set of $\mathcal{L}$, containing the identity, ter-

minal and bottom maps, and preserving products
(i.e. $\mathcal{I}(A, B \times C) = \{\langle f, g\rangle \mid f \in \mathcal{I}(A, B), g \in \mathcal{I}(A, C)\}$).
All further requirements for $\mathcal{I}$ are contained in the following axioms (it need not be assumed to be a category, for instance).

Recall also the notion of strict map; morphisms $f : A \to B$ such that$\perp_A; f = \perp_B$, and the subcategory $\mathcal{C}_S$ with the objects of $\mathcal{C}$, and strict maps as morphisms between them. In the various categories of games, the (non-$\perp$) strict strategies from A to B are those which follow the initial move with a move in A.

The axioms have a uniform flavour — each takes the form of a statement that a canonical mapping between hom-sets of $\mathcal{L}$ is a bijection on the restriction to maps in $\mathcal{I}$ which preserves or reduces their norms. Hence they are presented as an isomorphism between sets of $\mathcal{I}$-morphisms, although the additional structural details are crucial. (The first three characterize the answer object, and the latter two are more general lemmas about sequentiality which have appeared in some form in the decompositions of [6], [43], [59].) The intended interpretation of $\mathcal{I}$ in the category of games is the innocent strategies. Although the 'linear decomposition' for $\mathcal{G}$ described in Section 3.4 is not standard, the properties required by the following axioms are present, so the proof goes through for $\mathcal{G}$.

3.7.2.1 Minimality of the answer-object

$$\mathcal{I}(\mathbf{1}, \mathbf{a}) = \{\perp_{\mathbf{a}}\}.$$

Anticipating the use of $\mathbf{a}$ as an 'answer object' for a continuations monad, this condition can be seen as a necessary condition for preventing 'junk' from being introduced into the semantics by a choice of answer object which is 'too large', since an interpretation of a term over $(A \multimap \mathbf{a}) \multimap \mathbf{a}$
must either be $\perp$ or the lifting of an element of A. An answer object which includes other possibilities, such as the choice of the flat domain of natural numbers as an answer object in [80] violates this axiom.

Lemma 3.7.9 *It is equivalent to the following condition from [1]:*
All maps into the answer object are strict:
for any A, $\mathcal{I}_S(A, \mathbf{a}) = \mathcal{I}(A, \mathbf{a})$.

PROOF: For a non-strict map into $\mathbf{a}$ to exist, there must be a non-$\perp$ map from $\mathbf{1}$ to $\mathbf{a}$, whilst if all maps into $\mathbf{a}$ are strict, in particular, if $f : \mathbf{1} \to \mathbf{a}$, is strict, then $\perp_{\mathbf{1}}; f = \mathtt{id}_{\mathbf{1}}; f = \perp_{\mathbf{1}}$. □

This axiom can in certain circumstances be subsumed into the following property.

3.7.2.2 Contravariance of (linear) continuations

$$\mathcal{I}_s(A \multimap \mathbf{a}, B \multimap \mathbf{a}) \cong \mathcal{I}(B, A)_*.$$

More precisely, there is an inclusion of maps of $\mathcal{L}(B, A)_*$ in $\mathcal{L}_s(A \multimap \mathbf{a}, B \multimap \mathbf{a})$:

$$\begin{array}{rl} * \longrightarrow & \perp_{A \multimap \mathbf{a}, B \multimap \mathbf{a}} \\ (f : B \to A) & \longrightarrow \Lambda(\mathtt{id}_{A \multimap \mathbf{a}} \otimes f; \mathrm{App}), \end{array}$$

and the axiom stipulates that:

- its restriction to the maps in $\mathcal{I}$ is an isomorphism,
- it is strictly increasing with respect to the norm function (i.e. $\sharp(f : B \to A) < \sharp(\Lambda(\mathtt{id}_{A \multimap \perp} \otimes f; \mathrm{App}))$)

(In $\mathcal{G}$, any strict, non-bottom strategy from $A \multimap \mathbf{a}$ to $B \multimap \mathbf{a}$ responds to the initial move in $B \multimap \mathbf{a}$ by playing the initial move in $A \multimap \mathbf{a}$, Opponent plays an initial move in A, henceforth Player must play according to a (smaller, innocent) strategy on $B \multimap A$.)

This axiom is essentially a form of Abramsky's 'linear functional extensionality'[1]. In a control setting it can be seen as stipulating that a (non-$\perp$) morphism between linear continuations is always in the image of a the functor $\_ \multimap \mathbf{a}$. In the situation in which $\mathcal{I}$ itself is given to be a SMCC, however, it also characterises $\mathbf{a}$ as a minimal 'standard datatype', making the first axiom redundant.

Proposition 3.7.10 *If the contravariance axiom holds for the SMCC $\mathcal{L}$, then it entails minimality of $\mathbf{a}$, and hence that all maps into $\mathbf{a}$ are strict.*

Proof: $\mathbf{a}$ and $\mathbf{1} \multimap \mathbf{a}$ are still isomorphic in $\mathcal{L}_s$, as all isomorphisms are strict maps, and so by the contravariance axiom

$\mathcal{L}_s(\mathbf{a}, \mathbf{a}) = \mathcal{L}_s(\mathbf{1} \multimap \mathbf{a}, \mathbf{1} \multimap \mathbf{a}) \cong \mathcal{L}(\mathbf{1}, \mathbf{1})_*$. Therefore there are just two distinct maps in $\mathcal{L}_s(\mathbf{a}, \mathbf{a})$, $\mathtt{id}_{\mathbf{a}}$ and $\perp_{\mathbf{a}}$ (these are distinct, as otherwise

$\perp_{\mathbf{a}} : \mathbf{1} \to \mathbf{a}$ is an isomorphism, and $\mathbf{a}$ is terminal, contradicting the fact that there are two maps from $\mathbf{a}$ to itself).

Hence if $f : \mathbf{1} \to \mathbf{a}$, $t_{\mathbf{a}}; f = \perp_{\mathbf{a},\mathbf{a}}$, (where t_A is the terminal map from $\mathbf{a}$) as f is not an isomorphism either.

So $\perp_{\mathbf{a}} = \perp_{\mathbf{a}}; \perp_{\mathbf{a},\mathbf{a}} = (\perp_{\mathbf{a}}; t_{\mathbf{a}}); f = \mathtt{id}_{\mathbf{1}}; f = f$ as required. $\square$

3.7.2.3 π-atomicity of continuations

$$\mathcal{I}_S(B, A \multimap \mathbf{a}) + \mathcal{I}_S(C, A \multimap \mathbf{a}) \cong \mathcal{I}_S(B \times C, A \multimap \mathbf{a}).$$

Specifically, there is a map into $\mathcal{L}_S(B \times C, (A \multimap \mathbf{a}))$ from the coalesced sum $\mathcal{L}_S(B, (A \multimap \mathbf{a})) + \mathcal{L}_S(C, (A \multimap \mathbf{a}))$, which sends $f : B \to A \multimap \mathbf{a}$ to $\pi_l; f$ and $f : C \to A \multimap \mathbf{a}$ to $\pi_r; f$. For morphisms of $\mathcal{I}$, this is required to be a bijection such that $\sharp(f) = \sharp(\pi_i; f_i)$

This property of (target) objects is called π-atomicity in [1], following Joyal. In $\mathcal{G}$, all objects $A \multimap o$ are π-atomic, because they have a unique initial move. So the first Player move in a strict strategy on $B \times C \multimap (A \multimap o)$ is always in B or C, and by definition of the cartesian product, the remainder of the play is a play in $B \multimap (A \multimap o)$ or $C \multimap (A \multimap o)$

3.7.2.4 Linearization of head occurrence

$$\mathcal{I}_s(A, !A \multimap B) \cong \mathcal{I}_s(!A, B).$$

i.e. the canonical map from $\mathcal{L}_s(A, !A \multimap \mathbf{a})$ into $\mathcal{L}_s(!A, \mathbf{a})$:
$f \longrightarrow \mathtt{con_A}; ((\mathtt{f} \otimes \mathtt{der_A}); \mathrm{App})$ is a bijection on maps in $\mathcal{I}$, and
$\sharp(f) = \sharp(\mathtt{con_A}; ((\mathtt{f} \otimes \mathtt{der_A}); \mathrm{App}))$.
Linearization of head occurrence in a strict strategy $\sigma : !A \multimap B$ in $\mathcal{G}$ is simply a matter of relabelling the first opened thread in $!A$ as a play in an additional premise A, to yield $\sigma : A \multimap (!A \multimap B)$. This preserves innocence, as in any play according to a strict strategy on $!A \multimap B$, the initial move in the first thread opened in $!A$ appears in every subsequent Player view. (Compare with e.g. Danos, Herbelin and Regnier's notion of 'linear head reduction' [21].)

$$\begin{tikzcd}
!A \arrow[r, "\mathtt{con}"] \arrow[d, "f"'] & !A \otimes !A \arrow[d, "\mathtt{id} \otimes \mathtt{der}"] \\
B & !A \otimes A \arrow[l, "\hat{f}"]
\end{tikzcd}$$

Figure 3.1: Linearization of head occurrence ($f \longrightarrow \hat{f}$)

3.7.2.5 Uniformity of threads

$$\mathcal{I}(!A, \overline{(B \multimap \mathbf{a})}) \cong \mathcal{I}(!A, !\overline{(B \multimap \mathbf{a})}).$$

That is, the 'promotion' mapping from $\mathcal{L}(!A, \overline{(B \multimap \mathbf{a})})$ to $\mathcal{L}(!A, !\overline{(B \multimap \mathbf{a})})$ which sends $f : !A \to \overline{(B \multimap \mathbf{a})}$ to $f^\dagger : !A \multimap !\overline{(B \multimap \mathbf{a})} = \mathtt{prom}_A; !f$ is a bijection on maps in $\mathcal{I}$. Since by definition of promotion, $f^\dagger; \mathtt{der}_{\overline{(B\multimap \mathbf{a})}} = f$, this means that for all $g \in \mathcal{I}(!A, !\overline{(B \multimap \mathbf{a})})$, $(g; \mathtt{der}_{\overline{(B\multimap \mathbf{a})}})^\dagger = g$. The fact that this holds for all innocent strategies in $\mathcal{G}$ has already been noted as the 'Bang lemma' (Proposition 3.6.13).

Definition 3.7.11 *A partial sequential category is a pointed CCC, $\mathcal{C}$, together with a specified non-terminal object,* $\mathbf{a}$, *which decomposes into a pointed affine category $\mathcal{L}$.*

The category of AJM games and strategies (where $\mathcal{I}$ is the subset of history free strategies) provides another example of a sequential pointed CCC, — almost!

3.7.3 Axiomatic decomposition

Theorem 3.7.12 (Decomposition in a sequential, pointed CCC) *Let $(\mathcal{C}, \mathbf{a})$ be a category with a linear decomposition $\mathcal{L}$, and normed subsets of maps $\mathcal{I}$ satisfying the axioms above.*
If $A_1, A_2, \ldots A_n$ are objects of S such that $A_i = \Pi_{j\leq m_i} B_{i,j} \Rightarrow \mathbf{a} = !(\Pi_{j\leq m_i} B_{i,j}) \multimap \mathbf{a}$ then

$$\mathcal{I}(!\Pi_{i\leq n} A_i, \mathbf{a}) \cong (\Sigma_{i\leq n} \Pi_{j\leq m_i} \mathcal{I}(!(\overline{A}), B_{i,j}))_*.$$

Proof: (along similar lines to the decompositions in [1])
By Minimality of answer-object, or Proposition 3.7.10

$$\mathcal{I}(!\Pi_{i\leq n} A_i, \mathbf{a}) \cong \mathcal{I}_S(!\Pi_{i\leq n} A_i, \mathbf{a})$$

by Linearization of head occurrence, this is isomorphic to

$$\cong \quad \mathcal{I}_S(\Pi_{i\leq n}(!(\Pi_{j\leq m_i} B_{i,j}) \multimap \mathbf{a}), !(\overline{A}) \multimap \mathbf{a})$$

by π-atomicity of $(!(\overline{A}) \multimap \mathbf{a})$, to

$$\cong \quad (\Sigma_{i\leq n} \mathcal{I}(!(\Pi_{j\leq m_i} B_{i,j}) \multimap \mathbf{a}, !(\overline{A}) \multimap \mathbf{a}))$$

so by Contravariance of continuations, to

$$\cong \quad \Sigma_{i\leq n} \mathcal{I}(!(\overline{A}), !(\Pi_{j\leq m_i} B_{i,j}))_*$$

and hence by Uniformity of Threads, to

$$\cong \quad \Sigma_{i\leq n} \mathcal{I}(!(\overline{A}), \Pi_{j\leq m_i} B_{i,j})_*$$

and by definition of the product, to

$$\cong \quad \Sigma_{i\leq n} \Pi_{j\leq m_i} \mathcal{I}(!(\overline{A}), B_{i,j})_*$$

(Note also that the norm of each map is strictly greater than the greatest norm in its decomposition.)
Moreover, the inverse of each component of the isomorphism has been characterized as a canonical map in the linear category, so by composing these maps the decomposition can be given as follows:
For any $f \in \mathcal{I}(!\overline{A}, \mathbf{a})$, either $f = \perp_{\overline{A},\mathbf{a}}$ or there is a unique $i \in n$ and family of morphisms $\{g_j \in \mathcal{I}(\overline{A}, B_{i,j}) \mid j \leq m_i\}$ such that

$$f = \mathtt{con}_{\overline{\mathtt{A}}}; (\mathtt{der}_{\overline{\mathtt{A}}}; \pi_{\mathtt{i}}) \otimes \langle \mathtt{g}_1, \ldots \mathtt{g}_{\mathtt{j}}, \ldots \rangle^{\dagger}; \mathrm{App}$$

where $\sharp(f) < \mathtt{max}\{\sharp(\mathtt{g}_1), \ldots \sharp(\mathtt{g}_{\mathtt{n}})\}$. $\square$

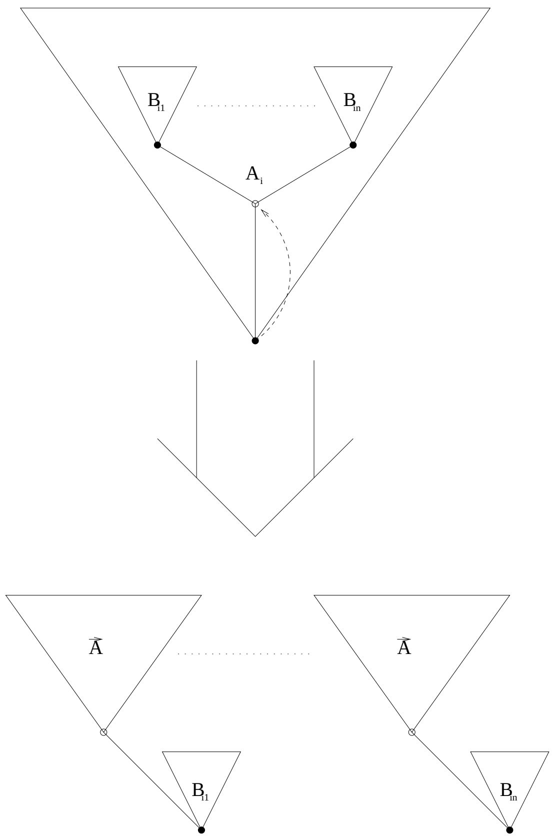

Figure 3.2: Decomposition of an innocent strategy on $\overline{A} \to o$

This simplifies to a decomposition in $\mathcal{C}$ as follows.

Corollary 3.7.13 *In any pointed sequential category with objects* $\overline{A} = \Pi_{i \leq n} A_i$ *where* $A_i = (\Pi_{j \leq m_i} B_{i,j} \Rightarrow \mathbf{a}) \Rightarrow \mathbf{a}$

$$\mathcal{I}(\overline{A}, \mathbf{a}) \equiv \Sigma_{i \leq n} \Pi_{j \in m_i} \mathcal{I}(\overline{A}, B_{i,j})$$

where the bijection is characterized

$$\langle f, i\rangle \longrightarrow \langle \pi_i, \langle g_1, \ldots g_{m_i}\rangle\rangle; \mathrm{App}$$

for unique $g_j \ : \ j \leq m_i$ *such that* $\sharp(g_j) < \sharp(f)$.
Hence the category which has objects generated form $\mathbf{a}$ *by* $\times$ *and* $\Rightarrow$*, and morphisms from* $\mathcal{I}$ *is isomorphic to the free CCC.*

Corollary 3.7.14 *A normed sequential partial CCC is isomorphic to the free partial CCC, and is a fully and faithfully complete model of* $\Lambda(\Omega)$.

Danos, Herbelin and Regnier [21] show that the correspondence with λ-terms extends to the interaction between strategies. They describe a process of evaluation of λ-terms to weak head-normal form by 'linear head reduction', and an abstract machine (due to Krivine) for implementing it. The pointers generated by the machine in the course of evaluating the application of one term to another are in bijective correspondence to the 'uncovering' of the interaction between the strategies which denote the terms.

3.8 The games models of control

The constructions described in the previous chapter can be applied to the categories of games and innocent and knowing strategies, to give games models of call-by-name and call-by-value control.

Proposition 3.8.1 *The models of control specified by* $(\mathcal{G}, o)$ *(by the strong monad of o-continuations on* $\mathsf{fam}(\mathcal{G})$*) are initial.*

PROOF: is by the proof of full completeness above, together with the proof in the previous chapter (Proposition 2.3.7) that any model of control given by $(\mathcal{C}, \mathbf{a})$ is initial if $\mathcal{C}$ is the free pointed CCC over $\mathbf{a}$. $\Box$

A significant point is that (unlike the abstract, categorical case) the completion of the games presentation of the initial model of control as a cpo-enriched 'continuous model of control' is now straightforward.

Proposition 3.8.2 $(\mathcal{G}_{alt}, o)$ *specifies a continuous computational model of control.*

PROOF: is as for the CCC of games and thread independent strategies (Proposition 3.5.7), noting that the least upper bound of a chain of innocent strategies is innocent. $\Box$

By Proposition 2.5.9, every definable morphism of a control model is a supremum of an ω-chain of morphisms from the CCC, and by the Definability Theorem, these are the innocent strategies with finite norms (view-functions). This generalises simply as follows:

Proposition 3.8.3 *$\mathcal{G}$ is ω-algebraic, with the compact strategies being precisely those with finite view-function.*

Thus the denotational part of the definability problem has been solved; all that remains is to give a complete syntactic and operational characterization of models of control. However, the non-reductionist project of characterizing control within the intensional hierarchy remains. The notion of 'weak co-product' monads on $\mathsf{Fam}(\mathcal{G})$ which yield call-by-name and call-by-value semantics provides a convenient structuring principle for this problem. Three such monads will be defined, each of which gives a different level of access to control flow. These can be characterized as higher-order control, first-order control, and strictly local. The can be derived from the 'linear decomposition' of the category of games which has already been described, based (albeit with various degeneracies) on the translations of intuitionistic and classical types into linear logic described in Figure 3.3 (to which there are several alternatives; [29], [22]). The obstacle to

Intuitionistic (call-by-name)	Intuitionistic (call-by-value)	Classical (call-by-name)	Classical (call-by-value)
$A \times B = A \& B$	$A \times B = !(A \& B)$	$A \times B = A \& B$	$A \times B = ?!A \& ?!B)$
$A \Rightarrow B = !A \multimap B$	$A \Rightarrow B = !A \multimap !B$	$A \Rightarrow B = !A \multimap B$	$A \Rightarrow B = !A \multimap ?!B$
$A + B = !A \oplus !B$	$A + B = A \oplus B$	$A + B = ?(!A \oplus !B)$	$A + B = A \oplus B$

Figure 3.3: Linear decompositions of call-by-name and call-by-value types ($\oplus$ is the dual to $\&$).

defining sums as dual to products in the category of alternating games, however, is its 'asymmetry'; Opponent always starts, so the true involution operator which just swaps the roles of Player and Opponent does not give a well-defined game. (There are ways to allow Player to start — see [7], but in this case the possibility of modelling the additives (cartesian closure) is lost.) However, true involution is in this case not necessary (or desirable, being incompatible with the pointedness which is a crucial feature of the weak co-product monad). Instead, the one-move game (which is a weakly initial object) is used as a 'weakly dualizing object' which both *lifts* and *dualizes* at the same time,

defining $(A)^{\perp} = A \multimap o$.
(This operation, forming *linear* continuations, — is a self-adjoint 'not-functor' [82] on the SMCC of games (see Proposition 3.8.7).)
Thus the 'intuitionistic' sum-monad is modelled by $((!A \multimap o) \times (!B \multimap o)) \multimap o$, and the 'classical' sum-monad by $!((!A \multimap o) \times (!B \multimap o)) \multimap o$, which is equal to $((A \Rightarrow o) \times (A \Rightarrow o)) \Rightarrow o$ — the sum given by the continuations monad. Recalling the definition of the ! as the lifting of the linearity condition on initial moves, the difference between the classical and intuitionistic sum games is therefore the following:
in the intuitionistic sum, the move justified by the initial move is linear, in the classical sum, it is not.

In the following section, these alternative notions of weak co-product and the associated monads are studied. The linearity conditions placed on the second move in the monad are shown to correspond to different types of control behaviour, in particular:

- there is a finer distinction between sum types, based on refining the linearity rule, to exclude control behaviour corresponding to 'logical stability'.
- the 'bracketing condition' of [43], [6] etc. corresponds to a stricter version of the linearity rule, which enforces local control flow.

3.8.1 The unbracketed sum

First, consider the monad of 'one-move continuations' $\mathbf{T}X = (X \Rightarrow o) \Rightarrow o$, and the associated 'weak co-product', $A + B = ((A \Rightarrow o) \times (B \Rightarrow o)) \Rightarrow o$
The natural transformation $\mu_T : \mathbf{T}^2 \rightarrow \mathbf{T}$ can itself be used as the denotation of the control operator $\mathcal{C}$ (see Chapter 4). However, the games interpretation of the **call/cc** operator can be given a more intuitive description. (Recall that this is typed with $(((S \Rightarrow T) \Rightarrow S) \Rightarrow S)$ which is the simplest type which can be given to control operators but not (generally) to functions; it is also the simplest example of a classical (minimal) tautology which is not provable in intuitionistic logic.)

Definition 3.8.4 *Let $A = \Sigma_{i \in I} A_i$, then for arbitrary B,*

$$\mathtt{peirce}^n_A : ((A \Rightarrow B) \Rightarrow A) \Rightarrow A)$$

(Figure 3.4) is the strategy which plays copycat between the outermost occurrences of A, until forced into $(A \Rightarrow B)$ by a move in B, after which it plays the opening

move in the innermost occurrence of A and then copycat between the outermost A, and both of the other A components.
For call-by-name type-objects other than weak co-products, the denotation corresponding of **call/cc** *can be derived from these instances via cartesian closure, just as the operator itself can be 'bootstrapped' from instances at sum-types in the λ-calculus.*
The call-by value version can be described directly for all families of games: for a family $A = \{A_i \mid i \in I\}$,

$$\mathtt{peirce}^v_A : \Pi_{i\in I}(A_i \Rightarrow \Sigma_{j\in J}B_j) \Rightarrow \Sigma_{i\in I}A_i \Rightarrow \Sigma_{i\in I}A_i$$

is the strategy which plays copycat between the positive and negative occurrences of $\Sigma_{i\in I}A_i$, until Opponent plays an initial move in the ith component of $\Pi_{i\in I}(A_i \Rightarrow \Sigma_{j\in J}B_j)$. Player then selects the ith component in the (positive) $\Sigma_{i\in I}A_i$, and plays copycat between the positive A_i and whichever of the negative A_i Opponent chooses to play in.

A control-specific analogy for the Player/Opponent duality, consistent with the general notion of System versus Environment, is to consider the role of Opponent as representing the current continuation. This is suggested by the correspondence between Player strategies on $A \multimap o$ and Opponent strategies (odd-branching, odd-prefix closed sets of moves) on A. `peirce` takes a strategy interpreting a term of type S with a free 'continuation variable' of type $S \Rightarrow T$, and binds it to the current continuation by copying Opponent's strategy on $[\![S]\!]$ as an input.

3.8.2 An alternative monad for control

To understand control flow in games, it is useful to return to the notion of a dialogue, and consider the distinction between demands for information, and information made in direct response to such a demand.
Call the initial move in the game $\mathbf{T}A$ a *question*, and any moves enabled by it, *answers*. (Note that no move can be both a question and an answer.) The intuition (see [59]) is that the weak co-product $\Sigma_{i\in I}A_i$ adds the following protocol:

- an initial Opponent question O_q: 'which A_i shall we play in ?'
- followed by a Player answer P_i, allowing play to proceed in A_i.

In order for this to define the usual 'separated sum' (and in the unary case, a lifting) it is necessary that what follows is a play in A_i. This is the case just if answer moves must be played linearly, as Player cannot revisit this choice and

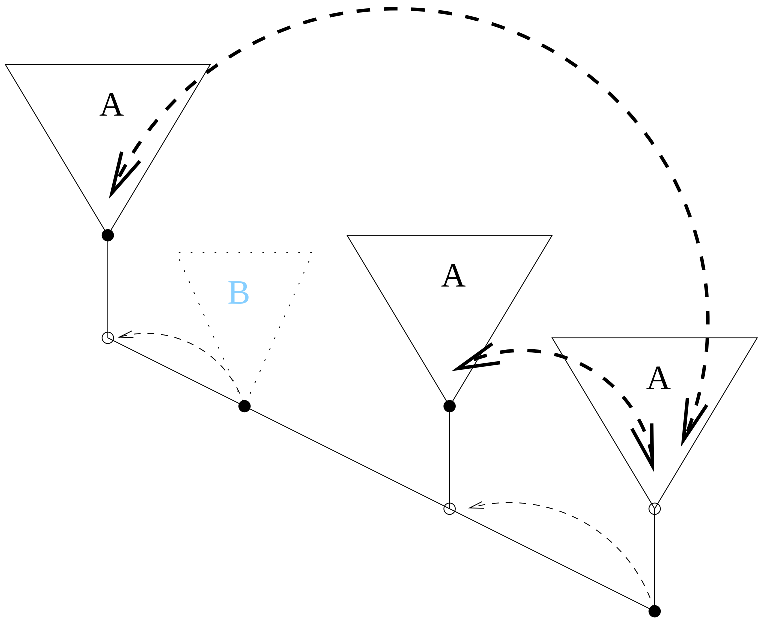

Figure 3.4: The strategy $\texttt{peirce}^n_A$

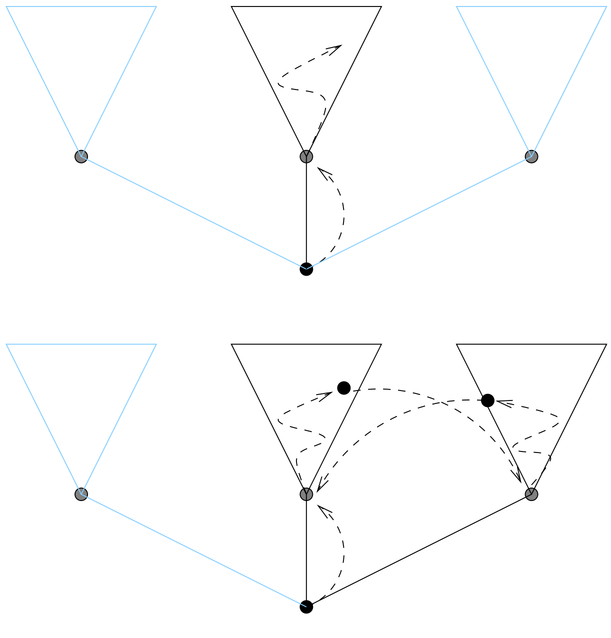

Figure 3.5: Contrasting plays in the different sum games.
Top: in the 'linear' or well-bracketed sum
Below: in the unbracketed sum, or continuations monad

either choose a new component, or open a new thread in A_i. In other words, non-$\perp$ elements in $A + B$ correspond either to an element in A, or one in B.

Note that the result of composing a strategy on $(A \Rightarrow \mathbf{T}B) \Rightarrow \mathbf{T}A$ which is non-linear in its argument, with $\mathtt{peirce}^v_A$, will be a strategy on $\mathbf{T}A$ which repeats the answer to the initial question each time the initial question in $(A \Rightarrow \mathbf{T}B)$ is asked (so the current continuation is 'called'). This multiple 'upward continuation passing' behaviour violates the principle that a term of 'lifted' type should correspond denotationally and operationally to a value, or to the undefined object $\perp$. Game semantically, it corresponds to the condition that answers should be unique and non-repeatable, — strategies cannot 'change their minds' about answers. By constraining answers to behave linearly it can be excluded from the model whilst retaining some level of non-local control.

Definition 3.8.5 *For a family of arenas $\{A_i \mid i \in I\}$ define the linear (affine), or 'weakly bracketed' sum $\Sigma^L_{i\in I} A_i$ to be the arena consisting of a single initial (linear) opponent move, enabling each one of a family of linear Player moves $\{l_i \mid i \in I\}$, each of which enables the initial moves of A_i.*

This can be defined in a more general setting as a continuations monad of sorts.

Definition 3.8.6 *Given a category $\mathcal{C}$ with a 'linear decomposition', of morphisms on $\mathcal{C}(A, B)$, as morphisms on $\mathcal{L}(!A, !B)$ for some linear category $\mathcal{L}$ and an object $\mathbf{a}$ of $\mathcal{L}$, define the (strong) 'linear $\mathbf{a}$-continuations monad' on $\mathcal{C}$*

$$\mathbf{T}^L_{\mathbf{a}} X = (X \multimap \mathbf{a}) \multimap \mathbf{a}$$

Proposition 3.8.7 *This is a strong monad on $\mathcal{L}$ (and on $\mathcal{C}$).*

Proof: is by observing that $T^L_{\mathbf{a}}$ can be resolved into the self-adjunction of a contravariant 'not-functor' which forms *linear* $\mathbf{a}$-continuations: $\neg A = A \multimap \mathbf{a}$. The fact that $\mathcal{L}(A, B \multimap o) \cong \mathcal{L}(B, A \multimap o)$ is immediate; the definitions for the Kleisli triple and monadic strength are 'linearized' versions of the standard continuations monad described in Section 2.2.6. $\square$

(Alternatively, in categories with a linear decomposition of $A \Rightarrow B$ as $!A \multimap B$, the monad can be defined as $TX = X \Rightarrow \mathbf{a}) \multimap \mathbf{a}$ (see [47]).)

Proposition 3.8.8 *In the category of (families of) linear games,*

$$\mathbf{T}^L_o\{A_i \mid i \in I\} = \{\Sigma^L_{i\in I} A_i\}.$$

(In fact, a notion of tensor product and closure on $\mathsf{Fam}(\mathcal{C})$ has not been defined here, but this is not really necessary, as only singleton families are involved.)

The fact that, given $\sigma\colon (A \multimap o) \multimap o$, either $\sigma = \bot$, or σ is the 'lifting' of some strategy on A has already been observed in the axiomatic characterization of the answer object. Thus at a local level the linear continuations monad defines a separated or intuitionistic sum type. However, control is a non-local issue: the linearity (or rather affinity) condition does not prevent 'jumps in the flow of control' in games. At the level of *atomic* games, the control strategies $\mathtt{peirce}$ described above are still well-defined:

Definition 3.8.9 *For any set X, define the* atomic *game $\widetilde{X_*}$ to be the game consisting of a single (Opponent) question and a set of answers, indexed over X. (So these correspond to 'flat domains' or weak co-products over the a family of terminal objects, $\{\mathbf{1}_x \mid x \in X\}$.)*

Proposition 3.8.10 *If X is a family of terminal objects, $\{\mathbf{1}_x \mid x \in X\}$, so that $\mathbf{T}^L_o X$ is the atomic game $\widetilde{X_*}$, then the strategies $\mathtt{peirce}^n_{\Sigma X}$ and $\mathtt{peirce}^v_X$* are *valid strategies, which give unique and non-repeated answers to questions.*

Proof: $\mathtt{peirce}^n_X$ (see Figure 3.6) copies the initial $\widetilde{X_*}$-question in the second component, and copies back any answer. If none is forthcoming, it copies the $\widetilde{X_*}$-question again in response to Opponent's $\widetilde{X_*}$ question, until it receives an $\widetilde{X_*}$-answer, which it copies as an answer to the initial question, closing the game. $\square$

$$\begin{array}{cccc}
((\mathbb{N} \Longrightarrow & \mathbb{N}) \Longrightarrow & \mathbb{N}) \Longrightarrow & \mathbb{N} \\
 & & & \mathrm{O}_q \\
 & & \mathrm{P}_q & \\
 & \mathrm{O}_q & & \\
\mathrm{P}_q & & & \\
\mathrm{O}_n & & & \\
 & & & \mathrm{P}_n
\end{array}$$

Figure 3.6: A typical play of $\mathtt{peirce}^n$ at atomic types

In fact, this is the only type of control behaviour possible in the arenas of the Hyland-Ong model of call-by-name PCF, which uses sums (implicitly) only to define ground types. In fact, by definition, the answer moves of Hyland-Ong games do not enable any other moves, and as a consequence, must be linear moves.

Proposition 3.8.11 *In a sequence satisfying visibility, any move which does not enable other moves*

- *occurs at most once,*
- *does not share its justifying move with any other such moves.*

Proof: is by induction on the length of sequence, with the hypothesis that if d is a Player move justified by c in the sequence $sc \cdot td \cdot r$ and d does not justify any subsequent move, then the following joint hypotheses hold:

- if it is Opponent's turn to move, none of the moves between c and d (exclusive) are in his view.
  This is clearly the case when the move d has just been made.
  Suppose d' is some subsequent Player move. Then by the induction hypothesis on Player views (below), and visibility, it cannot be justified by a move between c and d. If d' is justified by a move prior to c, then all moves between c and d are hidden in the Opponent view.
  If d' is justified by a move e after d, then this move occurs in the Opponent view at d' by the visibility condition. Hence by induction hypothesis applied at e, the moves between c and d are not in the Opponent view of this sequence.

- if it is Player's turn to move, none of the moves between c and d (inclusive) are in his view. (Hence, by visibility, he cannot make a move justified by c, QED).
  If the last move is an Opponent move, then it cannot be justified by d itself by assumption, and by induction hypothesis and visibility, it is justified either by a move before c or one after d. In the former case, the moves between c and d are hidden from the Player view, whereas in the latter case, the induction hypothesis can be applied to give the same result.

$\square$

This is closely connected with the fact that call-by-name PCF cannot be augmented with 'higher-order' control features expressively distinct from ground-type **call/cc**. The factorization described by the author in [48], shows that all bracketing violations in the call-by-name model of PCF can be represented as the application of the ground-type `peirce` strategy. The situation is slightly more complicated in the call-by-value model, as this contains answers which do justify questions, and strategies which are do not factorize via `peirce`. However, it can be shown (by factorization) that all weakly bracketed strategies can be simulated *up to observational equivalence* as the composition of a well-bracketed strategy, and the Peirce's law strategy at atomic types. Thus from a proof-theoretical standpoint, this model corresponds to 'stability'; classical rules for atomic formulas, from which all classical rules for the negative fragment of propositional logic can be derived.

Thus the category of games with linear answers can be seen as a semantics of 'first-order' control, with 'downward continuation' passing behaviour allowing jumps in the flow of control corresponding to functional variations of `GOTO`; aborts, escapes, exceptions etcetera, contrasting with the 'upwards continuation passing' of higher-typed **call/cc** in its (usual) call-by-value setting. This could be said to mark a boundary between control as a simple extension of functional languages, and continuation-passing as a programming style in its own right.

3.9 Linearity and the bracketing condition

An explanation for the existence of ground type 'control strategies' such as `peirce` in the weakly bracketed model, is that the linearity rule is not, strictly speaking, linear (precisely once), but affine (at most once). In addition to validating weakening (as the terminal object is a unit for $\otimes$), this also introduces the possibilities of limited contractions associated with the 'weak initiality' of the one-move game. As shown in Figure 3.6, the strategy `peirce` works at ground types (i.e. $o^X \multimap o$) by *missing out* answers rather than repeating them. `peirce` is also a total strategy (via its view function) on $((A \multimap B) \multimap A) \multimap A$, for atomic A, although this is *not* a tautology of affine logic.

By giving a refined, stricter definition of linearity, this behaviour can be excluded. This can be done most simply by considering only the case in which linear and non-linear moves alternate according to the question/answer discipline dictated by the sum game, giving a neat connection with the bracketing condition. This correspondence between intuitionistic and linear behaviour suggests, however, that an analysis of local control flow via decomposition into linear logic is possible.

The connection between linear moves and questions and answers in bracketed arenas can be formalized as follows.

Definition 3.9.1 *A non-initial (strictly) linear move is an answer. Any move which is not an answer is a question, — that is, initial moves and non-linear moves.*
An arena is bracketed *if only questions may justify answers (the original requirement of [43]).*

Note that the constructions $A \Rightarrow B$, $A \times B$ and, importantly, the linear sum, can be carried out in bracketed arenas. The linearity rule can be strengthened, to require that linear moves also occur at least once, by imposing the following 'liveness' condition.

Definition 3.9.2 *A legal sequence s is* strictly linear *if it can be extended to a (finite) legal sequence in which every question justifies exactly one answer.*

Interestingly, in the presence of visibility, it is possible to characterize this condition solely in terms of the moves already played, rather than by quantification over possible extensions. The following condition states that question-answer pairs must be 'nested' in an appropriate way, and was used in both game semantics of PCF but is older (see e.g. Felscher's paper [26]).

Definition 3.9.3 (Bracketing Condition) *Suppose A is a bracketed arena; a legal sequence over A is well-bracketed if every answer is always justified by the last-asked, unanswered question, — formally, assuming a $\mathcal{L}_J$ definable predicate $a \prec_s b$ - ' a precedes b in s', define*
$WB(sab)$ if $WB(sa)$ and $(\lambda^{WB}(b) = A$ implies
$(\lambda^{WB}(a) = Q \Longrightarrow \phi_{sab}(b) = a) \wedge (\lambda^{WB}(a) = A \Longrightarrow \phi_{sab}(b) \prec_{sab} \phi_{sab}(a))$.
Only one label, for answers, is really required. The condition is equivalent to requiring that there are equal numbers of questions and answers between each question and its answer.

Well-bracketed sequences are clearly (weakly) linear, as questions can be answered only once — in fact they are strictly linear, as questions can be answered in the reverse of the order that they were asked. Sequences over Hyland-Ong games which are not well-bracketed are not strictly linear, as unanswered questions become 'hidden' between two answers as described in Proposition 3.8.2 — in fact (surprisingly) this holds in general; once the balance between questions and answers has been lost by violating the bracketing condition, there is no way to regain it (see Figure 3.7).

Proposition 3.9.4 *If s is a (odd/even-length) well-bracketed legal sequence, then the last-asked open question in s is in the view (Player/Opponent) of s.*

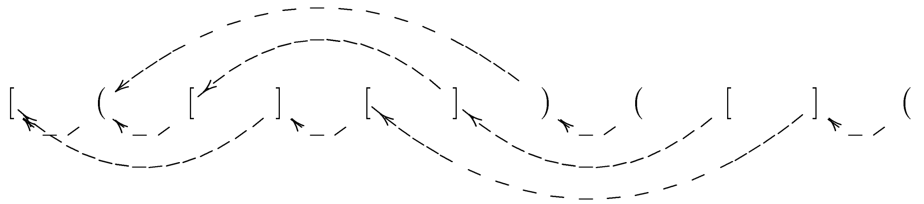

Figure 3.7: A sequence violating the bracketing condition (with justification pointers): Opponent questions are denoted '[', Player questions ')'.
Note how the first violation of well-bracketing destabilises the sequence so that there is always a prematurely closed unanswered question.

PROOF: is by induction on sequence length. Suppose sb is an odd-length well-bracketed sequence, then if b is a question, the hypothesis holds directly. If b is an answer, then $sb = s'q \cdot tb$, where q is a question justifying b. As s is well-bracketed, every question between q and b has been answered, and so the most recently asked unanswered question is in s', and by induction hypothesis appears in the view of $\ulcorner s' \urcorner$ and so in $\ulcorner s \urcorner = \ulcorner s' \urcorner qb$. $\square$

Definition 3.9.5 *Say that a question is 'prematurely closed' (whether it has been answered or not) if a previously asked question has already been answered, and 'open' if it is unanswered and not prematurely closed.*

Proposition 3.9.6 *Suppose A is an arena in which every question has an answer, then a legal sequence over A is strictly linear if and only if it is well bracketed.*

PROOF: Suppose $s \in J_A$ is well-bracketed, then every question in s is answered at most once. Moreover, s can be extended to a sequence in which all of the questions have exactly one answer by answering open questions in the reverse order to which they were asked. (Note that if s is a well-bracketed legal sequence, then the most recently asked open question is in the relevant view of s.)
The converse is a corollary of the following:

Lemma 3.9.7 *Any extension qt (to some s such that $s \cdot qt$ is a legal sequence) in which q is a prematurely closed question contains an unanswered, prematurely closed question.*

PROOF: is by induction on the length of extension.
Suppose qta is an extension to s in which q is prematurely closed. Then if a is not an answer to a prematurely closed question in qta, qt contains an unanswered, prematurely closed question by induction hypothesis, which is still unanswered in $sq \cdot ta$.

It is also safe to assume that a is an answer to q, as if it is an answer to some later prematurely closed question, then the induction hypothesis can be applied to the shorter extension starting from this question.
So suppose a answers q. Now consider the move preceding a; if this is a question, then it is prematurely closed by a, and unanswered, and we are done. So suppose that it answers some question q'; by the visibility condition, the justifier of a, which is q, must occur prior to q'. There are two possibilities:

- the answer prematurely closing q occurs after q', so that it also prematurely closes q'. Then the induction hypothesis can be applied to the extension starting at q'.

- the answer prematurely closing q occurs before q'. Then applying the induction hypothesis to the extension from q to q', there is a prematurely closed question in this segment, unanswered (at q'). If this is answered in t (between q' and a), then the answer prematurely closes q', and applying the induction hypothesis to the segment from q to a gives an unanswered prematurely closed question as required.

$\square$

Consequently, any (finite) sequence containing a prematurely closed question contains an unanswered question. So no sequence which is not well-bracketed can be extended to a finite sequence in which all questions are answered.

Proposition 3.9.8 *Well-bracketing preserves the identity and associativity conditions and hence defines a category of games.*

PROOF: is straightforward. In the presence of visibility, and the switching condition, compositionality can be proved as follows.
Note that as a consequence of the switching condition, a move from A in $(A \multimap B) \multimap C$ can *never* be directly followed by one from C and vice versa, but must be followed by an odd number of B moves.

So suppose sa is a justified sequence in $(A \multimap B) \multimap C$, and $sa \upharpoonright (A, B)$and $sa \upharpoonright (B, C)$ are well-bracketed. If a is an answer in A, then it answers the most recently asked open question in $s \upharpoonright A, B$. Hence any questions which were asked in C since this question was asked have been answered, as to 'get back' to A, an odd number of moves must have been made in B, the last of which must have answered an earlier question. By well-bracketedness of $sa \upharpoonright B, C$, all of the questions asked in this spate of moves in C have been answered.

Similarly, if a answers a question in C, there are no more recent open questions in A. $\square$

As it is a rule, the bracketing condition preserves the symmetric monoidal closed and co-monad structure (and the product is still cartesian).

Proposition 3.9.9 *The category of well-bracketed games and innocent strategies is cartesian closed.*

Thus the original categories of Hyland-Ong and McCusker games has been recovered (although not requiring well-openedness simplifies the definition of composition). A crucial property of the bracketing condition for both versions of the game semantics of PCF is that it prevents innocent strategies from making intensional observations.

Proposition 3.9.10 *The intrinsic preorder on the categories of well-bracketed games and innocent (and history-free) strategies is order extensional.*

PROOF: is given in detail in [43] and [59], but observe that a (strict) well-bracketed strategy on $A \rightarrow \widetilde{\mathbb{N}_*}$ must have an answer to the first question in A before it gives an answer in $\widetilde{\mathbb{N}_*}$, and if it is also innocent, then this answer is the only one in the view. $\square$

As well-bracketing is a well-defined rule, the category of well-bracketed games and innocent functions can be embedded in the category of unbracketed games via their innocent functions (Proposition 3.6.15). Moreover, there is a simple characterization of the image of this embedding in terms of view-functions.

Proposition 3.9.11 *An innocent strategy is well-bracketed if and only if it always answers the most recently asked open question in the view.*

PROOF: It is immediate that any innocent strategy on a well-bracketed game always answers the most recently asked question in the view, so it remains to show that any strategy σ which does so will never violate the bracketing condition before Opponent. But by Proposition 3.9.4, if s is a well-bracketed sequence, then the most recently asked open question in s is visible. $\square$

Answering the most recently asked question in the view can then be adopted as the *definition* of well-bracketedness for strategies which are not innocent.
The well-bracketed sum is shown in [59] to define a pointed weak co-product, so it is a strong monad on $\mathsf{Fam}(\mathcal{G}_{\mathsf{WB}})$, (with the same copycat strategies yielding

a Kleisli triple and strength as for the continuations monad). It thus yields models of call-by-name and call-by-value PCF as described in Section 2.4.4. The call-by-name model is equivalent to the original model described by Hyland and Ong in [43], as the atomic types denote the flat domain of natural numbers, and exponentiation. The well-bracketed sum was used by Abramsky and McCusker to give a fully abstract semantics of PCF_v in [3]. In each case, finite definability results (based on an inductive decomposition similar to the axiomatic result in this chapter) yield a fully abstract model by collapse under the (extensional) intrinsic preorder).

Theorem 3.9.12 (Hyland and Ong [43], Abramsky and McCusker [3]) *The semantics of call-by-name and call-by-value* PCF *in the category of well-bracketed games and innocent strategies has the finite definability property (and its extensional collapse is fully abstract).*

Dropping the bracketing condition transforms the well-bracketed sum from a linear 'double negation translation' on games, to an intuitionistic double negation: the interpretation of function types (the exponential in the Kleisli category of the lifting monad), $A \Rightarrow (B \Rightarrow o) \multimap o$ becomes $A \Rightarrow (B \Rightarrow o) \Rightarrow o$.

Remark 3.9.13 *The connection between intuitionistic functional types, and the monad $(\_ \Rightarrow o) \multimap o$ which has been established by showing the connection between well-bracketing and strict linearity leads to the following conjecture:*
there is an invertible 'linear cps' translation from (call-by-name and call-by-value) PCF *into the linear λ-calculus (with partiality and ω-indexed products) based on the single ground type ι. This is analogous the translation for the 'classical (un-bracketed) sum monad' and μPCF described in the next chapter (Section 4.5.1).*

Remark 3.9.14 (The stack discipline as an analogy for well-bracketing) *A further intuition connecting the bracketing condition with control, is that it represents a stack-discipline for the (computation represented by the) interaction of Player and Opponent. A formal connection with stack-based computation via abstract machines have been established in [21]. There it is shown how the interactions between strategies in unbracketed Hyland-Ong games can be computed using a stack-based abstract machine.*

The correspondence between bracketing and the stack discipline suggests how this work can be extended to games with the additional control flow information represented by the question/answer distinction. Suppose that the stack is used to hold only the 'open questions', — those demands for information which remains *to be supplied in order for the computation to be completed. Giving some of this information by making an answer move corresponds to 'popping the answered question' from the stack, together with any other questions which no longer need to be answered.*

Well Bracketed Games *correspond to rigidly maintained stack discipline; only the question at the top of the stack can be popped, (answered).*

Weakly bracketed Games *In games in which answers cannot be repeated, but can be missed out, prematurely answering a question 'closes' all of the more recently asked questions as far as Player is concerned. This corresponds to 'popping the stack* downwards*' (to the point of the last unclosed, unanswered question). The following lemma shows that the weak bracketing condition can be seen as a way to prevent an innocent strategy from violating linearity by answering the same question twice.*

Lemma 3.9.15 *If* Player *is playing according to an innocent linear strategy on a bracketed arena, then he cannot answer any questions prematurely closed by himself.*

PROOF: *(sketch) Suppose Player answers a prematurely closed (Opponent) question. Then this question appears in Opponent's view and can be repeated, with the same justifier. Player will then have the same view as when the question was first asked, and will make the same response. Opponent can then copy his initial response to this move, and so on, until Player is forced to repeat the answer move which closed the Opponent question, which is a violation of linearity.* □

Unbracketed games *As well as 'popping the control stack downwards', which simply discards some of the current demands for information, call/cc can be used to reset the stack to other previous states. This means that demands for information which has already been supplied can become 'live' once again, and new information (not necessarily the same) given in response. This corresponds to the situation in unbracketed games, where any previous (visible) questions can be 'answered'. The notion of 'open' question is now extended, to say that a question is open if the answer to any previous question occurs before the answer to some succeeding question. (So a question is open if it has not been closed, or if it has been closed, but the closing move, considered as a question, has itself been closed.)*

Answering a closed question thus corresponds to 'Popping the stack upwards' as it always opens up some closed questions.

Chapter 4

The syntax and semantics of μPCF

4.1 A minimal basis for computation

What is the simplest way to define a functional language for doing higher typed, sequential computation? PCF, or something similar, in which datatypes are 'built in' at ground types, with constants for construction $(0, \texttt{pred}, \texttt{succ})$ and decomposition (a conditional), is a possible and widely accepted answer to this question. Control operators fit in to this perspective as a useful non-functional extension to this minimal language. There is another possibility, however, which is to *define* data objects as simple functions, in the style of the original Church numerals, and latterly System F, the Calculus of Constructions, etc.

Example 4.1.1 (Finitary PCF**)** *Recall* $\Lambda(\Omega)$*, the (call-by-name)* λ*-calculus with single base type* ι *with a constant for non-termination. Define a type of 'Church Booleans':* $\texttt{bool} := \iota \Rightarrow (\iota \Rightarrow \iota)$*. Non-termination, call-by-name* λ*-abstraction and application based on this 'ground type' can be extended with the constants of boolean* PCF *defined as macros:*

- $\mathbf{tt} =_{df} \lambda x.\lambda y.x$
- $\mathbf{ff} =_{df} \lambda x.\lambda y.y$
- IF r then s else $t =_{df} \lambda x.\lambda y.r\ (s\ xy)(t\ xy)$.

However there are terms of $\Lambda(\Omega)$ *which are not (*$\beta\eta$ *equivalent to) the translation of a term of boolean* PCF*. Such terms can use the intensional way in which data has been defined to extract intensional information about their arguments (pace* PCF*). For example, the following strictness test returns* $\texttt{tt}$ *on a function which evaluates its argument (as the only way for a* $\Lambda(\Omega)$*-term to use argument is by* *applying it to something, which will be discarded). If f is not strict, then the test returns* **ff** *as it applies its result (which is a projection) twice to its second argument.*

strict?: $(\texttt{bool} \Rightarrow \texttt{bool}) \Rightarrow \texttt{bool} = \lambda f : \texttt{bool} \Rightarrow \texttt{bool}.\lambda x\colon \iota.\lambda y\colon \iota.f\ (\lambda z.\lambda z.x)\ y\ y$

The strictness test can, of course, be written using control operators: — this chapter will show that they bridge the gap between PCF and computation by continuation-passing by supplying the missing expressive power. The cps translation can thus be viewed as a *compilation* which structures procedures of the low-level control language $\lambda(\Omega)$ so that they can be written as functional programs using control operators at lower types. This gives the best of both worlds; a simple semantics for a powerful language.

The second section of the chapter describes the syntax of the $\lambda\mu$-calculus and μPCF, the third gives a unified call-by-name and call-by-value operational semantics, which is proved complete with-respect-to a minimal notion of adequate model in the fourth part. The fifth section describes a denotational semantics for the call-by-name language via a cps translation. Full abstraction for the games model is proved by showing that the translation restricts to a bijection between 'finite normal forms' in the next part. Sections 7, 8 and 9 define denotational models of the call-by-value language, show that $\lambda\mu_v$ is a complete syntax of control, and give definability and full abstraction results.

4.2 Representing functional control

Control operators such as *strict?*, or the **catch** operators of SPCF are a simple way to add escapes to call-by-name PCF. However, more complex control manipulation involving continuations, requires a more expressive syntax. The $\lambda\mu$-calculus has been chosen here as it defines control manipulations in a way which is easy to represent in a model of control, and can be given an operational semantics *and* an equational theory in a straightforward way. It originated as a calculus for classical natural deduction in the work of Parigot [69], who noted the existence of terms corresponding to control operators. $\lambda\mu$ has migrated via the extended Curry-Howard correspondence: it has subsequently been proposed as a 'foundation for functional computation with control' by Ong and Stewart [67], who developed a call-by value version, and used it to extend PCF with control features. Important confirmation of its suitability as a syntax for control has been provided recently by Selinger [79], who has proved that it is the *internal language* of a 'control category' (as outlined in Section 2.3.2).

$$\frac{}{x{:}A \vdash x{:}A}\ \mathrm{Ax}$$

$$\frac{\Gamma, x{:}A \vdash t{:}B;\Delta}{\Gamma \vdash \lambda x.t{:}A \Rightarrow B;\Delta}\ \Rightarrow\text{-intro} \qquad \frac{\Gamma \vdash s{:}A \Rightarrow B;\Delta \quad \Gamma \vdash t{:}A;\Delta}{\Gamma \vdash s\ t{:}B;\Delta}\ \Rightarrow\text{-elim}$$

$$\frac{\Gamma \vdash s{:}A;\Delta}{\Gamma \vdash [\alpha{:}A]s;\Delta,\alpha{:}A}\ \mathbf{0}\text{-intro} \qquad \frac{\Gamma \vdash s{:}\mathbf{0};\Delta,\beta{:}B}{\Gamma \vdash \mu\beta.s{:}B;\Delta}\ \mathbf{0}\text{ -elim}$$

Figure 4.1: Terms-in-context of the $\lambda\mu$ calculus

$\lambda\mu$ is a convenient, and in some senses canonical choice of syntax but it does not provide any greater expressive power than **call/cc**, or its idealized brother, Felleisen's $\mathcal{C}$ [25]. To make this claim more precise: one of the advantages of μPCF is that it can be considered with or without admitting the empty type. It will be shown that the closed terms which can be written with the empty type are expressible using $\mathcal{C}$ (and vice-versa), whilst those which can be written without the empty type are expressible using **call/cc**. $\lambda\mu$ is syntactically more complicated than either, as it adds a new 'continuation binding' operation to the λ-calculus, (this step was already present in Reynolds' definition of the **Escape** operator [76] (1972)). This has the advantage of making the control aspect of the calculus more explicit, and separate from the functional part.

4.2.1 The $\lambda\mu$ calculus: call-by-name and call-by-value

Terms of the $\lambda\mu$ calculus are obtained by adding the operations of *naming* and *μ-abstraction* to the λ-calculus. A term-in-context t is supplied with sets Γ containing the free variables of t, and Δ containing its free names (represented as small-case greek letters) so weakening, permutation and contraction hold for both contexts. The variable convention of [8] is adopted for names as well as variables. Terms-in-context are formed as set out in the table (Figure 4.1). To name a term of type T requires a name α of type T, the named term has type $\mathbf{0}$ (the empty type). There are no names of type $\mathbf{0}$. Any free name $\alpha{:}\,T$ in a term $t{:}\,\mathbf{0}$ can be bound by a μ-abstraction, written $\mu\alpha.t{:}\,T$.

Partiality is added to the $\lambda\mu$-calculus by extending it with the single constant $\Omega : \mathbf{0}$, from which $\Omega^T : T = \mu\alpha.\Omega^{\mathbf{0}}$ can be derived at any type. Extension with product types and pairing is not described, but is completely straightforward.

The intuition upon which the continuation-passing semantics of $\lambda\mu$ is based is that a term of type T can be interpreted as a map from continuations of type T, to the answer object. This is just the standard interpretation of a functional language in the Kleisli category of the monad of continuations described in Chapter 2. Names of type T are simply variables of type $T \Rightarrow \mathbf{0}$ representing continuations of type T. Naming corresponds to applying the current denotation to a continuation variable, and getting something of answer type. μ-abstraction is the λ-abstraction of such a variable from a term of answer type, giving a denotation of type T.

In order to define an operational interpretation based on this intuition, it is useful to have a syntactic representation of the current continuation. This is provided by the notion of 'evaluation context'. (Note that there are a variety of subtly different definitions of this in the literature.)

Definition 4.2.1 *Values (written $u, v, \ldots$), range over variables and λ-abstractions, i.e.*

$$v ::= x \mid \lambda x.t.$$

Contexts with a hole of type T will be written as $E^T[\cdot]$, (as distinct from $E[\cdot] : T$, where the result of filling the hole is a term of type T).
Call-by-name evaluation contexts are defined inductively over the following grammar (respecting the typing rules for $\lambda\mu$):

$$E^T[\cdot] ::= [\cdot] : T \mid [\alpha]E^T[\cdot] \mid E^T[\cdot]\ t$$

and call-by-value evaluation contexts over:

$$E[\cdot] ::= [\cdot] : T \mid [\alpha]E^T[\cdot] \mid E^T[\cdot]\ t \mid v\ E^T[\cdot]$$

where t ranges over general terms, and v over values.

The equational theory of each version of the $\lambda\mu$-calculus is given (up to α-equivalence of names and variables) by the standard $\beta\eta$-equalities of the λ-calculus, together with the following rules.

$(\mu\eta)$ $\quad \mu\alpha.[\alpha]t =_\mu t$ (α not free in t),

$(\mu\zeta)$ $\quad E[\mu\alpha.t] : \mathbf{0} =_\mu t[E[\cdot]/\alpha]$ (note the type restriction on $E[\cdot]$).

In the call-by-name version, β, η, and the notion of evaluation context are call-by-name, in the call-by-value version, they are call-by-value. (Completeness of the call-by-value theory for control models requires minor additional axioms discussed in Section 4.9.)
The operation which requires some explanation is the substitution of a context for a name used in $\mu\zeta$. (This is essentially a simplified form of the *mixed substitution*

of [69].)
$t[E[\cdot]/[\alpha]]$ means: replace occurrences of named subterms $[\alpha]r$ in s with $E[r]$. i.e. suppose $t = C[[\alpha]r_1]\ldots[[\alpha]r_n]$, where α does not occur free in $C[\cdot]\ldots[\cdot]$ or $r_1,\ldots r_n$; then $t[E[\cdot]/\alpha] = C[E[r_1]]\ldots[E[r_n]]$.
Formally, it can be defined by induction on term-structure:

Definition 4.2.2 (Named-substitution)

$$\begin{array}{rcl} x[E[\cdot]/\alpha] = x & & [\alpha]t[E[\cdot]/\alpha] = E[t[E[\cdot]/\alpha]] \\ (s\ t)[E[\cdot]/\alpha] = s[E[\cdot]/\alpha]\ t[E[\cdot]/\alpha] & & (\lambda x.t)[E[\cdot]/\alpha] = \lambda x.(t[E[\cdot]/\alpha]) \\ (\mu\beta.t)[E[\cdot]/\alpha] = \mu\beta.(t[E[\cdot]/\alpha]) & & ([\beta]t)[E[\cdot]/\alpha] = [\beta](t[E[\cdot]/\alpha]) \end{array}$$

This is a slight modification to the axiomatizations proposed by Parigot [69] and by Ong [66], firstly in its presentation via evaluation contexts instead of single applications (but this is a minor change, which is implicit in Ong and Stewart's paper [67]). There is also a change to the rules: the presentations cited above have an additional rule,
$(\mu\beta)\quad [\gamma]\mu\alpha.t = t[\gamma/\alpha]$
and a separate $\mu\zeta$-rule covering non-zero typed contexts, which is (presented using evaluation contexts)
$\mu\zeta:\quad E^T[\mu\alpha.t] = \mu\beta.t[[\beta]E[\cdot]/\alpha]$.

Proposition 4.2.3 *The $\lambda\mu$ theory proposed here is equivalent to those in [69] and [66].*

PROOF: $\mu\beta$ is derivable as an instance of $\mu\zeta$ as $[\alpha][\cdot]$ is an evaluation context: $[\alpha]\mu\beta.t = t[\beta/[\alpha][\cdot]]$.
$\mu\zeta$ is derivable using $\mu\zeta^{\mathbf{0}}$ and $\mu\eta$:
$E^T[\mu\alpha.t] = \mu\beta.[\beta].E[\mu\alpha.t]^T = \mu\beta.t[[\beta]E[\cdot]/\alpha]$. $\square$

The presentation given here is closer to the continuation passing interpretation which will be developed, making soundness and adequacy proofs for the semantics a little simpler. (Although the original rules were derived from the normalization of classical proofs, so it could be argued that this is moving further from the Curry-Howard correspondence, and towards a purely control-based analysis.)

4.2.2 Sum types in $\lambda\mu$

One advantage of $\lambda\mu$ is that it provides a natural syntax for terms at sum types, compared to the λ-calculus alone. (As one would hope, moving to a multiple conclusion logic simplifies the behaviour of disjunction.) Among several possibilities for annotating sums, rather than allowing names to take disjunctive types, naming of a term of type $A + B$ requires a pair (tuple) of names of type A and type B. (Thinking of names as variables of type $A \Rightarrow \mathbf{0}$ an $B \Rightarrow \mathbf{0}$, naming of $t\colon A + B$ with $\alpha\colon A, \beta\colon B$ is by applying the *co-pair* $[\alpha, \beta]$.)

Definition 4.2.4 *Form the $\lambda\mu^+$ calculus by extending the types over which formulas (and variables) may range with finite sum types $A + B$.*
Names may take only types with a main connective which is negative (i.e. $A \Rightarrow B$, $A \times B$), which is a natural extension of the rule that names may not take the empty type.
Extend the typing judgements of $\lambda\mu$ as follows:

$$\frac{\Gamma \vdash t\colon A + B; \Delta}{\Gamma \vdash [\alpha, \beta]t\colon \mathbf{0}; \alpha\colon A, \beta\colon B, \Delta}$$

$$\frac{\Gamma \vdash t\colon \mathbf{0}; \alpha\colon A, \beta\colon B, \Delta}{\Gamma \vdash \mu[\alpha, \beta].t\colon A + B; \Delta}$$

This extends naturally to all finite sums The equational theory of $\lambda\mu_v^+$ is given by adapting the (Ong and Stewart) theory for $\lambda\mu$ by

- *retaining $\mu\eta$ at all types,*
- *restricting $\mu\zeta$*
  $E[\mu[\alpha, \beta, \ldots].t] : \mathbf{0} =_\mu t[E[\cdot]/[\alpha, \beta, \ldots]]$
  to cases where $\alpha, \beta, \ldots$ only occur conjointly in t,
- *Adding $\mu\beta$, without the above restriction*
  i.e. $[\gamma : A, \delta : B, \ldots]\mu[\alpha\colon A, \beta\colon B].t \quad =_\mu t[\gamma/\alpha, \delta/\beta]$.

Standard constructors for sum types can be derived; for instance, define:
$\mathtt{in_l}(t\colon A) = (\mu[\alpha, \beta].[\alpha]t)\colon A + B$, $\mathtt{in_l}(t\colon A) = (\mu[\alpha, \beta].[\beta]t)$ (assuming α, β not free in t),
and given $t\colon A \Rightarrow C$, $s\colon B \Rightarrow C$, $r\colon A + B$, define
$(\mathtt{case}\ r\ \mathtt{of}\ t, s)\colon C = \mu\gamma\colon C.[\gamma]t\ (\mu\beta.[\gamma\colon C]s\ (\mu[\alpha].[\alpha, \beta]r))$
(so $\mathtt{case}\ \mathtt{in_l}(t)\ of\ u, v\ =_\mu u\ t$, and $\mathtt{case}\ \mathtt{in_r}(t)\ of\ u, v =_\mu v\ t$).

One could therefore use $\lambda\mu^+$ to add a sum type constructor to PCF. However an additional reason for introducing it is that by including sum types in the call-by-value language, a cps translation can be given which is surjective onto types and terms of $\Lambda(\Omega)$, showing that $\lambda\mu_v^+(\Omega)$ is a complete syntax of control (see Section 4.9).

4.2.3 μPCF: variations on the control theme

Definition 4.2.5 *Form the language* μPCF *by extending* $\lambda\mu$ *with the constants of* PCF*, forming terms-in-context over function types generated from the atomic basis* $\mathbf{0}, \mathtt{nat}$*. The equational theory of* μPCF *is given by the extension of the* $\lambda\mu$ *theory with the rules for the constants (and fixpoint operator) of* PCF*.*

In a further, minor departure from the standard presentation of PCF*, the conditional is given at type* $\mathbf{0}$*. i.e.*

$$\frac{\Gamma \vdash N\colon \mathtt{nat},\ \ M\colon \mathbf{0}, L\colon \mathbf{0}}{\Gamma \vdash \mathrm{IF0}\ N\ \mathrm{then}\ M\ \mathrm{else}\ L\colon \mathbf{0}}$$

and similarly for each `case` *statement.*
This proves to be a canonical choice in the semantics, but the original conditional can be recovered as $\mu\alpha.[\alpha]\mathrm{IF0}\ N\ \mathrm{then}\ [\alpha]M\ \mathrm{else}\ [\alpha]L$*, where* α *is not free in* L, M, N*.*

An equational theory for μPCF can be given by extending the theory of PCF with the rules $\mu\eta$, and $\mu\zeta$, extending the notion of evaluation context appropriately as follows.

Definition 4.2.6 (Evaluation contexts of μPCF) *Call-by name evaluation contexts are defined inductively as::*

$$\begin{aligned} E[\cdot] ::= {} & [\cdot] \mid \\ & [\alpha]E[\cdot] \mid E[\cdot]\ M \mid \\ & \mathrm{IF0}\ E[\cdot]\ \mathrm{then}\ M\ \mathrm{else}\ N \mid \mathtt{succ}\ E[\cdot] \mid \mathtt{pred}\ E[\cdot] \end{aligned}$$

and call-by-value evaluation contexts as:

$$\begin{aligned} E[\cdot] ::= {} & [\cdot] \mid \\ & [\alpha]E[\cdot] \mid E[\cdot]\ M \mid V\ E[\cdot] \mid \\ & IF0\ E[\cdot]\ then\ M\ else\ N \mid \mathtt{succ}\ E[\cdot] \mid \mathtt{pred}\ E[\cdot]. \end{aligned}$$

The type $\mathbf{0}$ (or $\bot$) appears to have an important role in 'idealized' control calculi such as $\lambda\mathcal{C}$ and $\lambda\mu$. However, 'real' control operators like **call/cc** do not use such an empty type. It is the very fact that $\mathbf{0}$ has no values which makes it useful in the semantics of control; if the current continuation takes arguments of type $\mathbf{0}$ to results of type $\mathbf{0}$ then it can safely be discarded. In terms of the Curry-Howard correspondence, the idealized control operators correspond to full classical logic, and (typed) **call/cc** to minimal logic. Both systems are of interest, and one of the advantages of the $\lambda\mu$ calculus (as opposed to Felleisen's $\mathcal{C}$ and **call/cc**) is that it is flexible in allowing both within the same framework. Thus a single denotational and operational semantics proves to be sufficient.

One can admit the type $\mathbf{0}$ to the type-system of PCF as a base type alongside `nat`. Or one can form the language, designated $\mu\mathrm{PCF}^-$ by Ong and Stewart, in which the use of the type $\mathbf{0}$ is restricted to namings.

Definition 4.2.7 *Form the language* μPCF$^-$ *(call-by-name or call-by-value) by restricting the typing judgements of* μPCF *as follows:*
the terms-in-context of μPCF$^-$*;* $\Gamma \vdash t : T, \Delta$*, are formed as for* μPCF *with the restriction that* Γ, Δ *contain only* PCF*-types, and* T *is* $\mathbf{0}$*, or a* PCF *type.*
(So every subterm of t *has a* PCF*-type or the empty type.)*

(The only well-formed terms of type $\mathbf{0}$ in μPCF$^-$ are namings and conditionals, and any naming must be immediately followed by a μ-abstraction, and any μ-abstraction must follow a naming.)
There will be some further comment on the semantic distinctions between μPCF and μPCF$^-$, but as the operational semantics and associated proofs can be presented so as to apply to both, they will be assumed to do so unless otherwise noted.
The following system with 'first-order' control can also be defined.

Definition 4.2.8 *Form the language* μPCF^1 *(call-by-name or call-by-value, with or without* $\mathbf{0}$*-types) as for* μPCF*, but restrict the use of names to the type* `nat`*.*
(This system is closed under the operational semantics which will be introduced.)

Three different variations on the $\lambda\mu$ calculus, giving eight different languages have now been defined, however, the distinction between first and second order is empty in the call-by-name versions. This can be seen as a syntactic counterpart to the observation that the only violations of the bracketing condition possible in the games models of call-by-name PCF are weak, representing only downward continuation passing [82]. Or, from the logical standpoint, that all of propositional classical (or minimal) implicational logic can be deduced from intuitionistic logic with 'atomic stability', — classical rules at atomic (or disjunctive) types. Rules 'bootstrapping' double-negation to higher types (as described below) were given by Prawitz [75].

Proposition 4.2.9 *With or without* $\mathbf{0}$*-types,* μPCF^1_n *is equivalent to full* μPCF$_n$ *(in that the latter can be soundly translated into the former).*

Proof: Naming at arrow-types in call-by-name can be eliminated as follows.
For a μPCF-type $T = A \Rightarrow B$, assuming $B \neq \mathbf{0}$, to each μ-name α of type T, associate a μ-name α' of type B and a variable $x_\alpha : A$. Now translate higher-type naming and μ-abstraction in a term M as follows:
Replace every named subterm of M, $[\alpha^T]N$ with $[\alpha'](N\ x_\alpha)$, and every μ-abstraction with $\lambda x^\alpha.\mu\alpha'.M$. (This can be defined formally as an inductive translation on terms.)

To show that all of the theory of $\lambda\mu$ holds with respect to this translation, the following simple lemma can be proved by structural induction.

Lemma 4.2.10 *For any* μPCF *term* M,
$M[[\alpha'][\cdot]]\ x_\alpha/\alpha][N/x_\alpha][E[\cdot]/\alpha'] = M[E[[\cdot]\ N]/\alpha]$.

Hence the translation is sound:
$\mu\zeta$ holds, because if $E^{A\Rightarrow B}[\cdot] : \mathbf{0}$ is a μPCF_n evaluation context, then *either* $E[\cdot] = E'[[\beta]\cdot]$, in which case $E^{A\Rightarrow B}[\mu\alpha.M]$ translates as $E[[\beta'](\lambda x.\mu\alpha'.M'\ x_\beta)] = E[\beta']\mu\alpha'.M'] = M'[E[\cdot]/\alpha]$ as required.
Or $E[\cdot] = E'[\cdot\ N]$, for some term $N : A$, so ignoring other names, $E[\mu\alpha.M]$ translates as $E'[\lambda x_\alpha.\mu\alpha'.M[([\alpha'][\cdot]\ x_\alpha)/\alpha]\ N] =_\mu M[([\alpha'][\cdot]\ x_\alpha)/\alpha][N/x_\alpha][E'[\cdot]/\alpha']$ and by the lemma, this is equal to $M[E'[[\cdot]\ N]/\alpha]$ as required.
The rule $\mu\eta$ translates as $\lambda x_\alpha.(\mu\alpha.[\alpha]M)\ x_\alpha =_\mu M$ by $\mu\eta$, and $\lambda\eta$.
Hence μ-names of type $T = A_1 \Rightarrow (A_2 \Rightarrow \ldots \Rightarrow (A_n \Rightarrow At)\ldots)$, can be replaced by a ground-type name, and a set of n variables, $x_1^\alpha : T_1, \ldots x_n^\alpha : T_n$. $\square$

With respect to call-by-value, (or call-by-name with disjunctive types) the situation is different: the above method of 'bootstrapping' first order $\lambda\mu$ to higher order gives terms which are type-correct, but which are not observationally equivalent to true higher-order μ-abstraction. (As pointed out to me by Hayo Thielecke.) Moreover, one can show by the fact that the fully abstract semantics of μPCF_v^1 and μPCF_v are inequivalent, that any such attempt at defining higher-order **call/cc** using first-order instances is doomed to failure.

4.3 Operational Semantics

The main difficulty with giving an operational semantics of μPCF is that it is necessary to evaluate terms 'inside' a μ-abstraction. Ong and Stewart described a one-step operational semantics of μPCF_v in [67] which is quite complicated (it requires two separate evaluation relations). The simplified semantics described here is based on evaluation-contexts. Thus it dovetails with the presentation of the equational theory of $\lambda\mu$. It is intuitive, as these contexts give an obvious notion of the 'next redex' to be reduced in evaluating a program. And it is general; adapted to both call-by-name and call-by-value semantics by the trade-off between the more restricted notion of β-reduction in call-by-value, and the richer notion of evaluation context that it allows.

Definition 4.3.1 (PCF **reductions)** *Let the small-step* PCF *reduction relation*

$\longrightarrow$, be given by the following rules:

$$(\beta) \quad (\lambda x.M)\ N \longrightarrow M[N/x]$$

$$\text{IF0} \ \ 0 \text{ then } M \text{ else } N \longrightarrow M$$

$$\text{IF0} \ \ \texttt{succ}\ n \text{ then } N \text{ else } N \longrightarrow N$$

$$\mathbf{Y}M \longrightarrow M\ (\lambda x.(\mathbf{Y}M)\ x)$$

$$\texttt{pred}\ (\texttt{succ}\ n) \longrightarrow n$$

$$\texttt{pred}\ 0 \longrightarrow \mathbf{Y}\lambda x.x$$

The call-by-value reductions are given by stipulating that N must be a value in the β-reduction clause.

The following proposition partially captures the way in which evaluation contexts 'pick out' the next reduction step.

Proposition 4.3.2 *Any term M of μPCF or $\mu\mathrm{PCF}^-$ (call-by-name or call-by-value) with no free variables (but possibly free names) is uniquely in one of the following forms:*
$M = n \mid \lambda x.N \mid E[\mu\alpha.N] \mid E[r] \mid E[[\alpha]V]$
where N is any term, r is a PCF*-redex, V is a value, and $E[\cdot]$ an evaluation context.*

Proof: is by structural induction on terms.
If $M = \lambda x.N$, $M = \mu\alpha.N$, $M = n$, $M = [\alpha]V$, or $M = \mathbf{Y}N$, then the hypothesis holds directly.
Otherwise, if $M = \text{IF0 } N \text{ then } L \text{ else } L'$, and N is either a numeral (so M is a PCF-redex, or $N = E[N']$, where N' is a naming, μ-abstraction, or PCF-redex, so $M = E'[N']$, is also in this form. The cases $M = \texttt{succ}\ N$, and $M = \texttt{pred}\ N$ are similar.
If $M = [\alpha]N$, where N is not a value, then N is $E[N']$ where N' is a named value, redex, or μ-abstraction. If $M = L\ N$, then it is necessary to distinguish call-by-name and call-by-value cases.
For call-by-name, either $L = \lambda x.L'$, and so M is a β-redex. Or $L = E[r]$ for some redex r, and so M is also of this form.
For call-by-value, suppose $L = \lambda x.L'$, then if N is a value, M is a β_v redex. Otherwise, $N = E[N']$, where N' is a named value, PCF-redex or μ-abstraction, in which case M is also of this form. If L is not a value, then by induction hypothesis, it is equal to $E[L']$, where L' is a named value, μ-abstraction or PCF-redex, and so M is of the same form. $\square$

Corollary 4.3.3 *μPCF programs (closed terms of type* `nat`*) are either numerals, or uniquely in one of the following 'reducible forms':*
$M = E'[\mu\alpha.N] \mid E[r] \mid \mu\alpha.[\alpha]n \mid \mu\alpha.E[\mu\beta.N] \mid \mu\alpha.E[r]$
where $E[\cdot]$ ranges over all evaluation contexts, and $E'[\cdot]$ only over non-trivial *ones, and r over* PCF *redexes.*

PROOF: By Proposition 4.3.2 above, M is either a numeral, or in the form $E[r]$ for a PCF redex r, or in the form $E[\mu\alpha.N]$ for some N, where $E[\cdot]$ is either trivial ($E[\cdot] = [\cdot]$) or non-trivial.
If $E[\cdot]$ is trivial, so $M = \mu\alpha.(N : \mathbf{0})$, then by Proposition 4.3.2 either $N = [\alpha]n$ or N is (uniquely) of the form $E[r]$ for a PCF redex r, or $N = E[\mu\beta.N']$. $\square$

Corollary 4.3.3 means that the operational semantics can be given in the following simple format, based on a combination of Plotkin's 'Structured Operational Semantics' [72], and Felleisen's use of unique decomposition of programs into a redex and an evaluation context, with one clause for each of the 'reducible forms'.

$$n \Downarrow n \qquad\qquad \mu\alpha.[\alpha]n \Downarrow n$$

$$\frac{r \longrightarrow s \quad E[s]\Downarrow n}{E[r]\Downarrow n} \qquad\qquad \frac{r \longrightarrow s \quad \mu\alpha.E[s]\Downarrow n}{\mu\alpha.E[r]\Downarrow n}$$

$$\frac{\mu\alpha.M[[\alpha]E'[\cdot]/\beta]\Downarrow n}{E'[\mu\beta.M]\Downarrow n} \qquad\qquad \frac{\mu\alpha.M[E[\cdot]/\beta]\Downarrow n}{\mu\alpha.E[\mu\beta.M]\Downarrow n}$$

($E'[\cdot]$ is non-trivial.)

Figure 4.2: One-step operational semantics of μPCF

(Note that the terms of μPCF[1] are closed under reduction, and that the first two rules in the left column give a sound and complete PCF semantics).

Proposition 4.3.4 (Determinism) *If $M \Downarrow$, (i.e. $M \Downarrow n$ for some n) then the derivation in the operational semantics is unique. Hence $M \Downarrow n$ implies $M \not\Downarrow n'$ for $n \neq n'$.*

PROOF: is direct by Corollary 4.3.3. $\square$

For the purposes of this thesis (considering fully abstract models), it is quite sufficient to consider only the evaluation of ground-type terms to numerals. Giving

an operational semantics of programs also has the advantage that all terms either converge to a value, or diverge (this is a corollary of computational adequacy). Significantly, this is not the case for higher-order terms of μPCF$_v$, — for terms of the form $\mu\alpha.[\alpha]\lambda x.N$, where α occurs in N (see [67]).
However, the operational semantics given here can be used to evaluate higher-order terms, (change the first rule in the right hand column to $\mu\alpha.[\alpha]V \Downarrow V$ *if* α *is not free in* V) and it is complete (in the sense that evaluation of any term which denotes a value terminates at avalue). Note also that the notion of observational equivalence used (program-type contextual equivalence), is equivalent to all-type contextual equivalence in an operational semantics based on evaluation of higher-type terms to values.

Proposition 4.3.5 *For terms* $M, N : T$,
$M \sqsubseteq_T^{OBS} N \iff \forall C^T[\cdot]\ C[M] \Downarrow V \implies C[N] \Downarrow V.$

PROOF: The implication from right to left is trivial, for the converse, suppose there is some higher-typed context $C[\cdot]$ such that $C[M] \Downarrow V$, and $C[N] \not\Downarrow V$, then $(\lambda x : S.0)\ C[M] \Downarrow 0$, and $(\lambda x : S.0)\ C[N] \not\Downarrow 0$, so $M \not\sqsubseteq_T^{OBS} N$ as required. □

The operational semantics is complete, and hence sufficient for the principal aim of proving full abstraction, however, it does not make any connection between the interactions of the game semantics, and the computation of programs. This possibility offers a new line of enquiry into the *dynamics* of computation with control, in light of the fact that the games models of μPCF are based on the CCC of one-move games. Danos, Herbelin and Regnier [21] have already given a close correspondence between interaction of strategies in both AJM and Hyland-Ong versions of these categories of games, and abstract machines for performing 'linear head reduction' in the λ-calculus. At a different level, Streicher and Reus [81] and Bierman [12] have suggested that an appropriate way of evaluating languages with control is to give an abstract machine which stores continuations (as evaluation contexts). Connecting these developments could be a way to optimize reduction strategies for languages which can be implemented using continuations, such as μPCF.

4.3.1 Expressing control in μPCF

Although μPCF is an 'idealized' language, it is suitable to represent a variety of control constructs based on continuation passing; this has been known since Parigot devised the $\lambda\mu$-calculus [69], and is a theme of the paper of Ong and

Stewart [67], in which μPCF is introduced. The call-with-current-continuation operator of Scheme, which is perhaps the most natural control extension of a functional language from a programming point of view, provides a useful example showing how μ-abstraction and naming correspond to 'catch and throw' of the current continuation. Parigot observed that the term of $\lambda\mu$ corresponding to a natural deduction proof of Peirce's law has the same operational behaviour as **call/cc** [69], and a more formal connection with the `callcc` of SML of New Jersey [35] is described in [67]. In the other direction, there is the question of whether $\lambda\mu$ terms can be written using **call/cc**. The result proved in [48], that unbracketed games give fully complete model of PCF + **call/cc** establishes that all *closed* $\mu\mathrm{PCF}_n^-$ terms are definable with **call/cc**. The definability of terms $\lambda\mu_v$ using $\mathcal{C}$ (Felleisen's 'idealized call/cc'), and vice-versa, is shown in Section 4.8.

Expressing the catch and error terms of Cartwright and Felleisen's SPCF establishes a link with the fully abstract semantics of this language in the category of sequential algorithms, which will be studied semantically in the next chapter. However, μPCF cannot express control operators such as the exception handler of ML, which use dynamic, rather than static variable bindings (such as μ-abstraction and λ-abstraction).

Definition 4.3.6 (Call-with-current-continuation) *The operational semantics of* PCF *can be extended with* **call/cc** *by evaluating terms within a context of 'continuation variables' bound to evaluation contexts (a similar semantics for* μPCF *is described by Bierman in [12]). The role of* **call/cc** *is to bind the current continuation to a variable; applying a continuation variable replaces the current continuation with the one which has been bound to the variable. As in Lemma 4.3.2, every λ-closed term is uniquely in the form $E[r]$, where r is the next redex to be evaluated, or $E[$***call/cc***$M]$, or $E[kM]$, where k is a continuation variable. So terms can be completely evaluated with the following rules.*

$$\frac{}{V,\emptyset \Downarrow_c V} \qquad \frac{r \longrightarrow s, \quad E[s],\mathcal{K}\Downarrow_c V}{E[r],\mathcal{K}\Downarrow_c V}$$

$$\frac{E[M],\mathcal{K}\cup\{k\to E[\cdot]\}\Downarrow V}{E[\mathbf{call/cc}\ \lambda k.M],\mathcal{K}\Downarrow_c V} \quad \frac{E[M],\mathcal{K}\Downarrow V, \quad k\to E[\cdot]\in\mathcal{K}}{E'[k\ \ M],\mathcal{K}\Downarrow_c V}$$

Definition 4.3.7 *For any term $M: (T \Rightarrow S) \Rightarrow T$,* $($**call/cc** $M)$ *will be written in* μPCF *as* $(\mu\alpha\colon T.[\alpha]M\ \ (\lambda y\colon T.\mu\beta\colon S.[\alpha]y))$.

Proposition 4.3.8 *Let M be any program of* PCF + **call/cc** *(interpreted in* μPCF *as defined above), then*

$$M,\varnothing \Downarrow_c \Leftrightarrow M \Downarrow_\mu$$

PROOF: To prove this claim, generalize it to include non-empty sets of continuation variables as follows.
Let M be a term such that there is an assignment $\mathcal{K}_M$ of appropriately typed evaluation contexts without free variables, $E_k[\cdot]$ to each of the continuation variables $k \in Cont(M)$.
Define the substitution $\mathcal{K}_\mu$ such that if $\mathcal{K}(k\colon T \Rightarrow S)) = E[\cdot]$ then $\mathcal{K}_\mu(k) = \lambda x\colon T.\mu\beta^S.[\alpha]E[x]$. Then

$$M, \mathcal{K} \Downarrow_c \iff \mu\alpha.[\alpha]M_{\mathcal{K}_\mu} \Downarrow_\mu .$$

Proof is by induction on the length of evaluation, of which the key induction steps are like this.

- Suppose $M = E[\mathbf{call/cc}\ \lambda k.s]$, then (proving the implication from left to right)
  $E[s], \mathcal{K} \cup (k \to E[\cdot]) \Downarrow_c$, with a shorter derivation, by definition of the evaluation relation for **call/cc**.
  Hence by hypothesis, $\mu\alpha.[\alpha]E[s[\lambda x.\mu\beta.[\alpha]x/k]]_{\mathcal{K}_\mu} \Downarrow_\mu$, so
  $\mu\alpha.[\alpha]E[\mu\gamma.[\gamma]\lambda k.s\ (\lambda x.\mu\beta.[\gamma]x)] \Downarrow_\mu$ as required.
  In the opposite direction, $\mu\alpha.[\alpha]E[s[k/\lambda x.\mu\beta.[\alpha]E[x]] \Downarrow_\mu$ by definition of $\Downarrow_\mu$ and by hypothesis $E[s]\mathcal{K} \cup k \to E[\cdot] \Downarrow_c$, and so $E[\mathbf{call/cc}\ \lambda k.s], \mathcal{K} \Downarrow_c$

- Suppose $M = E[k\ V]$ for some $k \in Cont(M)$ such that $\mathcal{K}(k) = E'[\cdot]$,
  then $E'[V], \mathcal{K} \Downarrow_c$ and so by hypothesis, $\mu\alpha.[\alpha]E'[V]_{\mathcal{K}_\mu} \Downarrow_\mu$,
  and so $\mu\alpha.[\alpha]E[\mu\beta.[\alpha]E'[V]]_{\mathcal{K}_\mu} \Downarrow_\mu$ and $\mu\alpha.[\alpha]E[k\ V]_{\mathcal{K}_\mu} \Downarrow_\mu$ as required.
  In the opposite direction:
  $\mu\alpha.[\alpha]E[\lambda x.\mu\beta.[\alpha]E'[x]V]_{\mathcal{K}_\mu} \Downarrow_\mu$ implies $\mu\alpha.[\alpha]E'[V]_{\mathcal{K}_\mu} \Downarrow_\mu$
  and by hypothesis, $E'[V], \mathcal{K} \Downarrow_c$, hence$E[k\ V], \mathcal{K} \Downarrow_c$

□

4.3.2 μPCF and SPCF

SPCF(E) is call-by-name PCF extended with a set of errors E (possibly empty), and a group of 'catch' constructs which take a term
$M\colon (S_1 \Rightarrow (S_2 \Rightarrow \ldots (S_n \Rightarrow \mathtt{nat}) \ldots))$ and return n if the first argument consulted by M is the nth, and $(t\ \Omega) + n + 1$ if M is non-strict. This procedure can be defined in μPCF, as

$$\mathbf{catch}^T = \lambda f\colon T.\mu\alpha\colon \mathtt{nat}.[\alpha](((f\ (\mu\beta.[\alpha]0)\mu\beta.[\alpha]1) \ldots \mu\beta.[\alpha]n) + (n+1)).$$

A control feature of SPCF of a somewhat different kind is the inclusion of constants for reporting run-time errors by terminating evaluation and returning an 'error value'. It is not possible to simulate errors directly in μPCF, as a whole new notion of evaluation is required, but it is possible to represent errors in $\mu\mathrm{PCF}^E$, which is a generalization of the errors of SPCF, as 'uncaught' continuations.

Definition 4.3.9 *Form the language $\mu\mathrm{PCF}^E$ (call-by-name or call-by-value μPCF with errors) by extending the syntax of μPCF with an ω-indexed set of error terms $\{\mathbf{e_i} \mid i \in \omega\}$.*
Errors are assumed to be polymorphic, taking any type (as they are intuitively of type $\forall X.X = \mathbf{0}$). The unique reducible forms Lemma 4.3.2 can be extended to show that a program of $\mu\mathrm{PCF}^E$ is uniquely in one of the the following forms

$$M = v \mid E'[\mu\beta.N] \mid E[r] \mid \mu\alpha.[\alpha]n \mid \mu\alpha.E[\mu\beta.N] \mid \mu\alpha.E[r] \mid E[\mathbf{e_i}] \mid \mu\alpha.[\alpha]E[\mathbf{e_i}]$$

where $E[\cdot]$ ranges over all evaluation contexts, and $E'[\cdot]$ only over non-trivial *ones.*
The range of the termination relation $\Downarrow$ is extended to include error-terms and the operational semantics can then be extended with the rules

$$\frac{}{E[\mathbf{e_i}] \Downarrow \mathbf{e_i}}$$

$$\frac{}{\mu\alpha.[\alpha]E[\mathbf{e_i}] \Downarrow \mathbf{e_i}.}$$

One can show with a routine induction that reduction of $\mu\mathrm{PCF}^E$ programs is deterministic, and either non-terminating, terminated by convergence to a value, or terminated by abortion to an error term.

Errors can be represented in μPCF itself by allowing 'closed' terms to contain a single free name ε of type `nat`, which can only be used as an error generator, writing $\mathbf{e_i} = \mu\alpha.[\varepsilon]\mathbf{2.i}$, where α is typed appropriately for the context in which the error appears. Hence for any evaluation context $E^T[\cdot]\colon \mathtt{nat}$,
$E[\mathbf{e_i}] =_{df} E[\mu\alpha.[\varepsilon]i] =_{\mu} \mu\alpha\colon \mathtt{nat}.[\varepsilon]2i =_{df} \mathbf{e_i}$.

Proposition 4.3.10 *This can be formalised as a translation $\mu\mathrm{PCF}^E$ terms in context $\Gamma \vdash t\colon A;\Delta$ to μPCF terms $\Gamma \vdash t^*;\Delta,\varepsilon$ such that $M =_{\mu(e)} N$ if and only $M^* = N^*$*

Evaluation of *Programs* (ground-type closed terms) with errors can be simulated by translating, and binding the free μ-variable as follows.

$$M \quad \longrightarrow \mu\varepsilon.[\varepsilon](\mathtt{succ}\ (2.M^*))$$

Proposition 4.3.11 *$M \Downarrow n$ if and only if $\mu\varepsilon.[\varepsilon](\texttt{succ}\ (2.M^*)) \Downarrow 2n+1$, and $M \Downarrow \mathbf{e_i}$ if and only if $\mu\varepsilon.[\varepsilon](\texttt{succ}\ (2.M^*)) \Downarrow 2i$.*

Errors can be represented semantically in much the same way as the $\perp$-elements were added to the free CCC. A semantics of μPCF based on names as continuation variables can be generalised to a semantics of μPCF^E by 'freely adjoining' a type-object representing a continution of type `nat` to Δ-contexts.

(In other words, attaching a set of 'top' elements to each domain: see Section 5.3.2.) A simple generalization of a context lemma for μPCF proved in Chapter 5 can be used to show a fact stressed by Cartwright, Curien and Felleisen, that operational equivalence in the presence of errors is order extensional (see Corollary 5.1.7). Error values alone are clearly less expressive than full **call/cc**: they allow the current continuation to be aborted ('thrown'), but there is no recovery, it cannot be 'caught'.

4.4 Completeness of the operational semantics

Like PCF (see [14]), the proof of adequacy of the μPCF semantics can be abstracted from any particular denotational framework, and based on soundness, together with two inequational conditions. In fact, these requirements are (unsurprisingly, but pleasingly) the same as those which have been used to axiomatize adequacy for PCF. A further pleasant feature of the dual purpose call-by-name/call-by-value operational semantics is that a single proof suffices to establish the completeness of the semantics for all of the variants of μPCF which have been defined.

The adequacy conditions must exclude the trivially sound model of μPCF in which all terms are identified. In other words, it is necessary for termination and non-termination to be distinguishable at ground type, which can be achieved with the following definition:

Definition 4.4.1 (Standard Datatypes) *A semantics for* μPCF *has standard datatypes if each numeral is interpreted as a distinct denotation $\overline{n}$, and $\perp_{[\![\texttt{nat}]\!]} \neq \overline{n}$. (In fact, in a sound model, this is equivalent to requiring only that $\perp_{[\![\texttt{nat}]\!]} \neq \overline{n}$ for all $\overline{n}$.)*

The following lemma partially formalises the intuition that any term appearing inside an evaluation context *must* be evaluated.

Lemma 4.4.2 (Strictness of evaluation contexts) *Let $\mathcal{M}$ be a semantics of* μPCF *(call-by-name or call-by-value), such that the $\mu\zeta$ rule is soundly interpreted* *in $\mathcal{M}$. Then for every* `nat`*-typed evaluation context $E^T[\cdot]$:* `nat`*:*

$$[\![E[\Omega^T]]\!]_{\mathcal{M}} = [\![\Omega^{\mathtt{nat}}]\!]_{\mathcal{M}}.$$

PROOF: Recall that Ω^T for $T \neq \mathbf{0}$ is defined to be $\mu\alpha^T.\Omega^{\mathbf{0}}$. By $\mu\zeta$, $[\![E^T[\mu\alpha.\Omega^{\mathbf{0}}]\!]_{\mathcal{M}} = [\![\mu\alpha : \mathtt{nat}.\Omega^{\mathbf{0}}]\!]_{\mathcal{M}} = [\![\Omega^{\mathtt{nat}}]\!]_{\mathcal{M}}$. □

A second condition (adapted from [14]) is based on the fact that the $\mathbf{Y}$ combinator is interpreted as a *least fixpoint*. Hence it can be 'unwound'.

Definition 4.4.3 (Continuous Observability) *A semantics of μPCF is continuously observable if for every program of the form $C[\mathbf{Y}M]$:* `nat`

$$[\![C[\mathbf{Y}M]]\!] \neq \bot \implies \exists k \in \omega\colon\ [\![C[M^k]]\!] \neq \bot$$

where $M^0 = \Omega$, and $M^{k+1} = M(\lambda x.(M^k x))$.
(So $M^{k+1} = M\ M^k$ in call-by-name μPCF.)

Lemma 4.4.4 *For all $k \in \omega$, and (closed) μPCF contexts $C^T[\cdot]$:* `nat`*,*
$C[M^k] \Downarrow n$ implies $C[\mathbf{Y}M] \Downarrow n$.

PROOF: is by induction first on k, and then on the length of derivation of $C[F^k] \Downarrow n$.
The important case is where $C[\cdot]$ is an evaluation context $E[\cdot]$. Then $E[M^{k+1}] \Downarrow n$ implies $E[M\ \lambda x.M^k\ x] \Downarrow n$, so $C'[M^k] \Downarrow n$, where $C'[\cdot] = C[M\ \lambda x.[\cdot]\ x]$, hence by inductive hypothesis, $E[M\ \lambda x.\mathbf{Y}M\ x] \Downarrow n$ thus $E[\mathbf{Y}M] \Downarrow n$. □

Theorem 4.4.5 *Let $\mathcal{M}$ be a sound denotational semantics for μPCF which is continuously observable, and has standard datatypes. Then the operational semantics of μPCF is computationally adequate with respect to $\mathcal{M}$ i.e.*
For all terms M of type `nat`*, $[\![M]\!] = \overline{n}$ if and only if $M \Downarrow n$.*

PROOF: The soundness part ($M \Downarrow n$ implies $[\![M]\!] = \overline{n}$) follows from a simple induction on the derivation of M, using the assumption that $\mathcal{M}$ is a sound model of the equational theory.
It remains to show completeness ($[\![M]\!] = \overline{n}$ implies $M \Downarrow n$). The following lemma shows that it is sufficient to consider terms without the $\mathbf{Y}$ combinator.

Lemma 4.4.6 *Suppose $\mathcal{M}$ is a continuously observable semantics which is complete for $\mathbf{Y}$-free terms, then $\mathcal{M}$ is complete for all terms.*

PROOF: that $[\![N]\!] = \overline{n}$ implies $N \Downarrow n$ is by induction on the number of occurrences of $\mathbf{Y}$ in N.
Assuming this is greater than zero, choose a context such that $N = C[\mathbf{Y}M]$, where M is $\mathbf{Y}$-free. By continuous observablility, there exists $k \in \omega$ such that $C[M^k] = \overline{n}$, and then by inductive hypothesis, $C[M^k] \Downarrow n$, and by Lemma 4.4.4 above, $C[\mathbf{Y}M] \Downarrow n$ as required. □

So it remains to prove completeness for recursion-free terms (as for PCF [71], by a now standard reducibility style argument based on a 'computability predicate'). This is complicated, however, by the fact that the operational semantics (unavoidably) allows reduction inside a μ-abstraction. So it is useful to distinguish a single name α of type `nat`, and define the notion of computability first for terms containing (at most) α free, which will be called α-closed.
Say that a term M of type $T_1 \Rightarrow (T_2 \ldots (T_n \Rightarrow \mathtt{nat}) \ldots)$, with no free variables or names (except α) is *computable* if whenever $N_1, \ldots N_k$ are computable α-closed terms,

$$[\![(\mu\alpha.[\alpha](MN_1, \ldots N_k)]\!] = \overline{n}$$

implies

$$\mu\alpha.[\alpha](MN_1, \ldots N_k) \Downarrow n$$

Define a *computable evaluation context* (call-by-name or call-by-value) to be an evaluation context $E[\cdot]$ of type `nat` with no free variables or names except α, such that every α-closed subterm of $E[\cdot]$ is computable.

Lemma 4.4.7 *An α-closed term M is computable if and only if $[\![\mu\alpha.E[M]]\!] = \overline{n}$ implies $\mu\alpha.E[M] \Downarrow n$ for all computable evaluation contexts.*

PROOF: The right to left implication is trivial, left to right can be proved by a simple induction on the structure of $E[\cdot]$. □

Definition 4.4.8 *For any term M, a computable variable-substitution for M is an assignment of computable α-closed terms to variables at each type T,*
a computable name-substitution for M is an assignment of a computable evaluation context $[\alpha]E^T[\cdot]$ to the free names of M (other than α) at each type T.
For each computable substitution σ of names and variables, M_σ is given by carrying out the replacements specified by σ.

Now say that term M with free variables and names is computable if M_σ is computable for every computable substitution σ such that M_σ is α-closed.

Proposition 4.4.9 *Every term of μPCF is computable.*

By the lemma above, this is equivalent to the proposition that for every term $M : T$, every computable substitution σ such that M_σ is α-closed, and every computable evaluation context $E^T[\cdot] : \mathbf{0}$, $[\![\mu\alpha.E[M_\sigma]]\!] = \overline{n}$ implies $\mu\alpha.E[M_\sigma] \Downarrow n$.

Proof: is by structural induction on M.
Let σ be any computable substitution for M, such that $[\![\mu\alpha.[\alpha].E[M_\sigma]]\!] = \overline{n}$
By Lemma 4.3.2 $\mu\alpha.E[M_\sigma]$ is in one of the 'reducible forms' (and one of the latter three). Hence it is (uniquely) covered by one of the following cases.

- $\mu\alpha.[\alpha].E[M_\sigma] = \mu\alpha.E'[\Omega]$ for some computable context $E'[\cdot]$. But in this case $[\![\mu\alpha.[\alpha].E[M_\sigma]]\!] = \bot_n$ by strictness of evaluation contexts, contradicting the hypothesis above.

- If $\mu\alpha.[\alpha].E[M_\sigma] = \mu\alpha.[\alpha]n$, $\mu\alpha.[\alpha].E[M_\sigma] \Downarrow n$ as required.

- If $\mu\alpha.E[M_\sigma] = \mu\alpha.E'[r]$, for a computable context $E'[\cdot]$, and PCF-redex r, which is either `pred` (`succ` n), IF0 n then N else N', or a β redex. The former case is trivial.
  If r is a conditional, it reduces to N or N', both of which are computable by the induction hypothesis as they are subterms of M_σ.
  In case r is a β redex, it reduces to a computable substitution $r' = N[N'/x]$ which is therefore computable.
  So $\mu\alpha.[\alpha]E'[r'] \Downarrow n$, and also $\mu\alpha.[\alpha]E'[R] \Downarrow n$ as required.

- If $\mu\alpha.E[M_\sigma] = \mu\alpha.E'[\mu\beta.N]$, for some non-trivial evaluation context $E'[\cdot]$, then N is shorter than M_σ, and computable by inductive hypothesis.
  $[\![\mu\alpha.E'[\mu\beta.N]]\!] = \overline{n} = \mu\alpha.N[E'[\cdot]/\beta]$, by soundness. Since N is computable, and $[E'[\cdot]/\beta]$ is a computable substitution,
  so $\mu\alpha.N[E'[\cdot]/\beta] \Downarrow n$, and hence $\mu\alpha.E'[\mu\beta.N] \Downarrow n$ as well.

$\square$

To prove the completeness theorem, a simple lemma can first be established by induction on the length of evaluation.

Lemma 4.4.10 *For any closed term M, if $\mu\alpha.[\alpha]M \Downarrow n$ then $M \Downarrow n$.*

Proof of the completeness theorem
Suppose M is a closed term such that $[\![M]\!] = \overline{n}$. Then $[\![\mu\alpha.[\alpha]M]\!] = \overline{n}$ and so by computability, $\mu\alpha.[\alpha]M \Downarrow n$, and by the above lemma $M \Downarrow n$ as required.

Corollary 4.4.11 *Any sound model of* PCF + **call/cc** *with standard datatypes and continuous observability is computationally adequate.*

4.5 Denotational semantics of μPCF via cps translation

It can now be shown that $\mu\mathrm{PCF}_n$ can be adequately interpreted in a computational control model, along the lines outlined in Chapter 2. That is, given a pointed and rational cartesian closed category with ω-indexed products, and a non-terminal answer object R, form the co-product completion $\mathsf{Fam}(\mathcal{C})$, and strong monad of continuations $\mathbf{T} = ((\_ \Rightarrow \mathbf{a}) \Rightarrow \mathbf{a})$, and interpret product and function types as product and exponential in $\mathsf{Fam}(\mathcal{C})$, and co-product types $A+B$ as $\mathbf{T}([\![A]\!] + [\![B]\!])$. Naming and μ-abstraction are interpreted as application and abstraction of continuation variables. Note, however, that only $\mathbf{T}(\{\}) \cong \mathbf{a}$, interpreting the empty type, and $\mathbf{T}(\{\mathbf{1}_n \mid n \in \omega\} \cong \mathbf{a}^\omega \Rightarrow \mathbf{a}$, interpreting `nat` are required for the full abstraction result. This yields a rational CCC $head, tail, ::$ (and hence every embedding-projection pair e_i, p_i) is definable.

The call-by-name cps translation of $\mu\mathrm{PCF}_n$ into the simply-typed λ-calculus with ω-indexed products and least fixed points is probably the simplest route to this interpretation. It is described here both for its intrinsic interest (it is fully abstract), and because it provides a clearer way to understand naming and μ-abstraction as catch and throw of continuations. The role of names in call-by-name $\lambda\mu$ is *more* subtle than in call-by-value; in the latter, they can be replaced by lambda-variables of 'negated type' (see Section 4.8.1), together with a double negation elimination. The call-by-name cps translation is simpler, however, as μ-abstraction and naming need be defined at ground type only.

To interpret infinite datatypes such as the natural numbers by cps translation requires infinite datatypes in the target language: it is sufficient to add a type constructor $(\_)^\omega$ (T^ω being the infinite streams, or lists or products of type T), and its constructor ::, and deconstructors *head* and *tail*, to the simply-typed λ-calculus, to form the infinitary calculus $\Lambda(\Omega)^\omega$.

Definition 4.5.1 *Types and terms of $\Lambda(\Omega)^\omega$ are given by the following grammar:*
$T ::= \iota \mid T^\omega \mid S \Rightarrow T$
$t := x : T \mid (\lambda x : S.t : T) : S \Rightarrow T \mid (t : S \Rightarrow T)\ (s : S) : T \mid \ \ t : T{::}s : T^\omega : T^\omega \mid head(t\colon T^\omega)\colon T \mid tail(t\colon T^\omega)\colon T^\omega \mid \mathbf{Y}(t\colon T \Rightarrow T)\colon T.$
The equational theory of $\Lambda(\Omega)^\omega$ is given by the standard call-by-name $\beta\eta$ equality, and the following rules for lists and fixpoints:

- $head(t{::}s) =_{\Lambda(\Omega)^\omega} t$
- $tail(t{::}s) =_{\Lambda(\Omega)^\omega} s$

- $\mathbf{Y}f =_{\Lambda(\Omega)^\omega} f(\mathbf{Y}f)$.

Definition 4.5.2 *The following familiar operations on infinite lists can be defined:*
For each type T there is a diverging element of type T, $\Omega^T = \mathbf{Y}(\lambda x\colon T.x)$.
Finite products $\langle t_1, \ldots, t_n\rangle$ are defined as $t_1{::}t_2{::}\ldots t_n{::}\Omega^{\iota^\omega}$
and projections are defined by iterating the tail-operation: $\pi_i t = head(tail^{i-1}(t))$.
For any $t\colon \iota$, an infinite list of copies of t, $\delta(t)$ can be defined using the recursion combinator: $\delta(t) = \mathbf{Y}(\lambda x\colon \iota^\omega.t{::}x)$.

Proposition 4.5.3 *The following equalities can be derived from the theory of $\Lambda(\Omega)^\omega$:*
$\pi_i(\langle t_1, t_2, \ldots t_n\rangle) =_\pi t_i$,
$head(\delta(t)) = t$,
$tail(\langle t_1, t_2, \ldots t_n\rangle) = \langle t_2, \ldots, t_n\rangle$.

It is clear that any rational pointed cartesian closed category with ω-indexed products $(\mathcal{C}, [\![\iota]\!])$ is a sound model of $\Lambda(\Omega)^\omega$, and in particular that:

Proposition 4.5.4 *The model of $\Lambda(\Omega)^\omega$ in the category of unbracketed games, $(\mathcal{G}, o)$ is sound.*

4.5.1 Continuation passing translation of $\mu\mathrm{PCF}_n$

This can be seen as an application to μPCF of previous work on the pure calculus. The novel feature here is the use of infinite lists to model continuations of type `nat`. de Groote [23] described a wholly syntactic continuation-passing-interpretation of the $\lambda\mu_n$-calculus which is similar to the one given here. Hofmann and Streicher [40] gave a continuation semantics for $\lambda\mu_n$, and showed that it is universal in that any $\lambda\mu_n$ model is isomorphic to such an interpretation (see the discussion in Remark 4.5.10 below).

Definition 4.5.5 (CPS translation of call-by-name μPCF) *The translation on types is given by*

- $\mathbf{0}^\Box = \iota$
- $\mathtt{nat}^\Box = \iota^\omega \Rightarrow \iota$
- $(S \Rightarrow T)^\Box = S^\Box \Rightarrow T^\Box$

Note that to translate μPCF$_n$ *requires only* ω*-indexed products in* $\Lambda(\Omega)^\omega$ *at ground-type, (written* $\Lambda(\Omega)^\omega_1$*).*
Terms-in context are translated by assuming a correspondence between μPCF *variables* $x\colon T$ *and* $\Lambda(\Omega)^\omega$ *variables* $x^\Box\colon T^\Box$*, and between names* $\alpha : \mathtt{nat}$*, and* $\Lambda(\Omega)^\omega$*-variables* $\alpha^\Box\colon \iota^\omega$*.*
So $(\Gamma \vdash t;\Delta)^\Box = \Gamma^\Box, \Delta^\Box \vdash t^\Box$*, where* $t^\Box$ *is defined by induction:*

- $(x)^\Box = x^\Box$
- $(\lambda x.M)^\Box = \lambda x^\Box.M^\Box$
- $(M\ N)^\Box = M^\Box\ N^\Box$
- $0^\Box = \lambda x\colon \iota^\omega.head(x)$
- $(\mathtt{succ}\ M)^\Box = \lambda x\colon \iota^\omega.M^\Box\ (tail(x))$
- $(\mathtt{pred}\ M)^\Box = \lambda x\colon \iota^\omega.M^\Box\ (\Omega{::}x)$
- $(\mathrm{IF0}\ M\ \mathrm{then}\ L\ \mathrm{else}\ N)^\Box = M^\Box\ (L^\Box{::}\delta(N^\Box))$
- $(\mathbf{Y}M)^\Box = \mathbf{Y}M^\Box$
- $([\alpha]M)^\Box = M^\Box\ (\alpha^\Box)$
- $(\mu\alpha.M)^\Box = \lambda\alpha^\Box.M^\Box$

The interpretation of terms of μPCF$_n$ in the games model can now be given via translation into $\Lambda(\Omega)^\omega$, and interpretation in $\mathcal{G}$; i.e.

$$[\![M]\!]_\mathcal{G} = [\![M^\Box]\!]_\mathcal{G}$$

The well-bracketed model of PCF embeds into the semantics of μPCF$_n$ as described in Chapter 3.

Proposition 4.5.6 *The denotations of the* PCF *types (as games) and* PCF *terms (as innocent functions) under the above interpretation are play-equivalent to those in the Hyland-Ong model of* PCF *without the bracketing condition.*

The proof of soundness of the translation uses the following lemma.

Lemma 4.5.7 *For every* $\mathbf{0}$*-typed,* $\mathtt{nat}$*-holed* μPCF$_n$ *evaluation context* $E^{\mathtt{nat}}[\cdot] : \mathbf{0}$*, there is a term* $N : \iota^\omega$ *of* $\Lambda(\Omega)^\omega$ *such that for any* μPCF *term* $M : \mathtt{nat}$*,* $E[M]^\Box =_{\Lambda(\Omega)^\omega} M^\Box\ N$*.*

PROOF: is by induction on μPCF$_n$-evaluation contexts. There are two base cases, the conditional and naming:
(IF0 M then L else $N)^\Box = M^\Box\ L^\Box{::}\delta(N^\Box)$
$([\alpha]M)^\Box = M^\Box\ \alpha^\Box$
For the inductive step in which $E^{\texttt{nat}}[\cdot] : \mathbf{0} = E'[\texttt{succ}\ [\cdot]]$ for some smaller $E'[\cdot]$, the induction hypothesis gives $N' : \iota^\omega$ such that $E'[M']^\Box = M'^\Box\ N'$ for any M'.
Let $N = \mathit{tail}(N')$, then $E'[\texttt{succ}\ M]^\Box =_{\Lambda(\Omega)^\omega} (\texttt{succ}\ M)^\Box\ N'$
$= (\lambda z : \iota^\omega.M^\Box\ \mathit{tail}(z))\ N' =_{\Lambda(\Omega)^\omega} M^\Box\ N$ as required.
The case $E[\cdot] = E'[\texttt{pred}\ \cdot]$ is similar. $\Box$

Lemma 4.5.8 *If $E^{\texttt{nat}}[\cdot] : \mathbf{0}$ is a μPCF$_n$ evaluation context, and N the associated term of $\Lambda(\Omega)^\omega$ such that for all $M : \texttt{nat}$, $E[M]^\Box =_{\Lambda(\Omega)^\omega} M^\Box N$, then for all μPCF terms L, $L[E[\cdot]/\alpha]^\Box = L^\Box[N/\alpha^\Box]$.*

PROOF: is by induction over the structure of L, for example,
$([\alpha]M[E[\cdot]/\alpha])^\Box = E[M[E[\cdot]/\alpha]]^\Box$
$= (M[E[\cdot]/\alpha])^\Box N = (M^\Box\ \alpha^\Box)[N/\alpha^\Box] = ([\alpha]M)^\Box[N/\alpha^\Box]$. $\Box$

Proposition 4.5.9 *The translation is sound with respect to the equational theories of μPCF$_n$ and $\Lambda(\Omega)^\omega$.*

PROOF: The $\beta\eta$ equalities are preserved by the translation, and it is easy to check that the Peano axioms and equalities for the conditional are satisfied.
$\mu\eta :\quad (\mu\alpha.[\alpha]M)^\Box =_\mu \lambda\alpha^\Box.M^\Box\ \alpha^\Box = M^\Box$
The $\mu\zeta$ rule translates to a β-substitution: by Lemma 4.5.7 above,
$(E[\mu\alpha.M] : \mathbf{0})^\Box =_{\Lambda(\Omega)^\omega} \lambda\alpha^\Box.M^\Box\ N =_{\Lambda(\Omega)^\omega} M^\Box[N/\alpha^\Box]$.
By Lemma 4.5.8, $M^\Box[N/\alpha^\Box] = M[E[\cdot]/\alpha]^\Box$ as required. $\Box$

Remark 4.5.10 (Relation to Hofmann-Streicher Models of $\lambda\mu$) *By the completeness theorem of Hofmann and Streicher mentioned above, every sound model of $\lambda\mu$ is isomorphic to a category of continuations constructed from a category $\mathcal{C}$ as follows: select an answers object R from $\mathcal{C}$.*
For each type T, define the continuations of type T as
$[\![\mathbf{0}]\!]_c = \mathbf{1}_\mathcal{C}$, *and* $[\![S \Rightarrow T]\!]_c = R^{[\![S]\!]_c} \times [\![T]\!]_c$.
(Leaving open the interpretation of continuations of ground types other than $\mathbf{0}$.)
The denotations *of type T are given in $R^{[\![T]\!]_c}$ (So that a denotation corresponds to a map from a continuation to an answer, and a continuation at arrow type to pair of a denotation of argument type and a continuation of result type.)*
Hence for a PCF*-type $T = A_1 \Rightarrow (A_2 \Rightarrow \ldots (A_n \Rightarrow \texttt{nat})\ldots)$,*

$[\![T]\!]_c \cong [\![A_1]\!]_D \times \ldots \times [\![A_n]\!]_D \times [\![\mathtt{nat}]\!]_c$,
$[\![T]\!]_D \cong R^{[\![A_1]\!]_D \times \ldots \times [\![A_n]\!]_D \times [\![\mathtt{nat}]\!]_c} \cong ([\![A_1]\!]_D \times \ldots \times [\![A_n]\!]_D) \Rightarrow [\![\mathtt{nat}]\!]_D$.
So denotations at PCF*-types are given the standard interpretation, and a continuation at higher type is given by a product of denotations with a continuation at ground type. This is how higher-type names are interpreted using the 'bootstrapping' described in Proposition 4.2.9.*

Another perspective on the semantics of call-by-name $\lambda\mu$ is given by Ong in 'A Semantic view of Classical Proofs' [66]. This paper describes categorical and game semantic notions of $\lambda\mu$-model, both of which are quite different to anything which appears here. The categorical model characterizes the operation of 'mixed substitution' directly, rather than by continuation-passing. Ong's '$\lambda\mu$-categories', are split fibrations; CCC's (i.e. λ-models) fibred over cartesian products representing μ-contexts. (So continuations models are a trivial example of such a fibration, in which the fibre for each μ-context is the CCC with the corresponding context of continuation variables adjoined to it.) Because of the various completeness results for cps semantics (such as [40]) it is known that every $\lambda\mu$-category can be written as a category of continuations in this way.

The relationship of the games semantics of $\lambda\mu_n$ in [66] to the unbracketed one is less clear. Ong's model retains the well-bracketing condition, but adds a notion of state via 'scratchpads'. These allow Player to give 'dummy' answers to questions which he would just skip over in the corresponding unbracketed strategy. It seems fair to say that the unbracketed games model, based on relaxing conditions rather than adding structure, is a simpler category; it has also been possible to connect local and non-local control semantically in a natural way. However the extra structure of scratchpads and control questions could prove useful in studying control-flow in games, using information which could be extracted from a formal connection between the games via the relationship between the corresponding categorical models.

Proposition 4.5.11 *If $(\mathcal{C}, \mathbf{a})$ specifies a computational model of control (i.e. $\mathcal{C}$ is a rational CCC with ω-indexed products, and* $\mathbf{a}$ *is non-terminal) then the semantics of $\mu\mathrm{PCF}_n$ in $(\mathcal{C}, \mathbf{a})$ given by translation into $\Lambda(\Omega)^\omega$ is computationally adequate with respect to the operational semantics.*

Proof: The semantics in $\mathcal{G}$ meet the adequacy criteria stated in Theorem 4.4.5, as the translation into $\Lambda(\Omega)^\omega$ is sound, and $\mathcal{G}$ has standard datatypes and is continuously observable.

Datatypes are standard Suppose $\mathbf{a}$ is non-terminal: then $\perp_{\mathbf{a},\mathbf{a}} = t_{\mathbf{a}}; \perp_{\mathbf{a}} \neq \mathtt{id}_{\mathbf{a}}$.

Hence for every $i \in \omega$, $[\![n]\!] = `\pi_n' : \mathbf{a}^\omega \Rightarrow \mathbf{a} \neq `\perp_{\mathbf{a}^\omega,\mathbf{a}}'$.

Continuous observability If $\mathcal{C}$ is rational, then the semantics of $\mu\mathrm{PCF}_n$ given by $(\mathcal{C}, \mathbf{a})$ is continuously observable.

Given a closed term of ground type, $C[\mathbf{Y}(M : T \Rightarrow T)]$,

$C[\mathbf{Y}M] = [\![\lambda x.C[x\ M]\ (\lambda y : T \Rightarrow T.\mathbf{Y}y)]\!] = F^{\nabla}; [\![\lambda x.C[x\ M]]\!]$.

(where $F\colon [\![((T \Rightarrow T) \Rightarrow T))]\!] \to [\![(T \Rightarrow T) \Rightarrow T))]\!] = [\![\lambda H.\lambda f.f(Hf)]\!]$)

By rationality, $C[\mathbf{Y}M] = \bigsqcup_{i\in\omega} F^i; [\![\lambda x.C[x\ M]]\!] = \bigsqcup_{i\in\omega}[\![C[M^i]]\!]$.

Hence if $C[\mathbf{Y}M] \neq \perp$, there exists $k \in \omega$ such that $C[M^k] \neq \perp$, as required.

□

4.6 Definability and full abstraction in $\mu\mathrm{PCF}_n$

Definability of finitary morphisms now follows from the axiomatic characterization of the category of unbracketed games and finite strategies as a fully complete model of $\Lambda(\Omega)$. It is also, trivially, a fully complete model of the $\lambda\mu_n(\Omega)$-calculus over the empty type $\mathbf{0}$, because by Lemma 4.2.9, naming and μ-abstraction at all types can be simulated using λ-variables. The extension to the infinitary types are given by the numerals and `case` statements.

Definability for $\mu\mathrm{PCF}_n$ can be established in greater detail by describing an inverse to the □ translation with respect to finitary evaluation trees. Recall (Proposition 2.5.9) that all definable morphisms of a computational control model $(\mathcal{C}, \mathbf{a})$ are least upper bounds of the form $\bigsqcup_{i\in\omega}(p_i^A; f_i; e_i^B)$, where each $f_i : A_i \to B_i$ is a definable morphism of the free CCC over $\mathbf{a}$. In conjunction with the unique representation of morphisms in the CCC as $\beta\eta$-long forms of $\Lambda(\Omega)$, this gives a unique representation of a finitary basis of a model of $\Lambda(\Omega)^\omega$.

Definition 4.6.1 *The finitary normal forms of $\Lambda(\Omega)^{\omega}{}_1$ are given by the following inductive definition.*

$$\frac{}{\Omega \in N(\Gamma; \iota)}$$

$$\frac{t \in N(\Gamma, x\colon S; T)}{\lambda x.t \in N(\Gamma; S \Rightarrow T)}$$

$$\frac{x : \iota^\omega \in \Gamma}{\pi_n(x) \in N(\Gamma; \iota)}$$

$$\frac{s_1, s_2, \ldots, s_n \in N(\Gamma; \iota)}{\langle s_1, s_2, \ldots, s_n\rangle \in N(\Gamma; \iota^\omega)}$$

$$\frac{s_i \in N(\Gamma; S_i) \quad x\colon S_1 \Rightarrow (S_2 \Rightarrow \ldots S_n \Rightarrow \iota)}{(\ldots((x\ s_1)\ s_2)\ldots) \in N(\Gamma; \iota)}$$

Proposition 4.6.2 *If $(\mathcal{C}, \mathbf{a})$ is a model of $\Lambda(\Omega)^\omega$, every finitary definable morphism is the interpretation of a (unique) finitary normal form.*

PROOF: is by a straightforward structural induction, proving that every denotation of a finite evaluation tree is equal to $e_i^A; f_i; p_i^A$, where f_i is the denotation of a unique $\beta\eta\pi$-long normal form of $\Lambda(\Omega)$ (given by Definition 3.7.4 extended with finite products at ground type). $\square$

The Definability Theorem (Theorem 3.7.2) then yields the following corollary.

Corollary 4.6.3 *If $(\mathcal{C}, \mathbf{a})$ defines a computational control model, and satisfies the axioms for definability, then every finitary morphism of $(\mathcal{C}, \mathbf{a})$ is the interpretation of a finitary normal form of $\Lambda(\Omega)^\omega$.*

The evaluation trees of $\mu\mathrm{PCF}_n$ follow essentially the same pattern.

Definition 4.6.4 (Evaluation trees of $\mu\mathrm{PCF}_n$) *The set of evaluation trees $E(\Gamma, \Delta; T)$ of type T over the contexts $\Gamma = x_1 : T_1, \ldots, x_n : T_n, \Delta = \alpha_1 : \mathtt{nat}, \alpha_2 : \mathtt{nat}, \ldots \alpha_n : \mathtt{nat}$ is generated (up to α-equivalence) by the following inductive definition:*

$$\frac{}{\Omega \in E(\Gamma, \Delta; \mathbf{0})}$$

$$\frac{\alpha \in \Delta}{[\alpha]n \in E(\Gamma, \Delta; \mathbf{0})}$$

$$\frac{M \in E(\Gamma, \Delta, \alpha; \mathbf{0})}{\mu\alpha.M \in E(\Gamma, \Delta; \mathtt{nat})}$$

$$\frac{M \in E(\Gamma, x : T; U)}{\lambda x : T.M \in E(\Gamma; T \Rightarrow U)}$$

$$\frac{M_1, M_2, \ldots, M_n \in E(\Gamma, \Delta; T_i), \quad f : (T_1 \Rightarrow (\ldots (T_n \Rightarrow \mathbf{0}) \ldots)) \in \Gamma}{(((f\ M_1)\ M_2)\ \ldots)\ M_n \in E(\Gamma, \Delta; \mathbf{0})}$$

$$\frac{M_i \in E(\Gamma, \Delta; T_i) : i \leq n, \quad f : \overline{T} \Rightarrow \mathtt{nat}) \in \Gamma, \quad N_i \in E(\Gamma, \Delta; \mathbf{0}) : i \leq k}{\mathtt{case}\ _k((((f\ M_1)\ M_2)\ \ldots)\ M_n|_i N_i \in E(\Gamma, \Delta; \mathbf{0})}$$

For the evaluation trees of $\mu\mathrm{PCF}_n^-$, delete the fifth clause.

Proposition 4.6.5 *Suppose $t \in E(\Gamma^\Box, \Delta^\Box; T^\Box)$ is a finitary normal form of $\Lambda(\Omega)^\omega$, over a context of translated variables and names. Then there is a unique finitary evaluation tree of $\mu\mathrm{PCF}_n$ $M \in E(\Gamma, \Delta; T)$ such that $M^\Box =_{\Lambda(\Omega)^\omega} t$.*

PROOF: is by induction on finite normal forms of $\Lambda(\Omega)^\omega$, which is routine as there is a direct correspondence with evaluation trees, as follows.

- $\Omega^{\iota} = (\Omega^{\mathbf{0}})^{\Box}$
- $y^{\Box}\colon \iota = (y\colon \mathbf{0})^{\Box}$
- $\lambda y^{\Box}.M = (\lambda y.M)^{\Box}$
- $\pi_n(\alpha\colon \iota^{\omega})^{\Box} =_{\Lambda(\Omega)^{\omega}} ([\alpha]n)^{\Box}$
- $\lambda\alpha^{\Box}.M^{\Box} = (\mu\alpha.M)^{\Box}$
- $(((y^{\Box}\ M_1^{\Box})\ M_2^{\Box})\ldots)\ M_n^{\Box} =_{\Lambda(\Omega)^{\omega}} ((((y\ M_1)\ M_2)\ldots)\ M_n)^{\Box}$
- $(((y^{\Box}\ M_1^{\Box})\ M_2^{\Box})\ \ldots)\ M_n^{\Box}\langle N_1^{\Box}, N_2^{\Box}, \ldots N_k^{\Box}\rangle$
  $= (\mathtt{case}\ _k((((y\ M_1)\ M_2)\ldots)\ M_n)|_i N_i)^{\Box}$

$\Box$

Corollary 4.6.6 *Every finite morphism in the games model of μPCF$_n$ (and any model constructed from $(\mathcal{C}, \mathbf{a})$ satisfying the axioms for definability) is the denotation of a unique evaluation tree.*

Theorem 4.6.7 (Full abstraction for μPCF$_n$) *Any model of control specified by a category and answer-object $(\mathcal{C}, \mathbf{a})$ satisfying the axioms for definability will collapse under its intrinsic preorder to give a fully abstract model.*

Proof: by Corollary 4.6.6 above, any such model will have the definability property, which by Proposition 2.4.11 means that its collapse will be fully abstract for closed terms. It remains only to show that full abstraction for closed terms extends to full abstraction for all terms. $\Box$

Lemma 4.6.8 *Suppose full abstraction holds for all closed terms; i.e for closed $s, t\colon A$, $s \sqsubseteq^{obs} t$ if and only if $[\![s]\!] \sqsubseteq_A^{OBS} [\![t]\!]$. Then full abstraction holds for all terms.*

Proof: It is observed in [6], for example, that it is sufficient to prove soundness and completeness for terms without free variables, as given any terms $M, N : T$ with free variables $\Gamma = x_1\colon A_1, x_2\colon A_2, \ldots, x_n\colon A_n$, then
$[\![\Gamma \vdash M]\!] \lesssim_{[\![\Gamma]\!] \Rightarrow [\![T]\!]} [\![\Gamma \vdash M]\!]$ if and only if $[\![\lambda\overline{x}.M]\!] \lesssim_{[\![\Gamma]\!] \Rightarrow [\![T]\!]} [\![\lambda\overline{x}.N]\!]$, and
$M \sqsubseteq_{\Gamma;T}^{OBS} N$ if and only if $\lambda\overline{x}.M \sqsubseteq_{\Gamma;T}^{OBS} \lambda\overline{x}.N$.
Here, it is shown how this observation extends to free names.
Suppose that $\Gamma \vdash M : A, \beta\colon B, \Delta$ and $\Gamma \vdash N : A, \beta\colon B, \Delta$ are terms in context with a free name $\beta\colon B$.

Let $x : A \Rightarrow \mathbf{0}$ be a variable not free in s, t, and define
$M' = \lambda x\colon A \Rightarrow \mathbf{0}.\mu\beta.(x\ M)$ and $N' = \lambda x\colon A \Rightarrow \mathbf{0}.\mu\beta.(x\ N)$
Then $M \sqsubseteq^{OBS}_{\Gamma;\Delta,B;A} N$ if and only if $M' \sqsubseteq^{OBS}_{\Gamma;\Delta;A\Rightarrow\mathbf{0}\Rightarrow B} N'$.
The implication from right to left is obvious; to prove the converse, suppose that $M' \sqsubseteq^{OBS}_{\Gamma;\Delta;\Rightarrow A} N'$, and $C[\cdot]$ is a program context such that $C[M] \Downarrow$.
Then let $D[\cdot] = C[[\cdot]\ \lambda z\colon.[\beta]z]$, so that
$[\![D[M']]\!] = [\![C[M[\lambda y.[\beta]y/\beta]]]\!] = [\![C[M]]\!]$, and $D[M'] \Downarrow$ by computational adequacy, hence $D[N'] \Downarrow$ and $C[N] = D[N']$, so $C[N] \Downarrow$ as required. $\square$

Hence another example of a fully abstract model of $\mu\mathrm{PCF}_n$ is given by dropping the bracketing condition on the Abramsky-Jagadeesan-Malacaria games model of PCF, since this coincides with the semantics given by translation into $\Lambda(\Omega)^\omega$, and interpreting ι as the one-move game. Because these games satisfy the axioms for definability, this model will also give rise, by collapse, to a fully abstract model.

4.7 Call-by-value: higher-order control

Call-by-value μPCF can be soundly interpreted in a control model, giving a fully complete games model, and a fully abstract translation into the simply typed λ-calculus, just as for call-by-name. However, although the continuation-passing interpretation based on the monad is in principle simple, translation of $A \Rightarrow_v B$ as $A \Rightarrow_n ((B \Rightarrow_n \iota) \Rightarrow_n \iota)$ has the effect of radically increasing the depth of the types concerned, making the direct translation into the λ-calculus less than transparent. What is required is a set of higher-level reasoning principles to structure the translation.

As remarked by Moggi [62], control flow is the key to a general system of reasoning about function-based programming; in the presence of phenomena such as partiality, state, non-determinism, exceptions, etcetera, the order in which computation occurs is critical in determining its result. Monads provide a way to structure this information. A variant of one of Moggi's metalanguages for reasoning about the semantics of call-by-value functional langauges in general Kleisli categories, — the 'computational λ-calculus' has been described in Chapter 2. By making the typing of the continuations monad explicit, a 'control' version of λ_c can be described (which is effectively Felleisen's $\lambda\mathcal{C}$ [25]). $\lambda\mu_v$ can be translated soundly and invertibly into this calculus, so its interpretation in models of control is sound and complete. Using the computational λ-calculus to structure the cps translations of $\lambda\mathcal{C}$ and $\lambda\mu$ thus has two advantages:
it comes with a complete set of high-level reasoning principles, which yield a sim-

$\texttt{let } x = t \texttt{ in } x$	$=_c$	t
$\texttt{let } y = (\texttt{let } x = t_1 \texttt{ in } t_2) \texttt{ in } s$	$=_c$	$\texttt{let } x = t_1 \texttt{ in } (\texttt{let } y = t_2 \texttt{ in } s)$
$\texttt{let } x = v \texttt{ in } t$	$=_c$	$t[v/x]$
$\mu(t)$	$=_c$	$\texttt{let } \ x = t \ in \ \ \mu(x)$
$\mu([t])$	$=_c$	t
$[\mu(x)]$	$=_c$	x
$\lambda x.t \ s$	$=_c$	$\texttt{let } x = s \texttt{ in } t$
$(\lambda x.v \ (x))$	$=_c$	v

Figure 4.3: Equational theory in λ_c

ple equational proof that the intended model of $\mu\mathrm{PCF}_v$ is sound,
it also allows the project described in Chapter 3, of giving fully abstract semantics to a variety of computational features via the intensional hierarchy, to be connected to this elegant and well-established paradigm for modelling such side-effects using monads.

There is in fact quite a venerable tradition of using the monadic nature of continuations to study them; Moggi gave continuations as one of the first examples of a computational monad, and Hatcliff and Danvy [36] used Moggi's *computational metalanguage* (not the computational λ-calculus) as a unifying framework for factoring the cps translation, but not control operators. It has also been known [27],[39] that the action of the μ-transformation on continuation models is that of a control operator such as Felleisen's $\mathcal{C}$. In this section this observation is combined with the power of `let` -based reasoning about the flow of control to describe the semantics and theory of $\lambda\mu$, and show that it too can prove stronger equalities for purely functional terms than $\beta\eta$-equality alone.

4.7.1 Control in the computational lambda-calculus

Recall the language λ_c defined in section 2.2.2.

Definition 4.7.1 *A value of λ_c is a variable, or λ-abstraction, or lifting, i.e.*

$$v ::= x \mid \lambda x.t \mid [t]$$

where t ranges over all terms of λ_c.

The equational theory of $=_c$ is given by the reflexive, transitive, compatible closure of the rules in table 4.3.

Proposition 4.7.2 *The above theory is sound with respect to the interpretation of λ_c in a computational model (the theory is also complete)(Figure 2.3).*

PROOF: is direct from the definition of λ_c model, see [62]. $\square$

Even without identifying continuation-passing as the notion of of computation, λ_c permits reasoning about the flow of control which is unavailable in the call-by value λ-calculus. In particular, the intuition behind the definition of evaluation contexts, that t is always evaluated in computing $E[t]$, can be formalised, by requiring that the meanings of $E[t]$ and $\lambda x.E[x]\ t$ (evaluate t, substitute it into $E[x]$, and compute that) should be the same in a computational model, and so $E[t] =_c (\lambda x.E[x])\ t$ should be derivable from the above theory. Since the evaluation context has been used as a syntactic representation of the current continuation, this is a directly useful fact.
First define the let-free λ_c evaluation contexts by induction over the following grammar.

Definition 4.7.3 (Let-free evaluation contexts)

$$E[\cdot] ::= [\cdot] \mid E[\cdot]\ t \mid v\ E[\cdot] \mid \mu E[\cdot].$$

As in μPCF, terms can be uniquely decomposed into an evaluation context and a value.

Proposition 4.7.4 *Let t be any term of λ_c. Then there is a unique value v, and evaluation context $E[\cdot]$, such that $t = E[v]$*

PROOF: is by routine induction along the lines of Proposition 4.3.2. $\square$

Proposition 4.7.5 (Evaluation Contexts Lemma) *For all evaluation contexts $E^T[\cdot]$ with x not free in $E[\cdot]$, and terms $t\colon T$,*

$$(\lambda x.E[x])\ t =_c E[t].$$

PROOF: First, observe that the proposition is equivalent to the following :

Lemma 4.7.6 *For all t:*

i $(\lambda x.x)\ t =_c t,$

ii $(\lambda x.(x\ s))\ t =_c t\ s,$

iii $(\lambda x.v\ (u\ x))\ t =_c v\ (u\ t),$

iv $(\lambda x.\mu(x))\ t =_c \mu(t)$

(where u, v denote values, s, t arbitrary terms, and x is not free in s, u, v).

PROOF: Each of the above equations is an instance of the evaluation contexts lemma.
On the other hand, one can prove by induction on the definition of evaluation context that they jointly entail the axiom schema. Given an evaluation context $E[\cdot]$, either:
$E[\cdot] = [\cdot]$, so $E[t] = t$, and by (i), $\lambda x.x\ t =_c t$.
or $E[\cdot] = E'[K[\cdot]]$ where $K[\cdot]$ is just one level deep.
Then $E[t] =_c \lambda x.E'[x]\ K[t]$ by inductive hypothesis.
$K[t] =_c \lambda y.K[y]\ t$ by the appropriate clause above,
and $E[t] =_c \lambda z.(\lambda x.E[x]\ (\lambda y.K[y]\ z))\ t =_c \lambda z.E[z]\ t$ by clause *iii* above. $\square$

Now it remains to show that $i - iv$ hold in λ_c:

i $(\lambda x.x)\ t =_c \texttt{let}\ y = t\ \texttt{in}\ (\lambda x.x)\ y =_c \texttt{let}\ y = t\ \texttt{in}\ t =_c t$

ii $(\lambda x.x\ s)\ t =_c \texttt{let}\ y = t\ \texttt{in}\ (\lambda x.x\ s)\ y =_c \texttt{let}\ y = t\ \texttt{in}\ y\ s =_c t\ s$

iii $(\lambda x.v\ (u\ x))\ t =_c \texttt{let}\ y = t\ \texttt{in}\ (\lambda x.v\ (u\ x))\ y$
$=_c \texttt{let}\ y = t\ \texttt{in}\ v\ (u\ y)$
$=_c \texttt{let}\ y = t\ \texttt{in}\ (\texttt{let}\ z = u\ y\ \texttt{in}\ v\ z)$
$=_c \texttt{let}\ z = (\texttt{let}\ y = t\ \texttt{in}\ u\ y)\ \texttt{in}\ v\ z =_c \texttt{let}\ z = u\ t\ \texttt{in}\ v\ z =_c v\ (u\ t)$

iv is by direct application of $\texttt{let}_{\mu}$.

4.8 Relating $\lambda\mu$ to λ_c via $\lambda\mathcal{C}$

λ_c is a potential basis for a metalanguage for reasoning about a variety of effects in games via sum monads and the $\mathsf{Fam}(\mathcal{C})$ construction. However, in the case of control, the calculus can be simplified by observing that for any continuation monad $\mathbf{T}$ with answer object $\mathbf{a}$, over a category with an initial object $\mathbf{0}$,

$$[\![\mathbf{T0}]\!]_c \cong \mathbf{a}$$

and hence for any type X, $\mathbf{T}X$ can be equated with $(X \Rightarrow \mathbf{T0}) \Rightarrow \mathbf{T0}$. So all references to $\mathbf{T}$ in the type system can be suppressed, by hiding the use of $\mathbf{T}$ in function types (as in the standard call-by-value interpretation, $[\![A \Rightarrow B]\!] = [\![A]\!] \Rightarrow \mathbf{T}[\![B]\!]$), and replacing the type-constructor $\mathbf{T}$ with $((\_ \Rightarrow \mathbf{0}) \Rightarrow \mathbf{0})$. So the typing rules for μ and $[\cdot]$ become:

$$\frac{\Gamma \vdash t \colon A}{\Gamma \vdash [t] \colon (A \Rightarrow \mathbf{0}) \Rightarrow \mathbf{0}}$$

$$\frac{\Gamma \vdash t\colon (A \Rightarrow \mathbf{0}) \Rightarrow \mathbf{0}}{\Gamma \vdash \mu(t)\colon A.}$$

A further simplification is that the lifting operation, $[\_]$ can be replaced by $\lambda x.x\ \_$, as for any term in context, $[\![\Gamma \vdash [t]]\!]_{\mathbf{T}} = [\![\Gamma \vdash t]\!]_T; \eta = [\![\Gamma \vdash \lambda x.x\ t]\!]_{\mathbf{T}}$.
Finally, note that the typing of μ given above is that suggested by Griffin for the $\mathcal{C}$-operator introduced (in an untyped setting) by Felleisen and Hieb [25]. This allows the equational theory of λ_c to be appropriated as an equational theory of for $\lambda\mathcal{C}$, with β_v, η_v, and the additional axioms below: the Evaluation contexts lemma, together with the rules defining $\mathcal{C}$ and $\lambda x.x\ \_$ as a monadic reflection, — an isomorphism between values of type $\mathbf{T}A$, and terms of type A (see [27]).

$\beta_{E[\cdot]}$ $(\lambda x.E[x])\ t =_{\mathcal{C}} E[t]$

$\mathcal{C}_\eta$ $\mathcal{C}\lambda x.x\ t =_{\mathcal{C}} t$

$\mathcal{C}_\beta$ $\lambda x.(x\ (\mathcal{C}v)) =_{\mathcal{C}} v.$

This is still not quite a complete axiomatization of continuation models of $\lambda\mathcal{C}$, — it is also necessary to capture the initiality of $\mathbf{0}$ — see [39], and Section 4.9. It is, however, sufficient to establish soundness of $\lambda\mu_v$ models by translation.

Remark 4.8.1 (Relationship to other axiomatizations of $\lambda\mathcal{C}$-models) *The equational theory of $\lambda\mathcal{C}$ supplied by Felleisen and Hieb is sound but not fully complete with respect to continuations models. (So it is properly contained in the above theory.) The description by Hofmann [39] of models of $\lambda\mathcal{C}$ essentially similar to control models (without co-products) has already been mentioned. This paper goes on to give an axiomatization of $\lambda\mathcal{C}$-theory which is sound and complete, and similar to the one which has been derived here. (The real difference with Hofmann's account is that here the rules are derived from the general notion of monad.) Sitaram and Felleisen [80] also gave a similar axiomatization of $\lambda\mathcal{C}$, and a syntactic completeness result in the form of invertible cps translations.*

4.8.1 The call-by-value de Groote Translation

$\lambda\mu_v$ can now be translated into $\lambda\mathcal{C}$, and the equational theory used to prove that any continuations monad on a CCC is a sound $\lambda\mu_v$-model. In fact, the translation has already been described by de Groote in [23], where it was considered for the call-by-name case only. As noted by Ong and Stewart [67], soundness of the translation into call-by-value $\lambda\mathcal{C}$ seems more problematic. This not due to the inadequacy of the rules for $\mathcal{C}$ itself, but the weakness of β_v on its own, for reasoning about the flow of control. This can be remedied by adopting the `let` -rules of

λ_c (or Hofmann's complete axiomatization of $\lambda\mathcal{C}$), which are encapsulated in the single rule $\beta_{E[\cdot]}$.

Using the presentation of $\lambda\mu$-theory via evaluation contexts, it is a simple matter to show that the translation is sound with respect to the $\lambda\mathcal{C}$ and $\lambda\mu_v$ theories. Hence the apparatus of naming and μ-abstraction can be seen as simply a (useful) syntactic sugaring of call-by-value $\lambda\mathcal{C}$. The correspondence of the call-by-value calculi can thus be said to be closer than in the call-by-name case, as the rules of $\lambda\mu_n$ (in particular $\mu\beta$) *cannot* be consistently translated to a sound rule of call-by-name $\lambda\mathcal{C}$ in this style (see Proposition 4.8.7).

Definition 4.8.2 (The 'de Groote translation') *from terms of $\lambda\mu$ to $\lambda\mathcal{C}$ is defined by structural induction as follows:*
Define a distinguished collection of variables $x_\alpha, x_\beta, \ldots$ in bijective correspondence with μ-names, such that if $\alpha\colon A$ then $x_\alpha\colon A \Rightarrow \mathbf{0}$.
For each $\lambda\mu$ term $t\colon A$, define the $\lambda\mathcal{C}$ term $\lfloor t \rfloor\colon A$:

- $\lfloor x \rfloor = x$,
- $\lfloor s\ t \rfloor = \lfloor s \rfloor\ \lfloor t \rfloor$,
- $\lfloor \lambda x.t \rfloor = \lambda x.\lfloor t \rfloor$,
- $\lfloor [\alpha]t \rfloor = x_\alpha \lfloor t \rfloor$,
- $\lfloor \mu\alpha.t \rfloor = \mathcal{C}\lambda x_\alpha.\lfloor t \rfloor$.

Definition 4.8.3 *The semantics of terms-in-context $\Gamma \vdash t; \Delta$ of $\lambda\mu_v$ in a control model $(\mathcal{C}, \mathbf{a})$ is given by translation into $\lambda\mathcal{C}$ and interpretation in the Kleisli category of continuations on $\mathsf{Fam}(\mathcal{C})$:*

$$[\![\Gamma \vdash t; \Delta]\!]_\mu = [\![\lfloor \Gamma \rfloor, \lfloor \Delta \rfloor \vdash \lfloor t \rfloor]\!]_{\mathcal{C}}.$$

4.8.2 Correctness of the translation

Using the axioms obtained for $\lambda\mathcal{C}$ as a particular form of the λ_c calculus, and in particular the Evaluation Contexts Lemma, it is simple to show the translation is sound with respect to the equational theory of call-by-value $\lambda\mu$. First extend the translation to (many-holed) *contexts* by adding the clause $\lfloor [\cdot] \rfloor = [\cdot]$ to its inductive definition.

Lemma 4.8.4 *For any context of $\lambda\mu$, $C[\cdot]$, and for any compatible term t, $\lfloor C[t] \rfloor = \lfloor C[\lfloor t \rfloor] \rfloor$.*
And *if $C[\cdot]$ is an evaluation context then so is $\lfloor C[\cdot] \rfloor$.*

PROOF: is by simple structural induction. $\square$

Proposition 4.8.5 (Correctness of the Translation) *For any $\lambda\mu_v$-terms s, t, $s =_{\lambda\mu_v} t$ implies $\lfloor s \rfloor =_{\mathcal{C}} \lfloor t \rfloor$.*

PROOF: it is necessary to show that the translations of $\mu\eta$ and $\mu\zeta$ of $\lambda\mu_v$ hold in $\lambda\mathcal{C}$ ($\lambda\beta, \lambda\eta$ translate directly to their $\lambda\mathcal{C}$ counterparts).

μ_η $\mu\alpha^A[\alpha]s = s$ $(\alpha \notin FN(s))$
$\lfloor \mu\alpha^A[\alpha] \rfloor = \mathcal{C}^A \lambda x_\alpha . x\ t =_{\mathcal{C}} t$ by $\mathcal{C}_\eta$.

μ_ζ $E[\mu\alpha^T.t] : \mathbf{0} = t[E[\cdot]/\alpha]$
$\lfloor E[\mu\alpha^A.t] \rfloor = \lfloor E[\lfloor \mu\alpha^A.t \rfloor] \rfloor =_{\mathcal{C}} \lfloor E[\mathcal{C}^A \lambda x_\alpha . \lfloor t \rfloor] \rfloor$ (by the lemma above)
$\lfloor E[\mathcal{C}^A \lambda x_\alpha . \lfloor t \rfloor] \rfloor =_{\mathcal{C}} (\lambda y^A . \lfloor E[y] \rfloor)\ \mathcal{C}^A \lambda x_\alpha . \lfloor t \rfloor)$, by the Evaluation Contexts Lemma
$=_{\mathcal{C}} (\lambda x_\alpha . \lfloor t \rfloor)\ (\lambda y^A . \lfloor E[y] \rfloor)$ by $\mathcal{C}_\beta$
$=_{\mathcal{C}} \lfloor t \rfloor [(\lambda y^A . \lfloor E[y] \rfloor)/x_\alpha]$, by β_v
Now make the α-named terms of t explicit:
suppose $t = C[[\alpha]s_1][[\alpha]s_2] \ldots [[\alpha]s_n]$.
Then $\lfloor t \rfloor = \lfloor C[x_\alpha\ \lfloor s_1 \rfloor] \ldots [x_\alpha\ \lfloor s_n \rfloor]$, and by $\beta_v, \beta_{E[\cdot]}$,
$\lfloor t \rfloor [(\lambda y^A . \lfloor E[y] \rfloor)/x_\alpha] =_{\mathcal{C}} \lfloor C[x_\beta\ \lfloor E[s_1] \rfloor] \ldots [x_\beta\ \lfloor E[s_n] \rfloor] \rfloor$
$= \lfloor t[\lfloor E[\cdot] \rfloor/\alpha] \rfloor$, as required.
(That the variable substitution of $\lambda y . \lfloor E[y] \rfloor$ for x_α is equivalent to the named substitution of $E[\cdot]$ for α can be proved formally by structural induction using the definition of substitutions as in Lemma 4.5.8.)

$\square$

Remark 4.8.6 *By way of comparison with the original call-by-name translation, note that the deGroote translation is not sound with respect to any non-trivial axiomatization of call-by-name $\lambda\mathcal{C}$. The translation of the $\mu\beta$ rule is inconsistent with $\lfloor [\alpha]\mu\beta.t \rfloor = x\ \mathcal{C}\lambda y . \lfloor t \rfloor$.*

Proposition 4.8.7 *Suppose T is a theory of $\lambda\mathcal{C}$ such that the deGroote translation is sound with respect to T. Then for any terms $s, t : S$ of $\lambda\mathcal{C}$, $s =_T t$.*

PROOF: *If $\lfloor \_ \rfloor$ is sound w.r.t. T, then T contains all instances of β_n, η_n, together with $\lfloor \mu\eta \rfloor$,*
i.e. $\mathcal{C}\lambda x.x\ t =_T t$ for all t not containing x free,
and $\lfloor \mu\beta \rfloor$, i.e. $x^{S \Rightarrow \mathbf{0}}\ \mathcal{C}\lambda y.t =_T t[x/y]$ for all t.
Consider the term $\mathcal{C}\lambda x : S \Rightarrow \mathbf{0}.(\lambda y : S \Rightarrow \mathbf{0}.y\ \mathcal{C}\lambda z : S \Rightarrow \mathbf{0}.(x\ t))\ \lambda w : S.(x\ s)$

$=_\beta \mathcal{C}\lambda x.(\lambda w.(x\ s))\ \mathcal{C}\lambda z : S \Rightarrow \mathbf{0}.(x\ t) =_\beta \mathcal{C}\lambda x.(x\ s) =_T s$
But also $\mathcal{C}\lambda x : S \Rightarrow \mathbf{0}.(\lambda y : S \Rightarrow \mathbf{0}.y\ \mathcal{C}\lambda z : S \Rightarrow \mathbf{0}.(x\ t))\ \lambda w : S.(x\ s)$
$=_T \mathcal{C}\lambda x.(\lambda z.(x\ t))\ \lambda w.(x\ s) =_\beta \mathcal{C}\lambda x.(x\ t) =_T t.$

□

4.9 Completeness and full abstraction in $\mu\mathrm{PCF}_v$

To form a complete theory of $\lambda\mu$, add the following rules to the call-by-value equational theory $\lambda\beta, \lambda\eta, \mu\eta, \mu\zeta$.

(η_μ) $(\lambda x.[\alpha]x)\ t =_\mu [\alpha]t$

This makes the correspondence between naming and applying a continuation variable direct; that it is sound in a model of control is clearly a consequence of η_v.

(0-init) For any values $v, u \colon \mathbf{0} \Rightarrow T$, $v =_\mu u$.

This makes the status of $\mathbf{0}$ as initial in a model of control explicit by stating that all values taking arguments of type $\mathbf{0}$ are equivalent.

Finally, it can be observed directly that the rules for the co-product extension are sound, establishing the following

Proposition 4.9.1 *Let $(\mathcal{C}, \mathbf{a})$ be a model of control, then the semantics of $\lambda\mu_v^+$ in $(\mathcal{C}, \mathbf{a})$ is sound.*

(As de Groote notes) an inverse to the translation into $\lambda\mathcal{C}$ can be given by translating $\mathcal{C}t$ as $\mu\alpha.t\ \lambda x.[\alpha]x$, (and $\mathcal{C}^{\mathbf{0}}t$ as $t\ \lambda x : \mathbf{0}.x$).

Proposition 4.9.2 *The inverse of the de Groote translation is sound with respect to the rules of $\lambda\mathcal{C}$.*

PROOF: The monadic reflection rules translate to sound rules of $\lambda\mu$ as follows:

- $\mu\alpha.((\lambda y.y\ t)\ (\lambda x.[\alpha]x)) =_\mu \mu\alpha.\lambda x.[\alpha]x)\ t =_\mu \mu\alpha.[\alpha]t =_\mu t$
- $\lambda y.y\ (\mu\alpha.v\ (\lambda x.[\alpha]x)) =_\mu \lambda y.v\ (\lambda x.y\ x) =_\mu \lambda y.v\ y =_\mu v.$

That the `let` -rules translate to valid equations of $\lambda\mu$ is equivalent to showing that $\beta_{E[\cdot]}$ holds in $\lambda\mu_v$.
First note that for all $t \colon T\ t =_\mu \mu\alpha.(\lambda z.[\alpha]z)\ t$ by η_μ, and so
$\lambda x.E[x]\ t =_\mu \lambda x.E[x]\ (\mu\alpha.(\lambda z.[\alpha]z)\ t)$
$=_\mu \mu\alpha.((\lambda z.[\alpha](\lambda x.E[x])\ z)\ t) =_\mu \mu\alpha.((\lambda z.[\alpha]E[z])\ t)$
$=_\mu E[(\mu\alpha.(\lambda z.[\alpha]z)\ t)] =_\mu E[t]$ as required. □

Note in particular that the theory of $\lambda\mu_v$ is not a conservative extension of $\beta\eta_v$. So although the rules $\mu\eta$ and $\mu\beta$ are manifestly about manipulation of first-class continuations, they can be used to reason about control flow even in purely functional programs.

Completeness of the $\lambda\mu$-theory can thus be achieved as a corollary of Hofmann's completeness theorem for the axiomatization for $\lambda\mathfrak{C}$. However, the addition of co-products allows a simple proof of completeness to be given more directly, based on the following observation:
any model of $\lambda\mu_v^+(\Omega)$ over the empty type is a model of the simply-typed *call-by-name* λ-calculus $\Lambda(\Omega)$.
This can be proved by giving an 'inverse' to the cps translation into $\Lambda(\Omega)$ which is implicit in the continuations model, and hence proving that $\lambda\mu_v(\Omega)$ is a complete syntax of control (recall Definition 2.3.11). Notwithstanding the brutal nature of the syntax, this is merely the syntactic counterpart of the observation in Chapter 2, Proposition 2.3.7, that every object generated from $\mathbf{a}$ in $(\mathcal{C}, \mathbf{a})$ corresponds to a call-by-value type in the associated control models.

Definition 4.9.3 *Define the following translation from the simply-typed $\Lambda(\Omega)$-calculus to the $+, \_ \Rightarrow \mathbf{0}$ fragment of $\lambda\mu_v^+(\Omega)$*
Types translate as follows

- $\iota^\blacktriangle = \mathbf{0}$
- $(A \Rightarrow \iota)^\blacktriangle = A^\blacktriangle \Rightarrow \mathbf{0}$
- $(A \Rightarrow B)^\blacktriangle = (A^\blacktriangle \Rightarrow \mathbf{0}) + B^\blacktriangle$ *$(B \neq \iota)$.*

Terms in context $\Gamma \vdash t$ are translated to $\lambda\mu^+$ terms in context $\_ \vdash t^\blacktriangle; \Gamma^\blacktriangle$ as follows:

- $\Omega\colon \iota^\blacktriangle = \Omega^{\mathbf{0}}$
- $(x\colon T)^\blacktriangle = \mu\gamma\colon T^\blacktriangle.[\alpha_x\colon T^\blacktriangle \Rightarrow \mathbf{0}]\lambda y[\gamma]y$
- $((t\colon S \Rightarrow T)\ s\colon S)^\blacktriangle = \mu\beta\colon T^\blacktriangle.((\mu\alpha.[\alpha\colon S^\blacktriangle \Rightarrow \mathbf{0}, \beta\colon T]t^\blacktriangle)\ s\colon S^\blacktriangle)$
- $(\lambda x\colon S.t\colon T)^\blacktriangle = \mu[\alpha_x\colon S^\blacktriangle \Rightarrow \mathbf{0}, \beta\colon T^\blacktriangle].[\beta]t^\blacktriangle$

Proposition 4.9.4 *The above translation is sound with respect to $\beta\eta$-equality.*

PROOF: β-equality: $((\lambda x\colon S.t\colon T)\ s\colon S)^\blacktriangle$
$=_\mu \mu\beta\colon T^\blacktriangle.(\mu\alpha\colon S^\blacktriangle \Rightarrow \mathbf{0}.[\alpha, \beta](\mu[\alpha_x\colon S^\blacktriangle \Rightarrow \mathbf{0}, \beta\colon T^\blacktriangle].[\beta]t^\blacktriangle)\ (s^\blacktriangle))$

(where the $\Lambda(\Omega)$-variable x is translated in t as $\mu\gamma.[\alpha_x]\lambda y.[\gamma]y$)
$=_\mu \mu\beta.((\mu\alpha.[\beta]t^\blacktriangle)\ s^\blacktriangle) =_\mu \mu\beta.[\beta]t^\blacktriangle[s^\blacktriangle/(\mu\gamma.[\alpha](\lambda y.[\gamma]y)] =_\mu t[s/x]^\blacktriangle$ as required. (As $\mu\gamma.(\lambda y.[\gamma]y)s^\blacktriangle = s^\blacktriangle$.)
η-equality: (it is sufficient to show that this holds for (the translation of) variables, i.e. $(\lambda x.y\ x)^\blacktriangle = y^\blacktriangle$)
$(\lambda x.y\ x)^\blacktriangle = \mu[\alpha_x,\beta].[\beta]\mu\gamma.((\mu\delta.[\delta,\gamma]\mu[\theta,\varepsilon].[\alpha_y]\lambda w.[\theta,\varepsilon]w)\ (\mu\psi.[\alpha_x]\lambda z.[\psi]z))$
$= \mu[\alpha,\beta].((\mu\delta.[\alpha_y]\lambda w.[\delta,\beta]w)\ \mu\psi.[\alpha_x](\lambda z.[\psi]z))$ (by repeated application of $\mu\beta$)
Using the derived rule $s\ t =_\mu (\lambda x.x\ t)\ s$, this is equal to
$\mu[\alpha_x,\beta].(\lambda u.[\alpha_x]y)\ (\mu\delta.[\alpha_y]\lambda w.[\delta,\beta]w)$
$=_\mu \mu[\alpha_x,\beta].[\alpha_y](\lambda w.[\alpha_x,\beta]w)$ which is equal by α-conversion to $y^\blacktriangle$ as required. □

Sum types, and the continuation forming operation $\_ \Rightarrow \mathbf{0}$ are thus, in a sense 'expressively complete' for the simply-typed call-by-value calculus.

Corollary 4.9.5 *The initial model of the $\_ \Rightarrow \mathbf{0}, +$ fragment of $\lambda\mu^+(\Omega)_v$ is isomorphic to the free pointed CCC.*

Completeness for the whole of $\lambda\mu_v(\Omega)$ can now be established by translating implicational types into the $\neg, +$ part of the language.

Proposition 4.9.6 *The initial model of $\lambda\mu(\Omega)$ is isomorphic to the initial call-by-value control model.*

Proof: Having shown completeness for the $(\_ \Rightarrow \mathbf{0}, +)$ fragment, it remains only to show that every function type of $\lambda\mu_v(\Omega)$ is equivalent to one formed from $+$ and $\_ \Rightarrow \mathbf{0}$, (essentially by defining a sort of co-currying operation between sum and arrow types).
So let $(\mathbf{0} \Rightarrow B)^\triangle = \mathbf{0}$, and
$(A \Rightarrow \mathbf{0})^\triangle = A^\triangle \Rightarrow \mathbf{0}\ (A \neq \mathbf{0})$
$(A \Rightarrow B)^\triangle = (A^\triangle + (B^\triangle \Rightarrow \mathbf{0})) \Rightarrow \mathbf{0}\ (A, B \neq \mathbf{0})$.
Note that under the interpretation of $\lambda\mu_v^+$ in a control model,

$$[\![T]\!] = [\![T^\triangle \Rightarrow \mathbf{0}]\!].$$

Lemma 4.9.7 *There is a ($\lambda\mu^+$ definable) bijection (up to $\lambda\mu$-equality) between values of type T and values of type $(T^\triangle \Rightarrow \mathbf{0})$.*

Proof: is by structural induction, defining an isomorphism ϕ_T from values of type T to values of type $T^\triangle \Rightarrow \mathbf{0}$ and an explicit inverse ϕ^{-1}.

- For $T = \mathbf{0} \Rightarrow A$, define ϕ_T as the constant $\lambda x\colon \mathbf{0}.x$ and $\phi^{-1} = \lambda x\colon \mathbf{0}.\mu\alpha\colon A.x$, as all terms at these types are equal by $\mathbf{0}$-init in any case.

- For $T = A \Rightarrow \mathbf{0}$, define $\phi_T(v) = \lambda x : A^{\triangle}.v\ (\phi_A^{-1}(x))$ and $\phi_T^{-1}(t) = \lambda x\colon A.(\phi_A(x)\ t)$. so that $\phi_T^{-1}(\phi(v)) = v$, and $\phi_T^{-1}(\phi_T(v))$

- For $T = A \Rightarrow B$ ($B \neq \mathbf{0}$), $\phi_T(v) =$

$$\mu\gamma : T.\mu[\alpha\colon A^{\triangle}, \beta\colon ((B^{\triangle} \Rightarrow \mathbf{0})].\phi_{B\Rightarrow\mathbf{0}}^{-1}(\lambda z.[\beta]z)\ (v\ \phi_A^{-1}(\lambda y.[\alpha]y))$$

  and define the inverse $\phi_T^{-1}(v) =$

  $\lambda x\colon A.\mu\beta\colon B.\phi_{B\Rightarrow\mathbf{0}}(\lambda z.[\beta]z)\ ((\mu\gamma.\phi_A(x)\ (\mu\alpha.v\ (\lambda y.[\alpha\colon A^{\triangle}, \gamma\colon B^{\triangle} \Rightarrow \mathbf{0}]y))))$

  So $\phi_T^{-1}(\phi_T(v)) =_\mu \lambda x.\mu\beta.\phi(\phi^{-1}(\lambda z.[\beta]z))\ (v\ \phi^{-1}(\phi(x))) =_\mu v$,
  and similarly $\phi_T(\phi_T^{-1}(v))$
  $=_\mu \lambda w.w\ \mu[\alpha, \beta].\phi^{-1}(\phi(\lambda z.[\beta]z))\ ((\phi(\phi^{-1}(\lambda y[\alpha]y)))\ \mu\gamma.v\ (\lambda u.[\gamma, \beta]u))$
  $= \lambda w.w\ \mu[\alpha, \beta].v\ (\lambda u.[\alpha, \beta]u)) =_\mu v$.

□

Hence the maps $[\![x : T \vdash \phi_T(x)]\!] : [\![T]\!] \to \mathbf{T}[\![T^{\triangle} \Rightarrow \mathbf{0}]\!]$, and $[\![x : T^{\triangle} \Rightarrow \mathbf{0} \vdash \phi_T(x)]\!] : \mathbf{T}[\![T^{\triangle} \Rightarrow \mathbf{0}]\!] \to [\![T]\!]]$ define an isomorphism on type-objects in the Kleisli category of the continuations monad.

Corollary 4.9.8 *If $(\mathcal{C}, \mathbf{a})$ satisfy the axioms for definability, then the control models of $\lambda\mu_v(\Omega)$ constructed from $(\mathcal{C}, \mathbf{a})$ are fully complete.*

4.9.1 Control models of $\mu\mathrm{PCF}_v$

The interpretation of $\mu\mathrm{PCF}_v$ in a computational control model $(\mathcal{C}, \mathbf{a})$ can now be given by combining the semantics of $\lambda\mu_v$ with the semantics of call-by-value PCF outlined in Chapter 2. As in the call-by-name case, the semantics is *adequate*, — the assumption of non-terminality of $\mathbf{a}$ yields standard datatypes, and rationality entails continuous observability. If $(\mathcal{C}, \mathbf{a})$ also satisfies the axioms for definability, then the semantics also has the finite definability property. Hence dropping the bracketing condition on the Honda and Yoshida [41], and Abramsky and McCusker [3] models of pure PCF yields such a model.

Recall the definition of a computational control model (Section 2.4); a rational pointed CCC with ω-indexed products and non-terminal answer-object. This gives a semantics of call-by-value PCF in the Kleisli category of the monad of $\mathbf{a}$ continuations(see Section 2.4.4).

Proposition 4.9.9 *The semantics of $\mu\mathrm{PCF}_v$ in a computational control model $(\mathcal{C}, \mathbf{a})$ (rational $\mathcal{C}$, non-terminal $\mathbf{a}$) is adequate.*

PROOF: Soundness follows from the soundness of the translation of $\lambda\mu$ into $\lambda\mathfrak{C}$. (That the Evaluation Contexts Lemma extends to the $\mu\mathrm{PCF}_v$ evaluation contexts is straightforward.)
Completeness follows from the application of the Adequacy Theorem (Theorem 4.4.5) for the operational semantics, together with the fact that the adequacy conditions of continuous observability and standardness of datatypes apply, for which the proof is entirely similar to the call-by-name case (Proposition 4.5.11). $\Box$

Definability is a consequence of the full completeness of the $\lambda\mu(\Omega)_v$ semantics, together with definability of embeddings and projections into PCF-types. As in the the call-by-name case, more detail can be given by defining the finitary evaluation trees of $\mu\mathrm{PCF}_v$ and showing that they correspond bijectively via denotation to unique $\beta\eta\pi$-long normal forms of $\Lambda(\Omega)^\omega$.

Definition 4.9.10 (Finite Evaluation trees of $\mu\mathrm{PCF}_v$) *Thus there are two disjoint sets of trees of type T over the higher-type context Γ, Δ; the values, $V(\Gamma, \Sigma; T)$, and the non-values $NV(\Gamma, \Sigma; T)$, formed by mutual induction as follows.*

$$\frac{}{\Omega \in NV(\Gamma; \Delta; \mathbf{0})}$$

$$\frac{}{n \in V(\Sigma; \Delta; \mathtt{nat})}$$

$$\frac{M \in NV(\Sigma, \Delta; B)}{\lambda x.M \in V(\Sigma; \Delta; A \Rightarrow B)}$$

$$\frac{\{M_i \in NV(\Sigma; \Delta; T) \mid i \leq k\},}{(\lambda x.\mathtt{case}\ _k\ x\ |_{i \leq k}\ \ M_i) \in V(\Sigma; \Delta; \mathtt{nat} \Rightarrow T)}$$

$$\frac{M \in NV(\Sigma; \Delta, \alpha \colon A; \mathbf{0})}{\mu\alpha.M \in NV(\Sigma; \Delta; A)}$$

$$\frac{M \in V(\Sigma, x : S \Rightarrow T; \Delta; T \Rightarrow \mathbf{0}), N \in V(\Sigma; \Delta, S)}{M\ \ (x\ \ N) \in NV(\Sigma, x : S \Rightarrow T; \Delta, \mathbf{0})}$$

$$\frac{M \in V(\Sigma, x : S \Rightarrow T; \Delta, \alpha : T; T)}{[\alpha]M \in NV(\Sigma, x : S \Rightarrow T; \Delta, \alpha : T; \mathbf{0})}$$

These normal forms are complicated by the 'eager evaluation' of ground-type values. To simplify matters, $\mathbf{0}$ is assumed not to occur to the right of an arrow, or as the type of a variable, (as by initiality of $\mathbf{0}$, all values at such types are equivalent). Variables of type `nat` are tested once, as soon as they are introduced, and so contexts are restricted to higher-type variables.

In addition, trees are split between values and non-values. It is only possible to reach a higher-type value by lambda-abstraction from a non-value, and it is only possible to reach a higher-type non-value by μ-abstraction.

This gives a notion of 'normal form' for continuation passing (which is very different, for instance, from the version arising from the well-bracketed games model given in [41]): apply a variable to an argument, and a continuation to the result, abstract a continuation variable, abstract an argument, and so on.

There is a cps translation from $\Lambda(\Omega)^\omega$ into $\mu\mathrm{PCF}_v$ along the lines of the call-by-name $\Box$ translation but it is substantially more complicated, due to the fact that variables of type `nat` do not translate to $\Lambda(\Omega)$ variables but operations on infinite lists. As there are no free variables in the finite evaluation trees of $\mu\mathrm{PCF}_v$, however, an invertible cps translation into the finitary normal forms of $\Lambda(\Omega)^\omega$ *can* be described simply, and used to show finite definability. First a (slightly non-standard) cps translation on types is defined.

Definition 4.9.11 *Define the $\Diamond$-translation on* μPCF *function types as follows.*

- $(\mathtt{nat} \Rightarrow \mathbf{0})^\Diamond = \mathbf{0}^\omega$
- $(\mathtt{nat} \Rightarrow \mathtt{nat})^\Diamond = (\iota^\omega \Rightarrow \iota)^\omega$
- $(\mathtt{nat} \Rightarrow T)^\Diamond = ((T^\Diamond \Rightarrow \iota) \Rightarrow \iota)^\omega$, *if* $T \neq \mathtt{nat}, \mathbf{0}$
- $(S \Rightarrow \mathbf{0})^\Diamond = S^\Diamond \Rightarrow \iota$, *if* $S \neq \mathtt{nat}, \mathbf{0}$
- $(S \Rightarrow \mathtt{nat})^\Diamond = S^\Diamond \Rightarrow (\iota^\omega \Rightarrow \iota)$, *if* $S \neq \mathtt{nat}, \mathbf{0}$ $(S \Rightarrow T)^\Diamond = S^\Diamond \Rightarrow (T^\Diamond \Rightarrow \iota) \Rightarrow \iota$, *if* $S, T \neq \mathbf{0}, \mathtt{nat}$.

Thus for any higher μPCF type T, the interpretation of T in a model of control is isomorphic to the interpretation of $T^\Diamond$ in a model of $\Lambda(\Omega)^\omega$.

Proposition 4.9.12 *If $\mathcal{C}, \mathbf{a}$ is a CCC with ω-indexed products specifying a (call-by-value) model of control, then for any $\mu\mathrm{PCF}_v$ type T, $[\![T^\Diamond]\!]_{(\mathcal{C},\mathbf{a})} \cong [\![T]\!]_{(\mathcal{C},\mathbf{a})}$.*

The finitary normal forms of $\Lambda(\Omega)^\omega$ (Definition 4.6.1) are extended to translations of $\mu\mathrm{PCF}_v$ types by adding the clause:

$$\frac{y : (S \Rightarrow \iota)^\omega \in \Gamma,\ t \in N(\Gamma; S)}{\pi_i(y)\ t \in (\Gamma; \iota).}$$

A correspondence is assumed at each type T between $\mu\mathrm{PCF}_v$ variables $x : T$ and $\Lambda(\Omega)^\omega$-variables, $x^\Diamond : T^\Diamond$, and between names $\alpha : T$ and $\Lambda(\Omega)^\omega$-variables, $\alpha^\Diamond : T^\Diamond \Rightarrow \mathbf{0}$.

Definition 4.9.13 *The $\diamondsuit$-translation takes trees of type* $\mathbf{0}$ *to finitary normal forms of type* ι*,* μPCF$_v$ values *of type* T *to normal forms of type* $T^{\diamondsuit}$*, and* μPCF non-values *of type* T *to normal forms of type* $(T^{\diamondsuit} \Rightarrow \iota) \Rightarrow \iota$*. i.e.*

$$NV(\Gamma, \Delta; \mathbf{0}) \longrightarrow N(\Gamma^{\diamondsuit}, \Delta^{\diamondsuit}; \iota)$$

$$V(\Gamma, \Delta; T) \longrightarrow N(\Gamma^{\diamondsuit}, \Delta^{\diamondsuit}; T^{\diamondsuit})$$

$$NV(\Gamma, \Delta; T) \longrightarrow N(\Gamma^{\diamondsuit}, \Delta^{\diamondsuit}; (T^{\diamondsuit} \Rightarrow \iota) \Rightarrow \iota)$$

- $\Omega^{\diamondsuit} = \Omega : \iota$
- $(\lambda x : \mathtt{nat}.\mathtt{case}_k\ x\ |_{i \leq k}\ M_i)^{\diamondsuit} = \langle M_1^{\diamondsuit}, M_2^{\diamondsuit}, \ldots, M_k^{\diamondsuit} \rangle$
- $(\lambda x : T.M)^{\diamondsuit} = \lambda x^{\diamondsuit}.M^{\diamondsuit}$, *for* $T \neq \mathtt{nat}$
- $(\mu\alpha.M)^{\diamondsuit} = \lambda\alpha^{\diamondsuit}.M^{\diamondsuit}$
- $([\alpha]n)^{\diamondsuit} = \pi_n(\alpha^{\diamondsuit})$
- $([\alpha](V : T))^{\diamondsuit} = \alpha^{\diamondsuit}V^{\diamondsuit}$, *for* $T \neq \mathtt{nat}$
- $(U\ (x\ n)^{\diamondsuit} = \pi_n(x^{\diamondsuit})\ U^{\diamondsuit}$
- $(U\ (x\ (V : T)))^{\diamondsuit} = (x\ v^{\diamondsuit})\ U^{\diamondsuit}$, *for* $T \neq \mathtt{nat}$.

Proposition 4.9.14 *The $\diamondsuit$ translation is sound and surjective.*
If $(\mathcal{C}, \mathbf{a})$ *is a computational model of control, then for any* $M \in NV(\Gamma, \Delta; T)$*,*

$$[\![\Gamma \vdash M; \Delta]\!]_{(\mathcal{C},\mathbf{a})} \cong [\![\Gamma^{\diamondsuit}, \Delta^{\diamondsuit} \vdash M^{\diamondsuit}]\!]_{(\mathcal{C},\mathbf{a})}$$

and for any $V \in V(\Gamma, \Delta; T)$*,*

$$[\![\Gamma \vdash V; \Delta]\!]_{\mathcal{C},\mathbf{a}} \cong [\![\Gamma^{\diamondsuit}, \Delta^{\diamondsuit} \vdash V^{\diamondsuit}]\!]_{(\mathcal{C},\mathbf{a})}; \eta_{[\![T]\!]}.$$

PROOF: is direct by induction over the definition of the translation. □

Proposition 4.9.15 *If* $(\mathcal{C}, \mathbf{a})$ *is an computational model of control satisfying the axioms for definability, then every morphism over a* μPCF$_v$ *type-object is the least upper bound of a chain of definable elements.*

PROOF: Is by surjectivity of the $\diamondsuit$-translation, together with the Definability Theorem (Theorem 3.7.2), and Proposition 2.5.9 (all morphisms are least upper bounds of the form $\bigsqcup_{i \in \omega}(p_A^i; f_i; e_i^B)$). □

Applying Proposition 2.4.11 yields a full abstraction result (which extends to open terms as in the call-by-name case described in Lemma 4.6.8).

Corollary 4.9.16 (Abstract full abstraction) *If* $(\mathcal{C}, \mathbf{a})$ *is a continuous control model satisfying the axioms for definability, the semantics for* $\mu\mathrm{PCF}_v$ *constructed from the collapse of* $\mathcal{C}$ *under its intrinsic preorder is fully abstract.*

Corollary 4.9.17 (Concrete full abstraction) *The call-by-value control model constructed by dropping the bracketing condition on the Abramsky and McCusker model of* PCF *and collapsing under the intrinsic preorder is a fully abstract semantics for* $\mu\mathrm{PCF}_v$*. The collapse of the games model of* $\mu\mathrm{PCF}_v$ *under its intrinsic preorder is fully abstract.*

Chapter 5

Analysis of the fully abstract model

This chapter is a study of the fully abstract models of control in greater depth, establishing further semantic properties such as universality, and also some of the syntactic information which can be extracted from them; in particular a context lemma for μPCF. The most important step towards a significant analysis of syntax and semantics, however, is that the fully abstract model can be presented effectively, — in fact directly, in the category of sequential algorithms.

Conceptually, the existence of a model which satisfies the (μPCF version of the) 'Jung and Stoughton criterion' is important because it contrasts with the purely functional case; by Loader's result [54], a PCF semantics cannot pass this test. One could consider extending the application of the criterion, to ask which (if any) meaningful sequential functional languages have effectively presentable models, and using it to delimit the applicability of full abstraction and of denotational semantics in general. The negative result for PCF reveals a deep incompatibility between sequentiality and full abstraction on the one hand, and a certain conception of denotational semantics in terms of (appropriately constrained) set-theoretic functionals on the other.

The nature of the of the analogous problem for μPCF is somewhat different because observational equivalence is not extensional, so there cannot be a presentation of the fully abstract model without (implicit) reference to its *behaviour*. Nonetheless, it is significant that there is such a construction which is both independent of the syntax, and mathematically more tractable. (Another advantage of a direct presentation of the model is that it can be seen to be cpo-enriched; this is still not known in the case of PCF). Moreover, the problem of determining observational equivalence in the presence of non-local control is a significant one in its own right, as most languages *can* use non-local control flow to distinguish between extensionally equivalent procedures. Another feature of the direct presentation described here is that it makes use of the category of sequential algorithms, and thus forms a connection between the 'intensional hierarchy' and another attempt to define a category of intensional 'realizers' for sequential functions. The existence of fully abstract models of of SPCF (in essence equivalent to $\mu\mathrm{PCF}_n$) based on sequential algorithms, and due to Cartwright, Curien and Felleisen [16] has already been noted. What is lacking in this semantics is a connection of its analysis of sequential control flow to other, more general accounts, in terms of the syntax (how does `catch` relate to call-by-value **call/cc**?) and the semantics (is there a connection with continuation passing?).

Although an effectively presented fully abstract model is technically a solution to the problem of deciding equivalence of finitary terms, as one can compute the denotation of terms and compare them, this is not especially illuminating. The problem with limiting attention to full abstraction is that in order to achieve it, the fine structure, which is not internally observable but which distinguishes terms which are intuitively intensionally distinct, has been thrown out. (Specifically, there is *no repetition of moves* in the sequential algorithms model, so a function which consults its argument once can be indistinguishable from one which does so several times). The finer structure is restored by establishing a connection with the innocent strategies model (which identifies denotations at the level of $\beta\eta$-equivalence). The unique structure-preserving functor from the games model, which exists by initiality, is given a syntax-free characterization and shown in a precise sense to be a removal of copied queries. Moreover, it is surjective — there is a way of saturating sequential algorithms with copies so that they become innocent strategies. Since the sequential algorithms model is equivalent to its own collapse, the 'copy removing' functor maps two strategies to the same algorithms if and only if they are observationally equivalent. In other words, considered as a functor on equivalence classes, it is the unique isomorphism between the fully abstract models. Thus the copy-removing functor characterizes intrinsic equivalence of innocent strategies, and the the copy-saturating translation gives a unique normal form for each equivalence class. This is the semantic invariant which truly characterises observational equivalence in control models.

5.1 Observational equivalence and control

Recall from Chapter 2 (Corollary 2.5) that effective presentability of the fully abstract models of control is equivalent to decidability of the intrinsic preorder and equivalence on the free pointed CCC (the strongest non-trivial equivalence containing $\beta\eta$-equality). The precise syntactic analogue of this problem is decidability of contextual equivalence on the language $\Lambda(\Omega)$, for contexts of type $\iota \Rightarrow \iota$. (And in fact in the pure simply-typed λ-calculus, for contexts of type $\iota \Rightarrow (\iota \Rightarrow \iota)$.)

By the full completeness result of Chapter 3, this is equivalent to a question of game semantics; to decide the intrinsic preorder on (finite) strategies of the unbracketed games model. With the lifting of the bracketing condition comes the power to make observational distinctions between extensionally equivalent strategies by 'exploring' them with a strategy on $A \to (o \Rightarrow o)$ which behaves in A as an Opponent, until some intensional feature is detected, then jumping out of A into $o \Rightarrow o$ to record this. The problem is to determine which intensional differences between strategies are detectable in this way.

Example 5.1.1 *As a simple example, consider the* μPCF *terms*
$L, M, N : \mathtt{nat} \Rightarrow (\mathtt{nat} \Rightarrow \mathtt{nat})$*, where*
$L = \lambda x.\lambda y.$IF x then (IF y then 0 else 0) else 0
$M = \lambda x.\lambda y.$IF y then (IF x then 0 else 0) else 0
$N = \lambda x.\lambda y.$IF y then (IF x then (IF y then 0 else 0) else 0) else 0
L, M, and N are all observationally equivalent with respect to PCF *contexts, — a consequence of Milner's Context Lemma for* PCF *[61], which states that function-type terms are observationally distinguishable if and only if they have they have distinguishable results when applied to the same argument.*
L and M are, however, distinguishable in μPCF*, using the 'catch' term defined in Section 4.3.2:*
$\mu\alpha.[\alpha]((L\ (\mu\beta.[\alpha]0))\ (\mu\beta.[\alpha]1)) \Downarrow 0$*, whilst* $\mu\alpha.[\alpha]((M\ \mu\beta.[\alpha]0)\ \mu\beta.[\alpha]1) \Downarrow 1$.
But M and N are μPCF *observationally equivalent; unnecessary repeated calls to a variable are not detectable using only control flow. But how to prove this?*

5.1.1 A Context Lemma for μPCF

Proving 'ad hoc' that no context exists which can distinguish equivalent terms is difficult even in the trivial example above. However, definability in the games model, and its axiomatic characterization, can be used to extract a useful practical reasoning principle; context lemmas for $\Lambda(\Omega)$ and μPCF.

Lemma 5.1.2 (Linear observers suffice) *Suppose $\sigma, \tau : A$ are innocent unbracketed strategies such that $\sigma \not\lesssim_A \tau$, i.e. there is some (innocent) $\rho : !A \multimap (o \multimap o)$ such that $\sigma; \rho \downarrow$ and $\tau; \rho \not\downarrow$. Then there is some $\phi : A \multimap (o \multimap o)$ such that $\sigma; \phi \downarrow$ and $\tau; \phi \not\downarrow$*

PROOF: is by induction on the size of the *uncovering* of $\rho \| \sigma$.
Applying the 'linearization of head occurrence' of Section 3.7.2 to ρ (which must be strict), yields a strict strategy $\rho' : A \multimap (!A \multimap (o \multimap o))$ such that:
$\sigma; (\langle \rho', \sigma \rangle; \mathrm{App}) = \sigma; \rho$ (simply by relabelling the first-opened thread as play in the new component).
Either $\tau; (\langle \rho', \sigma \rangle; \mathrm{App}) \not\downarrow$ *or* $\tau; (\langle \rho', \sigma \rangle; \mathrm{App}) \downarrow$.
In the former case let $\phi = \rho' \times \sigma; \mathrm{App}$, then $\sigma; \phi \downarrow$ and $\tau; \phi \not\downarrow$ as required. In the latter case, apply the induction hypothesis to $\mathtt{id}_A \times (\sigma; \rho'); \mathrm{App} : A \to (o \multimap o)$, as $\sigma \| (\mathtt{id}_A \times (\sigma; \rho'); \mathrm{App}) = \sigma \| \rho / (s \restriction a)$ (where $s \restriction a$ is the initial thread in A), — so this is a strictly smaller uncovering than $\sigma \| \rho$, whilst $\sigma; (\mathtt{id}_A \times (\sigma; \rho'); \mathrm{App}) \downarrow$ and $\tau; (\mathtt{id}_A \times (\sigma; \rho'); \mathrm{App}) \not\downarrow$. □

Proposition 5.1.3 *Suppose $A = \overline{B} \Rightarrow o$, then for innocent strategies $\sigma, \tau : A$, $\sigma \lesssim_A \tau$ if and only if for all innocent strategies $\rho : o \Rightarrow \overline{B}$,*

$$\rho; \sigma \downarrow \implies \rho; \tau \downarrow .$$

PROOF: Given $\sigma, \tau : A$, by Lemma 5.1.2,
$\sigma \lesssim_A \tau$ if and only if for all $\rho : \overline{B} \to (o \multimap o)$, $\sigma; \rho \downarrow$ implies $\tau; \rho \downarrow$.
Recall the 'Contravariance of continuations' axiom for the category of innocent strategies from 3.7.2. This states that the map from strategies on $D \multimap C$ to strict strategies on $(C \multimap o) \multimap (D \multimap o)$:

$$(f : D \to C) \quad \longrightarrow \Lambda(id_{C \multimap \perp} \otimes f; \mathrm{App})$$

is an isomorphism. So $\sigma \lesssim_A \tau$ if and only if for all $\varrho : o \to \overline{B}$ (in the linear category)
$\varrho; \sigma = \sigma; \Lambda(\mathtt{id}_{\overline{B} \multimap o} \otimes \rho; \mathrm{App}) \downarrow$ implies $\varrho; \tau = \sigma; \Lambda(\mathtt{id}_{\overline{B} \multimap o} \otimes \rho; \mathrm{App}) \downarrow$. □

A simple context lemma along the lines of the Context Lemma for PCF is a corollary of full completeness, together with the above result.

Corollary 5.1.4 (Context Lemma for $\Lambda(\Omega)$) *For any closed terms of $\Lambda(\Omega)$, $s, t : S_1 \Rightarrow (S_2 \Rightarrow \ldots (S_n \Rightarrow \iota) \ldots)$*
$s \sqsubseteq^{OBS} t$ if and only if
$\forall r(x)_1 : T_1, r(x)_2 : T_2, \ldots r(x)_n : T_n$,
$\lambda x.(s\ r(x)_1\ r(x)_2 \ldots r(x)_n) \Downarrow \implies \lambda x.(t\ r(x)_1\ r(x)_2 \ldots r(x)_n) \Downarrow$
(where $t \Downarrow$ if and only if $t =_{\beta\eta} \lambda x.x$).

Corollary 5.1.5 *There are finitely many observational equivalence classes at each type of $\Lambda(\Omega)$.*

PROOF: is by a straightforward induction on type, using the Context Lemma, together with a hypothesis that given maximum *depth* and *size* measures for types as follows,
$d(\iota) = 0$, $d(\overline{S} \Rightarrow \iota) = \mathtt{max}\{d(S_i) \mid i \leq n\} + 1$,
and $s(\iota) = 0$ $s(\overline{S} \Rightarrow \iota) = \mathtt{max}\{s(S_i) \mid i \leq n\} \cup \{n\}$
then there are at most $2^{d(T)}.s(T)\hat{}\hat{}d(T)$ observational equivalence classes in $\iota \Rightarrow T$ (that is, a tower of $s(T)$ exponents, of height $d(T)$).
This is obvious for $T = \iota$, so suppose $\iota \Rightarrow T = (\overline{S}, \iota) \Rightarrow \iota$. Then $s \not\sqsubseteq^{OBS}_{\iota \Rightarrow T} t$ if and only if there exist $q(x) : \iota, p_1(x) : S_1, \ldots p_n(x) : S_n$ (with $x : \iota$ free) such that $\lambda x.(s\ q(x))\ p_1(x) \ldots p_n(x) \Downarrow$ and $\lambda x.(tq(x))p_1(x) \ldots p_n(x) \not\Downarrow$. By hypothesis, there are at most $2^{d(T)-1}.s(S_i)\hat{}\hat{}d(S_i)$ equivalence classes of $\iota \Rightarrow S_i$ for each i (and 2 for $\iota \Rightarrow \iota$), hence at most $2^{d(T)}.s(T)\hat{}\hat{}d(T)$ many inequivalent combinations of $q(x), p_1(x), \ldots, p_n(x)$. $\square$

The Context Lemma for μPCF itself is reminiscent of the Context Lemma for PCF. It says that evaluation contexts plus a single top-level continuation suffice to distinguish inequivalent terms.

Proposition 5.1.6 (Context Lemma for μPCF) *For any closed terms $M, N : T$ of μPCF (call-by-name or call-by-value), $M \sqsubseteq^{OBS}_T N$ if and only if for all ground-type evaluation contexts $E^T[\cdot] : \mathbf{0}$,*

$$\mu\alpha.E[M] \Downarrow \implies \mu\alpha.E[N] \Downarrow .$$

PROOF: Suppose $M, N : T$ are terms of call-by-value μPCF, such that $M \not\sqsubseteq^{OBS}_T N$, then $[\![M]\!] \not\lesssim_{[\![T]\!]} [\![N]\!]$ By Proposition 5.1.1 there is a (finitary) element $\rho : o^\omega \to ([\![T]\!] \Rightarrow o)$ such that $\Lambda(\rho \times [\![M]\!]; \mathrm{App}) \downarrow$ and $\Lambda(\rho \times [\![N]\!]; \mathrm{App}) \not\downarrow$.
By definability, there is a value with a free name $\alpha : \mathtt{nat} \vdash V : T \Rightarrow \mathbf{0}$ such that $[\![V]\!]$ is the lifting of ρ.
$[\![\mu\alpha.V\ M]\!] = \Lambda(\rho \times [\![M]\!]; \mathrm{App})$, and $[\![\mu\alpha.V\ N]\!] = \Lambda(\rho \times [\![N]\!]; \mathrm{App})$ and hence by adequacy $\mu\alpha.VM \Downarrow$ and $\mu\alpha.V\ N \not\Downarrow$ as required.
Similarly if $M, N : T = \overline{S} \Rightarrow \mathtt{nat}$ are terms of call-by-name μPCF such that $M \not\sqsubseteq^{OBS}_T N$, then there are finite strategies $\rho_1 : o \to [\![S_1]\!], \ldots, \rho_n : o \to [\![S_n]\!]$, and $\rho' : o \Rightarrow o$ such that
$\Lambda(\langle \rho_1, \ldots, \rho_n \rangle \times [\![M]\!]; \mathrm{App}) \downarrow$ and $\Lambda(\langle \rho_1, \ldots, \rho_n \rangle \times [\![N]\!]; \mathrm{App}) \not\downarrow$.
By definability, there are terms $L_1(\alpha), \ldots, L_n(\alpha)$ denoting $\rho_1, \ldots, \rho_n$.
Either $\rho' = id_o$ or $\rho' = \bot_{o,o}$, — in the former case set $E[\cdot] = \mu\alpha.[\alpha][\ ]L_1 \ldots L_n$, — in the latter $E[\cdot] = \mu\alpha.\mathrm{IF}\ [\cdot]$ then $\Omega^{\mathbf{0}}$ else $\Omega^{\mathbf{0}}$, to give the required evaluation context. $\square$

The Context Lemma allows a simple proof of the following observation of Cartwright and Felleisen [15], that adding errors makes all control behaviour observable and hence:

Corollary 5.1.7 *Observational equivalence in* μPCF^E *is extensional. i.e.* $M \sqsubseteq_T^{OBS} N$ *if and only if for all evaluation contexts* $E^T[\cdot] : \mathtt{nat}$*,* $E[M] \Downarrow \mathbf{e_i} \implies E[N] \Downarrow \mathbf{e_i}$.

Proof: Recall that terms M^* of μPCF^E are soundly translated into terms M^* of μPCF by including an additional free variable α, labelling errors $\mathbf{e_i}$ as $\mu\beta.[\alpha]2i$. Hence $M \not\sqsubseteq_T^{OBS} N$ if $\mu\alpha.E[M^*] \Downarrow$ and $\mu\alpha.E[M^*] \not\Downarrow$ for some evaluation context $E[\cdot]$, containing only α free, by the Context Lemma. As $E[\cdot]$ can be assumed to consist of the finite evaluation trees given in the Definability Theorem, α is used to name only integers, and can be replaced with error terms to give an evaluation context $E'[\cdot]$ such that $E'[M] \Downarrow \mathbf{e_i}$ and $E'[N] \not\Downarrow \mathbf{e_i}$ as required. $\square$

Although the search for witnesses to observational inequivalence can be restricted using the Context Lemma, this is still not sufficient to show decidability of the intrinsic preorder in the games model, as the set of plays in most innocent strategies over arenas of depth greater than one is unbounded, as Opponent can simply repeat a move ad infinitum. However, this is not constructive opposition; although it may be necessary to repeat moves to obtain intensional information, there is an intuition that in a finite arena, one can observe only finitely many different Player responses (as there are only finitely many different paths through the 'game tree'). Therefore it should be possible to bound the size of the Opponent strategies needed to distinguish inequivalent finitary strategies. This is a non-trivial but feasible task, as a comparison with the problem of decidability in the 'minimal model', below, shows.

5.1.2 Observational equivalence in the λ-calculus

Although finitary PCF is not effectively presentable, there are even weaker extensional languages which are; notably PCF over a ground type with a single value, and the 'minimal' functional language:
the λ-calculus with a finite number of constants at a single ground type ι.
($\Lambda(\Omega)$ fits into this definition but ground-type contextual equivalence for $\Lambda(\Omega)$ is trivial.) The simplest meaningful version of the problem is thus: determine observational equivalence for the λ-calculus $\Lambda(\Omega, \top)$ with *two* constants, one denoting divergence, Ω, and the other convergence, $\top$. The observational equivalence on this language was shown by Padovani to be decidable [68].

It is shown below that the contextual equivalence decision problems for $\Lambda(\Omega)$ (at first-order) (and hence finitary μPCF) and $\Lambda(\Omega, \top)$ (at ground type) are equivalent (and establishing this is much simpler than the solution to either problem). Thus the undecidability in the case of finitary PCF appears as an increasingly isolated result: not only are there decision procedures for the problem in both strictly weaker (unary PCF, linear PCF) and stronger versions (μPCF, idealized Algol), but in the case of $\Lambda(\Omega, \top)$ and μPCF there is also a simple connection between these results.

Definition 5.1.8 *As with $\Lambda(\Omega)$ one can define a notion of reduction for $\Lambda(\Omega, \top)$ (this time to ground type), defining an evaluation relation on closed terms of type ι using the strong normalization theorem,*
i.e. $M \Downarrow$, if $M : \iota =_{\beta\eta} \top$, and $M \not\Downarrow$ if $M =_{\beta\eta} \Omega$.

As one would expect, the observational preorder is extensional, in other words, the following context lemma holds.

Proposition 5.1.9 *For all closed terms $s, t : T_1 \Rightarrow (\ldots(T_n \Rightarrow \iota)\ldots)$,*
$s \sqsubseteq^{OBS}_{\Lambda(\Omega,\top)} t \iff \forall a_1 : T_1, \ldots a_n : T_n,$
$s\ a_1\ \ldots a_n \Downarrow\ \Rightarrow t\ a_1 \ldots a_n \Downarrow$.

Proof: is standard, simplified version of the proof of the Context Lemma for PCF [61]. Or it can be deduced from the Context Lemma for $\Lambda(\Omega)$. $\square$

Just as for $\Lambda(\Omega)$, the Context Lemma can be used to show that there are only finitely many observational equivalence classes at each type. Hence observational equivalence can be decided by giving an algorithm which generates a finite list of terms at each type, containing at least one representative from each equivalence class. Such an algorithm for $\Lambda(\Omega, \top)$ was described by Padovani (a simpler version is described by Loader [53], following Schmidt-Schaüss's analysis for unary PCF).

Theorem 5.1.10 (Padovani) *The 'minimal model' of $\Lambda(\Omega, \top)$ is effectively presentable.*

Despite the obvious differences between the two preorders, and hence the fully abstract models, the closeness of the syntax and notion of reduction allows each to be used to reason about the other. In fact, the problems of effective presentability for the two models are easily shown to be equivalent.

Proposition 5.1.11 *The observational preorder on $\Lambda(\Omega)$ is decidable if and only if it is decidable in $\Lambda(\Omega, \top)$.*

PROOF: **Proposition 5.1.12** *For all terms $s, t : T$ of $\Lambda(\Omega)$, (which are also terms of $\Lambda(\Omega, \top)$)*

$$s \sqsubseteq^{OBS}_{\Lambda(\Omega,\top)} t \iff s \sqsubseteq^{OBS}_{\Lambda(\Omega)} t.$$

PROOF: Suppose $s \not\sqsubseteq^{OBS}_{\Lambda(\Omega,\top)} t$, then for some context $C^T[\cdot] : \iota \Rightarrow \iota$, $C[s] \Downarrow$ and $C[t] \not\Downarrow$, hence $(C[s]\ \top) \Downarrow \top$ and $(C[t]\ \top) \not\Downarrow \top$, so $s \not\sqsubseteq^{OBS}_{\Lambda(\Omega)} t$ as required.
Suppose $s \not\sqsubseteq^{OBS}_{\Lambda(\Omega)} s$, then for some ground-typed $\Lambda(\Omega, \top)$ context $C^T[\cdot]$, $C[s] \Downarrow$ and $C[t] \not\Downarrow$. Then as there are no reduction rules for the constants, $C[s][x/\top]$ normalizes to x, and $C[t][x/\top]$ normalizes to Ω.
Hence $\lambda x.C[s][x/\top] \Downarrow$ and $\lambda x.C[t][x/\top] \not\Downarrow$, i.e. $s \not\sqsubseteq^{OBS}_{\Lambda(\Omega,\top)} t$ as required. □

Corollary 5.1.13 *Fully abstract models of $\Lambda(\Omega)$ (and thus μPCF) are effectively presentable.*

However, the converse can also be shown.

Proposition 5.1.14 *If the minimal model of $\Lambda(\Omega)$ is effectively presentable, then so is the minimal model of $\Lambda(\Omega, \top)$.*

PROOF: The observational preorder on $\Lambda(\Omega, \top)$ is decidable as follows.
Given terms of $\Lambda(\Omega, \top)$, $s, t : T$, by the Context Lemma, $M \not\sqsubseteq^{OBS}_{\Lambda(\Omega,\top)} N$ if and only if there exist $r_1, \ldots r_n$ such that $s\ r_1 \ldots r_n \Downarrow \top$ and $t\ r_1 \ldots t_n \not\Downarrow \top$.
Thus $s \not\sqsubseteq^{OBS}_{\Lambda(\Omega)} t$ if and only if there exist $\Lambda(\Omega)$-terms $p_1(x), \ldots p_n(x)$ such that $(s[x/\top]\ p_1 \ldots p_n)$ normalizes to x, and $(t[x/\top]\ p_1 \ldots p_n)$ normalizes to Ω (take $r_i = p_i[\top/x]$),
i.e. $\lambda x.s[x/\top]\ p_1(x) \ldots p_n(x)) \Downarrow$ and $\lambda x.(t[x/\top]\ p_1(x) \ldots p_n(x)) \not\Downarrow$.
If $p_i(x) \equiv_{\Lambda(\Omega)} p'_i(x)$, then $\lambda x.s(x)\ p_1 \ldots p_n(x) \Downarrow$ if and only if $\lambda x.s(x)\ p'_1 \ldots p'_n(x) \Downarrow$.
So it is only necessary to test $s[x/\top], t[x/\top]$ against all combinations of representatives of the different $\Lambda(\Omega)$ observational equivalence classes of $p_1, \ldots p_n$, of which there are finitely many, so assuming a listing algorithm for $\Lambda(\Omega)$, observational equivalence for $\Lambda(\Omega, \top)$ is decidable. □

The effective presentability problem for μPCF has been answered in the affirmative. However, reasoning syntactically and indirectly via a (quite complex) proof for another language is not a very informative analysis. The solution to the effective presentability problem, as for full abstraction, should be syntax independent.

Remark 5.1.15 *Padovani's result was achieved in the course of investigation into the 'higher order matching problem' (given $r : S \Rightarrow T$, and $T : T$, is there some $s : S$ such that $r\ s =_{\beta\eta} t$?) of which it gives a solution in the case where*

t is an atom. Thus the characterization of $\Lambda(\Omega)$-equivalence given here will be a semantic *solution to this problem, and will also solve a similar problem: decidability of higher-order matching with respect to the largest non-trivial congruence containing $\beta\eta$ equivalence.*
Given a listing algorithm, higher-order matching with respect to $\simeq$ is decidable as observational equivalence classes of $\simeq$ are finite.

Thus far little attention has been paid to the question of which are the *extensional* functions computed in models of control, with the justification that control is intrinsically an intensional affair. The order-extensionality of the fully abstract model of SPCF with errors might be considered as contradicting this last assertion. However, the significance of the extensionality of control with errors is that it exposes an incompatibility between the formal definition of extensionality (which it satisfies), and a more informal notion of extensional model as given by some order-theoretic or topological description of set-theoretic functions (such as continuity, stability etcetera). The SPCF result highlights the ambiguities created by conflating these notions, as it shows that using errors it is possible to determine precisely *how* a function computes purely from the results of applying it to different arguments. Because intensional behaviour is made explicit by the error values, there can be no way of giving such an abstract description of the model.

A different strategy is to consider the extensional functions which are 'computed' (in some sense) by a category of intensional objects. The fully abstract model of PCF for instance, can be obtained as the extensional collapse of the highly intensional games and strategies model.

Berry and Curien's definition of the sequential algorithms [9] is also in this vein. It developed from an attempt by Kahn and Plotkin [45] to define a notion of sequential function made by structuring domains as concrete data structures (CDS). This allows higher-order computation to be modelled as a process of information-gathering by filling cells with values. A function between CDSs is sequential if it can be computed in a concrete data structure in a deterministic way; that is, there are 'sequentiality indices' at each point in the computation specifying the next cell which must be filled in the argument, to give more information at the output. The category of sequential functions is not, however, cartesian closed; to gain this semantic 'good behaviour' it is necessary to specify them along with sequentiality indices bearing witness to their sequentiality, forming the category of sequential algorithms.

An alternative notion of sequential functional 'realized' by the sequential al-

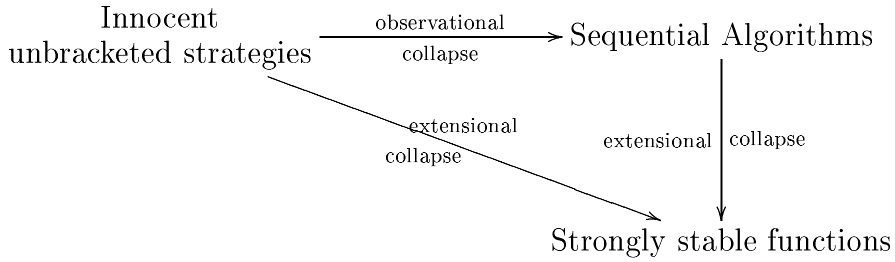

Figure 5.1: Relationship of three models of call-by-name PCF

gorithms has been considered by Ehrhard [24], who proved that the extensional collapse of the sequential algorithms model of PCF is isomorphic to Bucciarelli and Ehrhard's strongly stable model, or, in other words, that every strongly stable functional over a PCF-type is computed by a sequential algorithm. The proof that the intensional collapse of the unbracketed games model is isomorphic to the sequential algorithm can thus be 'composed' with this one, to yield the result that the extensional collapse of the unbracketed games model is also isomorphic to the strongly stable model (see Figure 5.1, and Corollary 5.4.23). Work by Longley [55], amongst others, has connected this sequentiality to the formal notion of realizability.

5.2 A sequential algorithms model of control

The description of the rich category of sequential algorithms, is necessarily somewhat terse, and slanted towards game semantics. A wealth of further material, including relationships with other semantics of for sequential languages is to be found in Curien's book [18], and article [19].

Following Cartwright and Felleisen's discovery that SPCF has a fully abstract semantics in terms of 'decision trees', it was observed by Curien that the model could be given an alternative presentation using sequential algorithms, which was adopted by Cartwright, Curien and Felleisen in [16]. This theme was developed in Curien's article [19], which also made some connections with the game semantics. The content of this connection is that at the affine level, the structures of Abramsky-Jagadeesan (and subsequently Hyland-Ong) games, and Sequential Data structures are essentially the same. The differences lie in the way constraints are used to enforce functional behaviour, whilst permitting non-linearity. In sequential data structures, access to data is *shared* (strategies can see the whole game history), but any given test can be applied to a variable only once, so it doesn't get the chance to behave inconsistently. By contrast, information which is not relevant to an *innocent* strategy, (such as how many times a variable has been used) is hidden from it, but it can (and must) copy information as required. This contrast between global and local constraints can be observed directly in the differing ways of modelling a co-monadic !, to yield an intuitionistic type-structure in the co-Kleisli category.

- *Plays* (or paths) in the sequential algorithms ! are more constrained, in that repeated queries are not allowed, and Player must respond to the most recent Opponent move.

- The *strategies* on dialogue games are constrained by *visibility* and *innocence*; although repetitious queries are allowed, Player must make the same response to them every time.

The games models of state [2], [4] are given by succesively dropping innocence and visibility, reflecting a setting where both copying and sharing of data is allowed, to model local binding of imperative variables. One consequence of interpreting functional behaviour in sequential algorithm style by allowing sharing but limiting copying is that this hierarchy of models is not available (sequential algorithms model of state could be constructed by allowing repetitions of moves but even strategies such as the identity will behave differently from the purely functional model).
The structure of arenas and games defined in Chapter 3 can be used to define a category of sequential data structures and sequential algorithms (as a particular category of games and strategies). In fact, they are rather 'over-defined', the roles of enabling and justification being somewhat superfluous. So the following definitions differ in terminology from Curien's definition of 'concrete sequential algorithms' [19], but they can readily be seen to be equivalent. ('addresses' correspond to Opponent moves, 'data' to Player moves, 'queries' to positions with Player to move (odd-length plays), and 'responses' to positions with Opponent to move (even length plays)).

Definition 5.2.1 *A Sequential Data Structure (SDS) arena*
$A = \langle M_A, \vdash_A, P \cup \{O, P\} \cup \{Q, A\}, \lambda_A \rangle$ *is an alternating, bracketed arena with an additional labelling of moves as* paths — *i.e. non-empty sequences of moves over an SDS arena (without their justification pointers). These will be called path-moves. Rather than extend the labelling function, each path-move will be written as the corresponding sequence.*

A sequential data structure is the game generated from a SDS-arena by alternation, the bracketing condition, and the following rule.

- *No repetitions of moves in plays: If $sa \sqsubset tb$, then $a \neq b$*

The product and function space constructions on SDS arenas are exactly as in Definition 3.2.2.

The path-moves for the SDS arenas used in the models of control will be constructed from the plays of a SDS, and hence will satisfy these rules as well (but they need not be *required* to do so). The above rules satisfy the identity and compositionality conditions and hence define a category. (This is easy to see, — as no moves can be repeated by any sequential algorithms, there can be no repetitions of moves in the 'interaction sequence' of two sequential algorithms, and hence none in the plays of their composition.) The standard constructions of Section 3.2.2 forming function-space and product arenas then give a symmmetric monoidal closed structure (together with the empty and one-move arenas).

Definition 5.2.2 *A sequential algorithm is a deterministic, history sensitive strategy over a sequential data structure A (i.e. a prefix-closed set of evenly-branching even-length paths of A, in which the final move can depend on the entire history of the play).*
The category of affine sequential algorithms has sequential data structures as objects, and sequential algorithms on $A \multimap B$ (defined in the standard way) as morphisms from A to B.

Comparison with the original definition of the Berry-Curien Sequential algorithms over concrete data structures is described formally and in detail in [19]. This shows that the sequential data structures are equivalent to *filiform* concrete data structures. The latter are defined by a quadruple $(C, V, E, \vdash)$ of *cells, values, events* $(\subseteq C \times V)$ *and enablings* $(\subseteq (E \times C) \cup C)$. An *event* consists of the filling of a cell with a value; this enables further cells to be filled, and so on. A (consistent and safe) state of the CDS is a set of events such that each filled cell is enabled and contains at most one value. Filiform CDS are assumed to be stable, — each cell which is filled within a state has a unique enabling in that state.
Each SDS defines a stable filiform CDS in which:

- *cells* are odd-length paths,
- *values* are Player moves, and
- *events* are even-length paths (filling a cell with a value is thus extending an position with player to move with a Player move),

- An event *enables* a cell if the latter (odd-length path) is the direct extension of the former (even-length path) with an Opponent move.

Thus states of the filiform CDS correspond via consistency and safety to even-prefix closed sets of plays; i.e. strategies.
Conversely, given a stable fcds, an equivalent sequential data structure can be constructed by taking Opponent moves to be cells, Player moves to be events, and the paths to be all alternating sequences of moves such that each Player move is the filling of the previous cell with a value, and each Opponent move (cell) is enabled by the previous Player move (event). Sequential algorithms over these equivalent structures can be defined in an abstract or concrete way; 'Concrete' (affine) algorithms from A to B are just strategies on $A \multimap B$ as in the standard linear category. Abstract algorithms (the original presentation) are sequential functions between the states of A and B together with sequentiality indices, which can be defined in various ways described in [19]; they form a CCC.

A subcategory, the abstract affine sequential algorithms over sequential data structures is axiomatized and shown to be equivalent to the concrete affine algorithms. The inclusion of the affine algorithms into the cartesian closed category has a left adjoint, and resolving this defines a co-monad on the affine (concrete) category, such that the co-Kleisli category is equivalent to the CCC of abstract algorithms. This can be described at the level of games semantics as follows.

As moves of a sequential data structure are, in general, sequences of moves, the paths of A can be used as the moves of $!A$. By definition of a sequential data-structure, paths in $!A$ are thus non-repetitive sequences of path-moves such that Opponent moves (questions) are odd-length and extend *some* previous move and Player moves (answers) are even-length and extend *the* preceding path-move. In other words, a path in $!A$ consists of a sequence of *different* interrogations of a single Player strategy on A.

Definition 5.2.3 *For a sequential data structure A, define $!_S A$ to be the SDS with the arena:*

- $M_{!_S A} = P_A$ *— the sequences over A satisfying the bracketing and no-repetitions rules.*

- $\lambda^{OP}(m) = O$*, and* $\lambda^{QA}(m) = Q$ *if m is an odd-length path in A.*
  $\lambda^{OP}(m) = P$ *and* $\lambda^{QA}(m) = A$ *if m is an even-length path in A.*

- $* \vdash_{!_S A} m$*, where m is a path in A consisting of a single move,*
  $p \vdash_{!_S A} pa$ *where pa is a direct extension of p.*

Note that this agrees with the convention that answers should be justified by questions. Defining moves of $!_S A$ as paths of A is not strictly necessary — one could use the same moves as A, and use justification pointers to reconstruct path-moves, but it allows the linearity (i.e. non-repetitivity) condition to be used in a straightforward way, by packaging the 'justification history' of a move up with it.

Proposition 5.2.4 *$P_{!_S A}$ consists of alternating, non-repetitive sequences of moves subject to the following condition:*
If $\lambda^{OP}(pa) = O$, then $s(pa) \in P_{!_S A}$ if and only if $s \in P_{!_S A}$ and pa is a path-move of length one, or there is some previous Player move p in s such that pa extends p.
If $\lambda^{OP}(pa) = P$, then $s(pa) \in P_{!_S A}$ if and only if $s \in P_{!_S A}$ and pa extends the preceding Opponent move.

PROOF: is direct by application of the bracketing condition □

Note that the rôle of the bracketing condition is rather different here than in the previous chapters. It does not enforce 'local control flow', but permits presentation of the sequential algorithms ! as a game generated on an arena. Further, whilst the 'switching condition' does not hold in general, it does hold for all games of the form $!(A \otimes B)$, and $!(A \otimes B) \multimap o$, which is sufficient for it to hold throughout the CCC with objects freely constructed from o (as defined below).

The sequential data structure and dialogue games versions of the ! both construct 'linearized' versions of the game tree, allowing Opponent to explore different branches of Player's strategy (see Figure 5.2. The key difference is that branches of the tree (from root to tip) are represented as interleaved 'threads' — (i.e. subsequences) of a dialogue, and as single *moves* of the SDS, without the same possibilities for interleaving.

Proposition 5.2.5 *$!_S$ is a co-monad, and its co-Kleisli category is cartesian closed.*

PROOF: is direct from the description of $!_S$ in [19] as the resolution of an adjoint inclusion of an affine category into a CCC. However, it will prove useful to have a concrete description of $!_S$ as a functor and co-monad.

Definition 5.2.6 *Define the following projection $\lceil \_ \rceil$ from sequences $J_{!_S A \multimap !_S B}$ to paths over $A \multimap B$ $P_{A \multimap B}$ which is similar to the view function, except that it also reduces path-moves to moves of A or B by extracting the last move.*

- $\lceil s(p) \cdot t(pa) \rceil = \lceil s(p) \rceil a$, *if pa is an Opponent move justifying (i.e. directly extending) p,*

- $\lceil s(pa) \rceil = \lceil s \rceil a$ *if pa is a Player move.*

Let $\chi : A \multimap B$ *be a sequential algorithm expressed as a function fromodd-length paths to moves, and define the function* $!_S\chi :!_S A \multimap !_S B$ *from sequences of path-moves to path-moves as follows:*
If $(\chi(\lceil s \rceil)$ *is a move in in B, then* $!_S\chi(s) = (\lceil s \rceil \upharpoonright A)\chi(\lceil s \rceil)$. *Otherwise, it is necessary to consider whether the move* $p = (\lceil s \rceil \upharpoonright A)\chi(\lceil s \rceil)$ *occurs in s or not. If it does, then it is succeeded by some Opponent move pa which extends it. Then let* $!_S\chi(s) = !_S\chi(s(p)(pa))$, *otherwise, let* $!_S\chi(s) = p$.
Note that $!_S\chi$ *is necessarily defined on sequences which contain repetitions, so that the sequential algorithm is in fact the restriction of* $!_S\chi$ *to sequential algorithm paths. Verification that* $!_S$ *is a functor is now straightforward. The co-monadic structure of the* $!_S$ *can be summarised as follows. For each SDS A, there are sequential algorithms*

- $\mathtt{der^S}_A :!_S A \multimap A$ *is essentially a copycat defined as follows:*

  - $\varepsilon \in \mathtt{der^S}_A$
  - $sa(sa\upharpoonright A) \in der_A$, *if* $s \in \mathtt{der^S}_A$ *and* $a \sqsubseteq A$ *(so* $(sa\upharpoonright A)$ *is a path move in* $!_S A$*)*

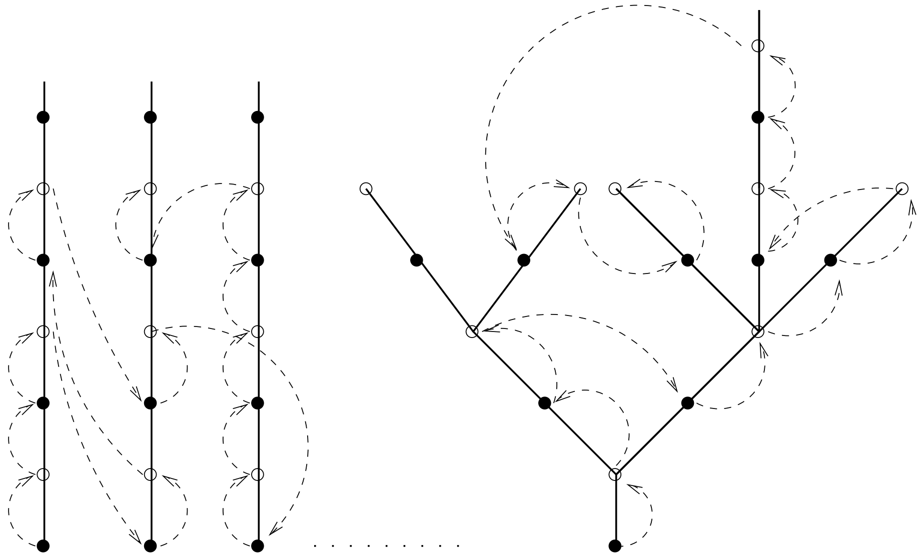

Figure 5.2: Plays in dialogue games and sequential algorithms 'banged' games: thick lines represent different *threads* of play

– $s(pab)b \in \mathtt{der^S}_A$, *if* $s \in \mathtt{der^S}_A$ *and* $pab \sqsubseteq !_S A$

- *A promotion algorithm* $\mathtt{prom^s}_A : !_S A \multimap !_S !_S A$

  – $s(p(ua))ua \in \mathtt{prom^s}_A$ *if* $s \in \mathtt{prom^s}_A$ *and* $p(ua) \in M_{!_S !_S} A$ *and* $ua \in M_{!_S A}$ *does not occur previously in* s

  – $s(p(ua))(p(uab)) \in \mathtt{prom^s}_A$ *if* $s \in \mathtt{prom^s}_A$ *and* $p(ua) \in M_{!_S !_S A}$ *and* $ua \in M_{!_S A}$ *does occur previously in* s*, where it is directly followed by* uab*.*

  – $s(p(ua))(ua)(uab)(p(uab))$ *if* $s(p(ua))(ua) \in \mathtt{prom^s}_A$

- *for any SDS* A, B*, there is an isomorphism* $!_S(A \times B) \multimap !_S A \otimes !_S B$*, yielding a contraction mapping* $\mathtt{con^S}_A : !_S A^- \multimap !_S A_1 \otimes !_S A_2$ *definable as follows:*
  $s(pa)_i pa^- \in \mathtt{con^S}_A$ *if* $s \in \mathtt{con^S}_A$ *and* $pa_i \sqsubseteq M_{A_i}$ *does not occur in* $s \restriction A^-$
  $spa_i pab_i \in \mathtt{con^S}_A$ *if* $s \in \mathtt{con^S}_A$ *and* $pa_i \sqsubseteq M_{A_i}$ *does occur in* $s \restriction A^-$*, where it is immediately followed by* pab*.*
  $spab^- pab_i \in \mathtt{con^S}_A$ *if* $s \in \mathtt{con^S}_A$ *and* pab *extends (uniquely) some move* $pa_i \sqsubseteq M_{A_i}$ *already played in* s*.*

These morphisms satisfy the conditions described in Chapter 3. □

Note that the natural transformations giving the co-monadic structure of $!_S$, — in particular promotion and contraction — are not true copycats; they rely critically on the history of the interaction to determine whether a move has already been played, and so avoid repetititions.

Having defined a CCC of sequential algorithms, $\mathcal{S}$, models of call-by-name and call-by-value μPCF are accessible by standard constructions. The one-move game o is a sequential data structure, and there are bottom maps (empty strategies), and ω-indexed products for each object. As in the rest of the category of games, set-inclusion of strategies is a cpo-enrichment.

Proposition 5.2.7 $(\mathcal{S}, o)$ *defines a computational (continuous) model of control: i.e.* $\mathcal{S}$ *is a cpo-enriched CCC with* ω*-indexed products, and* o *is non-terminal.*

The following example contrasts computational behaviour in the sequential algorithms and innocent strategy models.

Example 5.2.8 *Consider the sequential algorithms and innocent strategy interpretations of* $\lambda f : \mathtt{nat} \to \mathtt{nat}.f\ (f\ 0) : (\mathtt{nat} \to \mathtt{nat}) \to \mathtt{nat}$ *(Figure 5.3). This is the simplest term to exhibit imbrication (the application of a function variable to an argument which calls the variable itself). This 'diabolical' non-linearity is what* *makes determining observational equivalence a hard problem. In innocent strategies it corresponds directly to* interleaving*; returning to a thread after opening another in the same component. Plays in the innocent strategy (top) are uniform; it is just blindly,* innocently *copying moves, unlike the corresponding sequential algorithm, which makes a devious exit if it discovers that zero is a fixed point of its argument. (Otherwise the repetition condition would be broken). The sequential algorithm has already done some computation, — the innocent strategy can only compute by composition with other strategies.*

A significant point about the 'no-repetitions' condition is that it is symmetric between Player and Opponent: an Opponent strategy on A can be turned into a Player strategy on $A \multimap (o \Rightarrow o)$ just by adding additional initial and final moves. Consequently the category is isomorphic to its own 'observational collapse'.

Proposition 5.2.9 *The intrinsic preorder and equivalence on sequential algorithms is trivial — i.e. if $\subseteq_A$ is the subset relation on the concrete representations of algorithms as strategies, then*

$$f \lesssim_A g \iff f \subseteq_A g.$$

Proof: The implication from right to left is trivial; for the converse, suppose that $f \not\subseteq g$,
Then the promotion of f is not included in the promotion of g, i.e. $!f \not\subseteq !g$.
By prefix closure, there is some path $p \in P_{!_S A}$ of finite length, such that $p \in !f$

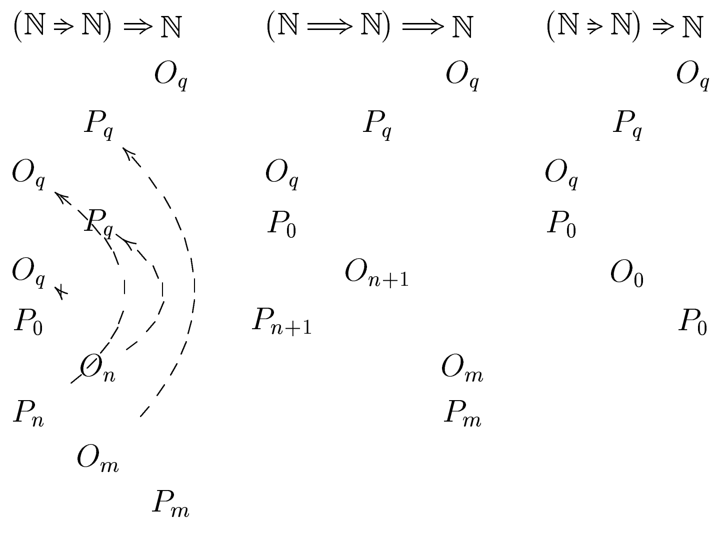

Figure 5.3: Contrasting plays for computing $f\ (f\ 0)$:
Left is the general innocent strategy, *Centre* is the sequential algorithm (assuming $f\ 0 \neq 0$), *Right* is the sequential algorithms play for $f\ 0 = 0$

but $p \not\in !g$.
Then form the sequential algorithm $h : !_S A \multimap (o \Rightarrow o)$ consisting of even prefixes of $op \cdot o'$ (where o, o' are the initial move and the response in $o \multimap o$).
Then $f; h \downarrow$ and $g; h \not\downarrow$ and so $f \not\lesssim_A g$ as required. $\Box$

Corollary 5.2.10 *The semantics of $\Lambda(\Omega)$ (and hence μPCF) in $(\mathcal{S}, o)$ is fully abstract if every finite sequential algorithm is definable.*

5.3 A fully abstract and universal model of μPCF

In fact, it is relatively straightforward to prove that every recursive element of the sequential algorithms model of control is μPCF definable. A universality result for the observably sequential functional model of SPCF has been achieved by Kanneganti, Cartwright and Felleisen [46]; this is dependent on the existence of errors, and is presented rather differently to the main result of this section, which is the following:

Theorem 5.3.1 *The semantics of $\mu\mathrm{PCF}_n$ in the category $\mathcal{S}$ of sequential data structures and* computable *sequential algorithms is universal.*

(The result is proved for $\mu\mathrm{PCF}_n$ for the sake of simplicity but there is no obstacle to extending it to include the call-by-value version.) As observed in Chapter 2, Section 2.5.1, this has the following corollary.

Corollary 5.3.2 *The semantics of $\mu\mathrm{PCF}_n$ in $\mathcal{S}$ is logically fully abstract.*

5.3.1 The computable sequential algorithms

The Universality Theorem is proved (as in [6],[43]) by defining a family of 'universal terms' for the language, relative to an effective enumeration of a finitary basis for the category.

Definition 5.3.3 *An effective coding of the computable sequential algorithms is a family of (effective) surjective functions indexed over the μPCF types, $\{\phi_T : \mathtt{nat} \to [\![T]\!]_{\mathcal{S}} \mid T \in TY_{\mu\mathrm{PCF}}\}$, giving a listing of the computable sequential algorithms on each type-object $[\![T]\!]$.*
A universal term *with respect to ϕ is a family of $\mu\mathrm{PCF}_n$ terms indexed over the μPCF types, $\{\mathcal{U}_T : \mathtt{nat} \to T \mid T \in TY_{\mu\mathrm{PCF}}\}$ such that*

$$[\![\mathcal{U}_T(n)]\!]_{\mathcal{S}} = \phi_T(n).$$

The first step in defining such a term is thus to fix a definition and encoding of the computable sequential algorithms.

Definition 5.3.4 *Suppose $p_0, p_1, \ldots, p_n, \ldots$ is an effective presentation of a finitary basis of a domain A (in this case a sequential data structure).*
i.e. a recursive enumeration of compact elements such that every sequential algorithm is a supremum of some directed set.
A sequential algorithm $\sigma : A$ is computable if it is the least upper bound of a recursively specified set, i.e.
there is a partial recursive function $f_\sigma : \mathbb{N} \to \mathbb{N}$ such that $\bigsqcup_{n\in\omega} p(f_\sigma(n)) = \sigma$.
Assuming a (computable) listing of the partial recursive functions on $\mathbb{N} \to \mathbb{N}$ in the form of a surjective recursive functional, $\psi : \mathbb{N} \to (\mathbb{N} \to \mathbb{N})$, this yields a listing of the computable elements of the domain as $\sigma_1, \sigma_2, \ldots$,
where $\sigma_i = \bigsqcup_{n\in\mathbb{N}} p_{\psi(i)(n)}$.

Thus it remains to give an effective presentation of a finitary basis for the sequential algorithms, (which differs from the standard notion of effective presentation only in that the effective operations defined on the basis are described so as to simplify the proof of universality). Every sequential algorithm is the supremum of its set of finite paths. Thus the effective presentation of this finite basis takes the form of an injection from paths over A to natural numbers.

Definition 5.3.5 (Encoding of paths) *Define* $\mathrm{path}_A : A \to \mathbb{N}$, *by structural induction on the SDS A.*
At the base cases: $\mathrm{path}_o(\varepsilon) = 0$, $\mathrm{path}_o(o) = 1$
$\mathrm{path}_{o^\omega}(\varepsilon) = 0$, $\mathrm{path}_{o^\omega}(m_i) = i + 1$.
Now suppose $A = B_1 \Rightarrow (B_2 \Rightarrow \ldots (B_n \Rightarrow o)\ldots)$, and define:
$\mathrm{path}_A(\varepsilon) = 0, \quad \mathrm{path}_A(o) = 1$
$\mathrm{path}_A(s(pa)) = (6.\mathrm{path}_A(s) + 1).2^{\mathrm{path}_{B_i}(p)}.3^i$, *where p is a path in B_i.*

Thus path_A is essentially an encoding of finite lists (of lists).

Proposition 5.3.6 *There are recursive (μPCF definable) functions for the following operations on the effective presentation:*

- $\mathrm{init} : \texttt{nat} \to \texttt{nat}$ *and* $\mathrm{last} : \texttt{nat} \to \texttt{nat}$ *for taking initial segments and last elements, such that:*
  $\mathrm{init}(\mathrm{path}_A(sa)) = \mathrm{path}_A(s)$ *and if a is a move in B_i,*
  $\mathrm{last}(\mathrm{path}_A(sa)) = \mathrm{path}_{B_i}(a)$

- $\mathrm{comp}_A : \texttt{nat} \to \texttt{nat}$ *for returning the subgame in which the last move was made.*
  For each type-object $A = B_1 \Rightarrow (B_2 \Rightarrow \ldots (B_n \Rightarrow \iota)))$, *if* a *is a move from* B_i *then* $\mathrm{comp}_A(\mathrm{path}_A(sa)) = i$

PROOF: is by definability of all recursive functions in μPCF. □

It is also possible to (recursively) extract the code for the substrategy manifested in a path over $\overline{B} \Rightarrow o$, in a given subcomponent B_i, using the representation of strategies as least upper bounds of directed sets.

Proposition 5.3.7 *For each sequential data structure* $\overline{B} \Rightarrow o$, *there are* μPCF-*definable functions* $\mathrm{strat}_i : \texttt{nat} \to \texttt{nat}$ *such that*
if s *is a path in* $\overline{B} \Rightarrow o$, *and* $R_i(s) = \{p \in \mathrm{Occ(s)} \mid \mathrm{p} \in \mathrm{P_{B_i}}\}$ *is the strategy on* B_i *manifested in* s, *then* $\phi(\mathrm{strat}_i(\mathrm{path}_A(s))) = R_i(s)$.

PROOF: is by recursive definition of strat_i as follows.
Assume that our listing function for the partial recursive functions comes with a partial recursive functional hotel : $\texttt{nat} \times \texttt{nat} \to \texttt{nat}$, which takes the code of a recursive function and, a value n, and returns the code for a function with n as the new value of $f(0)$, and $f(k)$ as the value at $\texttt{succ}\ k$. i.e. $\psi(\mathrm{hotel}(i,n))\ 0 = n$ and $\psi(\mathrm{hotel}(i,n))\ k+1 = \psi(i,k)$.
Then let:

- $\mathrm{strat}_i(0) = \phi(\bot_A) = 0$,
- $\mathrm{strat}_i(n+1) = \mathrm{strat}_i(\mathrm{init}(n+1))$, if $\mathrm{comp}(\mathrm{last}(n+1)) \neq i$,
- $\mathrm{strat}_i(n+1) = \mathrm{hotel}(\langle \mathrm{strat}_i(\mathrm{init}(n+1)), \mathrm{last}(n+1)\rangle)$,
  if $\mathrm{comp}(\mathrm{last}(n+1)) = i$.

□

5.3.2 Definability

The critical part of the definition of a universal term is a proof of finite definability by an inductive decomposition of strategies. The proof here is similar in essence to that given for SPCF in [16], but it is (a little) more abstract being based on some general properties of the category, more general, (in that it applies, via invertible cps translation, to $\mu\mathrm{PCF}_v$), and simpler, as there is no use of error elements. The basic idea is to add an extra premise of type ι to the interpretations of types-in-context. Instead of simply giving up, partial strategies can use their last move to jump into this part of the context. This allows the flow of control to be used in definability; each algorithm can be decomposed into strictly smaller ones which are 'glued together' by passing the flow of control from one to the next.

Definition 5.3.8 *A sequential algorithm $f : A$ is total if every maximal length path in f is also maximal in A, — i.e. f always has a response to Opponent's last move.*

Proposition 5.3.9 *The finite and total sequential algorithms form a cartesian closed category $\mathcal{S}^T$.*

PROOF: is by a routine check that composition etc. preserves totality (which requires the hypothesis of finiteness). □

Recall from Chapter 2, Section 2.1 that any CCC gives rise to a pointed CCC, $\mathcal{C}_\perp$ as the co-Kleisli category of a co-monad which freely adjoins a ground type-object. So consider the pointed CCC $(\mathcal{S}^T)_\perp$ of total sequential algorithms over the one move game. This is isomorphic to $\mathcal{S}^f$, the CCC of finite sequential algorithms. If $\mathcal{C}$ already has $\perp$-maps, then there is an obvious functor $\perp : \mathcal{C}_\perp \to \mathcal{C}$:
$\perp(f : o \times A \to B) \longrightarrow \langle \perp_{A,o}, \mathtt{id}_A \rangle; f : A \to B$.

Proposition 5.3.10 $\perp : (\mathcal{S}^T)_\perp \cong \mathcal{S}^f$ *i.e. $\perp_o$ has an inverse*
$\widetilde{(\_)} : (f : A \to B) \longrightarrow \widetilde{f} : o \times A \to B$.

PROOF: $\widetilde{(\_)}$ can be defined by a 'factorization of partiality' which turns bottom into top (Figure 5.4), — whenever a partial algorithm f has no response, and would just give up, $\widetilde{f}$ jumps out to the adjoined copy of o, after which play stops as neither side can make a move. Composition with $\perp_o$ clearly returns the original strategy. Formally, given a sequential algorithm f as a function from odd-length, f-reachable paths to moves, define
$\widetilde{f}(s) = o$, if $f(s) \uparrow$,
$\widetilde{f}(s) = f(s)$, otherwise. □

Lemma 5.3.11 *For any sequential algorithms $f, g : A \to o$ such that f, g are bounded above, $f \sqcup g = \langle g, \mathtt{id} \rangle; \widetilde{f}$.*

PROOF: $\langle g, \mathtt{id} \rangle; \widetilde{f}$ plays as f until it reaches a position to which f has no response. It then plays as g, *except* that by the non-repetition condition, it can only play path-moves in $\overline{A}$ which extend the path-moves already played by f. Hence $\langle g, \mathtt{id} \rangle; \widetilde{f} = f \sqcup g$ as required. □

Factorizing partiality allows definability of a finite basis to be extended to recursive definability. A series of sequential alogrithms $f_0, f_1, \ldots$ can be 'linked' by playing $\widetilde{f_0}$ until Opponent gives up, or f_0 gives up, in which case $\widetilde{f_0}$ passes out control to $\widetilde{f_1}$. Infinite chains of factorized algorithms can be joined in this way using least fixpoints represented by the $\mathbf{Y}$-combinator.

Proposition 5.3.12 (Universality Lemma) *Suppose $\mathcal{U}^f$ is a universal term for factorized partial maps from a finite basis (i.e. $[\![\mathcal{U}^f_T(n)]\!] = \widetilde{f_n} : o \Rightarrow [\![T]\!]$ for some enumeration of a finite basis $f_0, f_1, \ldots$ of $[\![T]\!]$). Then there is a universal term for $\mathcal{S}$ with respect to that basis, given by $\mathcal{U}_{\overline{S} \Rightarrow \mathbf{0}} =$*

$$\lambda w\colon \mathtt{nat}.\lambda \overline{y}\colon \overline{S}.(\mathbf{Y}\lambda F : \mathtt{nat} \Rightarrow \mathbf{0}.\lambda x : \mathtt{nat}.(\mathcal{U}^f(x)\ F(\psi(w)\ (\mathtt{succ}\ x)))\ \overline{y})\ 0.$$

Similarly $\mathcal{U}_{\overline{S} \Rightarrow \mathtt{nat}} =$

$$\lambda w\colon \mathtt{nat}.\lambda \overline{y}\colon \overline{S}.\mu\alpha : \mathtt{nat}.(\mathbf{Y}\lambda F.\lambda x.[\alpha](\mathcal{U}^f(x)\ F(\psi(w)\ (\mathtt{succ}\ x)))\ \overline{y})\ 0.$$

PROOF: Suppose $\phi(n) = h$, i.e. $h = \bigsqcup_{i \in \omega} f_{\psi(n)(i)}$.
Then by Lemma 5.3.11, for each $k \in \omega$,
$\bigsqcup_{i \leq k} f_{\psi(n)(i)} = \langle f_{\psi(n)(k)}, \mathtt{id}\rangle; \widetilde{(\ldots \langle f_{\psi(n)(1)}, \mathtt{id}_A\rangle; \widetilde{f_{\psi(n)(0)}} \ldots)}$
$= [\![\mathcal{U}^f(0)\ (\mathcal{U}^f(\psi(n)(1))\ (\ldots (\mathcal{U}^f(\psi(n)(k))) \ldots))]\!]$
$= [\![\lambda F : \mathtt{nat} \Rightarrow \mathbf{0}.\lambda x : \mathtt{nat}.(\mathcal{U}^f(x)\ \overline{y})\ F(\psi(n)(\mathtt{succ}\ x)))\ 0]\!]^k$.
Hence by definition of $\mathbf{Y}$ as a least fixed point, $h = [\![\mathcal{U}(n)]\!]$ as required. □

The above example allows partiality *together* with a top element introduced by adjoining an answer-object to the context. This allows non-terminating behaviour to co-exist with 'error-detection' (which is how explicit error values are modelled

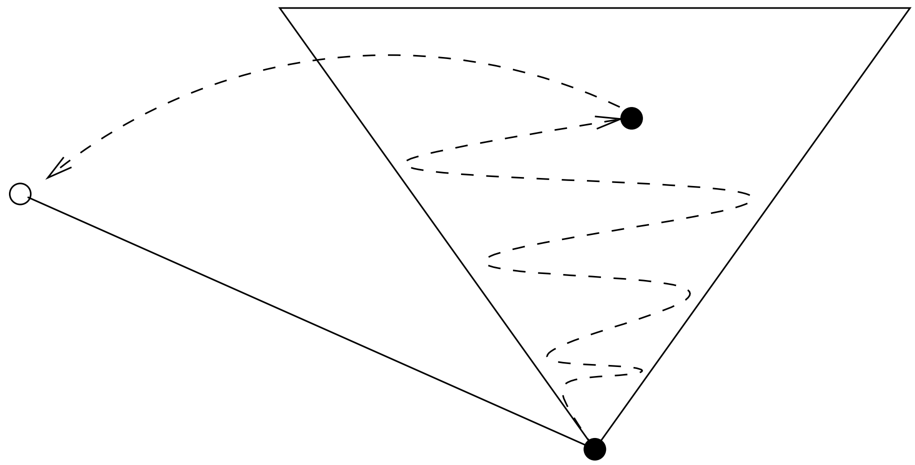

Figure 5.4: 'Partiality factorization'

in the CCF model). 'Escaping strategies' of this sort can be represented by strategies with (maximal) odd-length paths; a (maximal) even length play corresponds to non-termination (Player 'giving up') and a (maximal) odd-length play corresponds to an escape, or playing the top element.

Definition 5.3.13 *An escaping (deterministic) strategy on a game A is a non-empty, even-prefix-closed, evenly-branching subset of A such that if $s \in \sigma$ and $\exists t \in \sigma$ s.t. $s \sqsubset t$, then s is even-length.*
Then $\widehat{\sigma : o \Rightarrow} A = \sigma \upharpoonright A$ defines a bijection from (innocent) escaping strategies on A to (innocent) strategies on $o \Rightarrow A$ with inverse
$\sigma \longrightarrow \{s(even) \in \sigma\} \cup \{so \mid s(odd) \in \sigma \wedge \neg\exists(t \in \sigma) \sqsupset s\}$,
where o is the only move in o.
This extends to innocent strategies via the standard definition of innocence:
an escaping strategy σ is innocent if $sa, t \in \sigma \quad \wedge \ulcorner s \urcorner \in \sigma \implies ta \in \sigma$.

Composition of escaping strategies is defined as parallel composition with hiding as in Chapter 3, with the result that $\widehat{(\_)}$ is an isomorphism between the escaping strategies, and the category of games and strategies with a one-move game freely adjoined to the context (assuming a little currying).

Proposition 5.3.14 *Given $f : (o \times A) \to B, g : (o \times B) \to C$*

$$\widehat{f}; \widehat{g} = \widehat{\langle f, \pi_l \rangle}; g.$$

PROOF: is straightforward; note that if the last move of $s \in \widehat{f} \| \widehat{g}$ is in B and hence hidden, then $s \upharpoonright A, C$ is odd-length. Thus if an 'escape' occurs in a hidden part of the composition, then it corresponds to an escape from the unhidden part. $\square$

Escaping strategies on $A \to B$ inherit the order-enrichment on $o \times A \to B$,
— $f \leq g$ if and only if $\widehat{f} \leq \widehat{g}$.
Note that this is not longer simply inclusion, although observational equivalence of escaping sequential algorithms is still set-equality.

5.3.3 Decomposition of escaping strategies

A unique decomposition of escaping strategies can be found by considering strategies in which a *unique* Opponent strategy prompts an escape.

Definition 5.3.15 *A sequential algorithm $f : o \times \overline{A} \Rightarrow o$ is* uniquely escaping *if there is a* unique *(minimal) Opponent strategy* recognised *by σ; i.e. unique sequential algorithms $g_1 : A_1, g_2 : A_2, \ldots g_n$ such that*

$$\mathtt{id}_o \times \langle g_1 \ldots g_n \rangle; f \neq \bot_{o,o}.$$

Uniquely escaping strategies over A can therefore be identified with an even-prefix closed set of paths of which precisely one is odd-length, — corresponding to the Opponent-strategy which is recognised.

By the Universality Lemma, definability need only be proved for a particular class of (partially factorized) strategies — the basis corresponding to the finitary paths of the sequential data structure (and such strategies have the useful property of decomposing uniquely).

Definition 5.3.16 *For each a sequential data structure A, define the path-strategies over A to be uniquely escaping finitary sequential algorithms $g : o \to A$ such that if $f, f' \lesssim_A \perp_o; g$ then $f \lesssim_A f'$ or $f' \lesssim_A f$.*
So path-strategies are sequential algorithms which have a response to precisely one of the possible Opponent moves at any given point; at some stage, this is a move in o, and the interaction stops. Path strategies on A can therefore be uniquely represented by (and by abuse, identified with) an odd-length *path over A.*

Proposition 5.3.17 *If all of the (partial factorized) path-strategies over A are definable, then all of the finite sequential algorithms over A are definable.*

Proof: is via Lemma 5.3.11, as for the Universality Lemma. □

Definability of path strategies is proved by observing that each one has a unique decomposition into an initial segment followed by a Player move and a unique escape (the recognition of a unique Opponent move and a jump into the one-move game adjoined to the context). Note the contrast of this 'top down' induction with the decomposition of innocent strategies, which is 'from the bottom up'; using the response to the initial move to split the strategy into smaller strategies.

Proposition 5.3.18 (Decomposition Lemma) *Let $f : \overline{A} \times o \to o$ be a non-empty path-strategy. Then for some $i \leq n$ there is a path-strategy $f' \sqsubset f$ and unique minimal (uniquely escaping) sequential algorithms*
$g_{i1} : B_{i1} : o \to B_{i1}, \ldots g_{ib_i} : o \to B_{ib_i}$ such that

$$f = \langle \pi_l, \pi_i \times \langle g_{i1}, \ldots, g_{ib_i} \rangle; \mathrm{App} \rangle; f'.$$

Proof: Suppose $f = s(pa)(pab)$, where pa is a path in A_i, and so b is an (odd-length) path in some B_{ij} Then let $f' = s$, and $g_{i1} : o \Rightarrow B_{i1}, \ldots g_{ij}, \ldots g_{ib_i} : o \Rightarrow B_{ib_i}$ be the strategies manifested in pa;
i.e. $g_{ik} = \{c(even) \in \mathrm{Occ(pa)} \mid \mathrm{c} \in \mathrm{P_{B_{ik}}}\}$, for $k \neq j$.
Similarly g_{ij} is the *uniquely factorized* strategy on $o \Rightarrow B_{ij}$ which is manifested

in pab, with the unique escape given by the odd-length path b.
Then $\langle \pi_l, \pi_i \times \langle g_{i1}, \ldots, g_{ib_i}\rangle; \mathrm{App}\rangle; f'$ plays as f' until the position s is reached, whereupon it plays in B_i as $\pi_i \times \langle g_{i1}, \ldots g_{ib_i}\rangle; \mathrm{App} : \overline{A} \times o \to B_i$. But the only path-move in this strategy which can extend one already played, without repetition, is pa; if this is followed by the Opponent move pab, then Player escapes, otherwise he gives up. $\square$

Using this proposition, defining a universal term for path-strategies is simply a matter of re-constructing a path from the encoding of its decomposition.

Proposition 5.3.19 (Finite Definability) *There is a universal term of* $\mu\mathrm{PCF}_n$ *for the finite basis of path-strategies.*

PROOF: is by induction on type structure, defining a universal term $\mathcal{U}^f_T$ at each type for path-strategies, and using the recursive chain lemma above to give a term for general strategies.
Suppose $[\![T]\!] = A_1 \Rightarrow (A_2 \Rightarrow \ldots (A_m \Rightarrow o) \ldots)$, where $A_i = B_{i1} \Rightarrow (\ldots (B_{in_i} \Rightarrow o) \ldots)$, and that there are universal terms $\mathcal{U}_{ij}$ for the (partiality factorized) uniquely escaping partial sequential algorithms on each B_{ij}.
The following definition of $\mathcal{U}^f(n)$ is by course-of-values recursion on n which can then be implemented in μPCF. Supposing $T = S_1 \Rightarrow \ldots S_m \Rightarrow \mathbf{0}$, then for $i \leq m$, and $y_i : S_i = R_{i1} \Rightarrow \ldots R_{ik}$, define

$$M_i(x) = y_i\ (\mathcal{U}_{i1}(\mathrm{strat}_1(\mathrm{last}\ n))\ x) \ldots (\mathcal{U}_{ik_i}(\mathrm{strat}_k(\mathrm{last}\ n))\ x) : \mathbf{0}$$

and define $\mathcal{U}^f_T(n) =$

$$\lambda x : \mathbf{0}.\lambda \overline{y} : \overline{S}.(\mathcal{U}^f_T(\mathrm{init}^2 n)\ (\mathtt{case}_m\ \mathrm{comp}(n)|_{i \leq m}\ M_i))\ \overline{y}.$$

Supposing $\mathrm{path}_T(s(pa)(pab)) = n$, then $[\![\mathcal{U}^f_T(\mathrm{init}^2 n)]\!]$ is the path-strategy corresponding to s, and $\mathcal{U}_{ik}(\mathrm{strat}_k(\mathrm{last}\ n))$ is the Player strategy on B_{ik} manifested in pab. So $[\![M_i]\!]$ is the strategy on $B_i \to o$ manifested in pab (which is chosen by the conditional) and
$[\![\mathcal{U}^f_T(n)]\!] = \langle \pi_l, (\pi_i \times \langle [\![\mathcal{U}_{i1}(\mathrm{strat}_k(\mathrm{last}\ n))]\!], \ldots [\![\mathcal{U}_{in_i}]\!]\rangle); \mathrm{App}\rangle; [\![\mathcal{U}_T(\mathrm{init}^2 n))]\!]$.
So by the Decomposition Lemma (Proposition 5.3.18), $\mathcal{U}^f_T(n)$ is equal to the path strategy corresponding to pab, as required.
If $T = \overline{S} \Rightarrow \mathtt{nat}$, so $[\![T]\!] = \overline{[\![S_i]\!]} \Rightarrow (o^\omega \Rightarrow o)$ then define $\mathcal{U}_f(n) =$

$$\lambda x.\lambda \overline{y}.\mu\alpha.(\mathcal{U}^f_T(\mathrm{init}^2 n)\ (\mathtt{case}\ \mathrm{comp}(n)|_{i \leq m}\ M_i\ |_{m+1}\ [\alpha]\mathrm{last}\ n))\ \overline{y}$$

which is shown to be a universal term as above. $\square$

Thus the Universality Theorem has been proved. Finite definability for the (effectively presented) sequential algorithms also has the following consequence.

Corollary 5.3.20 *Observational equivalence for $\Lambda(\Omega)$, and for finitary μPCF is decidable.*

5.4 Relating models of control

Although full abstraction has been established, based on a decomposition of sequential algorithms, the characterization of observational equivalence of μPCF has been left wholly implicit in this direct presentation. By considering the relationship with the *initial* innocent strategies model, via the functor which collapses strategies under their intrinsic preorder, a much more informative analysis can be gained. Showing that this functor is surjective amounts to a second proof of full abstraction.

'Factoring' the proof of full abstraction through the proof of initiality of the games model in this way separates the properties of definability, and the denotational equality of observably equivalent objects. This has the potential to allow more syntactic information to be extracted, compared with the direct version given in Section 5.3 (or Cartwright, Curien and Felleisen's proof [16]). At the same time, this proof formalises and justifies the claim that innocent strategies are equivalent modulo duplication of queries, as the effect of the translation is to remove copies.

The Context Lemma for $\Lambda(\Omega)$ can be exploited to define inductively a kind of 'intensional logical relation' between pointed cartesian closed categories (and hence models of control).

Definition 5.4.1 *Let $(\mathcal{C}, \mathbf{a})$ and $(\mathcal{C}', \mathbf{a}')$ be pointed CCCs interpreting $\Lambda(\Omega)$, and define the following relation between 'escaping' morphisms over their type-objects, $\mathcal{R}_T : \mathcal{C}(\mathbf{a}, [\![T]\!]) \times \mathcal{C}'(\mathbf{a}', [\![T]\!])$, by induction on type structure as follows:*

- *$f\ \mathcal{R}_\iota f'$ if and only if $f := \mathtt{id}_\mathbf{a}$ and $f' = \mathtt{id}_{\mathbf{a}'}$, or $f = \bot_{\mathbf{a},\mathbf{a}}$ and $f' = \bot_{\mathbf{a}',\mathbf{a}'}$*
- *$f\ \mathcal{R}_{S \Rightarrow T} f'$ if and only if $\forall x \in \mathcal{C}(\mathbf{a}, [\![S]\!]), x' \in \mathcal{C}'(\mathbf{a}', [\![S]\!])$ such that $x\ \mathcal{R}_S x'$, $\langle f, x\rangle; \mathrm{App}\ \mathcal{R}_T = \langle f', x'\rangle; \mathrm{App}$ and $\langle \bot_{\mathbf{a},\mathbf{a}}; f, x\rangle \mathrm{App}\ \mathcal{R}_T = \langle \bot_{\mathbf{a}',\mathbf{a}'}; f', x'\rangle; \mathrm{App}$.*

Now define $\mathcal{R}_T$ on $\mathcal{C}(\mathbf{1}, [\![T]\!]) \times \mathcal{C}'(\mathbf{1}, [\![T]\!]_{\mathcal{C}'})$ by $e\ \mathcal{R}_T e'$ if and only if $\mathbf{1}_\mathbf{a}; e\ \mathcal{R}_T \mathbf{1}_{\mathbf{a}'}; e'$

Lemma 5.4.2 *For every term $t : T$ of $\Lambda(\Omega)$, $[\![t]\!]_\mathcal{C}\ \mathcal{R}_T [\![t]\!]_{\mathcal{C}'}$, and for every two pairs of related elements, $e\ \mathcal{R}_T f$, $e'\ \mathcal{R} f'$*
$e \equiv_{[\![T]\!]} e'$ and $f \equiv_{[\![T]\!]} f'$ if and only if $e\ \mathcal{R} f'$.

Proof: is by induction on type-structure, from which the first part is direct by soundness.

For the second part, applying the $\Lambda(\Omega)$ Context Lemma, $e, e' : [\![S \Rightarrow T]\!]_{\mathcal{C}}$ and $f, f' : [\![S \Rightarrow T]\!]_{\mathcal{C}'}$ are observationally equivalent if and only if for any $x : \mathbf{a} \to [\![S]\!]_{\mathcal{C}}$, $y : \mathbf{a} \to [\![S]\!]_{\mathcal{C}'}$ such that $x \ \mathcal{R}_S y$,

$(\langle e, x\rangle; \mathrm{App} \equiv (\langle e', x\rangle; \mathrm{App}$ and $(\langle f, y\rangle; \mathrm{App} \equiv (\langle f', y\rangle; \mathrm{App}$.

Hence by induction hypothesis, f, f' and e, e' are observationally equivalent if and only if for all $x \ \mathcal{R}_T y$,

$\langle e, x\rangle; \mathrm{App}\ \mathcal{R}_T (\langle f', y\rangle; \mathrm{App}$, and similarly

$\langle \bot; e, x\rangle; \mathrm{App}\ \mathcal{R}_T \langle \bot; f', y\rangle; \mathrm{App}$ if and only if $e \ \mathcal{R}_{S \Rightarrow T} f'$ as required. □

So the $\mathcal{R}$-relation between the games model and itself is observational equivalence. To show full abstraction, it is sufficient to show that this relation between the games and sequential algorithms models is a surjective function (which must be a functor, as it preserves meaning), by defining it as a relation on strategies. This also neatly clarifies the relationship between dialogue games and sequential algorithms.

5.4.1 The collapse of innocent strategies

Preparatory to describing the relationship between innocent strategies and sequential algorithms, it is useful to re-examine the notion of observational equivalence. Recall the notion of an 'escaping strategy' which contains odd-length (maximal) sequences, corresponding to a jump into a one-move game adjoined to the context. Two such strategies on the one-move game can be distinguished; the empty strategy $\{\varepsilon\}$, and one that 'recognises' Opponent's move, $\{\varepsilon, o\}$. This allows the following refinement of the characterization of observational equivalence via the following version of the Context Lemma.

Lemma 5.4.3 *Given innocent (non-escaping) strategies $\sigma, \tau : A$,*
$\sigma \lesssim_A \tau$ if and only if $\forall \rho : \overline{B_i}$, (escaping)
$\rho; \sigma = \{\varepsilon, o\}$ implies $\rho; \tau = \{\varepsilon, o\}$.

Proof: is direct by application of Proposition 5.1.1. □

It is also important to restrict attention to observers which are definable; innocent *Opponent* strategies:

Definition 5.4.4 *An innocent play according to an (innocent) strategy $\sigma : A$ is a finite play $s \in \sigma$ such that Opponent also plays innocently: i.e.*

$\forall s', t' \sqsubseteq_{even} s, \quad \llcorner s' \lrcorner = \llcorner t' \lrcorner \quad \wedge s'a \sqsubseteq \quad s \implies t'a \sqsubseteq s.$
The innocent plays in σ are denoted $I(\sigma)$.

Innocent Opponent strategies are represented as Player strategies using the following observation.

Proposition 5.4.5 *Let osb be an even-length justified sequence over $\overline{A} \Rightarrow o$. Then sb is a justified sequence in $\overline{A}$ and the Opponent view of osb is the 'lifting' of the Player view of sb in the sense that*
if $b \in M_{A_i}$, then $\llcorner osb \lrcorner = o\ulcorner sb \upharpoonright A_i \urcorner$.

PROOF: is by a simple induction on the definition of view. □

Definition 5.4.6 *For any non-empty innocent play $s \in \overline{B} \Rightarrow o$, the Opponent strategy manifested in s is represented as a player strategy $\tau : \overline{B}$, by erasing the initial moves from all of the O-views of s.*
For any player strategy $\sigma : A$, the interaction of σ with the Opponent strategy τ, written as an uncovering $\sigma \| \langle \tau_1, \tau_2, \ldots \tau_n \rangle$ (see Definition 3.2.7) is the maximal sequence $s \in \sigma$ such that $s \upharpoonright B_i \in \tau_i$.

Opponent strategies are ordered (by inclusion, and the intrinsic preorder) by considering them as Player strategies.

Lemma 5.4.7 *Given (non-escaping) $\sigma, \tau : A$*
$\sigma \lesssim_A \tau$ if and only if $\forall s \in I(\sigma). \exists t \in I(\tau) : \mathrm{O-strategy}(s) = \mathrm{O-strategy}(t)$.

PROOF: Suppose $\sigma \not\lesssim_A \tau$, then by the new presentation of the Context Lemma (Lemma 5.4.3), there is an (escaping) Opponent strategy $\rho : \overline{B}$ such that $\rho; \sigma = \{\varepsilon, o\}$ and $\rho; \tau = \{\varepsilon\}$.
Let $s = \rho \| \sigma$, then $\mathrm{O-strategy}(s); \sigma = \{\varepsilon, o\}$, and $\mathrm{O-strategy}(s); \tau = \{\varepsilon\}$. If t is an even-length innocent play such that $\mathrm{O-strategy}(s) = \mathrm{O-strategy}(t)$, then $\mathrm{O-strategy}(t); \tau = \{\varepsilon\}$, and so $\mathrm{O-strategy}(t) \| \tau \neq t$, so $t \notin \tau$.
Hence $\mathrm{O-strategy}(s) \neq \mathrm{O-strategy}(t)$ for all $t \in I(\tau)$.
Conversely, suppose $\sigma \lesssim_A \tau$, then for all $s \in I(\sigma)$, $\mathrm{O-strategy}(s); \sigma = \{\varepsilon, o\}$, and hence $\mathrm{O-strategy}(s); \tau = \{\varepsilon, o\}$. Hence $\mathrm{O-strategy}(s) \| \tau$ is an even length sequence in τ.
Moreover, it can be proved by induction on the number of O-views in s, that $s \in I(\sigma)$ implies $\mathrm{O-strategy}(\mathrm{O-strategy}(s) \| \tau) = \mathrm{O-strategy}(s)$ because: $\llcorner \mathrm{O-strategy}(s) \| \tau \lrcorner = \llcorner s \lrcorner$, as this is (the lifting of) the unique odd-length view (escape) in $\mathrm{O-strategy}(s)$. Let s' be the greatest even innocent prefix of s, then by inductive hypothesis, $\mathrm{O-strategy}(s') = \mathrm{O-strategy}(\mathrm{O-strategy}(s') \| \tau)$, and hence $\mathrm{O-strategy}(s) = \mathrm{O-strategy}(\mathrm{O-strategy}(s) \| \tau)$ as required. □

Corollary 5.4.8 *Given (non-escaping) innocent strategies* or *sequential algorithms, $\sigma, \tau : A = \overline{B} \Rightarrow o$,*
$\sigma \lesssim_A \tau$ *if and only if* $\forall s \in I(\sigma).\exists t \in I(\tau) : \mathrm{O}-\mathrm{strategy}(s) \equiv_{\overline{B}} \mathrm{O}-\mathrm{strategy}(t)$

The symmetry between Player and Opponent in sequential algorithm paths (both subject to the same non-repetitivity condition) means that the translation is well-defined only for innocent plays. The following lemma establishes that definition by induction on subsequences of innocent sequences is possible.

Lemma 5.4.9 (Projection condition for innocent plays) *If s is a (finite) innocent play and a an occurrence of a move in s, then $s{\upharpoonright}a$ is an innocent play.*

PROOF: shows that in plays over arenas of the form $\overline{A} \Rightarrow o$, every thread of $!A$ is an innocent play. (Recall from Chapter 3 (Proposition 3.3.8) that every game is in fact specified by an arena of this form.) Proof is by induction on the number of threads of $!A$ occurring in s.
Suppose $s \in \sigma$ (an innocent player strategy). By the 'bang lemma', (Proposition 3.6.13), $\mathrm{O}-\mathrm{strategy}(s) = !\tau$ for some innocent $\tau' : \overline{A}$.
By definability axiom 'linearization of head occurrence' there is a strict, innocent strategy $\sigma' : A \multimap (!A \multimap o)$, obtained by relabelling the first thread of $!A$ opened by σ as a play in the new linear premise, and similarly an innocent strategy $\sigma'' : !A \multimap (A \multimap o)$ obtained by switching premises.
Consider the uncovering of $!\tau; \sigma' : A \multimap o$ (which is innocent) played off against τ, — this is equal to the first thread opened in $!A$,
i.e. $(!\tau; \sigma')\|\tau = (\sigma\|\tau){\upharpoonright}a$, which is therefore innocent.
The remainder of the play, $s/(s \upharpoonright a)$ is innocent, as it is the uncovering of the composition of $!\tau$ with $\tau; \sigma' : !A \multimap o$, which is an innocent strategy. It also contains fewer threads of $!A$, hence by inductive hypothesis, each thread is an innocent play. □

Definition 5.4.10 (Copy-removing translation) *The translation is defined by a double induction, first on type-structure, and secondly on sequence length of (prefixes of) innocent plays. For any type $T = S_1 \Rightarrow (S_2 \Rightarrow \ldots (S_n \Rightarrow \iota) \ldots)$, assuming the translation is defined for each S_i, define the translation of the justified sequences of $[\![T]\!]$ as follows:*

- $\varepsilon^T = \varepsilon$
- $o^T = o$ *(where o is the initial move)*

- *For occurrences a of non-initial moves, there is some unique move occurrence b in s which is initial in some $[\![S_i]\!]$ hereditarily justifying (or equal to) a, so let*
$(sa)^T = s^T(sa\upharpoonright b)^{S_i}$ *if* $(sa\upharpoonright b)^{S_i}$ *does not occur in* s^T*, and*
$(sa)^T = s^T$ *otherwise.*

Definition 5.4.11 *This induces a map on sets of innocent plays, and hence on innocent (partial) strategies (restricted to their innocent plays):*
for $\sigma : [\![T]\!]$*, define* $\sigma^T = \{s^T \mid s \in I(\sigma)\}$.

It is clear from the definition that the translation of each justified sequence is a non-repetitive sequence starting with a move in o, followed by paths from the S_i. The difficult property to prove is that play alternates between Player and Opponent, — and this is clearly not the case if either fails to play innocently. Intuitively, one can argue roughly as follows:
if a move has occurred previously in the translated path, then the view of the original play has occurred previously; and consequently the subsequent move is a repetition (because of innocence) so its translation is removed from the sequential algorithms path. This is, however, patently an oversimplification of the proof, as there may be whole sequences of moves repeated due to innocence.

Definition 5.4.12 *For a sequential data structures path s over $\overline{A} \Rightarrow o$, define the 'Opponent strategy' manifested in s^T to be the set of even-length path moves in A, together with the most recent path-move (which may be odd, representing an escape).*
i.e. $\mathrm{O-strategy}(s) = \{p \in \mathrm{Occ(s)}\upharpoonright\overline{\mathrm{B}}) \mid \mathrm{Even(p)}\} \cup \{\mathrm{s}\upharpoonright\mathrm{b_i^{S_i}}\}$.

Proposition 5.4.13 (The translation is well-defined) *The following hold for translated sequences and strategies.*

i *The Opponent strategy manifested in s^T is a well-defined sequential algorithm such that* $\mathrm{O-strategy}(s^T) = \mathrm{O-strategy}(s)^{\overline{B}}$.

ii *If sa^T (odd-length) extends s^T, then s is innocent.*

iii *If s is (even/odd)-length then s^T is an (even/odd)-length sequential-algorithms path.*

iv *For any innocent strategies $\sigma, \tau : A$,*
$\sigma \lesssim_T \tau$ *if and only if* $\sigma^T \subseteq \tau^T$.

v *For any innocent strategy $\sigma : A$, σ^T is a sequential algorithm.*

Proof: is by joint induction over type-depth (first) and sequence length (second).

i Suppose $p \in \mathrm{O}-\text{strategy}(s^T)$, then $(p = t\lceil b_i)^{S_i}$, for some $t \sqsubseteq s$, and $t\lceil b_i \in \mathrm{O}-\text{strategy}(s)$ by definition (note that $\mathrm{O}-\text{strategy}$ is a well-defined sequential algorithm by (v)).

ii If sa^T extends s^T, then $(sa\lceil b_i)^{S_i}$ is an even-length path in S_i distinct from all previous paths, and the Opponent sequential algorithm manifested in sa^T is not equal to the Opponent sequential algorithm manifested in s^T, and by (iv), the Opponent strategy manifested in sa is distinct from the Opponent strategy manifested in s. Hence $\ulcorner s\lceil b_i \urcorner$ must be distinct from every previous Opponent view, and so s is innocent ($\llcorner s \lrcorner$ is new) as required.

iii If s is an (odd/even-length) innocent play then s^T is an (odd/even-length play) As noted above, s^T is non-repetitive, and consists of well-defined path-moves by induction hypothesis.

Suppose s is odd-length and innocent, then every thread of $!\overline{B}$ in s is innocent and even-length by the innocence Lemma 5.4.9 above. Hence there are equal numbers of (non-initial) Player and Opponent moves in s^T, as each thread translates to an even-length path move. Since every Opponent path-move extends a Player path-move, the most recent path-move has been made by Opponent. By (ii), the play preceding this most recent extension is innocent and even-length, hence by induction hypothesis, its translation is an even-length path.

Suppose s is even-length and innocent. The threads of $!\overline{B}$ in s are all innocent, and even-length, except the current one, which innocent and odd-length, by Lemma 5.4.9. So the translations of the other threads are all even-length path-moves, the translation of the current thread is an odd-length path-move.

The most recent path-move in s^T is Player's. (Suppose the most recent move is Opponent's; then the play preceding this extension is innocent and even length, giving a contradiction). Hence there are equal numbers of (non-initial) Player and Opponent moves in the initial segment of s^T. The penultimate move must be an Opponent path-move (extending the translation of an innocent even-length play by ii) which translates to an even-length path by induction hypothesis.

iv $\sigma \lesssim_{[\![T]\!]} \tau$ if and only if $\sigma^T \subseteq \tau^T$.

This is shown using Lemma 5.4.7 and Corollary 5.4.8, and (iv) as induction hypothesis.

Suppose $\sigma^T \subseteq \tau^B$, then for all $s \in \sigma$, there exists $t \in \tau$ such that $s^T = t^T$, and hence by (iv), $\mathrm{O}-\mathrm{strategy}(s) \equiv \mathrm{O}-\mathrm{strategy}(t)$, so by Lemma 5.4.7, $\sigma \sqsubseteq^{OBS}_{\Lambda(\Omega)} \tau$.

Suppose $\sigma \lesssim_{[\![T]\!]} \tau$; then by the observational equivalence Lemma 5.4.7, for every $s \in \sigma$, there exists $t \in \tau$ such that $\mathrm{O}-\mathrm{strategy}(s) \equiv \mathrm{O}-\mathrm{strategy}(t)$, and hence $\mathrm{O}-\mathrm{strategy}(s^T) = \mathrm{O}-\mathrm{strategy}(t^T)$. So $\sigma^T \subseteq \tau^T$ as required.

v To prove that the translation of an innocent strategy is a sequential algorithm, it remains to show that the translations of $sa, tb \in \sigma$ are evenly branching. The induction step is to show that if $s^T = t^T$, then $sa^T = tb^T$.

$s^T = t^T$ implies $\mathrm{O}-\mathrm{strategy}(s) \equiv \mathrm{O}-\mathrm{strategy}(t)$ by (iv), and this implies $\mathrm{P}-\mathrm{strategy}(sa) = \mathrm{P}-\mathrm{strategy}(tb)$.

(Suppose $\mathrm{P}-\mathrm{strategy}(s) \neq \mathrm{P}-\mathrm{strategy}(t)$, then w.l.og. there is some Player-view in s not occurring in t, and this can be used to escape from s and distinguish $\mathrm{O}-\mathrm{strategy}(s)$ from $\mathrm{O}-\mathrm{strategy}(t)$.)

$\square$

Example 5.4.14 *Recall Example 5.2.8, comparing a play in the innocent strategy denoted by $\lambda f.f\ (f\ 0)$ to paths in the sequential algorithms denotation. As expected, the former translates to one of the latter, depending on the Opponent strategy manifested in the innocent play. This can be seen by observing that the effect of the translation can be 'simplified in this case to the following map on innocent plays in* $[\![(\mathtt{nat} \Rightarrow \mathtt{nat}) \Rightarrow \mathtt{nat}]\!]$:

$(sa)^\star = s^\star$, *if there are some pevious moves b, c such that b hereditarily justifies a, and $sa\restriction b = sa\restriction c$,*

$(sa)^\star = s^\star a$, *otherwise.*

Hence any repetition of the initial question 'What is the value of f?', and/or the succeeding Opponent move (which it justifies); 'What is the value of the argument to f?', will be removed. In the case that $f\ 0 = 0$, the repeated supplying of zero as the argument to f is removed, as is Opponent's next move, which, by innnocence, must be a repetition of the 'answer' zero as well.

5.4.2 Mapping sequential algorithms to innocent strategies

The relation $\mathcal{R}$ can now be shown to be a surjective function, by defining an innocent strategy from a sequential algorithm, which is in its pre-image with respect to $\mathcal{R}$. Hence this produces a unique representative for each equivalence class of the intrinsic preorder on $\mathcal{G}$.

It is presented in the form of a translation from sequential data structure paths to justified sequences which maps each sequential algorithm to a strategy which 'collapses' to that algorithm by the translation described above. The result of this operation is not unique, nor is it functorial, but it does provide a unique representative, a 'normal form', for each equivalence classes of strategies, which can be obtained by 'collapsing' to a sequential algorithm, and then 'fluffing up' to an innocent strategy.

Definition 5.4.15 (Copy saturating translation) *This is defined as a translation from sequential algorithm paths to justified, well-opened sequences (which are innocent plays), by a double induction over type structure, and over sequence length, following a similar pattern to the copy-removing translation.*
The translation of a path t over A is written t_T. Assuming $[\![T]\!] \cong (\Pi_{i \leq n}[\![S_i]\!]) \Rightarrow o$ for some $S_1, S_2, \ldots, S_n$.

- $\varepsilon_T = \varepsilon$
- $o_T = o$ *(where o is the unique initial move).*
- *If s is non-empty, and of even length, and hence of the form $t(pa)$, for some (odd-length) path pa in some S_i, then $(t(pa))_T = t_T \cdot (pa)_{S_i}$.*
  The justifier of the first move in $(pa)_{S_i}$ is the first move in t, an as pa_{S_i}

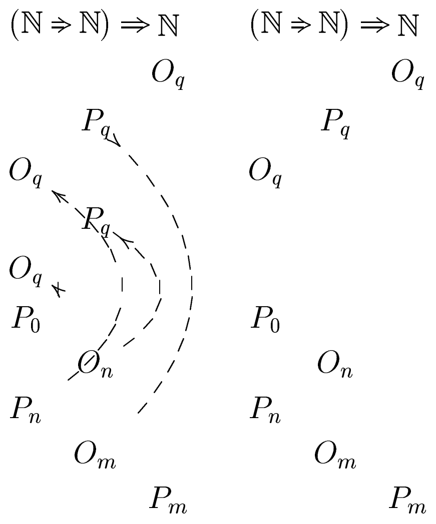

Figure 5.5: Copy-removing translation applied to $\lambda f.f\ f\ 0$ (Opponent represents a function s.t $f\ 0 \neq 0$).

is well-opened, all of its other moves are justified by occurrences of moves already in pa_{S_i}.

- *if* s *is of odd length (and longer than a single move), it is of the form* $t(pa)(pab)$*, where* pa *is a path in some* S_i*, and* pab *extends it. Then* $(t(pa)(pab))_T = t_T \cdot (pab)_{S_i}$*, with the justification pointers given as above.*

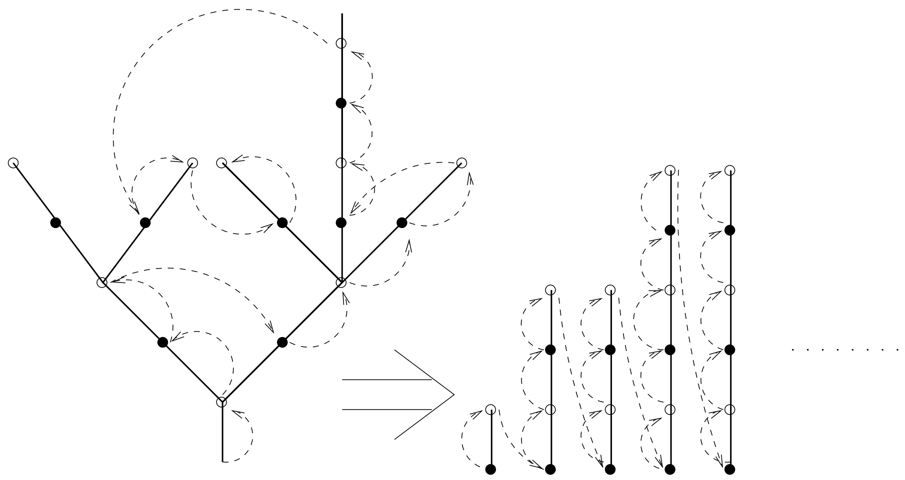

Figure 5.6: Translation from sequential algorithm to innocent strategy

Proposition 5.4.16 *The following two facts can be proved directly from the definition of the translation:*

- $s_T \in [\![T]\!]$*: the translation of an (even/odd -length) path in* $[\![T]\!]_{\mathcal{S}}$ *is an alternating (even/odd - length) sequence in* $[\![T]\!]_{\mathcal{G}}$ *satisfying visibility.*

- $(s_T)^T = s$*: removing the copies from a translated path returns the original path.*

Proof: is by induction following the pattern set by the definition of the translation:

- Suppose $s = t(pa)(pab)$, then by hypothesis, t_T is an odd-length, alternating sequence satisfying visibility, and pab_{S_i} is an even-length alternating sequence satisfying visibility starting with an initial player move in S_i (justified by the initial, visible move). Hence $t_T \cdot pab_{S_i}$ is odd-length, alternating, and satisfies visibility. (And similarly for $t_T \cdot pa_{S_i}$).

- proof is by induction that for an innocent play r and sequential algorithms path s, if $r \sqsubseteq s_T$, then $r^T \sqsubseteq s$, *and* if $s_T \sqsubseteq r$, then $s \sqsubseteq r^T$. Hence $s = (s_T)^T$. Suppose $r \sqsubseteq ((t(pa)(pab))_T)$, then either $r \sqsubseteq t_T$, so $r^T \sqsubseteq t$, or $r = t_T \cdot u$, where $u \sqsubseteq pab_{S_i}$. Then by hypothesis, $t \sqsubseteq r^T$, and for every prefix of $u' \sqsubseteq u$, $u'^T \sqsubseteq pab_{S_i}$ hence r^T is the extension of t by pa or pab (or the empty sequence).
  Suppose $((t(pa)(pab))_T) \sqsubseteq rc$, then either $t(pa)(pab))_T) \sqsubseteq r$ or $rc\restriction d = pab_{S_i}$, where d is the initial move of S_i hereditarily justififying c, and hence $(rc\restriction d)^{S_i} = pab$ by induction hypothesis, and $t(pa) = r^T$, hence $rc^T = t(pa)(pab)$ as required.

$\square$

Definition 5.4.17 *For a sequential algorithm $\chi : [\![T]\!]$, define*

$$\chi_T = \{\ulcorner t \urcorner \mid t_T \in \chi\}.$$

Example 5.4.18 *Continuing the theme of Examples 5.2.8 and 5.4.14, consider the translation of the sequential algorithm denoted by $\lambda f.f\ (f\ 0)$. The translation on paths over $[\![(\texttt{nat} \Rightarrow \texttt{nat}) \Rightarrow \texttt{nat}]\!]_S$ can be simplified to the following: $(s(pa)(pab))^{(\bigstar} = s^\bigstar pab$, where pa, pab are path-moves in $[\![\texttt{nat} \Rightarrow \texttt{nat}]\!]$, and $sa^\bigstar = s^\bigstar a$, otherwise.*

Applying this to the sequential algorithm denoted by of $\lambda f.f\ f\ 0$ yields an innocent strategy which is not the denotation of $\lambda f.f\ f\ 0$. (See Figure 5.4.18.) This strategy first tests f for strictness, returning its constant value if it is not strict. Otherwise it tests $f0$, and returns 0 if this is a fixedpoint of f. Otherwise it applies f to the value obtained from $f0$ and returns that. This strategy corresponds to the following term of PCF with `catch` and infinitary `case` statements:

$$\texttt{case}\ (\texttt{catch}f)\ |_0\ (\texttt{case}\ (f\ 0)\ |_0\ 0\ |_{i \neq 0}(f\ i))\ |_{j \neq 0}\ j - 1.$$

It is also possible to verify that applying the copy-removing translation to this strategy yields the sequential algorithm corresponding to both $\lambda f.f\ f\ 0$ and the above term.

Proposition 5.4.19 *For any sequential algorithm χ, χ_T is an innocent function over A.*

Proof: Arguing informally, suppose $t(pa)pab$ is a sequential algorithms path, so $(t(pa)(pab))_T = t_T \cdot pab_{S_i}$. Under an inductive assumption, t_T and pab_{S_i} are both innocent paths, so it is easy to see that Player plays innocently, as he can always 'see' the last move in t_T. Opponent plays innocently, because he plays the same in p_{S_i} as when it appeared before, whilst a must be new, by the 'no repetition' condition, so Opponent has a new view at this point, and safely play a new move of his own.

Formally, the proof proceeds by verifying inductively the following hypotheses.

(i) If $s \sqsubseteq u$ then $s_T \sqsubseteq u_T$.

(ii) If $u_T(odd) \sqsubseteq r \sqsubseteq s_T(odd)$, then $\ulcorner u_T \urcorner \sqsubseteq \ulcorner r \urcorner \sqsubseteq \ulcorner s_T \urcorner$.

(iii) If $u \not\sqsubseteq s$ then $\ulcorner u_T \urcorner \not\sqsubseteq \ulcorner s_T \urcorner$ (and $\llcorner u_T \lrcorner \neq \llcorner s_T \lrcorner$. The content of (ii,iii) is that Player has access via the view to the entire history of the (translation of) the sequential algorithms play.

(iv) Player plays innocently in s_T, i.e. if $ra, r' \sqsubseteq s_T$ such that $\ulcorner r \urcorner = \ulcorner r' \urcorner$ then $r'b \sqsubseteq s_T$.

(v) If s and u branch at an odd (Opponent) move, then s_T, u_T are coherent with respect to innocence; i.e. if $s_T \sqcap u_T$ is odd and $ra \sqsubseteq s_T$ and $r'b \sqsubseteq u_T$ such that $\ulcorner r \urcorner = \ulcorner r' \urcorner$ then $a = b$.

(vi) Opponent plays innocently in s_T, i.e. if $ra, r' \sqsubseteq s_T$ such that $\llcorner r \lrcorner = \llcorner r' \lrcorner$ then $r'b \sqsubseteq s_T$.

$$
\begin{array}{ccccccc}
(\mathbb{N} & \Longrightarrow \mathbb{N}) & \Longrightarrow \mathbb{N} & \qquad & (\mathbb{N} & \Longrightarrow \mathbb{N}) & \Longrightarrow \mathbb{N} \\
 & & O_q & & & & O_q \\
 & P_q & & & & P_q & \\
O_q & & & & O_q & & \\
 & & & & & P_q & \\
 & & & & O_q & & \\
P_0 & & & & P_0 & & \\
 & O_{n+1} & & & & O_{n+1} & \\
 & & & & & P_q & \\
 & & & & O_q & & \\
P_{n+1} & & & & P_{n+1} & & \\
 & O_m & & & & O_m & \\
 & & P_m & & & & P_m
\end{array}
$$

Figure 5.7: Copy saturating translation of $\lambda f.f\ f\ 0$

The base cases are are straightforward. On the assumption that $(i)-(vi)$ hold for t_T and for $(pab)_{S_i}$, it is shown below that they hold for $(t(pa)(pab))_T = t_T \cdot (pa)_{S_i}$. The proofs that they hold for $(t(pa))_T$ are in all cases similar.

(i) $s \sqsubseteq u \implies s_T \sqsubseteq u_T$ If $r \sqsubseteq t(pa)(pab)$, then either $r \sqsubseteq t$ (in which case $r_T \sqsubseteq t_T \sqsubseteq t_T \cdot (pa)_{S_i}$ by induction hypothesis), or $r = t(pa)(pab)$, so $r_T = (t(pa)(pab))_T$, or $r = t(pa)$, so $r_T = t_T \cdot (pa)_{S_i}$, $pa_{S_i} \sqsubseteq pab_{S_i}$ by induction hypothesis, therefore $r_T \sqsubseteq (t(pa)(pab))_T$ as required.

(ii) $u_T(odd) \sqsubseteq r(odd) \sqsubseteq s_T \implies \ulcorner u_T \urcorner \sqsubseteq \ulcorner r \urcorner \sqsubseteq \ulcorner s_T \urcorner$ Suppose $r \sqsubseteq_{odd} t_T \cdot (pab)_{S_i} = s_T$. If $r \sqsubseteq t$ then the induction hypothesis applies to give the result directly. So suppose $r = t_T \cdot v$, for some $v \sqsubseteq pab_{S_i}$. Then the initial (Player) move in S_i justifying all of the moves in v must occur in $\ulcorner r \urcorner$, so $\ulcorner r \urcorner = \ulcorner t_T \urcorner \cdot \llcorner v \lrcorner$, and the induction hypothesis applies giving $\ulcorner u_T \urcorner \sqsubseteq \ulcorner t_T \urcorner \sqsubseteq \ulcorner r \urcorner$, and $\llcorner v \lrcorner \sqsubseteq \llcorner pab_{S_i} \lrcorner$, so $\ulcorner r \urcorner \sqsubseteq \ulcorner s_T \urcorner$ as required.
Note in particular that $\ulcorner (t(pab))_T \urcorner = \ulcorner t \urcorner \cdot \llcorner pab_{S_i} \lrcorner$.

(iii) If $s(odd) \not\sqsubseteq s'(odd)$ then $\ulcorner s_T \urcorner \not\sqsubseteq \ulcorner s'_T \urcorner$
(and similarly $s'(even) \not\sqsubseteq s'(even) \implies \llcorner s_T \lrcorner \not\sqsubseteq \llcorner s'_T \lrcorner$)
Suppose $t(pa)(pab) \not\sqsubseteq s'$, then either $t \not\sqsubseteq s'$ and by hypothesis $\ulcorner t_T \urcorner \not\sqsubseteq \ulcorner \urcorner s'_T$, and by (ii), $\ulcorner t_T \urcorner \subseteq \ulcorner s_T \urcorner \not\sqsubseteq s'_T$ as required.
or $s' = tqc(qcd)$ for some qcd such that $pab \not\sqsubseteq qcd$
As remarked above, $\ulcorner (t(pa)(pab))_T \urcorner = \ulcorner t_T \urcorner \llcorner pab_{S_i} \lrcorner$, and
$\ulcorner (t(qc)(qcd))_T \urcorner = \ulcorner t_T \urcorner \llcorner qcd_{B_j} \lrcorner$.
By inductive hypothesis, $\llcorner pab_{S_i} \lrcorner \neq \llcorner qcd_{B_j} \lrcorner$, and so
$\ulcorner (t(pa)(pab))_T \urcorner \not\sqsubseteq \ulcorner (t(qc)(qcd))_T \urcorner$, as required.

(iv) By the inductive hypothesis (vi), that t_T and $(pab)_{S_i}$ are innocent, Player plays innocently in s_T:
Suppose $rc \sqsubseteq_{even} r' \sqsubseteq_{odd} t_T \cdot (pab)_{S_i}$, and $\ulcorner r \urcorner = \ulcorner r' \urcorner$,
If $r' \sqsubseteq_{odd} t_T$, then $r'c \sqsubseteq_{even} t_T \sqsubseteq t_T \cdot (pab)_{S_i}$ by innocence of t_T.
If $rc \sqsubseteq_{even} t_T$, but $r' \not\sqsubseteq_{odd} t_T$, then $r' = t_T q$, for some $q \sqsubseteq_{even} (pab)_{S_i}$ and so $\ulcorner r' \urcorner = \ulcorner t_T \urcorner \llcorner q \lrcorner$.
But since t_T is innocent $\ulcorner t_T \urcorner$ is not the view of any prefix of t_T, and so neither is $\ulcorner t_T \urcorner \llcorner q \lrcorner$, so $\ulcorner r \urcorner \neq \ulcorner r' \urcorner$, a contradiction.
If $rc \not\sqsubseteq_{even} t_T$, then $r = t_T \cdot u$, for some $u \subseteq_{even} (pab)_{S_i}$ and $r' = t_T \cdot u'$, for some $u' \subseteq_{even} (pab)_{S_i}$. $\ulcorner r \urcorner = \ulcorner t_T \urcorner \llcorner u \lrcorner$, and $\ulcorner r' \urcorner = \ulcorner t_T \urcorner \llcorner u' \lrcorner$. So if $\ulcorner r \urcorner = \ulcorner r' \urcorner$, then $\llcorner u \lrcorner = \llcorner u' \lrcorner$, and as $(pab)_{S_i}$ is innocent by hypothesis, and $um \sqsubseteq_{odd} (pab)_{S_i}$, $r'c \sqsubseteq_{even} t_T \cdot (pab)_{S_i}$, as required.

(v)Evenly branching paths map to coherent plays If $r \sqsubseteq_{odd} s_T$, $r' \sqsubseteq_{odd} s'_T$, and $r' \not\sqsubseteq s_T$, where $s' \sqcap s$ is even, then $\ulcorner r \urcorner \neq \ulcorner r' \urcorner$, — so there is no possibility of a violation of innocence.

Let $u = s' \sqcap s$ so there is an Opponent move c such that $uc_T \sqsubseteq r$ but $uc_T \not\sqsubseteq r'$ Then by (ii,iii) $\ulcorner uc_T \urcorner \sqsubseteq \ulcorner s_T \urcorner$, but $\ulcorner uc_T \urcorner \not\sqsubseteq r'$ so $\ulcorner r' \urcorner \not\sqsubseteq \ulcorner s_T \urcorner$, and $\ulcorner r \urcorner \sqsubseteq s_T$, hence $\ulcorner r \urcorner \neq \ulcorner r' \urcorner$ as required.

(vi) Opponent plays innocently in $s_T = t_T(pab)_{S_i}$ By inductive hypotheses (iv) and (vi), t_T and pab_{S_i} are innocent plays, so the case to consider is $r, r' \sqsubseteq^{even} s_T$ such that $r \sqsubseteq t_T$, and $r' \not\sqsubseteq t_T$, but $\llcorner r \lrcorner = \llcorner r' \lrcorner$.

So $r' = t_T \cdot u'$, for some $u' \sqsubseteq_{even} pab_{S_i}$

$\llcorner t_T \cdot u' \lrcorner = m \ulcorner u' \urcorner$, thus $r = v_T \cdot u$, for some paths v in A, and $u \sqsubseteq (qcd)_{S_i}$, where qcd in S_i is such that $v(qc)(qcd) \sqsubseteq t$, and $\ulcorner u \urcorner = \ulcorner u' \urcorner$.

If $qcd \sqsubseteq pab$ then $u \sqsubseteq (pab)_{S_i}$, and by the inductive hypothesis (v) Player (i.e. whoever makes even moves) plays innocently in S_i, thus if $um \sqsubseteq pab_{S_i}$ then $u'm \sqsubseteq pab_{S_i}$, and $rm \sqsubseteq (t(pa)(pab))_T$ implies $r'm \sqsubseteq (t(pa)(pab))_T$ as required.

If $qcd \not\sqsubseteq pab$, then they branch at an even move, as plays in $!_S S_i$ are *non-repetitive* (and this is the only point at this fact is used). Thus by hypothesis (v), they are coherent with respect to innocence, thus $\ulcorner u \urcorner = \ulcorner u' \urcorner$ and $um \sqsubseteq qcd$ implies $u'm \sqsubseteq pab$, and so $rm \sqsubseteq (t(pa)(pab))_T$ implies $r'm \sqsubseteq (t(pa)(pab))_T$ as required.

Finally, note that item (v) above entails that χ_T is an innocent function as required. $\square$

Proposition 5.4.20 *For every sequential algorithm* $\chi : [\![T]\!]_{\mathcal{S}}$, $\chi_T \; \mathcal{R}_T \chi$ *, and for each innocent strategy* $\sigma : [\![T]\!]_{\mathcal{G}}$, $\sigma \; \mathcal{R}_T \sigma^T$.

PROOF: is by induction on type-structure: suppose $T = \overline{S} \Rightarrow \iota$.
Given an Opponent sequential algorithm $\psi : [\![\overline{S}]\!]_{\mathcal{S}}$,
$(\chi \| \psi)_{\overline{S} \Rightarrow \iota} = \chi_{\overline{S} \Rightarrow \iota} \| \psi_{\overline{S}}$, hence $\psi; \chi \downarrow$ if and only if $\psi_T; \chi_T \downarrow$
Since every innocent strategy is observationally equivalent to some ψ_T, by hypothesis, it follows that $\chi_T \; \mathcal{R} \chi$.
Secondly, $((\sigma^T)_T)^T = \sigma^T$, hence $\sigma \equiv \sigma^T_T$, and $(\sigma^T)_T \; \mathcal{R}_T \sigma^T$, hence $\sigma \; \mathcal{R}_T \sigma^T$ as required. $\square$

Corollary 5.4.21 *The initial functor from the games to the sequential algorithms model is surjective.*

Corollary 5.4.22 *The semantics of $\Lambda(\Omega)$ in the CCC of Sequential Algorithms generated from freely generated from o is fully abstract, hence the models of control for μPCF given by $(\mathcal{S}, o)$ are fully abstract.*

Corollary 5.4.23 *The extensional collapse of the standard call-by-name* PCF *model in the category of unbracketed games is isomorphic to the strongly stable model.*

PROOF: is as a corollary of Ehrhard's proof [24] that the extensional collapse of the sequential algorithms model of PCF is isomorphic to the strongly stable model. The extensional collapse can be found by identifying the extensional elements of the semantics, and an extensional equivalence $\equiv$ on them as follows.
All objects at ground type are extensional, and equivalence is the identity relation.
$f : A \Rightarrow B$ is extensional if for all extensional $a, a' : A$, such that $a \equiv_A a'$, $f a \equiv_B f a'$.
$f \equiv_{A \Rightarrow B} f'$ if for all extensional $a : A$, $f a \equiv_B f' a$.
Considering the extensional collapse of the PCF type-structure of games it is clear that $\sigma \equiv_T \tau$ implies $\sigma \cong \tau$. Thus, by a trivial induction, σ is extensional if and only if σ^T is, and σ is extensionally equivalent to τ if and only if σ^T is extensionally equivalent to τ^T. Hence composing the functor collapsing innocent strategies into sequential algorithms (restricted to extensional objects) with Ehrhard's functor projecting (extensional) sequential algorithms to strongly stable functions, yields a projection from the extensional innocent strategies onto sequential algorithms, and this is injective on extensional equivalence classes (see Figure 5.1). $\square$

This gives a partial answer to the question: — what are the extensional functions which are computed in models of control? but leaves another: — which are the extensional morphisms which compute them? Longley [55] has shown that there is an extensional functional **H**, programmable with *either* control (call-with-current-continuation) or state (ML-style references), such that the strongly stable model of PCF+**H** is fully abstract. Although unbracketed games are by no means a unique way to 'realize' these *realizably sequential functions*, the richness of their structure, as expressed by the intensional hierarchy, together with the fact that they allow a connection to be made with the fully abstract model of PCF on the one hand, and the reduction to the simply-typed λ-calculus via cps translation on the other, indicates that they may be an informative place to study it.

The translation to sequential algorithms has confirmed the claim that each innocent strategy is intensionally equivalent to one which does not duplicate queries.

But in fact it gives a stronger characterization of the intrinsic equivalence: composing with the translation back to innocent strategies yields a unique strategy for each equivalence class (as described for the denotation of $\lambda f.f\ (f\ 0)$ in Example 5.4.18). Moreover, this uniqueness is easy to capture as a notion of 'normal form'.

Proposition 5.4.24 *Let $s \in [\![T]\!]$ be an innocent play. Then s is the translation of a sequential algorithms path if and only if for any move b in s, the hereditary subsequence $s \upharpoonright b$ satisfies the following criteria*

No interleaving of threads *$s \upharpoonright b$ is a contiguous subsequence of s,*

No repetition of threads *There is no previous occurrence b' of the same move in s, justified by the same move, such that $s \upharpoonright b = s \upharpoonright b'$,*

No overextension of threads *Suppose $s \upharpoonright b = tcd$. Then there is some previous occurence b' of b in s, justififed by the same move, such that $s \upharpoonright b' = t$.*

Proof: is straightforward by induction, using the definition of the translations to show that:

- all translated paths have the above properties,
- if s has the above properties, then $(s^T)_T = s$ so s is a translated path.

□

Open questions raised by this observation are:

- What are the evaluation trees corresponding to the normal forms for strategies?
- What is the syntactic translation from terms to their normal forms corresponding to the semantic translation?
- Is there simple axiomatization (or schematization) of the complete theory of (finitary) observational equivalence extending $\beta\eta$-equivalence?

Chapter 6

Conclusions

This thesis has presented and studied a semantics of sequential control which is simple, tractable, and above all, accurate. Rather than identifying a definitive notion of control flow, the analysis has been implicit, drawing together some more-or-less diverse elements which have a control thread running through them; continuation-passing, the bracketing condition of games semantics, $\lambda\mu$, and sequential algorithms. That control can be identified within this variety of formalisms confirms its ubiquity in sequential computation. That its fully abstract models are effectively presentable and universal supports a claim which is one of the main themes of 'intensional semantics', that behaviour, modelled through a combination of process and domain-theoretic ideas, can be as mathematically well-behaved as static sets and extensional functions. That control flow can be subject to different constraints itself, and can be combined with other computational features, means that no single fully abstract model can amount to a general account of control; identifying it with the bracketing condition within the intensional hierarchy, however, does have the making of such an account.

What then are the main limitations of this analysis, and how might they be rectified? Most significantly, the description of control (in particular, the categorical treatment) has been quite indirect. It has been shown that it is possible to reason about control accurately using continuation passing translation into the λ-calculus, but the usefulness of this observation has not been demonstrated, and is questionable, given the complexity of the translation. However, the consolidation of ideas represented by the fully abstract model is a step forward, and should be a useful setting to test and apply new work on control. A categorical understanding of control which is not more general than continuation passing translation, but is conceptually clearer is emerging, based on the work of [82]. Work by Levy [52] on a game semantics for continuation passing has the potential to make the connection between these ideas and intensional semantics clearer.

The study of games has been limited by the rather ad hoc nature of the framework on which it is based; hopefully it will in future be possible to prove results about dynamic behaviour (for instance, factorization) in a truly natural, clear and concise way. This thesis has attempted small steps towards this goal, — ground-breaking work by Abramsky on a more general notion of 'concurrent game' promises further progress. It should be remembered that one of the advantages of the axiomatic treatment of definability is that it should aid the transfer of both results and intuitions about control to such new settings.

The games semantics of control described here misses the point in another way; it fails to fully exploit their dynamicity. By contrast to the case of state, the games models of control are instances of a general categorical construction (cps), and it is not clear how much has been gained directly by giving game semantic instances. To some extent, this is due to the existence of a simple, general, notion of model of control (which is a good thing) by contrast to, say, Idealized Algol, and in any case the potential of a game semantics of control is still great. But to fully justify the use of games it will be necessary to go beyond merely constructing fully abstract models of control, and redeeem some of the promises made below.

6.1 Further Directions

The scope of the thesis has also been naturally limited by constraints of time and space to a single notion of control based on static bindings, in a simple functional setting. This leaves many avenues for further research.

Local versus non-local control flow The way in which data flows through programs is clearly crucial to understanding their behaviour, and the results which they generate. 'Control flow analysis' is about more than the locality or otherwise of control, although this is an important aspect of it — not least because programs which can access the flow of control can use this to distinguish between different inputs. The way in which control flow is made explicit in game semantics has been exploited by Malacaria and Hankin's 'flowchart' analysis of dataflow in a simple imperative language [58] — this could be extended with control operators, but there are further possibilities.

Another significant area of research is the analysis of locality of variables in imperative languages like Idealized Algol; — identifying expressions which can have side-effects on the store by using a distinction between *active* and *passive* types. The use of a rule (well-bracketing) to model locality of control in game semantics suggests that side-effects on the flow of control could also be governed by a typing distinction between activity and passivity. The connection between the bracketing condition and linearity suggests how this might be done with linear logic. Integrating the two kinds of behaviour, in order to reason about state and control will then be a real challenge!

Dynamically Bound Control features The most obvious gap in the description of control operators in this thesis, is that little has been said about those which create *dynamic bindings*, such as the exception-handling mechanism of Standard ML. An elegant general treatment of these operators can be found in the work by Gunter, Remy and Riecke on prompts [33], which have the expressive power of exceptions and **call/cc**. Unlike the case of **call/cc**, there are at present no general or denotational models of these features; this leaves real scope for the application of game theoretic ideas. Work by this author on a semantics of exceptions has been based on the intensional hierarchy, modelling the dynamic bindings by dropping innocence and visibility (and regaining them via factorization).

Recursive and Polymorphic Types If control flow in realistic functional languages is to be studied, it will be necessary to take account of their more elaborate typing systems. Games semantics for both recursive [59] and polymorphic [42] types have been considered, but the addition of control raises new problems. A semantics based simply on cartesian closure, a single answer object, and recursive types should be sufficient to define models of untyped control operators, such as Scheme's original **call/cc**, via continuation passing translation, though it is not clear whether it would be useful to factor this via a translation into a language ('μFPC') with control operators *and* recursive types.

An interesting case is Felleisen's original $\mathcal{C}$-operator, which is not fettered by typing restrictions, and hence is very expressive. This seems to be an instance where control in games cannot be understood via the bracketing condition, but by a very large choice of 'answer object'; it seems that the language (untyped) $\lambda\mathcal{C}$ can be given a fully abstract semantics by solving the domain equation $D = ((D \Rightarrow D) \Rightarrow D)$.

Towards a semantics of standard ML After combining the above ideas with work on the game semantics of state and higher-type references, active and passive types and so on, the goal of a fully abstract semantics for a realistic language such as standard ML (or Java) begins to look distinctly achievable.

Of course, many issues remain to be resolved, — not least the interaction of control with the other features.

Classical Logic The (extended) Curry-Howard correspondence seems to be an incomplete way of understanding control and classical logic; there are important issues in control which do not arise in logic, and vice-versa. Nonetheless, the interpretation of $\lambda\mu$ described here could form the basis of a semantics of classical natural deduction suited to a useful analysis of constructive classical logic. In particular, the bracketing condition allows strategies (proofs) with intuitionistic behaviour to be distinguished from classical ones within the same system, and the notion of cut-elimination as interaction between strategies means that there is a notion of 'information flow' through proofs which can be tapped, to extract constructive content.

Such a semantics could exploit a full completeness principle; a bijective correspondence between cut-free classical proofs and strategies. However, a large part of the interest (and complexity) of classical logic lies in its behaviour under cut-elimination. To capture this requires an even closer correspondence between syntax and semantics; non-confluence of classical cut-elimination is a well-known obstacle to such an understanding. Various possibilities for making classical proofs more semantically tractable have been discussed, including the 'disambiguation' of cuts by labelling them in some way, which can be described as various forms of 'decorating' of proofs with exponentials in linear logic [22]. Here, perhaps, the link with control can be reforged; it should also be possible to exploit the call-by-name/call-by-value duality of control models (and their linear analysis), to model the duality of classical cuts.

Bibliography

[1] S. Abramsky. Axioms for full abstraction and full completeness. In *Essays in Honour of Robin Milner*. MIT Press, to appear.

[2] 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. Tennent, editors, *Algol-like languages*. Birkhauser, 1997.

[3] S. Abramsky and G. McCusker. Call-by-value games. In *Proceedings of the eleventh international workshop in Computer Science Logic, CSL '97*, LNCS, pages 1–17. Springer-Verlag, 1998.

[4] S. Abramsky, K. Honda, G. McCusker. A fully abstract games semantics for general references. In *Proceedings of the 14th annual Symposium on Logic In Computer Science, LICS '98*, 1998.

[5] S. Abramsky, R. Jagadeesan. Games and full completeness for multiplicative linear logic. *Journal of Symbolic Logic*, 59:543–574, 1994.

[6] S. Abramsky, R. Jagadeesan and P. Malacaria. Full abstraction for PCF. Accepted for publication in Information and Computation, 1996.

[7] P. Baillot, V. Danos, T. Ehrhard, L. Regnier. AJM games are a model of classical linear logic. In *Proceedings of the twelfth International Symposium on Logic In Computer Science, LICS '97*, 1997.

[8] H. P. Barendregt. *The Lambda Calculus, its Syntax and Semantics, revised edition*. North-Holland, 1984.

[9] G. Berry and P.-L. Curien. Sequential algorithms on concrete data structures. *Theoretical Computer Science*, 20:265–321, 1982.

[10] G. Berry, P.-L. Curien and J.-J. Levy. Full abstraction for sequential languages: the state of the art. In M. Nivat, and J. Reynolds, editor, *Algebraic Semantics*, pages 89–132. Cambridge University Press, 1986.

[11] G. Bierman. What is a categorical model of intuitionistic linear logic. In *Proceedings of the Second International Conference on Typed Lambda Calculus*, number 902 in LNCS, 1995.

[12] G. Bierman. A computational interpretation of the $\lambda\mu$-calculus. Technical Report 448, Cambridge University Computer Laboratory, 1998.

[13] A. Blass. A game semantics for linear logic. *Annals of Pure and Applied logic*, 56:183 – 220, 1992.

[14] T. Braüner. A simple adequate categorical model for PCF. In *Proceedings of Third International Conference on Typed Lambda Calculi and Applications, Nancy, France*, number 1210 in LNCS. Springer Verlag, 1997.

[15] R. Cartwright, M. Felleisen. Observable sequentiality and full abstraction. Technical Report 91 - 167, Rice University Department of Computer Science, 1991.

[16] R. Cartwright, P.-L. Curien and M. Felleisen. Fully abstract semantics for observably sequential languages. *Information and Computation*, 1994.

[17] W. Clinger and J. Rees, eds. The revised^4 report on the algorithmic language Scheme. *ACM Lisp pointers IV, July–September 1991*, 1991.

[18] P.-L. Curien. *Categorical combinators, sequential algorithms and functional programming.* Progress in Theoretical Computer science series. Birkhauser, 1993.

[19] P.-L. Curien. On the symmetry of sequentiality. In *Mathematical Foundations of Computer Science*, number 802 in LNCS. Springer, 1993.

[20] P.-L. Curien and H. Herbelin. Computing with abstract Böhm trees, 1996. manuscript.

[21] V. Danos, H. Herbelin and L. Regnier. Games semantics and abstract machines. In *Proceedings of the eleventh International Symposium on Logic In Computer Science, LICS '96*, 1996.

[22] V. Danos, J.-B. Joinet and H. Schellinx. A new deconstructive logic, linear logic, 1996.

[23] P. de Groote. On the relation between the $\lambda\mu$-calculus and the syntactic theory of sequential control. In *Proc. 5th International Conference on Logic*

Programming and automated reasoning - LPAR'94, number 802 in LNCS, pages 31–43. Springer-Verlag, 1994.

[24] T. Ehrhard. Projecting sequential algorithms on strongly stable functions. *Annals of Pure and Applied Logic*, 77, 1996.

[25] M. Felleisen, R. Hieb. The revised report on the syntactic theories of sequential control and state. *Theoretical Computer Science*, 103:235–271, 1992.

[26] W. Felscher. Dialogues as a foundation for intuitionistic logic. In D. Gabbay, and F. Guenthener, editor, *Handbook of Philosophical Logic, Vol III*, pages 341 –372. D. Reidel Publishing Company, 1986.

[27] A. Filinski. *Controlling Effects*. PhD thesis, School of Computer Science, Carnegie Mellon University, 1996.

[28] C. Führmann. Relating two models of continuations. Submitted for publication, 1998.

[29] J.-Y. Girard. Linear logic. *Theoretical Computer Science*, 50, 1987.

[30] J.-Y. Girard. Towards a geometry of interaction. In *Categories in Computer Science and Logic*, volume 92 of *Contemporary Mathematics*, 1989.

[31] J.-Y. Girard. On the unity of logic. *Annals of Pure and Applied Logic*, 59:201–217, 1993.

[32] T. Griffin. A formulae-as-types notion of control. In *Proc. ACM Conference on the Principles of Programming Languages*. ACM Press, 1990.

[33] C. Gunter, D. Remy, and J. Riecke. A generalization of exceptions and control in ML like languages. In *Proceedings of the ACM conference on Functional Programming and Computer Architecture*, pages 12–23, 1995.

[34] R. Harmer. AJM games, self equivalence & non-determinism. In *Proceedings of the Fourth Workshop of the Theory and Formal Methods Group, Department of Computing, Imperial College London*, 1998.

[35] R. Harper, B. Duba and D. MacQueen. Typing first-class continuations in ML. *Journal of Functional Programming*, 3:465–484, October 1993.

[36] J. Hatcliff, O. Danvy. A generic account of continuation passing style. In *ACM Symposium on Principles of Programming Languages*, pages 458–471, 1994.

[37] C. T. Haynes, D.P. Friedman, and M. Wand. Obtaining coroutines from continuations. *Journal of Computer Languages*, 11:143–153, 1986.

[38] H. Herbelin. Games and weak-head reduction for classical PCF. In *Proceedings of TLCA '97*, 1998.

[39] M. Hofmann. Sound and complete axiomatisation of call-by-value control operators. *Math. Struct. in Comp. Sci.*, 5:461–482, 1995.

[40] M. Hofmann and T. Streicher. Continuation models are universal for the $\lambda\mu$-calculus. In *Proceedings of the twelfth International Symposium on Logic In Computer Science, LICS '97*, 1997.

[41] K. Honda and N. Yoshida. Game theoretic analysis of call-by-value computation. In *Proceedings of 24th International Colloquium on Automata, Languages and Programming*, volume 1256 of *Lecture Notes in Computer Science*. Springer-Verlag, 1997.

[42] D. Hughes. Games and definability for system F. In *Proceedings of the twelfth international syposium on Logic in Computer Science, LICS '97*. IEEE press, 1997.

[43] J. M. E. Hyland and C.-H. L. Ong. On full abstraction for PCF: I, II and III. to appear, 1995.

[44] A. Jung, and A. Stoughton. Studying the fully abstract model of PCF within its continuous function model. In *Proceedings of the International Conference on Typed Lambda Calculi and Applications*, volume 664 of *LNCS*, 1993.

[45] G. Kahn, and G. Plotkin. Concrete domains. *Theoretical Computer Science*, Böhm Festschrift special issue, 1993. First appeared as technical report 338 of INRIA-LABORIA, 1978.

[46] R. Kanneganti, R. Cartwright, M.Felleisen. SPCF: its model, calculus and computational power. In *Semantics: Foundations and Applications. Workshop of REX 92*, number 666 in LNCS, 1992.

[47] R. B. Kieburtz, B. Agapiev and J. Hook. Three monads for continuations. Technical Report CS/E 92-018, Oregon Graduate institute of Science and technology, 1992.

[48] J. Laird. Full abstraction for functional languages with control. In *Proceedings of the twelfth International Symposium on Logic In Computer Science, LICS '97*, 1997.

[49] J. Lambek, and P. J. Scott. *Introduction to Higher Order Categorical Logic.* Number 7 in Cambridge Studies in Advanced Mathematics. Cambridge University Press, 1986.

[50] P. Landin. A generalization of jumps and labels. Technical report, UNI-VAC Systems Programming Research, 1965.

[51] S. Mac Lane. *Categories for the working mathematician.* Springer–Verlag, 1971.

[52] P. Levy. Games semantics as continuation passing. Available from http://www.dcs.qmw.ac.uk/ pbl, 1998.

[53] R. Loader. An algorithm for the minimal model. draft, 1997.

[54] R. Loader. Finitary PCF is undecidable. Manuscript, 1997. To appear in Theoretical Computer Science.

[55] J. Longley. The sequentially realizable functionals. Technical Report ECS-LFCS-98-402, LFCS, Univ. of Edinburgh, 1998.

[56] J. Longley and G. Plotkin. Logical full abstraction and PCF. In *Proceedings of the Tblisi Symposium on Language, Logic and Computation.* SiLLI/CSLI, 1996.

[57] P. Lorenzen. *Normative Logic and Ethics.* Mannheim: Zurich: Bibliographisches Institut, 1969.

[58] P. Malacaria and C. Hankin. Generalised flowcharts and games. In *Proceedings of the 25th International Colloquium on Automata, Langugages and Programming*, 1998.

[59] G. McCusker. *Games and full abstraction for a functional metalanguage with recursive types.* PhD thesis, Imperial College London, 1996.

[60] R. Milner. Processes:a mathematical model. In *Logic Colloquium '73*, pages 157–173. North–Holland, 1975.

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

[62] E. Moggi. Computational lambda-calculus and monads. Technical Report ECS-LFCS-88-66, University of Edinburgh Department of Computer Science, 1988.

[63] E. Moggi. An abstract view of programming languages. Technical Report ECS-LFCS-90-113, University of Edinburgh Department of Computer Science, 1990.

[64] E. Moggi. Notions of computation and monads. *Information and Computation*, 93(1), 1991.

[65] H. Nickau. Hereditarily sequential functionals. In *Proceedings of the Symposium on Logical Foundations of Computer Science:Logic at St. Petersburg*, LNCS. Springer-verlag, 1994.

[66] C.-H. L. Ong. A semantic view of classical proofs: type-theoretical, categorical, denotational characterizations. In *Proceedings of 11th IEEE Symposium on Logic In Computer Science, New Jersey, 1996*, pages 230–241, 1996.

[67] C.-H. L. Ong and C. Stewart. A Curry-Howard foundation for functional computation with control. In *Proceedings of ACM SIGPLAN-SIGACT syposium on Principles of Programming Languages, Paris, January 1997*. ACM press, 1997.

[68] V. Padovani. Decidability of all minimal models. In M. Coppo , S. Berardi, editor, *Types for proofs and programs*, volume 1158 of *LNCS*. Springer, 1996.

[69] M. Parigot. $\lambda\mu$ calculus: an algorithmic interpretation of classical natural deduction. In *Proc. International Conference on Logic Programming and Automated Reasoning*, pages 190–201. Springer, 1992.

[70] G. Plotkin. Postgraduate lecture notes in advanced domain theory (incorporating the 'Pisa notes'. Dept. of Computer Science, Univ. of Edinburgh, 1981.

[71] G. D. Plotkin. LCF considered as a programming language. *Theoretical Computer Science*, 5:223 – 255, 1977.

[72] G. D. Plotkin. A structural approach to operational semantics. Technical Report DAIMI FN-19, Aarhus University Computer Science Department, 1981.

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

[74] J. Power and E. Robinson. Premonoidal categories and notions of computation. *Mathematical Structures in Computer Science*, 1997.

[75] D. Prawitz. *Natural Deduction.* Number 3 in Stockholm Studies in Philosophy. Almqvist and Wiksell, 1965.

[76] J. Reynolds. Definitional interpreters for higher-order programming languages. In *Proceedings of the* 25^{th} *ACM National Conference*, pages 717–740, 1972.

[77] J. Reynolds. The discoveries of continuations. *Lisp and Symbolic Computataion*, 6(3/4):223–247, 1993.

[78] D. S. Scott. A type-theoretic alternative to CUCH, ISWIM and OWHY. *Theoretical Computer Science*, 121:411–440, 1969.

[79] P. Selinger. Control categories: an axiomatic approach to the semantics of control in functional languages. Presented at MFPS workshop, 1998.

[80] D. Sitaram, and M. Felleisen. Reasoning with continuations II: Full abstraction for models of control. In *Proc. 1990 ACM conference on LISP and functional programming*, pages 161–175, 1991.

[81] T. Streicher, B. Reus. Continuation semantics: Abstract machines and control operators. Submitted to the Journal of Functional Programming, 1996.

[82] H. Thielecke. *Categorical Structure of Continuation Passing Style.* PhD thesis, University of Edinburgh Department of Computer Science, 1997.

Index