Some Intensional Models of Lambda Calculus

CANDIDATE: Gianluca Franco

SUPERVISOR: Furio Honsell

REFEREE: Chantal Berline
Luke Ong

20/04/2001
To Sabrina, to our son
This is my letter to the World
That never wrote to Me –
The simple News that Nature told –
With tender Majesty

Her Message is committed
To Hands I cannot see –
For love of Her – Sweet – countrymen
Judge tenderly – of Me

E. D.
Abstract

The advent of Linear Logic [Gir87] stimulated the introduction of an *intensional semantics* paradigm for programming languages. This semantics was intended to be intermediate between the syntax-dependent operational semantics and the abstract functional semantics. A very fruitful realization of this intensional paradigm has been *game semantics* [AJ94a, AJM96, HO00, Nic96], which has allowed for the definition of fully-abstract models – hence abstract models capturing exactly the operational features – of a rich variety of programming languages [AJM96, HO00, AM95a, AM97, AHM98, Nic96, KNO01].

In this thesis we study some models of the untyped λ-calculus arising as applications of the intensional semantics paradigm. In particular, we study game semantics for the untyped λ-calculus by introducing suitable tools to build models in the category $G$ of games and history-free strategies of Abramsky et al. [AJM96] and we characterize their local structure, *i.e.* the set of equations between terms enforced by them.

Finally we make a few initial steps towards an investigation of the models of the untyped λ-calculus obtained as linear combinatory algebras through the Geometry of Interaction construction of Abramsky [Abr96].

The results of this thesis enforce the idea that game semantics has a strong bias towards head and weak head reduction. This property can be considered as a kind of *rigidity* with respect to other semantical paradigms (as, for instance, the topological world of complete partial orders and continuous functions). In fact, only three λ-theories have a model in the game semantics setting. They are the maximal sensible theory $H^*$ induced also by the model $D_{\infty}$ of Scott, the theory $B$ that identifies two terms if and only if they have the same Böhm tree and the theory $L$ that identifies two terms if and only if they have the same Lévy-Longo tree.
Acknowledgments

As Beaumarchais has underlined “books are for authors as babies are for mothers: conceived in pleasure, carried with fatigue and given birth with pain.” This effort has been less harsh for me for the help and encouragement of many people. I want to thank my supervisor Furio Honsell, to whom I owe ideas and stimuli, for his immeasurable patience. A special mention deserves Pietro Di Gianantonio and Marina Lenisa for many useful suggestions. Fabio Alessi has introduced me to the categories world and has supported me in clearing many doubts. I had a lot of useful discussions with many people throughout the world. I would to mention, among many others, Samson Abramsky, Andrea Asperti, Corrado Böhm, Mariangela Dezani, Simone Martini, Luke Ong and Simona Ronchi della Rocca and the internal referees of the University of Udine Maria Staniszki and Simone Martini. I would like to thank the European HCM Research Network Lambda Calcul Typè, the European TMR Network Linear Logic in Computer Science, the Esprit working group TYPES, the Programmi di Ricerca scientifica Tecniche formali per la specifica, l’analisi, la verifica, la sintesi e la trasformazione di sistemi software and Teoria della Concorrenza, Linguaggi di Ordine Superiore e Strutture di Tipi and the Department of Mathematics and Computer Science of the University of Udine for supporting my work.
viii
# Contents

Introduction xiii

I Basic concepts 1

1 Category theory 3
  1.1 Basic category theory 4
  1.1.1 Categories 4
  1.1.2 Functors 6
  1.1.3 Natural transformations 7
  1.1.4 Adjunctions 8
  1.1.5 Products 9
     Tensor product 9
     Cartesian product 9
  1.1.6 Limits and colimits 10
  1.2 Categories with product 14
     1.2.1 Monoidal categories 14
     1.2.2 Cartesian closed categories 16
     1.2.3 Traced monoidal categories 18
     1.2.4 Monads 20

2 Lambda Calculus Issues 23
  2.1 Lambda theories 24
  2.2 Trees of λ-terms 27
  2.3 Labelled λ-calculus 33

3 Models of the Lambda Calculus 35
  3.1 Algebraic models of λ-calculus 36
     3.1.1 Environment models 38
     3.1.2 Other algebraic models 39
     3.1.3 Examples of environment models 40
        The model $D_\infty$ 41
        The model $P^n$ 43
        The model $D_\oplus$ 44
        The completeness of environment models 46
  3.2 Categorical models of λ-calculus 46
     3.2.1 Definitions 46
     3.2.2 The relation between algebraic and categorical models 47


3.2.3  The completeness of categorical models .................................. 48
3.3  General properties of models .................................................... 49
   3.3.1  Contexts and trees ............................................................. 49
   3.3.2  Böhm-out technique ........................................................... 49
   3.3.3  The approximation theorem ................................................ 50

II  Game semantics for Lambda Calculus 55

4  Game Semantics 57
   4.1  Basic definitions ................................................................. 58
      4.1.1  The notion of game ......................................................... 58
      4.1.2  Construction on games .................................................... 61
             Tensor product ......................................................... 61
             Linear implication ....................................................... 62
             Exponential game ...................................................... 63
      4.1.3  Strategies ................................................................. 63
             Set-theoretic description of strategies ................................ 64
             History-free strategies .................................................. 64
             Copy-cat strategies ...................................................... 65
             The application strategy ............................................... 66
             Composition of strategies .............................................. 67
   4.2  Categories of games ............................................................ 67
      4.2.1  The category Games ..................................................... 68
      4.2.2  The category \( \mathcal{G} \) .................................................. 68
      4.2.3  The category \( K_t(\mathcal{G}) \) ........................................... 72
      4.2.4  The category \( \mathcal{G}^s \) .................................................. 74
   4.3  Approximating strategies ..................................................... 75

5  Game Lambda Theories 79
   5.1  Some categorical game models for \( \lambda \)-calculus ....................... 80
   5.2  The theories of the game models .......................................... 84
      5.2.1  The theory of models \( \mathcal{D}^C \) ....................................... 85
      5.2.2  The theory of models \( \mathcal{D}^S \) ....................................... 85
      5.2.3  The theory of models \( \mathcal{D}^E \) ....................................... 86
      5.2.4  The main result ............................................................ 86

III  Further developments 87

6  A Finitary Logical Description of Game Semantics 89
   6.1  The type assignment system ............................................... 90
   6.2  Equivalence of semantics ................................................... 92

7  Geometry of Interaction 95
   and other stories
   7.1  Linear combinatory algebras ............................................... 95
      7.1.1  A case study: \( Pfn(\omega,\omega) \) ..................................... 96
             Application ............................................................... 96
## CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Combinators</td>
<td>97</td>
</tr>
<tr>
<td>The operator !</td>
<td>99</td>
</tr>
<tr>
<td>7.2 The general Geometry of Interaction construction</td>
<td>100</td>
</tr>
<tr>
<td>7.2.1 Weak linear categories</td>
<td>100</td>
</tr>
<tr>
<td>7.2.2 The Geometry of Interaction (GoI) construction</td>
<td>101</td>
</tr>
<tr>
<td>7.3 Graph models</td>
<td>102</td>
</tr>
<tr>
<td>7.3.1 Linear graph models</td>
<td>102</td>
</tr>
<tr>
<td>7.3.2 Linear and standard graph models</td>
<td>102</td>
</tr>
<tr>
<td>7.3.3 Some linear combinatory algebras arising from GoI construction</td>
<td>103</td>
</tr>
<tr>
<td>7.4 Game semantics</td>
<td>104</td>
</tr>
<tr>
<td>7.4.1 The category of resumptions</td>
<td>104</td>
</tr>
<tr>
<td>7.4.2 Specification structures</td>
<td>105</td>
</tr>
<tr>
<td>7.4.3 Finding again the category Games</td>
<td>106</td>
</tr>
<tr>
<td>Conclusions</td>
<td>109</td>
</tr>
<tr>
<td>Bibliography</td>
<td>113</td>
</tr>
</tbody>
</table>
Introduction

“...at the time through self-study I found out about the λ-calculus of Curry and Church which, literally, gave me nightmares at first.”

D. Scott [Sco87], p. 50

The advent of the French Revolution enabled by the introduction of the Linear Logic of Jean-Yves Girard [Gir87] has made possible a radical change in many fields of formal logics, in particular in the semantics of programming languages. Before this shake-up, the semantical section of the computer science universe, was divided (in a Manichean view) by a sharp dichotomy: the operational semantics world on one side (assigning a meaning to a program on the basis of its syntax) and the denotational semantics world on the other (assigning meaning to a program on the basis of its functionality).

Operational semantics was born with the intent to be a precise support for the implementer of a programming language. It roughly describes the steps a machine takes interpreting a program of a given programming language. It strongly relies, of course, on the syntax of the language. Further developments rendered operational semantics more formal; in particular the introduction of the Structured Operational Semantics (SOS) [Pl81, BH90] allowed for the investigation of many properties of programs but with the payoff of long, tedious and error-prone proofs.

Denotational semantics can be considered as the semantical paradigm which best approximates the ideal of a mathematical theory of computation of McCarty [McC63] and Scott [Sco70]. Despite its pretensions of broad applicability, it carries a bias towards functional computation: the computation world where the behavior of a program is adequately abstracted as the computation of a function. Denotational semantics has allowed for the introduction of many interesting features in the analysis of computation: types and type-checking, higher-order functions, recursive types, polymorphism [Gir72, Rey74b], continuations [Lan65, Rey74a, SF92], monads [Mog91, Wad92].

The appearance of Linear Logic brought from its beginning – with a denotational semantics of the formal system proposed by Girard itself [Gir87] – an innovative look to the semantics of programming languages, with a special accent on the usage of the resources of a program. This aspect allowed to introduce an intensional semantics for the programming languages – assigning a meaning to a program that does not rely exclusively on its external behavior – which lies between the operational and the denotational sides. This was also the main course of the Geometry of Interaction program initiated by Girard [Gir88a, Gir88b, Gir95] and which followed the introduction of Linear Logic. This development in the semantics of the programming languages, allowed to capture significantly the computation paradigms of reactive, real-time and distributed systems, that, as it was readily apparent, did not fit well in the functional scenario. Moreover, also in the functional com-
putational paradigm, some aspects of the computation such as the sequentiality (the PCF problem [Plo77, Mil77]), computational complexity and optimality of reduction strategies [Lév80], seemed not to be captured well by standard denotational semantics.

Among the different implemetations of the new semantics (the coherent spaces and stable and strongly-stable functions [Gir87, BE91, Bas96], the sequential algorithms [Ber78, BC82]), the game semantics implementation seemed to be the most promising. It was developed in two main dialects, both based on former ideas of Blass [Bla92]: one by Abramsky et al. [AJM96] with the starting intent of giving a fully abstract model for Linear Logic [AJ94a] and the other by Hyland and Ong [HO00], inspired by the works of Kleene [Kle78] and Berry and Curien [BC82]. Both succeeded in solving the outstanding problem of finding a fully-abstract model for the (sequential) language PCF. A slightly variation on the second dialect was introduced by Nickau [Nic96], with the intent of characterizing exactly the computability in higher types of Kleene [Kle69].

The strong capability of capturing exactly the inherent notion of sequentiality present in the prototypical language PCF, made game semantics suitable to investigate, with the same enthusiastic spirit, the reduction process to which the terms of $\lambda$-calculus undergo $\lambda$-calculus has been a preeminent tool for the analysis of programming languages. It allowed the formalization of many properties of programs and their study in a formal manner. Moreover, it has also a pragmatical interest, being a programming language itself (in the LISP version of McCarthy, for instance [McC60]) and being largely used as an intermediate form in the compilation process [App92].

As a metalanguage, $\lambda$-calculus allows for the study of suitable relations on $\lambda$-terms (congruences) that originate the $\lambda$-theories, each of which represents an operational semantics of the programming language under consideration. Since now, there is no semantical framework that is able capturing abstractly the preeminent features of each $\lambda$-theory, i.e. capable of giving a fully-abstract model for each of them. Many $\lambda$-theories have a topological model – a classical denotational semantical domain – but it has been shown by Honsell and Ronchi della Rocca [HR92] that there are $\lambda$-theories that cannot be induced by a topological model, a situation which is known as the topological incompleteness of $\lambda$-calculus. What seems to go wrong is the inherent parallel behavior of some continuous functions – the topological elements – that contribute to constitute models with too much points. It is natural to ask if the benefits obtained for the PCF by the usage of game semantics can be transposed also on the semantics of $\lambda$-calculus.

The objective of the thesis is the application of the new paradigm of intensional semantics to untyped $\lambda$-calculus. This purpose is achieved in different directions. First, we shall investigate which $\lambda$-theories have got a model in the category $G$ of games and history-free strategies of Abramsky et al. [AJ94a, AJM96]. What we have found, leaving out any unnecessary suspense, is that game semantics is even more rigid then topological semantics; only three $\lambda$-theories can be modeled using games and history-free strategies in spite of the rich class that can be modelled by topological and other models. We characterize completely all the $\lambda$-theories which have a model in $G$, applying then exhaustively the game semantics paradigm to untyped $\lambda$-calculus.

Game models (that are fully-complete) for two of these theories are built in [KNO01, KN099] in the different setting of games and innocent strategies. These papers contribute to enforce the feeling that the very notion of game represents the element of rigidity in the game semantics paradigms. Moreover, we are confident that the same results apply also in the game setting of games and history-free strategies.
From another perspective, starting from the fact that all the classical denotational semantics models of λ-calculus allows a finitary presentation in terms of an applicative structure whose elements are sets of properties (types) (the filter models [BCDC83, CDCHL84]), we investigate the possibility of giving a similar presentation also for the game semantics. This is not, of course, a rewriting of known results, as the category of games and history-strategies \( \mathcal{G} \) is not concrete and hence cannot be viewed as a category of sets and functions. A new notion of type and type assignment system needs to be introduced to accomplish the task.

Besides the topological world of denotational semantics for λ-calculus there is a general set-theoretic model construction for λ-calculus that allows to build mathematical structures known as the Plotkin-Scott-Engeler algebras (PSE-algebras) or graph-models [Lon83]. Such structures are very flexible: \( 2^\omega \) different λ-theories have their model in this setting.

In the context of the Geometry of Interaction paradigm, Abramsky introduced and studied the linear graph models and the associated notion of linear combinatory algebras [Abr97, AHP98]. Such structures are strongly connected to the standard combinatory algebras. As a third line of work, we shall study linear combinatory algebras induced by linear graph models, investigating their relation with standard combinatory algebras induced by standard graph models and with linear combinatory algebras induced by the Geometry of Interaction construction of Abramsky and Girard [Abr96, Gir89].

**Structure of the thesis.** The thesis is divided in three parts. In the first part the basic notions used throughout the rest of the thesis are introduced. We give also introductory remarks and provide insights for such notions. The second part is devoted to the complete characterization of the theories of untyped λ-calculus that are induced by game categorical models of λ-calculus. The third part concerns further developments on the main theme, the conclusions and discussion of open problems.

Chapter 2 is devoted to the introduction of all the categorical notions necessary to understand the following chapters.

Untyped λ-calculus together with some tools necessary to its study are introduced in Chapter 3, together with the proofs of some properties of the λ-theories.

In Chapter 4 the notions of categorical model and extensional categorical model of λ-calculus are presented together with a brief survey of the algebraic models. Some general results about the theories induced by some models are then shown as some considerations on existing works on the subject.

Second part starts with Chapter 5 where we introduce the basic notions of game semantics that shall be used in the following chapters.

The full characterization of the theories of untyped λ-calculus in the game semantics setting occupies Chapter 6.

A type assignment system for the game semantics is introduced in Chapter 7, and it is shown to be sound in the sense of inducing the same categorical interpretation as the standard one.

Chapter 8 introduces the world of Geometry of Interaction and presents some results on the linear combinatory algebras. Moreover, a category of games introduced in Chapter 4 is rebuilt in this setting by means of a general construction.

Conclusions and open problems are the subject matter of Chapter 9.
INTRODUCTION
I

Basic concepts
Category theory

"Perhaps the purpose of categorical algebra is to show that which is trivial is trivially trivial."

P. Freyd

Abstract

The basic category theory necessary for the interpretation of untyped \(\lambda\)-calculus and for carrying out the Geometry of Interaction construction is introduced. Two different axiomatic definitions of a category are presented. The first is the object-arrows definition, it emphasizes the "typed" nature of category theory. The second is the only-arrows definition, which emphasizes the algebraic aspect of category theory. We discuss in particular the "categories with product", monoidal, symmetric monoidal, symmetric monoidal closed and Cartesian closed as basic frameworks for the intensional semantics of \(\lambda\)-calculus. Finally recursively defined objects are presented.

Category theory is a convenient conceptual language for the studying of abstract properties of mathematical objects. It has progressively gained relevance in many fields of mathematics. It is based on the notion of category, which consists of a class of structures — called objects — and a class of relations between these structures — called arrows or morphisms. From the point of view of computer science, objects are "abstract data types" and the language of category theory prevents us to investigate their internal structure. Their usage is described only by their external behavior through the arrows that connect them to other objects. Category theory for theoretical computer science has been mainly used for giving denotational semantics of programming languages, to describe theories of types and models of logical systems. Other applications of category theory in computer science regard software engineering, artificial intelligence and automata theory.

Using an Object-oriented programming metaphor, we can say that the main advantage in defining an abstract property of structures is the "modularity" and "reusability" of the

\footnote{In S. Eilenberg, D. K. Harrison, S. Mac Lane and H. Röhrle editors, Proceedings of the Conference on Categorical Algebra, La Jolla, California, Springer-Verlag, New York, 1965}
property in different contexts. A statement valid for all Cartesian categories is nevertheless valid for all its different instances, like – for instance – the Cartesian closed categories. As a practical example, it is known that every reflexive object in a Cartesian closed category is a model of untyped λ-calculus. If we exhibit a reflexive object in a whichever Cartesian Closed category we are certain to have built such a model and do not need any further check.

The main reference for category theory is [Mac71]. Other useful general publications are [BW90, Cro93, FS91, AL91, Pie91]. The category theory machinery necessary to define models of λ-calculus can be found also in [Bar84], chapter 5.

1.1 Basic category theory

1.1.1 Categories

There are two axiomatic definitions of a category that aim to underline the different aspects of category theory: the typed view looks at objects as types and at arrows as abstractions of functions; the untyped view takes in consideration the “essentially algebraic nature” (quoted from [FS91]) of category theory, looking at arrows as abstract elements of an algebraic structure and at objects as their related identity arrows.

The typed definition is the most broadly used while the untyped one is more suitable for defining some aspects of the interpretation of λ-calculus inside a categorical framework. In particular, it gives more intuition on the theorem of Scott [Sco80b] which asserts that each λ-theory has a model in a suitable Cartesian closed category. The general Geometry of Interaction construction too (Chapter 7), as presented in the paper [AJ94b], is based on this view.

The two definitions are clearly equivalent, in a strong manner: a category defined with the axioms of one definition is a category also in the other sense.

Definition 1.1 (The objects-arrows category). A category consists of objects \( A, B, C, \ldots \) and arrows \( f, g, h, \ldots \) and four operations:

- **dom** which assigns to each arrow \( f \) its domain object \( \square f \);
- **cod** which assigns to each arrow \( f \) its codomain (or target) object \( \square f \);
- **id** which assigns to each object \( A \) an identity arrow \( 1_A \);
- **comp** that assigns to each ordered pair of arrows \( (f, g) \) with \( f \square = \square g \) an arrow \( g \circ f \) with \( \square (g \circ f) = \square f \) and \( (g \circ f) \square = g \square \).

If \( \square f = A \) and \( f \square = B \) we write \( f : A \rightarrow B \). The operations satisfy the following two axioms:

- **Associativity.** For any objects \( A, B, C, D \) and arrows \( f : A \rightarrow B, g : B \rightarrow C \) and \( h : C \rightarrow D \), \( h \circ (g \circ f) = (h \circ g) \circ f \).
- **Identity.** For each arrow \( f : A \rightarrow B \) and \( g : B \rightarrow C \), \( 1_B \circ f = f \) and \( g \circ 1_B = g \).

Given a category \( C \), the set of arrows \( f : A \rightarrow B \) for each pair of objects \( A, B \) of \( C \) is indicated with \( C(A, B) \). A subcategory \( C' \) of \( C \) consists of some objects and some arrows of \( C \) such that for each arrow \( f \) in \( C' \) the objects \( \square f \) and \( f \square \) are in \( C' \), for each object \( A \) in \( C' \) \( 1_A \) is in \( C' \) and for each pair of compoundable arrows \( f, g \) in \( C' \) the compound is in \( C' \). Analogously we can define the notion of super-category.
Definition 1.2 (The only-arrows category). A category consists of arrows \( f, g, h, \ldots \), of some ordered pairs \( (f, g) \) of arrows which can be composed and of an operation which assigns to each pair \( (f, g) \) the composition arrow \( g \circ f \).

An identity is an arrow \( u \) such that \( f \circ u = f \) whenever \( f \circ u \) is defined and \( u \circ g = g \) whenever \( u \circ g \) is defined.

The following axioms must be satisfied:

- \((h \circ g) \circ f \) is defined iff \( h \circ (g \circ f) \) is defined. If both are defined then \( (h \circ g) \circ f = h \circ (g \circ f) \triangleq hgf \);
- \( hgf \) is defined iff \( hg \) and \( gf \) are defined;
- for each arrow \( f \) there are identity arrows \( u \) and \( u' \) such that \( f \circ u = u' \circ f \) are defined.

Notice that an identity arrow , if exists, is unique, that is if \( u, w \) are arrows such that \( u \circ f \) is defined if and only if \( w \circ f \) is defined and \( u \circ f = w \circ f = f \) then \( u = w \).

Examples of Categories

- The empty category \( \mathbb{0} \): a category with no objects and no arrows.
- The category \( 1 \): a category with one object \( \bullet \) and the identity arrow \( 1 \).
- The category \( 2 \): a category with two objects \( \bullet \) and \( \circ \), their identity arrows and an arrow \( \bullet \to \circ \).
- Each monoid \( \langle M, \cdot, I \rangle \) is a category \( M \) with one object \( M \), the identity arrow \( 1_M = I \), arrows \( x : M \to M \) for each \( x \in M \) and composition given by \( x \cdot y = x \cdot y \).
- The collection of sets and total functions between them, where the compositions is the function composition and the identities are the identity functions forms the category \( \text{Set} \).
- The collection of sets and relations between them forms the category \( \text{Rel} \). The composition is defined in the usual way, that is given \( R : A \to B, R \subseteq A \times B \) and \( S : B \to C \), \( S \circ R : A \to C \) is defined as

\[
\{ \langle a, c \rangle \in A \times C \mid \exists b \in B : \langle a, b \rangle \in R, \langle b, c \rangle \in S \}\]

The identity arrows \( 1_A : A \to A \) are given by \( \{ \langle a, a \rangle \mid a \in A \} \).
- Monoids and monoid homomorphisms form the category \( \text{Mon} \), groups and group homomorphisms form the category \( \text{Grp} \).
- The category \( \text{Set} \) is a sub-category of the category \( \text{Rel} \); the category \( \text{Grp} \) is a sub-category of the category \( \text{Set} \).
- The collection of complete partial orders \( \langle A, \sqsubseteq \rangle \) and continuous functions between them forms the category \( \text{CPO} \). A function \( f : \langle A, \sqsubseteq_A \rangle \to \langle B, \sqsubseteq_B \rangle \) is continuous when, given a chain \( X \subseteq A \), \( f(\bigcup X) = \bigcup \{ f(x) \mid x \in X \} \).
- A discrete category is a category in which the only arrows are the identities.
Given categories $C$ and $D$ the *product category* $C \times D$ of $C$ and $D$ has as objects pairs $\langle C, D \rangle$ with $C \in \text{Obj}(C)$ and $D \in \text{Obj}(D)$, as morphisms pairs $\langle f, g \rangle : \langle A, A' \rangle \to \langle B, B' \rangle$ with $f : A \to B$ an arrow of $C$ and $g : A' \to B'$ an arrow of $D$, and the composition of two arrows

$$\langle A, A' \rangle \xrightarrow{\langle f, g \rangle} \langle B, B' \rangle \xrightarrow{\langle f', g' \rangle} \langle A'', B'' \rangle$$

is defined as $\langle f', g' \rangle \circ \langle f, g \rangle = \langle f' \circ f, g' \circ g \rangle$.

Given a category $C$, the *opposite category* $C^{\text{op}}$ has the same collection of objects of $C$ and for each arrow $f : A \to B$ in $C$ there is a corresponding arrow $f' : B \to A$ in $C^{\text{op}}$. The identity arrow for an object $A$ of $C^{\text{op}}$ is $1_A$ of $C$. If $f' : A \to B$ and $g' : B \to C$ are arrows of $C^{\text{op}}$ then $g' \circ f' : A \to C$ is $(f \circ g)'$.

Let $C$ be a category and $B$ an object of $C$. The *slice* of $C$ by $B$, denoted as $C / B$, is the category whose objects are arrows $f : A \to B$ in $C$, and whose arrows $g : f \to f'$ are those arrows $g : \Box f \to \Box f'$ in $C$ such that $f' \circ g = f$.

Let $F : C \to D$ be a functor and let $D$ an object of $D$. The *over-cone* category $[\text{Cro93}]$ or *super-comma* category [Mac71] $(F \downarrow D)$ is the category where objects are pairs $\langle A, f \rangle$, with $A$ an object of $C$ and $f : D \to FA$ an arrow of $D$, and where arrows $h : \langle A, f \rangle \to \langle B, g \rangle$ are arrows $h' : A \to B$ of $C$ which satisfy $F(h') \circ f = g$. Similarly the *under-cone* category $(D \downarrow F)$ is the category where objects are pairs $\langle A, f \rangle$, with $A$ an object of $C$ and $f : FA \to D$ an arrow of $D$, and where arrows $h : \langle A, f \rangle \to \langle B, g \rangle$ are arrows $h' : A \to B$ of $C$ which satisfy $g \circ F(h') = f$.

As is custom in category theory to abbreviate a long list of equations, we shall make use of diagrams. We remind that a diagram is *commutative* when each path between two vertices denotes the same arrow.

### 1.1.2 Functors

Functors are arrows (transformations) between categories satisfying some properties which assure that the transformation they act is *natural*, that is independent of the particular essence or shape of the objects and the arrows.

**Definition 1.3 (Functor).** A functor $F : C \to D$ from a category $C$ to a category $D$ assigns to each object $C \in C$ an object $FC \in D$ and to each arrow $f : A \to B$ of $C$ an arrow $F(f) : FA \to FB$ in $D$ in a such a way that the following conditions are satisfied:

$$F(1_C) = 1_{FC} \quad F(g \circ f) = F(g) \circ F(f)$$

A functor $F : C \to C$ is called *endofunctor* on $C$. A functor $F : C \times C \to C$ is called *bifunctor* on $C$. If $F : C \to D$ is bijective on both objects and arrows it is an isomorphism and the categories $C$ and $D$ are said to be isomorphic. A functor $F : C \to D$ is *full* if for each pair of objects $A, B$ of $C$ and for each arrow $g : FA \to FB$ there exists an arrow $f : A \to B$ such that $g = F(f)$, and is faithful if for each pair of objects $A, B$ and for each pair of arrows $f, g : A \to B$ of $C$, $F(f) = F(g)$ implies $f = g$. If for each $f : A \to B$ in $C$, $F(f) : FB \to FA$ in $D$, the functor $F$ is said to be *contravariant*, otherwise it is said to be *covariant*. 
Examples of Functors

- Given a category $C$, $\mathcal{I}_C : C \to C$ which maps each object $A$ of $C$ to itself and each arrow $f$ of $C$ to itself is the identity functor.

- Given categories $C$ and $D$, and an object $D$ of $D$, $\mathcal{U} : C \to D$ defined by $\mathcal{U}C = D$, $\mathcal{U}(f) = 1_D$ for each object $C$ and each arrow $f$ of $C$ is a constant functor.

- $U : \text{Grp} \to \text{Set}$ which maps each group $\langle A, \cdot \rangle$ to its underlying set $A$ and each group homomorphism $f : \langle A, \cdot \rangle \to \langle B, \cdot \rangle$ to its underlying map $f^\sharp : A \to B$ is a functor that is called forgetful since it forgets about the group structure.

- Given a category $C$ and an object $X$ in $C$, $\mathcal{C}(X, -) : C \to \text{Set}$ which assigns to each object $A$ of $C$ the set $\mathcal{C}(X, A)$ and to each arrow $f : A \to B$ the function $f^* : \mathcal{C}(X, A) \to \mathcal{C}(X, B)$ defined by $f^*(g) = f \circ g$ for each $g \in \mathcal{C}(X, A)$, is a covariant functor called hom-functor.

- Given a category $C$ and an object $X$ in $C$, $\mathcal{C}(-, X) : C \to \text{Set}$ which assigns to each object $A$ of $C$ the set $\mathcal{C}(A, X)$ and to each arrow $f : A \to B$ the function $f_* : \mathcal{C}(B, X) \to \mathcal{C}(A, X)$ defined by $f_*(g) = g \circ f$ for each $g \in \mathcal{C}(B, X)$, is a contravariant functor.

- Given a set $A$ let $A^*$ be its Kleene closure, that is $A^* = \{a_1a_2\cdots a_n \mid a_1, a_2, \ldots, a_n \in A, n \geq 0\}$. $\mathcal{K} : \text{Set} \to \text{Mon}$, the Kleene-closure functor is defined by $\mathcal{K}(A) = A^*$ and $\mathcal{K}(f) = \text{map}(f)$ where if $f : A \to B$ then $\text{map}(f) : A^* \to B^*$ was defined by $a_1a_2\cdots a_n \mapsto f(a_1)f(a_2)\cdots f(a_n)$.

1.1.3 Natural transformations

Natural transformations are arrows between functors. Their name indicates that the action they carry out is uniform on the functors they act on.

**Definition 1.4 (Natural transformation).** Given two functors $F, G : C \to D$, a natural transformation $\eta : F \Rightarrow G$ assigns to each object $A$ of $C$ an arrow $\eta_A : FA \to GA$ such that for each arrow $f : A \to B$ in $C$ the following diagram commutes

$$
\begin{array}{ccc}
FA & \xrightarrow{F(f)} & FB \\
\downarrow{\eta_A} & & \downarrow{\eta_B} \\
GA & \xrightarrow{G(f)} & GB
\end{array}
$$

If for each $A$ the arrow $\eta_A$ is invertible then the natural transformation is called natural isomorphism and it is indicated as $\eta : F \cong G$.

In particular, let $F : C \to D$ and $G : D \to C$ be two functors, consider their intuitive composition $F \circ G : D \to D$ or $G \circ F : C \to C$. If there exists a natural isomorphism $\eta : \mathcal{I}_D \cong F \circ G$ or $\eta' : \mathcal{I}_C \cong G \circ F$ then the categories $C$ and $D$ are said to be equivalent, which is a weaker notion of isomorphic. Precisely, two equivalent categories are similar but could have very different “dimensions”.
Examples of Natural Transformations

- Let $X$ be a fixed set. Let $F_X : \textbf{Set} \to \textbf{Set}$ be defined by $F_X(A) = (X \Rightarrow A) \times X$ and $F_X(f) = f^* \times 1_X$ where $X \Rightarrow A$ is the set of all the total functions from $X$ to $A$ and $f^*$ is defined as above. The map $A \mapsto ev_A$ where $ev_A : (X \Rightarrow A) \times X \to A$ is defined by $(f, x) \mapsto f(x)$ defines a natural transformation $ev : F_X \rightarrow I_{\textbf{Set}}$.

- Given a set $A$ define $rev_A : A^* \rightarrow A^*$ as $a_1a_2 \cdots a_{n-1}a_n \mapsto a_na_{n-1} \cdots a_1$. The maps $rev_A$ define a natural transformation $rev : \mathcal{K} \rightarrow \mathcal{K}$.

- Consider the functor $\Pi : \textbf{Set} \rightarrow \textbf{Set} \times \textbf{Set}$ defined by $\Pi(A) = \langle A, A \rangle$ and for $f : A \rightarrow B$, $\Pi(f) = \langle f, f \rangle : \langle A, A \rangle \rightarrow \langle B, B \rangle$ and the functor $* : \textbf{Set} \rightarrow \textbf{Set} \times \textbf{Set}$ defined by $*(A) = \langle A, \{\ast\} \rangle$ and for $f : A \rightarrow B$, $*(f) = \langle f, 1_{\{\ast\}} \rangle : \langle A, \{\ast\} \rangle \rightarrow \langle B, \{\ast\} \rangle$. The maps $\eta_A : \langle A, A \rangle \rightarrow \langle A, \{\ast\} \rangle$ defined by $\eta_A = \langle 1_A, !_A \rangle$, where $!_A : A \rightarrow \{\ast\}$ is the unique function from $A$ to $\{\ast\}$, define the only natural transformation $\eta : \Pi \rightarrow *$ between the functors $\Pi$ and $*$.

- Let $C$ and $D$ be two categories, $D$ and $E$ two objects of $D$ and $\overline{D}$ and $\overline{E}$ the related constant functors. Let $f : D \rightarrow E$ an arrow of $D$. The natural transformation $\eta^f : \overline{D} \rightarrow \overline{E}$ assigns to each object $C$ of $C$ the arrow $f$.

1.1.4 Adjunctions

A very important notion, which plays a preeminent role in the definition of monoidal closed and Cartesian closed categories, is that of adjunction between categories. It has been pointed out [Mac71] that adjunctions arise everywhere in mathematics. We are confident that the notion of adjunction is very useful to define concisely many categorical notions.

Definition 1.5 (Adjunction). An adjunction between two categories $C$ and $D$ is a triple $\langle F, G, \varphi \rangle$ where $F : C \rightarrow D$ and $G : D \rightarrow C$ are functors, and $\varphi : D(F(-), -) \cong C(-, G(-))$ is a natural isomorphism, that is it assigns for each pair of objects $C$ in $C$ and $D$ in $D$ a bijection

$$\varphi_{C,D} : D(FC,D) \cong C(C,GD)$$

$F$ is said to be the left adjoint while $G$ is said to be the right adjoint.

Examples of Adjunctions

- The categorical product functor is a right adjoint to the diagonal functor $\Delta$ defined as $\langle C \times C \rangle \circ \langle A, A \rangle \cong C(A, B \times C)$ (whose definitions are recalled below) since for each pair of arrows $f : A \rightarrow B$ and $g : A \rightarrow C$ there is only one arrow $\langle f, g \rangle : A \rightarrow B \times C$ which commutes with the projections.

- The categorical co-product functor is a left adjoint to the diagonal functor $\Delta$ defined as $C(A + B, C) \cong (C \times C) \circ \langle A, B \rangle \circ \langle C, C \rangle$ since for each pair of arrows $f : A \rightarrow C$ and $g : B \rightarrow C$ there is only one arrow $\langle f, g \rangle : A + B \rightarrow C$ which commutes with the injections.

- The unique functor $\top : \textbf{Set} \rightarrow 1$ is a left adjoint to the functor $T : 1 \rightarrow \textbf{Set}$ defined by $T(\ast) = \{\ast\}$ and $T(1) = 1_{\{\ast\}}$, $\bullet$ and $1_{\bullet}$ being respectively the unique object and arrow of $1$, since $1(\bullet, \ast) \cong \textbf{Set}(C, \{\ast\})$, the two sets having both only one element.
1.1.5 Products

Products are ways to combine objects and arrows in a category. There are two types of combinations considered: the “mobocratic” tensor product and the “conservative” Cartesian product.

Tensor product

A tensor product is a rule to combine objects and arrows which is “natural,” that is independent of the particular shape or essence of the objects to which it is applied. This naturality property suggests, of course, the notion of functor.

Definition 1.6 (Tensor product). Given a category $\mathcal{C}$, a tensor product on $\mathcal{C}$ is a bifunctor $\otimes: \mathcal{C} \times \mathcal{C} \to \mathcal{C}$.

If there exists an object $I$ in $\mathcal{C}$ such that $A \otimes I \cong A$ and $I \otimes A \cong A$ for each object $A$, $I$ is called the unit of the tensor product.

Cartesian product

The Cartesian product is a rule to combine objects and arrows which preserves the information carried by the original objects.

Definition 1.7 (Cartesian product). more briefly a product on $\mathcal{C}$) is a bifunctor $\times$ on $\mathcal{C}$ which fulfils the following requirement: for each pair of objects $A$ and $B$ in $\mathcal{C}$ there exist arrows $\pi_{A B}^A: A \times B \to A$ and $\pi_{A B}^B: A \times B \to B$ such that for each object $C$ in $\mathcal{C}$ and for each pair of arrows $f: C \to A$ and $g: C \to B$ there exists a unique arrow $\langle f, g \rangle: C \to A \times B$ s.t. the following diagram commutes:

\[
\begin{array}{ccc}
A & \xrightarrow{\pi_{A B}^A} & A \times B \\
\downarrow{f} & & \downarrow{\langle f, g \rangle} \\
C & \xrightarrow{\pi_{A B}^B} & B
\end{array}
\]

Given $f: A \to B$ and $g: A' \to B'$ in $\mathcal{C}$, $f \times g: A \times A' \to B \times B'$ is defined by $f \times g = \langle f \circ \pi_{A B}^A, g \circ \pi_{A' B'}^{A'} \rangle$. A Cartesian product is unique up to isomorphisms. With the notation $A^n$ it is intended the $n$-times product $(\cdots (A \times A) \times A) \cdots \times A)$. Moreover observe the following equations:

- $\langle f, g \rangle \circ h = \langle f \circ h, g \circ h \rangle$
- $(f \times g) \circ \langle h, k \rangle = \langle f \circ h, g \circ k \rangle$

Definition 1.8 (Terminal and initial objects). A terminal object (initial object) in a category $\mathcal{C}$ is an object $T$ (?V) such that for each object $A$ of $\mathcal{C}$ there is a unique arrow $!_A: A \to T$ (?A: V \to A).

A category may, of course, not have a terminal or an initial object. The category Rel has both. The dual categorical notion to that of product is the co-product.
Definition 1.9 (Co-product). Given a category $C$, a co-product on $C$ is a bifunctor $+ : C \times C \to C$ which satisfies the following requirement: for each pair of objects $A$ and $B$ in $C$ there exist arrows $\text{in}_{l}^{A,B} : A \to A + B$ and $\text{in}_{r}^{A,B} : B \to A + B$ such that for every object $C$ in $C$ and for each pair of arrows $f : A \to C$ and $g : B \to C$ there exists a unique arrow $[f,g] : A + B \to C$ s.t. the following diagram commutes:

![Diagram](image)

1.1.6 Limits and colimits

In the denotational semantics of programming languages, it is often easy to face the problem of defining infinite objects, either infinite data types or infinite functionals. Very often the problem amounts to give a recursive definition

$$ X \cong F(X) \quad (1.1) $$

where, in a categorical setting, $X$ ranges over the objects of a category $C$ and $F : C \to C$ is an endofunctor of that category. Solutions to the equation 1.1 are fixed points of the functor $F$. They are usually obtained by successive approximations with a limit construction process.

It is to emphasize that in category theory, in general, it is not sufficient to say that two objects are isomorphic but we also need to specify how they are isomorphic. While a fixed point for a function $f : A \to A$ is simply an element $a \in A$ such that $f(a) = a$ (we reason up to equality), in the categorical setting a fixed point for a functor $F$ is a pair $\langle A, \alpha \rangle$ where $\alpha : FA \cong A$.

Definition 1.10 (Functor category). Let $C$ and $D$ be two categories. The functor category $[C,D]$ or $D^{C}$ is the category where objects are functors $F : C \to D$, arrows are natural transformations $\eta : F \to G$ and the composition $\mu \circ \eta$ of two natural transformations $\eta : F \to G$ and $\mu : G \to H$ assigns to each object $A$ of $C$ the arrow $\mu_{A} \circ \eta_{A}$ of $D$.

Definition 1.11 (Diagonal functor). Let $J$ and $C$ be two categories. The diagonal functor $\Delta : C \to C^{J}$ maps each object $A$ of $C$ to the constant functor $\mathcal{A}$ and each arrow $f : A \to B$ to the natural transformation $\eta^{f} : \mathcal{A} \to \mathcal{B}$ which assigns to each object $D$ of $C$ the arrow $f$.

Definition 1.12 (Limit of a functor). Let $F : J \to C$ be a functor, where $J$ is a small category (the class of its objects is a set). Let $\Delta : C \to C^{J}$ be the diagonal functor. A limit for $F$ in $C$ is a terminal object in the over-cone category $\langle \Delta \downarrow F \rangle$.

Let us remember that objects in the over-cone category $\langle \Delta \downarrow F \rangle$ are pairs $\langle A, \eta \rangle$, where $A$ is an object of $C$ and $\eta : \mathcal{A} \to F$ is a natural transformation, and arrows $f^{\sharp} : \langle A, \eta \rangle \to \langle B, \mu \rangle$ are arrows $f : A \to B$ of $C$ such that

![Diagram](image)
commutes, where $\eta^f$ is the natural transformation which assigns to each object $C$ of $\mathcal{C}$ the arrow $f$. A limit for the functor $F$ is then a pair $(T, \nu)$, $T$ an object of $\mathcal{C}$ and $\nu : T \rightarrow F$, such that, for each object $A$ of $\mathcal{C}$, there is a unique natural transformation $\eta^A : A \rightarrow T$ such that the following diagram commutes:

$$
\begin{array}{ccc}
A & \xrightarrow{\eta^A} & T \\
\downarrow{\eta} & & \downarrow{\nu} \\
F & & \end{array}
$$

Alternatively, a limit for a functor $F : \mathcal{J} \rightarrow \mathcal{C}$ can be seen as a right adjoint $\text{lim} : \mathcal{C}^\mathcal{J} \rightarrow \mathcal{C}$ to the functor $\Delta : \mathcal{C} \rightarrow \mathcal{C}^\mathcal{J}$, which means that for each object $A$ of $\mathcal{C}$ and each functor $F : \mathcal{J} \rightarrow \mathcal{C}$ there is a natural bijection

$$
\eta_{A,F} : \mathcal{C}^\mathcal{J}(\Delta A, F) \cong \mathcal{C}(A, \text{lim} F)
$$

(1.2)

between the set of natural transformations in $\mathcal{C}^\mathcal{J}$ between the constant functor $A$ and $F$, and the set of arrows in $\mathcal{C}$ between $A$ and $\text{lim} F$. If $(\text{lim} F, \nu)$ is a limit of the functor $F : \mathcal{J} \rightarrow \mathcal{C}$, the natural transformation $\nu : \text{lim} F \rightarrow F$ originates, for each arrow $f : B \rightarrow C$ of $\mathcal{C}$, a diagram

$$
\begin{array}{ccc}
\text{lim} F & \xrightarrow{\nu} & \text{lim} F \\
\downarrow{\nu_B} & & \downarrow{\nu_C} \\
FB & \xrightarrow{F(f)} & FC \\
\end{array}
= 
\begin{array}{ccc}
\text{lim} F & \xrightarrow{\nu} & \text{lim} F \\
\downarrow{\nu_B} & & \downarrow{\nu_C} \\
FB & \xrightarrow{F(f)} & FC \\
\end{array}
$$

which is called cone and where $\text{lim} F$ is called the vertex. Let us observe, in fact, that the natural transformation $\nu$ can be characterized by the set of arrows $\{\nu_J | J \text{ object of } \mathcal{J}\}$ and that the above diagram expresses the requirements for $\nu$ to be a natural transformation. Equation 1.2 tells us that for each natural transformation $\eta : A \rightarrow \text{lim} F$, that is for each cone

$$
\begin{array}{ccc}
A & \xrightarrow{\eta_B} & FB \\
\downarrow{\eta} & & \downarrow{\nu_B} \\
F & \xrightarrow{F(f)} & FC \\
\end{array}
$$

with vertex $A$, there is a unique arrow $h : A \rightarrow \text{lim} F$ in $\mathcal{C}$ such that the following diagram commutes:

$$
\begin{array}{ccc}
A & \xrightarrow{h} & \text{lim} F \\
\downarrow{\eta_B} & & \downarrow{\nu_B} \\
FB & \xrightarrow{F(f)} & FC \\
\end{array}
$$

$$
\begin{array}{ccc}
A & \xrightarrow{\eta_C} & FC \\
\downarrow{\eta} & & \downarrow{\nu_C} \\
FB & \xrightarrow{F(f)} & FC \\
\end{array}
$$
such a consideration allows us to look at $\lim F$ as the least upper bound of the vertexes of the cones generated by $F$. This is the intuitive idea which underlies the notion of categorical limit. A limit for a functor $F$ is also called inverse limit.

The colimit (or direct limit) is defined symmetrically.

**Definition 1.13 (Colimit of a functor).** Let $F : \mathcal{J} \to \mathcal{C}$ be a functor, where $\mathcal{J}$ is a small category. Let $\Delta : \mathcal{C} \to \mathcal{C}^\mathcal{J}$ be the diagonal functor. A colimit for $F$ in $\mathcal{C}$ is an initial object in the under-cone category $(F \downarrow \Delta)$.

All the considerations made after Definition 1.12 extends, dualized, to the colimit. The colimit of a functor $F$ is usually indicated as $\lim F$. Observe that a colimit $(\lim F, \rho)$ of a functor $F : \mathcal{J} \to \mathcal{C}$ has the universal property that, for each object $(A, \eta)$ of $(F \downarrow \Delta)$, there is a unique arrow $k : \lim F \to A$ in $\mathcal{C}$ such that the following diagram commutes:

\[
\begin{array}{c}
A \\
\downarrow \eta_B \\
\downarrow \eta_C \\
\downarrow \rho_B \\
\downarrow \rho_C \\
F B \\
\rightarrow F(f) \\
\rightarrow FC
\end{array}
\]

This may suggest us the intuitive notion of colimit as a greatest lower bound to the vertexes of the under-cones generated by $F$.

**Definition 1.14 (Complete and co-complete categories).** A category $\mathcal{C}$ is said complete (co-complete) if each functor $F : \mathcal{J} \to \mathcal{C}$, with $\mathcal{J}$ a small category, has a limit (colimit).

**Examples of Limits and Colimits**

- **Colimits in $\text{Set}$.** Let $\omega$ denote the category \{0 \rightarrow 1 \rightarrow 2 \rightarrow \cdots\} where the compositions of the arrows and the identity arrows are omitted. A functor $F : \omega \to \mathcal{C}$ determines a diagram

\[
D_0 = F 0 \xrightarrow{f_0} D_1 = F 1 \xrightarrow{f_1} D_2 = F 2 \xrightarrow{f_2} \cdots
\]

which is called $\omega$-chain and is also indicated as $(D_0, f_1)$. A limit for $F$ exists and is $D_0$:

\[
\begin{array}{c}
D_0 \\
\downarrow f_0 \\
\downarrow f_1 \\
\downarrow f_2 \\
\downarrow \cdots
\end{array}
\]

where $\nu_0 = 1_{D_0}, \nu_1 = f_0 \circ \nu_0, \nu_2 = f_1 \circ \nu_1, \ldots$. Observe, in fact, that each $\nu_{n+1}$ is completely determined by $\nu_n$. If $(C, \mu)$ is another vertex for the diagram, there is a unique arrow, $\mu_0 : C \to D_0$, such that $\nu_n \circ \mu_0 = \mu_n$. 
• A functor $F : \omega \to \mathbf{Set}$, which maps each arrow of $\omega$ to an inclusion function in $\mathbf{Set}$, yields a chain $F_0 \subseteq F_1 \subseteq F_2 \subseteq \cdots$. A colimit $\lim F$ for $F$ is given by $\lim F = \bigcup_{i \geq 0} F_i$ and $\nu = \{ \nu_i : F_i \to \lim F \}$ is the inclusion function.

• Fixed points. Let $\omega^{op}$ be the opposite category of $\omega$, that is

$$\omega^{op} = \{ 0 \leftarrow 1 \leftarrow 2 \leftarrow 3 \leftarrow \cdots \}$$

Let $G : C \to C$ be a functor. Let $A_0$ be an object of $C$, $f_0 : G(A_0) \to A_0$ be an arrow of $C$ and let $F : \omega^{op} \to C$ be the functor defined as follows:

- $F(n) = G^n(A_0)$ where $G^0(A_0) = A_0$
- $F(f) = \begin{cases} 1_{G^n(A_0)} & \text{if } f : n \to n \\ G^{m-1}(f_0) \circ \cdots \circ G^{n+1}(f_0) \circ G^n(f_0) & \text{if } f : m \to n, m > n \end{cases}$

A limit of $F$ is characterized by the following diagram:

$$\begin{array}{c}
A_0 \xleftarrow{f_0} A_1 = G(A_0) \xrightarrow{G(f_0)} A_2 = G(A_1) \xrightarrow{G(f_0)} \cdots \\
\end{array}$$

Observe that also

$$\begin{array}{c}
A_0 \xleftarrow{f_0 \circ G(\nu_0)} A_1 = G(A_0) \xrightarrow{G(\nu_0)} A_2 = G(A_1) \xrightarrow{G(\nu_0)} \cdots \\
\end{array}$$

is commutative. If $G$ preserves the limits—maps objects which are limits to objects which still are limits—$G(\lim F)$ is still a limit, and since a limit is unique up to isomorphisms, $\lim F \cong G(\lim F)$, that is $\lim F$ is a fixed point of $G$.

Fixed points of functors can be obtained as a limit (colimit) construction. Stoy and Plotkin [SP82] gave sufficient conditions for a general functor to have fixed points. In the setting of game semantics, recursive objects are studied in [AM95b], where sufficient conditions for the existence are stated.

**Definition 1.15 (Fixed points).** Let $C$ be a category and $F : C \to C$ an endofuctor. A fixed point of $F$ is a pair $(A, \alpha)$, where $A$ is an object of $C$ and $\alpha : FA \cong A$ is an isomorphism.

**Definition 1.16 (Continuous functors).** A functor $F : C \to D$ is continuous (cocontinuous) if it preserves limits (colimits).

**Theorem 1.17 ([SP82]).** Each co-continuous functor $F : C \to C$ on a co-complete category $C$ admits a fixed point $(\lim F, \alpha)$, where $\lim F$ is the colimit of the $\omega$-chain $(F^n(V), F^n(\nabla_V))$, $V$ is the initial object of $C$, $\nabla_V$ is the unique morphism in $C(V, FV)$ and $\alpha : \lim F \cong F(\lim F)$.

Symmetrically, each continuous functor $F : C \to C$ on a complete category $C$ has a fixed point $(\lim F, \beta)$, where $\lim F$ is the limit of the $\omega$-chain $(F^n(T), F^n(\nabla_T))$, $T$ is the terminal object of $C$, $\nabla_T$ is the unique morphism in $C(FT, T)$ and $\alpha : F(\lim F) \cong \lim F$. 


1.2 Categories with product

Categorical denotational semantics of logic and programming languages consists in an association between terms of the language and arrows of the interpreting category. The categorical translation of typed languages is best understood following the typed view of categories. In such a framework, each arrow $f : A \to B$ has got its “source type” $A$ and its “target type” $B$. If the interpreting category is equipped with a product, a program who receives more than one argument can be interpreted by an arrow with source type the product of the types (objects) of the single arguments. This introduces the relevance of categories with product as frameworks for the categorical semantics of programming languages. If the product (as a combination operation on the objects) fulfills more properties (yields a monoidal structure, for instance), the resulting category is more interesting since enjoys pleasant additional properties.

1.2.1 Monoidal categories

Monoidal categories are monoids defined on categories rather than sets. This, of course, implies a more involved definition since a category is more complicated than a set. The main difficulty resides in the fact that, in category theory, objects are usually consider up to isomorphisms. For the rest, the definition of monoidal category is an aping of the definition of monoid in the context of categories: the binary operation $\otimes : M \times M \to M$ becomes a bifunctor $\otimes : C \times C \to C$ and the unit element becomes an object of $C$.

**Definition 1.18 (Monoidal category).** A structure $\mathcal{M} = (\mathcal{C}, \otimes, I, \alpha, \lambda, \rho)$ is a monoidal category when $\mathcal{C}$ is a category, $\otimes : C \times C \to C$ is a functor (called monoidal or tensor product), $I \in C$ is an object (the unit object) and

\[
\begin{align*}
\alpha_{A,B,C} & : (A \otimes B) \otimes C \cong A \otimes (B \otimes C) \\
\lambda_A & : I \otimes A \cong A \\
\rho_A & : A \otimes I \cong A
\end{align*}
\]

are natural isomorphisms such that, for every objects $A, B, C, D \in C$, the following diagrams commute:

\[
\begin{array}{ccc}
((A \otimes B) \otimes C) \otimes D & \xrightarrow{\alpha_{A,B,C,D}} & (A \otimes B) \otimes (C \otimes D) \\
(A \otimes (B \otimes C)) \otimes D & \xrightarrow{\alpha_{A,B,C,D} \otimes 1_D} & A \otimes ((B \otimes C) \otimes D) & \xrightarrow{1_A \otimes \alpha_{B,C,D}} & A \otimes (B \otimes (C \otimes D)) \\
(A \otimes I) \otimes B & \xrightarrow{\alpha_{A,I,B}} & A \otimes (I \otimes B) & \xrightarrow{\rho_A \otimes 1_B} & A \otimes B & \xrightarrow{1_A \otimes \lambda_B} & A \otimes (I \otimes B) \\
\lambda_I & \xrightarrow{\rho_I} & I \otimes I \cong I
\end{array}
\]

In category theory we usually reason about mathematical objects “up to isomorphism”. Of course, we want to think at isomorphic mathematical objects as being the same objects. Category theory tells us that, in general, we are not allowed to do this if we do not consider
the “right” isomorphism. In fact, only isomorphisms which satisfy “coherent conditions” should be considered. The diagrams of Definition 1.18 are coherence conditions diagrams. This means that the commutativity of those diagrams is a sufficient condition to state that all diagrams built up from $\otimes$ with occurrences of $\alpha, \lambda$ and $\rho$ commute. More clearly this means that, considering an isomorphism which satisfies the conditions of Definition 1.18, we can coherently think at isomorphic objects always as being the same.

**Definition 1.19 (Strict monoidal category).** A strict monoidal category $\mathcal{M}$ is a monoidal category in which all the natural isomorphisms $\alpha_{A,B,C}$, $\lambda_A$ and $\rho_A$ are identities for every objects $A, B, C$ of $\mathcal{M}$.

**Example 1.20.** The category $[\mathcal{C}, \mathcal{C}]$ of endofunctors on $\mathcal{C}$ and natural transformations between them, with the composition of functors as tensor product is a strict monoidal category.

The commutativity of the tensor product gives rise to the notion of symmetric monoidal category.

**Definition 1.21 (Symmetric monoidal category).** A symmetric monoidal category is a monoidal category equipped with a natural isomorphism $\gamma_{A,B} : A \otimes B \cong B \otimes A$ such that the following diagrams commute:

![Diagram](attachment:image.png)

A strict symmetric monoidal category is a strict monoidal category equipped with a natural isomorphism $\gamma_{A,B} : A \otimes B \cong B \otimes A$ which satisfies the diagrams of the previous definition. Notice that in a strict symmetric monoidal category we do not require that $A \otimes B = B \otimes A$.

**Example 1.22.** The category of abelian groups and homomorphisms with the tensor product of abelian groups is a symmetric monoidal category in which the unit object is $\mathbb{Z}$.

**Definition 1.23 (Monoidal functor $[\mathcal{E}, \mathcal{F}]$).** A monoidal functor from a monoidal category $\langle \mathcal{C}, \otimes, I, \alpha, \lambda, \rho \rangle$ to a monoidal category $\langle \mathcal{D}, \otimes', I', \lambda', \rho' \rangle$ is a triple $\langle F, \varphi, \varphi_1 \rangle$ where $F : \mathcal{C} \to \mathcal{D}$ is a functor, $\varphi : F(-) \otimes F(-) \to F(- \otimes -)$ is a natural transformation and $\varphi_1 \in \mathcal{D}(I', FI)$ is an arrow such that the following diagrams commute:

![Diagram](attachment:image.png)
Definition 1.24 (Symmetric monoidal functor). A monoidal functor \( \langle F, \varphi, \varphi_I \rangle \) from a symmetric monoidal category \( \langle C, \otimes, I, \alpha, \lambda, \rho, \gamma \rangle \) to a smc \( \langle D, \otimes', I', \alpha', \lambda', \rho', \gamma' \rangle \) is symmetric if the following diagram commutes:

\[
\begin{array}{ccc}
FA \otimes' FA & \xrightarrow{\lambda} & FA \\
\downarrow{\varphi_I \otimes' 1_{FA}} & & \downarrow{F(\lambda)} \\
F(1 \otimes A) & \xrightarrow{1_{FA} \otimes' \varphi_I} & F(A \otimes I)
\end{array}
\]

\[
\begin{array}{ccc}
FA \otimes' FI & \xrightarrow{\rho'} & FA \\
\downarrow{\varphi_{FI,FA}} & & \downarrow{F(\rho)} \\
F(A \otimes I) & \xrightarrow{\varphi_{FA,FI}} & F(A \otimes I)
\end{array}
\]

A monoidal functor \( \langle F, \varphi, \varphi_I \rangle \) is called strong if \( \varphi \) is a natural isomorphism and \( \varphi_I \) is an isomorphism. It is called strict if all components of \( \varphi \) and \( \varphi_I \) are identity arrows.

Definition 1.25 (Monoidal natural transformation [EK65]).
Given two monoidal functors \( \langle F, \varphi, \varphi_I \rangle \) and \( \langle G, \psi, \psi_I \rangle \) with the same source and target monoidal categories, a monoidal natural transformation from \( \langle F, \varphi, \varphi_I \rangle \) to \( \langle G, \psi, \psi_I \rangle \) is a natural transformation \( \eta : F \rightarrow G \) such that the following diagrams commute:

\[
\begin{array}{ccc}
FA \otimes' FB & \xrightarrow{\varphi} & F(A \otimes B) \\
\downarrow{\eta_A \otimes \eta_B} & & \downarrow{\eta_{A \otimes B}} \\
GA \otimes' GB & \xrightarrow{\psi} & G(A \otimes B)
\end{array}
\]

\[
\begin{array}{ccc}
F(I) & \xrightarrow{\varphi_I} & G(I) \\
\downarrow{\eta_I} & & \downarrow{\psi_I} \\
F(I) & \xrightarrow{\psi_I} & G(I)
\end{array}
\]

1.2.2 Cartesian closed categories

From a tensor product \( \otimes : C \times C \rightarrow C \) we can derive, for each object \( B \) of \( C \) a functor

\[- \otimes B : C \rightarrow C\]

defined by \((- \otimes B)(A) = A \otimes B\) for each object \( A \) in \( C \) and \((- \otimes B)(f) = f \otimes 1_B : A \otimes B \rightarrow A' \otimes B\) for each \( f : A \rightarrow A' \) in \( C \). This allows us to define the notion of symmetric monoidal closed category.

Definition 1.26 (Symmetric monoidal closed category). A category \( C \) is said symmetric monoidal closed if it is a symmetric monoidal category in which each functor \(- \otimes B\) has a right adjoint indicated as \( B \dashv -\), that is there is a natural isomorphism

\[\Lambda_{A,B,C} : C(A \otimes B, C) \cong C(A, B \rightarrow C)\]

for every objects \( A, B, C \).

A symmetric monoidal closed category is also called autonomous.
Example 1.27. Examples of symmetric monoidal closed categories are:

- \((\text{Set}, \times, \{\ast\})\): the category of sets and total functions with the direct product and the singleton set as unit object. The functor \(- \times B : \text{Set} \to \text{Set}\) has the right adjoint \(B \Rightarrow - : \text{Set} \to \text{Set}\).

- \((\text{Rel}, \times, \emptyset)\): the category \(\text{Rel}\) of sets and binary relations with the direct product and the empty set as unit object. The functor \(- \times B : \text{Rel} \to \text{Rel}\) has the right adjoint \(B \times - : \text{Rel} \to \text{Rel}\).

Definition 1.28 (Cartesian closed category). A Cartesian closed category is an autonomous category in which the tensor product is a Cartesian product.

More specifically, a Cartesian closed category (CCC for short) is a category \(C\) in which the bifunctor \(\times\) has a right adjoint indicated as \(\Rightarrow\) and for every objects \(A, B, C\) there exists a natural isomorphism

\[
\Lambda_{A,B,C} : C(A \times B, C) \cong C(A, B \Rightarrow C)
\]

The naturality of \(\Lambda\) implies, for each \(g : A \to D\) and \(h : D \times B \to C\), the validity of the following equation:

\[
\Lambda_{A,B,C}(h \circ (g \times 1_B)) = \Lambda_{D,B,C}(h) \circ g \quad (\Lambda_{\text{nat}})
\]

In fact, \(\Lambda : C(- \times -, -) \to C(-, - \Rightarrow -)\) is a natural transformation between two hom-functors. The naturality of \(\Lambda\) implies the naturality for each of the three arguments separately. Hence, fixed two of them, consider \(\Lambda_{B,C} : C(- \times B, C) \to C(-, B \Rightarrow C)\). Remember that for \(g : A \to D\)

\[
C(- \times B, C)(g) = h \quad \Rightarrow \quad h \circ (g \times 1_B)
\]

\[
C(-, B \Rightarrow C)(g) = h' \quad \Rightarrow \quad h' \circ g
\]

with \(h \in C(D \times B, C)\) and \(h' \in C(D, B \Rightarrow C)\). The following diagram:

\[
\begin{array}{ccc}
C(D \times B, C) & \xrightarrow{\Lambda_{A,B,C}(h \circ (g \times 1_B))} & C(A \times B, C) \\
\Lambda_{D,B,C} & & \Lambda_{A,B,C} \\
C(D, B \Rightarrow C) & \xrightarrow{\Lambda_{D,B,C}(h) \circ g} & C(A, B \Rightarrow C)
\end{array}
\]

commutes and then \(\Lambda_{A,B,C}(h \circ (g \times 1_B)) = \Lambda_{D,B,C}(h) \circ g\).

A direct consequence of equation \(\Lambda_{\text{nat}}\) is that, for every objects \(A\) and \(B\), there exists an evaluation map \(ev_{A,B} = \Lambda_{A,B,C}^{-1}(1_{A \Rightarrow B}, 1_{A \Rightarrow B})\), such that, for each \(f : A \times B \to C\), the following equation holds:

\[
ev_{B,C} \circ (\Lambda_{A,B,C}(f) \times 1_B) = f \quad (\beta_{\text{nat}})
\]

Notice, in fact, that

\[
\begin{align*}
\Lambda_{A,B,C}(ev_{B,C} \circ (\Lambda_{A,B,C}(f) \times 1_B)) &= \text{ by } \Lambda_{\text{nat}} \\
\Lambda_{B \Rightarrow C, B,C}(ev_{B,C}) \circ \Lambda_{A,B,C}(f) &= \text{ by definition of } ev_{B,C} \\
1_{B \Rightarrow C} \circ \Lambda_{A,B,C}(f) &= \text{ by definition of }\Lambda_{A,B,C}(f)
\end{align*}
\]
which implies the validity of $\beta_{\text{cat}}$ since $\Lambda_{A,B,C}$ is a bijection and hence

$$\Lambda_{A,B,C}(f) = \Lambda_{A,B,C}(g) \Rightarrow f = g$$

The name $\beta_{\text{cat}}$, which appears in [Mar92], reflects the very fact that it is this property which allows to define models of $\lambda$-calculus, that validate the $\beta$-rule, in a categorical setting.

Another instance of equation $\Lambda_{\text{nat}}$ is the following:

$$\Lambda_{A,B,C}(\text{ev}_{B,C} \circ (h \times 1_B)) = h$$

(\eta_{\text{cat}})

which reflects the fact that $\Lambda_{A,B,C}$ is an injective function, and hence left invertible, since equation $\eta_{\text{cat}}$ is valid only if

$$\Lambda_{B \Rightarrow C,B,C}(\text{ev}_{B,C}) = \Lambda_{B \Rightarrow C,B,C}(\Lambda_{B \Rightarrow C,B,C}^{-1}(1_{B \Rightarrow C})) = 1_{B \Rightarrow C}$$

**Definition 1.29 (Points in a CCC).** A point of an object $A$ of a CCC $C$ is a morphism $x : T \rightarrow A$. The set of points of $A$, $C(T,A)$, is indicated as $|A|$.

Category theory is a theory of functions in intension rather then a theory of functions in extension. This peculiarity is manifested by the following definition.

**Definition 1.30 (Enough points).** An object $A$ has enough points if

$$(\forall f,g : A \rightarrow B)(f \neq g \Rightarrow (\exists x \in |A|)(f \circ x \neq g \circ x))$$

The intensional character of category theory is evident for those objects $A$ which have not enough points. It would be the case that although $(\forall x \in |A|)(f \circ x = g \circ x)$ (with $f,g : A \rightarrow B$) $f \neq g$, while $f$ and $g$ exhibit the same extensional behavior if we regard the composition of $f$ with the point $x$ as the functional application $f(x)$.

In a CCC $C$ the relation $C(T \times A,B) \cong C(A,B) \cong C(T,A \Rightarrow B)$ witnesses the fact that the object $A \Rightarrow B$ represents the set of arrows $C(A,B)$ since there is a one-to-one correspondence between arrows $f : A \rightarrow B$ and points of the object $A \Rightarrow B$. For these considerations, a categorical model of $\lambda$-calculus in a CCC should be an object $D$ such that the object $D \Rightarrow D$, representing the set of arrows $f : D \rightarrow D$, is at least “embeddable” in $D$. Such an object is called reflexive.

**Definition 1.31 (Retract).** Given a category $C$, an object $A$ is a retract of an object $B$ if there are arrows $i : A \rightarrow B$ and $j : B \rightarrow A$ such that $j \circ i = 1_A$. To indicate that $A$ is a retract of $B$ via $i,j$ we shall write $(A \lhd B,i,j)$.

**Definition 1.32 (Reflexive object).** Given a CCC category $C$, a reflexive object $A$ is an object such that $A \Rightarrow A$ is a retract of $A$.

### 1.2.3 Traced monoidal categories

In the categorical denotational semantics of logic and programming languages, the execution of a program usually takes place in a given environment, which is described by a suitable object. For the purpose of the Geometry of Interaction [Gir88a, Abr96], which aims to give a semantics of computation which captures the dynamic aspects usually bypassed by traditional denotational semantics, it is intrinsic to consider some intensional aspects of the objects which model the semantical universe. In the traced monoidal categories, introduced in [JSV96], each arrow interpreting a term carries on the history of the interactions of the term with the environment, that is its trace.
As is custom [Has97, Abr96], we present the definition of traced symmetric monoidal category for the strict case (when the isomorphisms $\lambda$, $\rho$ and $\alpha$ are identities) to simplify the presentation without loss of generality since each symmetric monoidal closed category is equivalent to a strict one.

**Definition 1.33 (Traced symmetric monoidal category [JSV96]).** A symmetric monoidal category $(C, \otimes, 1, \gamma)$ is traced if for every objects $A, B, X$ of $C$ there are functions $\text{Tr}_X^{A,B} : C(A \otimes X, B \otimes X) \to C(A, B)$ such that the following conditions are satisfied:

- **Vanishing:**
  \[ \text{Tr}_A^B(f) = f \]
  where $f \in C(A,B)$ and
  \[ \text{Tr}_X^{A,B} \otimes Y(f) = \text{Tr}_A^B(\text{Tr}_X^{A,X,Y,B\otimes X}((A \otimes X \otimes Y, B \otimes X \otimes Y)) \]
  where $f \in C(A \otimes X \otimes Y, B \otimes X \otimes Y)$

- **Superposing:**
  \[ \text{Tr}_C^{A \otimes C \otimes B}(1_C \otimes f) = 1_C \otimes \text{Tr}_A^{B}(f) \]
  where $f \in C(A \otimes X \otimes B \otimes X)$

- **Yanking:**
  \[ \text{Tr}_X^{X,X}(\gamma X,X) = 1_X \]

- **Left Tightening (Naturality in $A$):**
  \[ \text{Tr}_A^{A,B}(f \circ (g \otimes 1_X)) = \text{Tr}_A^{A,B}(f) \circ g \]
  where $f \in C(A' \otimes X, B \otimes X)$ and $g \in C(A, A')$

- **Right Tightening (Naturality in $B$):**
  \[ \text{Tr}_A^{A,B}((g \otimes 1_X) \circ f) = g \circ \text{Tr}_A^{A,B}(f) \]
  where $f \in C(A \otimes X, B' \otimes X)$ and $g \in C(B', B)$

- **Sliding (Naturality in $X$):**
  \[ \text{Tr}_A^{A,B}((1_B \otimes g) \circ f) = \text{Tr}_A^{A,B}(f \circ (1_B \otimes g)) \]
  where $f \in C(A \otimes X, B \otimes X')$ and $g \in C(X', X)$

A related notion to that of traced symmetric monoidal category is that of compact closed category due to Kelly [Kel72, KL80].

**Definition 1.34 (Compact closed category).** A compact closed category is a symmetric monoidal category $(M, \otimes, 1, \alpha, \lambda, \rho, \gamma)$, with an endofunctor $(-)^* : M \to M$ and with two natural transformations $\eta : I \to - \otimes (-)^*$ and $\epsilon : (- \otimes (-)^*) \to I$ (where $I$ is the
constant endofunctor which maps each object \( A \) of \( C \) to the unit object \( I \) and each arrow \( f : A \to B \) to the identity arrow \( 1_f \) such that the following diagrams commute

\[
\begin{align*}
I \otimes A & \xrightarrow{\eta_A \otimes 1_A} (A \otimes A^*) \otimes A \\
& \xrightarrow{\alpha_{A,A^*}} A \otimes (A^* \otimes A) \\
& \xrightarrow{1_A \otimes \lambda_A} A \otimes I
\end{align*}
\]

\[
\begin{align*}
A^* \otimes I & \xrightarrow{1_{A^*} \otimes \eta_A} A^* \otimes (A^* \otimes A) \\
& \xrightarrow{\alpha_{A^*,A^*}^{-1}} (A^* \otimes A) \otimes A^* \\
& \xrightarrow{\epsilon_A \otimes 1_{A^*}} I \otimes A^*
\end{align*}
\]

The close structure is given by the bijection \( \varphi : \mathcal{M}(A \otimes B, C) \cong \mathcal{M}(A, B^* \otimes C) \). In [JSV96] it is shown that each compact closed category is traced and that each traced symmetric monoidal category can be fully and faithfully embedded into a compact closed category.

### 1.2.4 Monads

Monads are monoids defined over categories of functors. Precisely, given a category \( C \), let \([C, C]\) be the category of endofunctors on \( C \), with the tensor product given by composition of functors. This is a monoidal category. A monoid on \([C, C]\) is called monad over \( C \).

**Definition 1.35 (Monad).** Given a category \( C \), a monad \( \langle T, \eta, \mu \rangle \) over \( C \) consists of a functor \( T : C \to C \) and natural transformations \( \eta : I_C \to T \) and \( \mu : T^2 \to T \) such that the following diagrams commute for each object \( A \):

\[
\begin{align*}
T^2 A & \xrightarrow{T \mu_A} T A \\
& \xrightarrow{\mu A} T A
\end{align*}
\]

\[
\begin{align*}
T^3 A & \xrightarrow{T \eta_A} T^2 A \\
& \xrightarrow{T \mu_A} T A
\end{align*}
\]

The dual definition of co-monad will be extensively used in the referring category of games for the semantics of \( \lambda \)-calculus.

**Definition 1.36 (Co-monad).** Given a category \( C \), a co-monad \( \langle T, \eta, \mu \rangle \) over \( C \) consists of a functor \( T : C \to C \) and natural transformations \( \eta : T \to I_C \) and \( \mu : T \to T^2 \) such that the following diagrams commute for each object \( A \):

\[
\begin{align*}
T^3 A & \xleftarrow{T \eta_A} T^2 A \\
& \xleftarrow{T \mu_A} T A
\end{align*}
\]

\[
\begin{align*}
T^3 A & \xrightarrow{T \eta_A} T^2 A \\
& \xrightarrow{T \mu_A} T A
\end{align*}
\]
Examples of Monads

In [Mog91] several examples of monads on the category $[\text{Set}, \text{Set}]$ are introduced. The aim is that of capturing suitable notions of computation.

- $\langle T, \eta, \mu \rangle$ with
  - $T : \text{Set} \to \text{Set}$ defined by
    \[
    \begin{cases}
    T(A) = A + \{\bot\} & A \text{ a set} \\
    T(f) = [f, 1_{A}] & f : A \to B
    \end{cases}
    \]
  - $\eta_A : A \to A + \{\bot\}$ defined by $a \in A \mapsto \eta_l(a)$
  - $\mu_A : (A + \{\bot\}) + \{\bot\} \to A + \{\bot\}$ defined by
    \[
    \begin{cases}
    \eta_r(\bot) \mapsto \eta_r(\bot) \\
    \eta_l(\eta_r(\bot)) \mapsto \eta_l(\bot) \\
    \eta_l(\eta_l(a)) \mapsto \eta_l(a)
    \end{cases}
    \]
  is a monad – the partiality monad.

- $\langle T, \eta, \mu \rangle$ with
  - $T : \text{Set} \to \text{Set}$ defined by
    \[
    \begin{cases}
    T(A) = \mathcal{P}^{<\omega}(A) & A \text{ a set} \\
    T(f) = X \subseteq A \mapsto \{f(x) \mid x \in X\} & f : A \to B
    \end{cases}
    \]
  - $\eta_A : A \to \mathcal{P}^{<\omega}(A)$ defined by $a \in A \mapsto \{a\}$
  - $\mu_A : \mathcal{P}^{<\omega}(\mathcal{P}^{<\omega}(A)) \to \mathcal{P}^{<\omega}(A)$ defined by $Y \mapsto \bigcup \{X \subseteq A \mid X \in Y\}$
  is a monad – the nondeterminism monad.

- $\langle T, \eta, \mu \rangle$ with, given a set of states $S$,
  - $T : \text{Set} \to \text{Set}$ defined by
    \[
    \begin{cases}
    T(A) = S \Rightarrow (A \times S) & A \text{ a set} \\
    T(f) = g \in S \Rightarrow (A \times S) \mapsto (s \mapsto (f(\pi_1 \circ g(s)), s)) & f : A \to B
    \end{cases}
    \]
  - $\eta_A : A \Rightarrow S \Rightarrow (A \times S)$ defined by $a \in A \mapsto (s \in S \mapsto \langle a, s \rangle)$
  - $\mu_A : S \Rightarrow ((S \Rightarrow (A \times S)) \times S) \Rightarrow S \Rightarrow (A \times S)$
    defined by $f \in S \Rightarrow ((S \Rightarrow (A \times S)) \times S) \mapsto (s \in S \mapsto \pi_1 \circ f(s))$
  is a monad – the side-effects monad.

The importance of monads resides primarily in their connection with adjoint functors. For our purposes, however, we shall use a property of monads quite different. Provided we have a monoidal closed category $\mathcal{C}$ and a monad $\langle T, \eta, \mu \rangle$ over it, we can obtain a Cartesian closed category $K_T(\mathcal{C})$. The construction is due to Kleisli [Kle65] and can be also found in [Mac71, Mog91].

**Theorem 1.37 (Kleisli category over a monad).** Given a monad $\langle T, \eta, \mu \rangle$ over a category $\mathcal{C}$, consider for each object $A$ in $\mathcal{C}$ a new object $A^\sharp$ and for each arrow $f : A \to TB$ in $\mathcal{C}$ a new arrow $f^\sharp : A^\sharp \to B^\sharp$. If we define the composition of $f^\sharp$ with $g^\sharp : B^\sharp \to C^\sharp$ as $g^\sharp \circ f^\sharp \triangleq (\mu_C \circ T(g) \circ f)^\sharp$ we get a new category that we indicate as $K_T(\mathcal{C})$. 
Lambda Calculus Issues

"However, I recall seeing in Göttingen in 1928, but not comprehending, an uncirculated manuscript containing λ’s which was doubtless a progenitor of λ-calculus."

H. B. Curry [Cur80], p. 88

Abstract

Lambda calculus is a well-established formalism in the computer science community. The main comprehensive reference remains Barendregt’s book [Bar84]. In this chapter some peculiar aspects of λ-calculus will be introduced. We start by introducing some results on the λ-theories, deeply investigated in [HR92, HL95, HL99, Len97]. The trees of terms we shall utilize are introduced in the following section. Labelled λ-calculus, as the main tool in the proof of the approximation theorem of Chapter 3, constitutes the last section.

The main purpose for the introduction of λ-calculus [Chu32, Chu33] was the study of the concept of function intended as an “operation” or a “transformation”, i.e. a rule of correspondence by which an argument is given and a value is obtained [Chu41, Bar84]. A function is not applicable, in general, to whatsoever it has got a range of possible arguments. It is possible that a range of arguments for a function consists of other functions and it is even possible that a function should be applied to itself. The same objects can be, at the same time, functions and arguments. λ-calculus has a widespread sphere of employments. Refer to [Bar84] and [Bar97] for a survey of its applications. In this thesis we shall mainly consider untyped λ-calculus and its fruitful usage as a metalanguage for the study of properties of programs.

This issue has not only theoretic import but broad practical usage [App92]. λ-notation is, in fact, largely used in the compiling process as an intermediate representation between an high level programming language and its machine-dependent translation, to deeply analyze the program and generate optimized code. This, of course, has a great pith nowadays when applicative software is still becoming more and more sophisticated and resource-consuming.
In the study of properties of programs, the main notion to consider on \( \lambda \)-terms is \textit{observational equivalence}. It arises considering programs as \textit{black boxes} and take them to be equivalent if we cannot tell them apart by \textit{observing} that, for a given program context, the computation halts successfully when one is used as a subprogram but does not halt when the other is used as a subprogram.

There are mainly two roads in establishing observational equivalences on terms. The first consists in establishing an \textit{operational semantics} of \( \lambda \)-calculus by specifying a reduction strategy, that is a procedure for determining, for each \( \lambda \)-term \( M \), a particular redex to be contracted. A reduction strategy \( \rightsquigarrow_\sigma \) determines an \textit{evaluation relation} \( \Downarrow_\sigma \) on \( \lambda \)-terms which allows to define an observational equivalence \( \approx_\sigma \):

\[
M \approx_\sigma N \iff (\forall C[\ ])(C[M], C[N] \in \Lambda^0 \Rightarrow (C[M] \Downarrow_\sigma \equiv C[N] \Downarrow_\sigma))
\]

Such a relation is a \textit{congruence}, that is an equivalence relation compatible with the syntax of the calculus. Showing that two terms are observationally equivalent by induction on the computational steps can be, however, a lengthy and error-prone activity. For this reason, semantical techniques are called. Given a model \( M \) of \( \lambda \)-calculus and an interpretation function \( [\ ] : \Lambda \to M \), it induces an equivalence relation \( \approx_M \) defined as:

\[
M \approx_M N \iff [M] = [N]
\]

which is also a congruence. If it coincides with the observational equivalence induced by an operational semantics, we say that the model is \textit{fully abstract} with respect to the operational semantics. A fully-abstract model provides the same amount of information on the observational equivalence as the operational semantics does, and hence can be used to study all the properties of the relation. It is not always easy to build a fully-abstract model for a given observational equivalence. In this thesis we aim to carry out this duty in the semantical setting of game semantics.

### 2.1 Lambda theories

\( \lambda \)-theories are congruences on \( \lambda \)-terms which extend \( \beta \)-conversion and aim to represent theories of programs. As already mentioned above, they can be obtained operationally by choosing a notion of reduction or as a semantical quotient imposed by a computationally adequate model of \( \lambda \)-calculus. A \textit{notion of reduction} \( \mathcal{R} \) is a binary relation on \( \Lambda \). The basic notion of reduction is \( \beta \), defined as \( \{((\lambda x.M)N, M[N/x]) \} \subseteq \Lambda \times \Lambda \). Another widespread used notion of reduction is \( \eta \) defined as \( \{((\lambda x.Mx, M) \mid M \in \Lambda \land x \notin FV(M)) \} \subseteq \Lambda \times \Lambda \). Each notion of reduction induces different equivalences on \( \lambda \)-terms.

**Definition 2.1.** A notion of reduction \( \mathcal{R} \) induces the following relations:

- the \textit{compatible closure} of \( \mathcal{R} \) denoted as \( \rightarrow_\mathcal{R} \), which is called \textit{small step reduction} defined by
  - \( (M, N) \in \mathcal{R} \iff M \rightarrow_\mathcal{R} N \);  
  - \( M \rightarrow_\mathcal{R} N \iff PM \rightarrow_\mathcal{R} PN \);  
  - \( M \rightarrow_\mathcal{R} N \iff MP \rightarrow_\mathcal{R} NP \);  
  - \( M \rightarrow_\mathcal{R} N \iff \lambda x.M \rightarrow_\mathcal{R} \lambda x.N \)

for \( M, N, P \in \Lambda \) and \( x \in \text{Var} \).
• the reflexive and transitive closure $\rightarrow^*_R$ of $\rightarrow_R$, which is called big step reduction or simply reduction, defined by

- $M \rightarrow_R N \implies M \rightarrow^*_R N$;
- $M \rightarrow^*_R M$;
- $M \rightarrow^*_R N \& N \rightarrow^*_R P \implies M \rightarrow^*_R P$

for $M,N,P \in \Lambda$

• the $\mathcal{R}$-convertibility $=^\mathcal{R}$ defined as the equivalence relation generated by $\rightarrow^*_R$ also called $\mathcal{R}$-congruence:

- $M \rightarrow^*_R N \implies M =^\mathcal{R} N$;
- $M =^\mathcal{R} N \implies N =^\mathcal{R} M$;
- $M =^\mathcal{R} N \& N =^\mathcal{R} P \implies M =^\mathcal{R} P$

The relation $\rightarrow^*_R$ is indicated also as $\rightarrow_R$. Given a reduction relation $\rightarrow^*_R$ and a term $M$, it is possible to build a (non-deterministic) path $M_1 \rightarrow^*_R \cdots \rightarrow^*_R M_n$ where $M_1 \equiv M$. At each step, different redexes can be contracted. A procedure to determine what redex to contract is termed reduction strategy.

A (possibly non-deterministic) reduction strategy can be formalized as a relation $\leadsto_\sigma \subseteq \Lambda \times \Lambda$ such that if $M \leadsto_\sigma N$ (which stays for $(M,N) \in \leadsto_\sigma$) then $N$ is a possible result of applying the reduction strategy $\leadsto_\sigma$ to $M$. Reduction paths are defined by repeatedly (possibly zero times) applying the reduction strategy. The set of terms which do not belong to the domain of $\leadsto_\sigma$ are partitioned into two disjoint sets: the set of $\sigma$-values $\text{Val}_\sigma$ and the set of $\sigma$-deadlocks. The first is the set of terms which can be considered as the satisfactory results of a terminating reduction path while the $\sigma$-deadlocks are terms which should be considered meaningless.

Given a reduction strategy $\leadsto_\sigma$, we can define the evaluation relation $\downarrow_\sigma: \Lambda \times \Lambda$ such that $M \downarrow_\sigma$ holds if and only if there exists a reduction path leading from $M$ to $N$ and $N$ is a $\sigma$-value. If there exists $N$ such that $M \downarrow_\sigma N$ we say that $\rightarrow_\sigma$ terminates on $M$, otherwise we say that $\rightarrow_\sigma$ diverges on $M$ – or reaches a deadlock – and write $M \not\downarrow_\sigma$.

The pair $(\downarrow_\sigma, \text{Val}_\sigma)$ defines an operational semantics for $\lambda$-calculus from which we can derive an observational equivalence on the $\lambda$-terms.

**Definition 2.2 (Observational equivalence).** Let $\leadsto_\sigma$ be a reduction strategy and $M,N \in \Lambda^0$ be two closed terms. The observational equivalence relation $\approx_\sigma$ induced by $\leadsto_\sigma$ is defined as

$M \approx_\sigma N \iff (\forall C[\ ])(C[M], C[N] \in \Lambda^0 \Rightarrow (C[M] \downarrow_\sigma \Leftrightarrow C[N] \downarrow_\sigma))$

**Definition 2.3 (Lambda theory).** A $\lambda$-theory is a congruence on $\lambda$-terms that extends $\beta$-conversion. It is consistent if there exists terms $M,N \in \Lambda$ which are not congruent.

**Examples of $\lambda$-theories**

Here are some examples of $\lambda$-theories induced by reduction strategies. See also [Lengrand, Hirsbrunner, Louden 1995] for extensive references.
• Lazy call-by-name. The lazy call-by-name reduction strategy \( \sim_\ell \subseteq \Lambda^0 \times \Lambda^0 \), is the strategy which reduces the leftmost redex not inside a \( \lambda \)-abstraction. It is the strategy usually implemented by the lazy functional languages. Its observational equivalence was studied in [AO93, Ong88]. \( \text{Val}_\ell = \{ \lambda x.M \mid M \in \Lambda \} \cap \Lambda^0 \). The evaluation \( \Downarrow_\ell \) is the least binary relation over \( \Lambda^0 \times \text{Val}_\ell \) satisfying the following rules:

\[
\begin{array}{c}
\lambda x. M \Downarrow_\ell \lambda x. M \\
M \Downarrow_\ell \lambda x. P \\
M N \Downarrow_\ell Q
\end{array}
\]

• Lazy call-by-value. The lazy call-by-value reduction strategy, was introduced in [Plo75] and is the strategy implemented by the Landin’s SECD machine. The related observational equivalence was studied in [EHR92]. \( \sim_v \subseteq \Lambda^0 \times \Lambda^0 \) reduces the leftmost redex, not inside a \( \lambda \)-abstraction, whose argument is a \( \lambda \)-abstraction. \( \text{Val}_v = \{ \lambda x.M \mid M \in \Lambda \} \cap \Lambda^0 \). The evaluation \( \Downarrow_v \) is the least binary relation over \( \Lambda^0 \times \text{Val}_v \) satisfying the following rules:

\[
\begin{array}{c}
\lambda x. M \Downarrow_v \lambda x. M \\
M \Downarrow_v \lambda x. P \\
N \Downarrow_v Q \\
P[Q/x] \Downarrow_v U \\
M N \Downarrow_v U
\end{array}
\]

• Head call-by-name. The head call-by-name (or eager call-by-name) reduction strategy \( \sim_h \subseteq \Lambda \times \Lambda \) reduces the leftmost redex if the term is not in head normal form. Its observational strategy is studied in [Bar84, Wad76, Hy76]. \( \text{Val}_h = \{ \lambda x_1 \ldots x_n.M_1 \ldots M_m \mid M_1, \ldots, M_m \in \Lambda \} \). The evaluation \( \Downarrow_h \) is the least binary relation over \( \Lambda \times \text{Val}_h \) satisfying the following rules for \( n \geq 0 \):

\[
\begin{array}{c}
xM_1 \ldots M_n \Downarrow_h xM_1 \ldots M_n \\
M \Downarrow_h N \\
\lambda x. M \Downarrow_h \lambda x. N \\
(\lambda x. M)NM_1 \ldots M_n \Downarrow_h P
\end{array}
\]

• Normalizing. The normalizing reduction strategy \( \sim_n \subseteq \Lambda \times \Lambda \) reduces the leftmost redex. It is also known as the “complete” leftmost strategy. In [HL95] it is conjectured that its observational equivalence coincides with the equivalence induced by the model defined in [CDCZ87]. \( \text{Val}_n = \) the set of normal forms. The evaluation \( \Downarrow_n \) is the least binary relation over \( \Lambda \times \text{Val}_n \) satisfying the following rules:

\[
\begin{array}{c}
M_1 \Downarrow_n M_1' \\
\ldots \\
M_n \Downarrow_n M_n' \\
\lambda x. M \Downarrow_n \lambda x. N
\end{array}
\]

\[
\begin{array}{c}
M[N/x]M_1 \ldots M_n \Downarrow_n P \\
(\lambda x. M)NM_1 \ldots M_n \Downarrow_n P
\end{array}
\]

• Perpetual. The perpetual reduction strategy \( \sim_p \subseteq \Lambda \times \Lambda \) reduces the leftmost \( \beta \)-redex which is not contained in the operator of another redex, and which is either an \( I\beta \)-redex, or a \( K \beta \)-redex whose argument is a normal form. \( \text{Val}_p = \) the set of normal forms. The evaluation \( \Downarrow_p \) is the least binary relation over \( \Lambda \times \text{Val}_p \) satisfying the following rules:
\[
\frac{M_1 \Downarrow_p M'_1 \ldots M_n \Downarrow_p M'_n}{xM_1 \ldots M_n \Downarrow_p xM'_1 \ldots M'_n} \quad n \geq 0
\]
\[
\frac{M \Downarrow_p N}{\lambda x. M \Downarrow_p \lambda x. N}
\]
\[
\frac{N \Downarrow_p M[N/x]M_1 \ldots M_n \Downarrow_p V}{(\lambda x. M)N M_1 \ldots M_n \Downarrow_p V} \quad n \geq 0
\]

- **Parallel call-by-value.** The parallel call-by-value reduction strategy \(\sim_{pv}\) \(\subseteq \Lambda^0 \times \Lambda^0\) is a non-deterministic version of the lazy call-by-value reduction strategy \(\sim_v\). \(Val_{pv} = \{\lambda x. M \mid M \in \Lambda\} \cap \Lambda^0\). The evaluation \(\Downarrow_{pv}\) is the least binary relation over \(\Lambda^0 \times Val_{pv}\) satisfying the following rules:

\[
\frac{M \Downarrow_{pv} \lambda x. P \quad N \Downarrow_{pv} Q \quad P[Q/x] \Downarrow_{pv} U}{\lambda x. M \Downarrow_{pv} \lambda x. M}
\]

\[
\frac{MN \Downarrow_{pv} U}{MN \Downarrow_{pv} U}
\]

Each \(\lambda\)-theory \(\tau\) is induced by an equivalence relation \(\text{Eq} \subseteq \Lambda^0 \times \Lambda^0\) closed by \(\beta\)-equivalence (that is \(\beta \cap (\Lambda^0 \times \Lambda^0) \subseteq \text{Eq}\)) with only two equivalence classes \([HR92, HL99]\). Each equivalence relation \(\text{Eq}\) induces the following congruence on \(\lambda\)-terms termed contextual relation.

**Definition 2.4.** Let \(\text{Eq} \subseteq \Lambda^0 \times \Lambda^0\) be an equivalence relation. The contextual relation \(\approx_{\text{Eq}}\) is defined as follows:

\[
M \approx_{\text{Eq}} N \iff (\forall C[\_])(C[M], C[N] \in \Lambda^0 \Rightarrow C[M] \text{Eq} \text{ } C[N])
\]

**Theorem 2.5.** Let \(\tau\) be a \(\lambda\)-theory. There exists a set \(C \subseteq \Lambda^0\) such that:

\[
M \Rightarrow_{\tau} N \iff (\forall C[\_])(C[M], C[N] \in \Lambda^0 \Rightarrow (C[M] \in C \iff C[N] \in C))
\]

**Proof.** Put \(C = \{P \mid P \Rightarrow_{\tau} \lambda z.AB \mid A \Rightarrow_{\tau} B\} \cup C\). \(M \Rightarrow_{\tau} N \Rightarrow C[M] \Rightarrow_{\tau} C[N] \Rightarrow \exists C[\_] (C[N] \in C)\) for each context \(C[\_]\). Hence \(C[M] \in C \Rightarrow C[M] \Rightarrow_{\tau} \lambda z.AB \Rightarrow_{\tau} C[N] \in C\).

\(M \not\Rightarrow_{\tau} N \Rightarrow \exists C[\_] (\exists (\lambda z.z[\_] N) \text{ such that } C[N] \in C \text{ since } N \Rightarrow_{\tau} N \text{ and } C[M] \not\in C\).

\[
\square
\]

### 2.2 Trees of \(\lambda\)-terms

Syntactical equality of terms, is usually difficult to visualize, because induction on the syntactical structure of terms could be a tiring task and the terms could be very involved. It is then useful to associate to each term \(M \in \Lambda\) some tree, different from the syntactical one, which helps in clarifying the convertibility of two terms in a theory, allowing for a structural induction different from the purely syntactical one.

The trees of terms we shall utilize are the so-called Bohm trees [Böhm68, Bar84] and Lévy-Longo trees [Lévy75, Lon83]. Both definitions rely on the notion of solvability. A third type of trees – the Berarducci trees [B196] – also useful in the characterization of the equality in some theories, will not be taken in consideration, since, for the purpose of game semantics, they collapse onto the Lévy-Longo ones.

**Definition 2.6 (Solvability terms).** Let \(M \in \Lambda\) be a term and \(I\) the \(\lambda\)-term \(\lambda x.x\).

- \(M \in \Lambda^0\) is solvable if \(\exists n \exists N_1, \ldots, N_n \quad M N_1 \ldots N_n =_{\beta} I\);
- \(M\) is solvable if \(\lambda x_1 \ldots x_k. M\) is solvable with \(\text{FV}(M) = \{x_1, \ldots, x_k\}\);
• $M$ is unsolvable if it is not solvable.

**Example 2.7.** $K \equiv \lambda x y. x$ is solvable: $K I I =_\beta I$. $\Omega \equiv (\lambda x.x)(\lambda x.x)$ is unsolvable since $\Omega N_1 \ldots N_n \rightarrow^*_\beta \Omega N'_1 \ldots N'_n$ with $N_i \rightarrow^*_\beta N'_i$ for $1 \leq i \leq n$ and each $n \in \omega$ and terms $N_1, \ldots, N_n$, but $x\Omega$ is solvable: $(\lambda x. x\Omega)K =_\beta I$.

In the class of unsolvable terms we may distinguish syntactically terms which are abstractions from terms which are not.

**Definition 2.8.** Let $M \in \Lambda$ be an unsolvable term.

• $M$ is unsolvable of order $n$ if there exists $n \geq 0$ and $N \in \Lambda$ such that $M =_\beta \lambda x_1 \ldots x_n. N$

• $M$ is unsolvable of order $\infty$ if for all $n \in \omega$ there exists $N \in \Lambda$ such that $M =_\beta \lambda x_1 \ldots x_n. N$

The solvability property can be characterized syntactically with the notion of head normal form.

**Definition 2.9 (Head normal form).** A $\lambda$-term $M$ is in head normal form ($hnf$ for short) if it has the form

$$M \equiv \lambda x_1 \ldots x_n. y M_1 \ldots M_m$$

for suitable $n, m \geq 0$, variables $x_1, \ldots, x_n, y$ where $y$ can be one of the $x_1, \ldots, x_n$, and $\lambda$-terms $M_1, \ldots, M_m$. If $M_1, \ldots, M_m$ are in normal form then $M$ is in normal form. $M$ has a hnf if

$$\exists N \text{ in hnf and } M =_\beta N$$

$M$ has principal head normal form $\lambda x_1 \ldots x_n. y M_1 \ldots M_m$ if

$$M \rightarrow^*_\beta \lambda x_1 \ldots x_n. y M_1 \ldots M_m$$

following the head reduction strategy $\rightarrow^*_h$, that is reducing only the leftmost redexes until the term is not in hnf.

**Theorem 2.10.** $M$ is solvable if and only if $M$ has a hnf.

**Proof.** The proof is due to Wadsworth. See [Bar84], cap. 8.

Böhm and Lévy-Longo trees are labelled trees, that is partial functions $\varphi : \text{Seq} \rightarrow \Sigma$, where $\text{Seq}$ is the set of sequences of natural numbers, and $\Sigma$ is a set of labels. The empty sequence is denoted as $\epsilon$, while, for $s, t \in \text{Seq}$, $t \sqsubseteq s$ indicates that $t$ is a prefix of $s$. The concatenation of sequences $s$ and $t$ is indicated as $s \cdot t$. $|s|$ is the length of the sequence $s$. $\varphi(s) \downarrow$ says that $\varphi$ is defined on $s$ while $\varphi(s) \uparrow$ denotes the converse.

**Definition 2.11.** Given a set of labels $\Sigma$, a $\Sigma$-labelled tree is a partial function $\varphi : \text{Seq} \rightarrow \Sigma$ such that:

• $\varphi(\epsilon) \downarrow$

• $\varphi(s) \downarrow$ & $t \sqsubseteq s \Rightarrow \varphi(t) \downarrow$

The underlying tree $T_\varphi$ of $\varphi$ is defined as the set $\{ s \in \text{Seq} \mid \varphi(s) \downarrow \}$.

**Definition 2.12.** Given a $\Sigma$-labelled tree $\varphi$, the subtree $\varphi_s$ at some node $s$ is the tree defined by $\varphi_s(t) = \varphi(s \cdot t)$. 
The Böhm tree $BT(M)$ of a term $M \in \Lambda$ is informally described as follows:

• $BT(M) = \perp$ if $M$ is unsolvable

• $BT(M) = \lambda x_1 \ldots x_n.y$
  \[ BT(M_1) \quad \ldots \quad BT(M_m) \]
  if $M \rightarrow^* _h \lambda x_1 \ldots x_n.yM_1 \ldots M_m$

The Lévy-Longo tree $LLT(M)$ of a term $M \in \Lambda$ is informally depicted as:

• $LLT(M) = T$ if $M$ is unsolvable of order $\infty$

• $LLT(M) = \lambda x_1 \ldots x_n.\perp$ if $M$ is unsolvable of order $n$

• $LLT(M) = \lambda x_1 \ldots x_n.y$
  \[ LLT(M_1) \quad \ldots \quad LLT(M_m) \]
  if $M \rightarrow^* _h \lambda x_1 \ldots x_n.yM_1 \ldots M_m$

**Definition 2.13 (Böhm tree – formal definition).** Let $\Sigma_1$ be the set

\[ \{ \perp \} \cup \{ \lambda x_1 \ldots x_n.y \mid n \geq 0, x_1 \ldots x_n \text{ variables} \} \]

The Böhm tree of $M$, $BT(M)$ is the $\Sigma_1$-labelled tree defined as follows:

• if $M$ is unsolvable then
  - $BT(M)(e) = \perp$
  - $BT(M)(s) \uparrow$ for each $s \in \text{Seq}$

• if $M$ is solvable, and has principal hnf $\lambda x_1 \ldots x_n.yM_1 \ldots M_m$ then
  - $BT(M)(e) = \lambda x_1 \ldots x_n.y$
  - $BT(M)(k \cdot s) = \begin{cases} BT(M_k)(s) & \text{if } k < m \\ \uparrow & \text{if } k \geq m \end{cases}$

**Definition 2.14 (Lévy-Longo tree – formal definition).** Let $\Sigma_2$ be the set

\[ \Sigma_1 \cup \{ T \} \cup \{ \lambda x_1 \ldots x_n.\perp \mid n > 0, x_1 \ldots x_n \text{ variables} \} \]

The Lévy-Longo tree of $M$, $LLT(M)$ is the $\Sigma_2$-labelled tree defined as follows:

• if $M$ is unsolvable of order $\infty$ then
$$- LLT(M)(\varepsilon) = T$$
$$- LLT(M)(s) \uparrow \text{ for each } s \in \text{Seq}$$

- if $M$ is solvable of order $n \geq 0$ then
  $$- LLT(M)(\varepsilon) = \lambda x_1 \ldots x_n.\bot$$
  $$- LLT(M)(s) \uparrow \text{ for each } s \in \text{Seq}$$

- if $M$ is solvable, and has principal hnf $\lambda x_1 \ldots x_n.yM_1 \ldots M_m$ then
  $$- LLT(M)(\varepsilon) = \lambda x_1 \ldots x_n.y$$
  $$- LLT(M)(k \cdot s) = \begin{cases} LLT(M_k)(s) & \text{if } k < m \\ \uparrow & \text{if } k \geq m \end{cases}$$

**Definition 2.15.** A Böhm-like tree is a $\Sigma_1$-labelled tree. The set of all Böhm-like trees is denoted by $\mathcal{B}T$.

A Lévy-Longo-like tree is a $\Sigma_2$-labelled tree. The set of all Lévy-Longo-like trees is denoted by $\mathcal{L}LT$.

Each finite Böhm-like tree $A \in \mathcal{B}T$ is the tree of some term $M$ defined by induction on $d(A) = \sup\{|s| \mid s \in T_A\}$ as follows.

**Definition 2.16.** Given a finite Böhm-like tree $A \in \mathcal{B}T$, the term $M[A]$ is inductively defined as:

- $A = \bot \Rightarrow M \equiv \Omega \equiv (\lambda x.xx)(\lambda x.xx)$
- $A = \lambda x_1 \ldots x_n.y \Rightarrow M \equiv \lambda x_1 \ldots x_n.y$
- $A = \lambda x_1 \ldots x_n.yM_1 \ldots M_m \Rightarrow M \equiv \lambda x_1 \ldots x_n.yM[A_1] \ldots M[A_m]$

$$A_1 \quad \cdots \quad A_m$$

Notice that, if $BT(P)$ is finite, $M[BT(P)] = \beta P$. If $BT(P)$ is infinite, the related term is not defined since $P$ has no normal form and $d(BT(P)) = \infty$. A typical case is the fixed-point combinator $Y \equiv \lambda f.(\lambda x.f(xx))(\lambda x.f(xx))$ where

$$BT(Y) = \lambda f.f$$
$$\quad |$$
$$\quad f$$
$$\quad |$$
$$\quad f$$
$$\quad |$$
$$\quad \vdots$$

**Definition 2.17.** Let $A \in \mathcal{B}T$. For $k \in \omega$ define $A^k$ as

$$A^k(s) = \begin{cases} A(s) & \text{if } |s| < k \\ \uparrow & \text{if } |s| \geq k \end{cases}$$

$BT^k(M) - LLT^k(M) -$ will denote the truncated tree $(BT(M))^k - (LLT(M))^k -$ and $P^{(k)} = M[BT^k(P)]$. Moreover, for $A, B \in \mathcal{B}T$ or $(\mathcal{L}LT)$, $A =_k B$ means $A^k = B^k$. 
Since the two \(\eta\)-convertible terms \(\lambda x.x\) and \(\lambda xy.xy\) have different Böhm-trees

\[
\begin{array}{c}
\lambda x.x \\
\downarrow \\
y
\end{array}
\quad
\begin{array}{c}
\lambda xy.x \\
\downarrow \\
y
\end{array}
\]

it is necessary to introduce a notion of \(\eta\)-conversion also for the Böhm-like trees.

**Definition 2.18 (\(\eta\)-reduction for Böhm-like trees).** Let \(A \in \mathcal{B}T\) and let \(s \in T_A\) a node such that \(A(s) = \lambda x_1 \ldots x_n.y\) and it has \(m\) successors \(B_1, \ldots, B_m\). An \(\eta\)-expansion of \(A'\) of \(A\) at \(s\) (written \(A' \rightarrow_\eta A\)) is the result of replacing in \(A\) the subtree \(A_s\)

\[
\begin{array}{c}
\lambda x_1 \ldots x_n.y \\
\downarrow \\
B_1 \quad \cdots \quad B_m
\end{array}
\]

by

\[
\begin{array}{c}
\lambda x_1 \ldots x_nyz.y \\
\downarrow \\
B_1 \quad \cdots \quad B_m
\end{array}
\]

The reflexive and transitive closure of \(\rightarrow_\eta\) is denoted as \(\rightarrow_*\).

Wadsworth has noticed that we may have also infinite \(\eta\)-expansions of a Böhm-like tree. Consider the following example. Let \(M \equiv x\) and let \(N \equiv Jx\) where \(J \equiv \Theta(\lambda jxy.x(jy))\) and \(\Theta \equiv (\lambda xy.y(xxy))(\lambda xy.y(xx\lambda))\). It is easy to see (see [Bar84], Chapter 10) that

\[
BT(M) = x \quad BT(N) = \lambda z_0.x \\
\downarrow \\
\lambda z_1.z_0 \\
\downarrow \\
\lambda z_2.z_1 \\
\vdots
\]

where \(BT(N)\) can be considered as the infinite sequence of \(\eta\)-expansions

\[
BT(x) \eta \leftarrow BT(\lambda z_0.xz_0) \eta \leftarrow BT(\lambda z_0.x(\lambda z_1.z_0)) \eta \leftarrow \cdots
\]

and is called infinite \(\eta\)-expansion of \(A\). Thus we have to define also the transfinite closure \(\geq_\eta\) of \(\rightarrow_\eta\). The general definition of \(\eta\)-expansion of Böhm-like trees is the following.

**Definition 2.19.** Let \(A \in \mathcal{B}T\) and \(X \subseteq \text{Seq}\).

- **X extends A if and only if**
  - \(T_A \subseteq T_X\), \(X\) is a tree and is finitely branching
  - \(A(s) = \perp \Rightarrow s\) is a terminal node in \(X\)

- **If X extends A the Böhm-like tree \((A;X)\) is the tree with underlying tree \(T_{(A;X)} = X\) defined as follows:**
  - \((A;X)(s) = A(s)\) if \(A(s) \downarrow\) and \(s\) has the same number of successors in \(A\) and \(X\)
\( (A; X)(s) = \lambda x_1 \ldots x_n z_1 \ldots z_k. y \) if \( A(s) = \lambda x_1 \ldots x_n. y \) and \( s \) has \( m \) successors in \( A \) and \( m + k \) successors in \( X \)

\( (A; X)(s) = \lambda z_1 \ldots z_n. z_i \) if 

* \( s = t \cdot (k + i) \), \( X(s) \downarrow \), \( A(s) \uparrow \), \( A(t) \downarrow \), \( t \) has \( k \) successors in \( A \) and \( s \) has \( n \) successors in \( X \)

* \( s = t \cdot i \), \( X(s) \downarrow \), \( A(s) \uparrow \), \( A(t) \uparrow \) and \( s \) has \( n \) successors in \( X \)

\( (A; X)(s) \uparrow \) if \( X(s) \uparrow \)

- \( B \) is a (possibly) infinite \( \eta \)-expansion of \( A \) (\( A \leq_\eta B \)) if, for some \( X \) extending \( A \), \( B = (A; X) \)

Each Böhm-like tree \( A \in \mathcal{BT} \) has an infinite \( \eta \)-nf \( \eta(A) \).

**Definition 2.20 (Infinite \( \eta \)-nf).** Let \( A \in \mathcal{BT} \). Define \( \eta(A) \) as:

- \( \eta(\bot) = \bot \)

- \( \eta(\lambda x_1 \ldots x_n. y) = \begin{cases} \eta(\lambda x_1 \ldots x_{n-1}. y) & \text{if } x_n \leq_\eta A_m \text{ and } x_n \notin \text{FV}(A_i) \text{ for } 1 \leq i \leq m-1 \\ \eta(A_1) \cdots \eta(A_m) & \text{otherwise} \end{cases} \)

Refer to [Bar84], Chapter 10, for the verification that this is a good definition.

**Definition 2.21.** The following relations are defined on Böhm and Lévy-Longo trees:

- \( \mathcal{BT}(M) \subseteq \mathcal{BT}(N) \) \iff \((\forall s)(\mathcal{BT}(M)(s) = \bot \text{ and } \mathcal{BT}(N)(s) \downarrow) \text{ or } \mathcal{BT}(M)(s) = \mathcal{BT}(N)(s))\)

- \( \eta(\mathcal{BT}(M)) \subseteq \eta(\mathcal{BT}(N)) \) \iff \((\exists A \in \mathcal{BT})(\mathcal{BT}(M) \leq_\eta A \subseteq \mathcal{BT}(N))\)

- \( \mathcal{BT}(M)^n \subseteq \eta(\mathcal{BT}(N)) \) \iff \((\exists A, B \in \mathcal{BT})(\mathcal{BT}(M) \leq_\eta A \subseteq B \geq_\eta \mathcal{BT}(N))\)

- \( \mathcal{BT}(M) =_\eta \mathcal{BT}(N) \) \iff \( \mathcal{BT}(M)^n \subseteq \eta \mathcal{BT}(N) \subseteq \eta \mathcal{BT}(M) \)

- \( \mathcal{LLT}(M) \subseteq \mathcal{LLT}(N) \) \iff \((\forall s)(\mathcal{LLT}(M)(s) = \bot \text{ or } \mathcal{LLT}(M)(s) = \lambda x_1 \ldots x_n. \bot \text{ and } \mathcal{LLT}(N)(s) = \lambda x_1 \ldots x_{n'} \bot, n' > n \text{ or } \mathcal{LLT}(M)(s) = \lambda x_1 \ldots x_n. \bot \text{ and } \mathcal{LLT}(N)(s) = T \text{ or } \mathcal{LLT}(M)(s) = \mathcal{LLT}(N)(s))\)

The following theorem will be used in Chapter 3 to study the theory of some models.
Theorem 2.22. Let $A, B \in \mathcal{B}T$. The following statements are equivalent:

- $A \equiv \eta B$
- $\infty \eta(A) = \infty \eta(B)$
- $(\forall k)(\exists A', B' \in \mathcal{B}T)(A' \rightarrow^*_\eta A, B' \rightarrow^*_\eta B, A' =_k B')$

Proof. See [Bar84], Theorem 10.2.31.

2.3 Labelled $\lambda$-calculus

A large class of denotational models of $\lambda$-calculus enjoys the property that the interpretation $\lbrack M \rbrack$ of a term $M$ is the least upper bound of the interpretations of the approximate normal forms of $M$, that is of the terms obtained by a partial evaluation of $M$. To represent such terms, an applicative version (with constants) of $\lambda$-calculus is used, $\lambda\Omega$-calculus.

Definition 2.23 ($\Lambda(\Omega)$). The set of $\Omega$-approximate terms $\Lambda(\Omega) \ni M$ is obtained from $\Lambda$ by adjoining a constant $\Omega$ to the formation rules:

$$M ::= x \mid \lambda x. M \mid MM \mid \Omega$$

The following notions of reduction are considered on $\Lambda(\Omega)$:

- $(\beta)$ $(\lambda x. M) N \rightarrow M[N/x]$
- $(\Omega_1)$ $\lambda x. \Omega \rightarrow \Omega$
- $(\Omega_2)$ $\Omega M \rightarrow \Omega$

The aim of the constant $\Omega$ is to represent not yet evaluated subterms on which, at the moment, no information is provided. From a semantical point of view, this corresponds to a divergent term. $\Omega$ is redundant since it can be replaced by the prototypical term $(\lambda x.xx)(\lambda x.xx)$.

To obtain a (strongly) terminating calculus we need to introduce a typed version of $\lambda\Omega$, where types are indexes put on subterms of a term.

Definition 2.24 ($\Lambda(\Omega)^{\Omega}$). The set of (possibly) indexed terms $\Lambda(\Omega)^{\Omega} \ni M$ is the superset of $\Lambda(\Omega)$ defined as

$$M ::= x \mid MM \mid \lambda x. M \mid \Omega \mid M^\omega$$

A term is truly indexed if it is of the shape $M^\omega$. A term is completely indexed if all its subterms of the shape variable, abstraction, and application are immediate subterms of truly indexed terms.

The following notions of reduction are considered on $\Lambda(\Omega)^{\Omega}$:

- $(\beta_1)$ $(\lambda x.P)^{m+1} Q \rightarrow ((\lambda x.P)Q^m)^{m+1}$
- $(\beta_{i,j})$ $(M^i)^j \rightarrow M^{m^i}$
- $(\beta)$ $(\lambda x. M) N \rightarrow M[N/x]$
- $(\Omega_1)$ $\lambda x. \Omega \rightarrow \Omega$
- $(\Omega_2)$ $\Omega M \rightarrow \Omega$
- $(\Omega^0)$ $M^0 \rightarrow \Omega$
- $(\Omega^n)$ $\Omega^n \rightarrow \Omega$
Definition 2.25. The erasing function $\mid : \Lambda(\Omega)^n \rightarrow \Lambda(\Omega)$ is inductively defined as follows:

1. $\mid x \mid = x; \mid \Omega \mid = \Omega$
2. $\mid PQ \mid = \mid P \mid \mid Q \mid$
3. $\mid \lambda x. P \mid = \lambda x. \mid P \mid$
4. $\mid M^n \mid = \mid M \mid$

Proposition 2.26. A completely indexed term $Q$ is $\beta^1 \beta^3 \Omega^0 \Omega^n$-strongly normalizing.

Proof. Since each indexed reduction step decreases the index of the contractum or stops the reduction at $\Omega$, the proof is the same of Theorem 14.1.2 of [Bar84].
Models of the Lambda Calculus

“...there seems to be, firstly an assumption that the definition is too obvious to need stating, and secondly a disagreement about what the definition should be.”

Hindley and Longo, 1980 [HL80]

Abstract

Many mathematical (syntax-free) models of untyped \( \lambda \)-calculus appeared in the literature since the seminal example of Scott [Sc72]. All these models count as algebraic, admitting a set-theoretic definition. We shall introduce in this chapter the relevant notions and present three basic examples: the model \( D_\infty \), the model \( P^\infty \) and the model \( D_\bullet \) (an instance of the class of Scott-Plotkin-Engeler algebras). We focus on the environment presentation of algebraic models, showing a completeness result due to Meyer [Mey82]: for each \( \lambda \)-theory there is an environment model which enforces exactly the identities of the theory.

Then we present the categorical definition of models of untyped \( \lambda \)-calculus giving again a completeness result, due, this time, to Scott [Sc80b]. We compare algebraic and categorical models providing a transformation back and forth.

We conclude the chapter by introducing the notion of approximable model and presenting some basic properties common to all these models. The theory of the above mentioned basic examples is then studied and the equality on terms they induced defined by means of the tree form of terms.

Mathematical models of \( \lambda \)-calculus give a conceptual view of the syntax of the language and allow to grasp intuitions on the operational properties of the theory they induce. In fact, each model \( M \), through an interpretation function \( [\cdot] : \Lambda \rightarrow M \), allows to define on the \( \lambda \)-terms the congruence relation \( \approx_M \): 

\[
\forall M, N \in \Lambda, M \approx_M N \iff [M]^M = [N]^M
\]

which is closed by \( \beta \)-conversion. If this relation coincides with the congruence induced by a reduction strategy, we may say that the model is fully-abstract with respect to the operational
semantics related to the reduction strategy. In this case, the semantical model carries all
the necessary operational information on the related $\lambda$-theory. The semantical model can
then be freely used to state all the relevant properties of terms. In some cases, the proof of
some properties are greatly facilitate as is the case for the proof that, in the theory $\mathcal{H}_n$, all
fixed-point operators are identified. Syntax-independent models of $\lambda$-calculus are useful to
get information on the $\beta$-reduction, to exhibit proofs of consistency for equational extensions
of $\beta$, to produce examples for refutation arguments.

The first mathematical models of $\lambda$-calculus fallen in the universe of set-theoretic func-
tions (and then functions in extension) albeit Church admonished in [Chu41] that $\lambda$-calculus
was a calculus about functions in intension. This approach was later the origin of some
disagreement on what a model of $\lambda$-calculus should have been to be, i.e. $\lambda$-algebras $\vDash$
$\lambda$-models. It posed, however, the main challenge. Since each term has to be interpreted
as a function and each term can be applied to itself, it is necessary to develop a domain
of interpretation $D$ which “contains” the set of functions $\{f : D \to D\}$. “Contains” has to be inten-
ded more formally as “there is a subset $D'$ of $D$ which is isomorphic to the set of
functions from $D$ to $D$. For cardinality reasons, this cannot be true for the set of all
functions from $D$ to $B$ but only for a subset of them.

Dana Scott in 1969 [Scot72] built the first model utilizing the continuous functions between
continuous lattices. This model, which was named $D_{\infty}$, was rebuilt later in the more general
framework of continuous functions between complete partial orders (CPOs). It has been only
after a long discussion that the notion of model of $\lambda$-calculi proposed by Scott [Scot72] gained
general consensus in the computer science community. After then, many different models,
with different properties, appeared. Barendregt was the first to distinguish between two
types of models for $\lambda$-calculus: the $\lambda$-algebras and the $\lambda$-models. The objects in the first
class satisfy all the provable equations of $\lambda$-calculus, can be axiomatized by a finite set of
equational axiom schemes but are not closed under rule derivation. The $\lambda$-models satisfy
instead all the rules of $\lambda$-calculus but are not finitely axiomatizable, they can be defined only
by a set of first-order axioms.

Categorical models of $\lambda$-calculus allow to bypass the diatribe between $\lambda$-algebras and
$\lambda$-models [Ber81]. For each reflexive object $D$ in a CCC it is possible to define an interpre-
tation function $\Delta : A \to (D^A) \Rightarrow D$ with $\Delta = \{x_1, \ldots, x_n\}$, such that for each $M, N \in A$ with
$FV(MN) \subseteq \Delta$, $M =_\beta N \Rightarrow [M]_\Delta = [N]_\Delta$.

3.1 Algebraic models of $\lambda$-calculus

Both $\lambda$-algebras and $\lambda$-models are algebraic structures which can be seen as instances of a
more general class of objects: the combinatorial algebras.

Definition 3.1 (Applicative structure). $A = \langle A, \cdot \rangle$ is an applicative structure if $\cdot$ is a
binary operation on $A$. If $a, b \in A$ we write simply $ab$ for $a \cdot b$. The rule of association of $\cdot$
is on the left: $abc$ stands for $(ab)c$. An applicative structure is extensional if and only if for all $a, b \in A$ the following property is satisfied: $(\forall c \in A)(ac = bc \Rightarrow a = b)$.

Definition 3.2 (Representatives). Let $\langle A, \cdot \rangle$ be an applicative structure and let $n \geq 1$.
A function $f : A^n \to A$ is representable in $A$ if and only if there exists $a \in A$ such that
$(\forall b_1, \ldots, b_n \in A)(f(b_1, \ldots, b_n) = ab_1 \ldots b_n)$.

The element $a$ is called a representative of $f$. Notice that each element $a \in X$ represents
a function of each finite number of arguments. The unique one-place function $b \mapsto a \cdot b$
that \( a \in X \) represents is indicated as \( f_a \). Following an observation of Schönfinkel, each function of \( n + 1 \) arguments \( f : X^{n+1} \to X \) with values in \( X \) can be thought of a functional \( f' : X^n \to (X \to X) \) of \( n \) arguments which returns a function of one argument with values in \( X \). It is then sufficient to consider elements \( a \in X \) to represent only functions \( f : X \to X \) of one argument. This observation retains the relevance of Cartesian closed categories for the models of \( \lambda \)-calculus.

In general there is no assurance that all the functions defined by \( \lambda \)-terms are representable in an applicative structure \( A \). This is instead the case for combinatory algebras.

**Definition 3.3 (Combinatory algebra).** An applicative structure \( \langle A, \cdot \rangle \) is a combinatory algebra \( \langle A, \cdot, k, s \rangle \) if there are distinguished \( k, s \in A \), \( k \neq s \) such that \( kab = a \) and \( sabc = (ac)(bc) \) for each \( a, b, c \in A \).

Combinatory algebras are the basic structures for the interpretation of \( \lambda \)-calculus. Given a valuation \( \rho : \text{Var} \to A \), that is a function who assigns elements of \( A \) to the free variables, the interpretation of \( \lambda \)-terms can be defined as follows.

**Definition 3.4.** Given a combinatory algebra \( A = \langle A, \cdot, k, s \rangle \) and a valuation \( \rho : \Lambda \to A \) be the function defined as follows:

- \( ((x))_\rho = \rho(x) \)
- \( ((MN))_\rho = ((M))_\rho((N))_\rho \)
- \( ((\lambda x.M))_\rho = \begin{cases} 
  \text{skk} & \text{if } M \equiv x \\
  k((M))_\rho & \text{if } x \notin M \\
  s((\lambda x.P))_\rho((\lambda x.Q))_\rho & \text{if } M \equiv PQ 
\end{cases} \)

Unfortunately, this interpretation is not sound. Consider the \( \lambda \)-terms \( M \equiv \lambda x.(\lambda y.y)x \) and \( N \equiv \lambda x.x \). No doubt that \( M =_\beta N \) but, putting \( i = \text{skk} \), we have \( (M)_\rho = s(k)i \neq i = (N)_\rho \) for each \( \rho \).

The additional axioms which have to be added to a combinatory algebra to obtain a sound model are known as the **Curry axioms** \( A_\beta \) [Bar84]. There are, of course, equivalent formulations using different set of axioms [Lam80, Ros50].

**Definition 3.5 (Lambda algebra).** A combinatory algebra \( A = \langle A, \cdot, k, s \rangle \) such that \( M =_\beta N \Rightarrow ((M))_\rho = ((N))_\rho \) for each \( \rho \), is called \( \lambda \)-algebra. A \( \lambda \)-algebra \( A = \langle A, \cdot, k, s \rangle \) is extensional if the applicative structure \( \langle A, \cdot \rangle \) is extensional.

Notice that an extensional combinatory algebra is always a \( \lambda \)-algebra. In an extensional \( \lambda \)-algebra there is, in fact, only one element which denotes a particular function represented by some term. Turning an applicative structure into a combinatory algebra (when possible) amounts to define a method or an operator which always chooses a particular element among all the representatives of a function.

Albeit \( \lambda \)-algebras validate all the equations between \( \lambda \)-terms obtained by \( \beta \)-conversion, they do not validate all the rules of \( \lambda \)-calculus, in particular they do not validate the \( \xi \)-rule

\[ M = N \Rightarrow \lambda x.M = \lambda x.N \]

The different approach of Lambek [Lam80], based on the definition of polynomials, allows to deal also with the \( \xi \)-rule.
Definition 3.6 (Polynomials). The set of polynomials $P(A)$, over a combinatorial algebra $(A, \cdot, k, s)$ is the set of words obtained from a countably infinite set of variables $\text{Var}$ according to the following grammar:

$$W ::= x \mid a \mid (WW)$$

with $x \in \text{Var}$ and $a \in A$.

Definition 3.7 (Lambek interpretation). Given a combinatorial algebra $A = (A, \cdot, k, s)$, let $(\langle \rangle)^P : \Lambda \rightarrow P(A)$ be the function defined as follows:

- $(\langle x \rangle)^P = x$
- $(\langle MN \rangle)^P = ((M)^P (N)^P)^P$
- $(\langle \lambda x. M \rangle)^P = \begin{cases} \text{skk} & \text{if } M \equiv x \\ \text{k}(\langle M \rangle)^P & \text{if } x \not\in M \\ \text{s}(\langle \lambda x. P \rangle)^P (\langle \lambda x. Q \rangle)^P & \text{if } M \equiv PQ \end{cases}$

Definition 3.8 (Equality of polynomials). Equality of polynomials $=_P$ is the smallest equivalence relation $\approx \subseteq P(A) \times P(A)$ such that for $\chi, \chi_1, \chi_2, \psi, \psi_1, \psi_2, \varphi \in P(A)$

$$\begin{align*}
\chi_1 \approx \psi_1 & \quad \chi_2 \approx \psi_2 \\
\chi_1 \chi_2 \approx \psi_1 \psi_2 & \\
\text{k}\chi_1 \psi \approx \chi & \\
\text{s}\chi_1 \psi \varphi \approx (\varphi \psi) & \\
\end{align*}$$

Proposition 3.9. $M =_\beta N \Rightarrow (\langle M \rangle)^P =_P (\langle N \rangle)^P$.

Different axioms have been proposed to assure the validity of the $\xi$-rule in a $\lambda$-algebra (turning it into a model). The following one was proposed by Scott.

Definition 3.10 (Lambda models). Let $A = (A, \cdot, k, s)$ be a $\lambda$-algebra. Let $1 = s(k)$. $A$ is a $\lambda$-model if for all $a, b \in A$, the axiom

$$(\forall c \in A)(ac = bc) \Rightarrow 1a = 1b$$

holds. A $\lambda$-model $A = (A, \cdot, k, s)$ is extensional if $\langle A, \cdot \rangle$ is extensional.

It is clear that the following chain of inclusions holds: extensional $\lambda$-models $\subseteq$ $\lambda$-models $\subseteq$ combinatorial algebras. In [BK80] the strictness of all these inclusions is shown.

### 3.1.1 Environment models

The history of $\lambda$-calculus models is hardly linear. There is a general agreement to indicate the first mathematical model as the one construed by Scott in 1969 [Sco72]. After that, different models have been presented, each of them suitable for a particular purpose. All these presentations, however, turn out to be equivalent. We choose as representatives of the comprehensive class of algebraic models of $\lambda$-calculus, the environment models. Environment models have been largely used in the literature [HL80, HS86, Bar84, Mey82, Koy82] whenever it was preeminent to clarify the structure of the interpretation of a particular $\lambda$-term.

An environment model is defined by a set $D$ of values together with a mapping $[\quad]$ from any $\lambda$-term $M$ to a value $[M] \in D$, such that $\beta$-convertible terms receive the same
value. The interpretation of λ-terms is given following the syntax. Since the meaning of a λ-abstraction is intended to be a function \( f : D \rightarrow D \) rather than an element of \( D \), there is a need for an onto-function \( \Phi \) from \( D \) to the set \( (D \rightarrow D) \) of representable functions on \( D \). Moreover, since we need to represent each function in \( (D \rightarrow D) \) as an element of \( D \), we also need a function \( \Psi : (D \rightarrow D) \rightarrow D \) such that \( \Phi(\Psi(f)) = f \) for each \( f \in (D \rightarrow D) \).

Free variables of terms are managed by an environment, that is a valuation \( \rho : \text{Var} \rightarrow D \). As already pointed out above, the role of the environment is to pass values for free variables. It is then necessary that environments and λ-abstractions behave in the same way and, since environments are extensional in nature by definition, so there has to be also the interpretation of λ-abstractions.

Environment models have been presented in [Mey82, Koy82] and, in the equivalent form of the syntactical models, in [HL80, Bar84]. Syntactical models are called simply λ-models and models of λβ in [HS86]. Environment models are called combinatory models in [Lou83].

**Definition 3.11 (Environment λ-model).** An environment model is a structure \( E = \langle D, \Phi, \Psi \rangle \) with \( \Phi : D \rightarrow (D \rightarrow D) \) and \( \Psi : (D \rightarrow D) \rightarrow D \) such that \( \Phi(\Psi(f)) = f \) for each \( f \in (D \rightarrow D) \) and such that if λ-terms are interpreted as follows:

\[
\begin{align*}
[x]_\rho &= \rho(x) \\
[MN]_\rho &= (\Phi([M]_\rho))([N]_\rho) \\
[\lambda x. M]_\rho &= \Psi(d \mapsto [M]_{\rho[x:=d]})
\end{align*}
\]

for a valuation \( \rho \), each function \( d \mapsto [M]_{\rho[x:=d]} \in (D \rightarrow D) \).

Notice that in the above definition the interpretation function \( [\cdot] \) is not part of the definition of the model since it is uniquely determined by \( D, \Phi \) and \( \Psi \).

### 3.1.2 Other algebraic models

The attention driven on algebraic models of λ-calculus has produced a class of heterogeneous models. These are briefly introduced and compared in the following.

The syntactical models have been introduced in [HL80] as structures \( \langle D, \cdot \rangle \), an applicative structure and with a set of properties that each interpretation \([\cdot]\) of terms in a set of values \( D \) has to satisfy. The set of properties does not provide a determined interpretation function, as the interpretation of a λ-abstraction \( \lambda x.M \) is not established, but lists the minimal properties each interpretation must satisfy.

The syntax-free models of [HS86], arises by studying the quotient \( D/\sim \) of a given syntactical λ-model \( D = \langle D, \cdot \rangle \) for the extensional equivalence relation \( \sim \) on elements of \( D \):
\( d \sim e \iff (\forall a \in D)(da = ea) \). For each extensional equivalence class \( \bar{d} \) there are \( M, x, \rho \) such that \([\lambda x. M]_\rho \in \bar{d} \). It is then possible to choose an element of \( \bar{d} \) as the value of \([\lambda x. M]_\rho \) for all \( M, x, \rho \). This gives rise to a map \( \psi : D \rightarrow D \), which is representable in \( D \). A syntax-free model is then a structure \( \langle D, \cdot, \psi \rangle \) where \( \langle D, \cdot \rangle \) is a combinatory algebra.

The combinatory models of [Mey82] are syntax-free models where, instead of focusing on the map \( \psi \), the attention is driven on its representatives. They are structures \( \langle D, \cdot, \varepsilon \rangle \), where \( \langle D, \cdot \rangle \) is a combinatory algebra and \( \varepsilon \in D \) is the element which represents the choosing map \( \psi \).

The Scott-Meyer models [Mey82, Sco80], are combinatory models \( \langle D, \cdot, \varepsilon \rangle \) where new axioms are introduced to allow only one possible choice of \( k \) and \( s \) to make \( \langle D, \cdot \rangle \) into a combinatory algebra.
3.1.3 Examples of environment models

Among all the algebraic models of \( \lambda \)-calculus, two seminal examples deserve particular attention: the model \( D_\infty \) and the model \( P^\omega \). Their equational theories are induced also by some classes of models of \( \lambda \)-calculus built in the category of games \( G \). The comprehensive class of models \( D_A \) is instead used to extract a model, \( D_\oplus \), which induces a theory different from the previous ones, and is the theory of a third class of game models of \( \lambda \)-calculus.

Let us briefly remind the basic notions used in the examples that follow.

**Definition 3.12 (Partially ordered set).** A partially ordered set or poset is a pair \( \langle D, \sqsubseteq \rangle \) where \( D \) is a set and \( \sqsubseteq \) is a binary relation on \( D \) which for all \( a, b, c \in D \) fulfills:

\[
\begin{align*}
\quad a & \sqsubseteq a \quad \text{(reflexivity)}; \\
\quad a \sqsubseteq b \text{ and } b \sqsubseteq a & \Rightarrow a = b \quad \text{(anti-symmetry)}; \\
\quad a \sqsubseteq b \text{ and } b \sqsubseteq c & \Rightarrow a \sqsubseteq c \quad \text{(transitivity)}.
\end{align*}
\]

The least member of \( D \) (if any) is indicated as \( \bot \) and is called bottom:

\[
(\forall d \in D)(\bot \sqsubseteq d)
\]

**Definition 3.13 (Least upper bound).** Let \( \langle D, \sqsubseteq \rangle \) a partially ordered set.

1. A subset \( X \subseteq D \) is directed iff \( (\forall a, b \in X)(\exists c \in X)(a \sqsubseteq c \text{ and } b \sqsubseteq c) \).

2. An upper bound of a subset \( X \subseteq D \) is any \( b \in D \) such that \( (\forall a \in X)(a \sqsubseteq b) \).

The least upper bound (lub) or supremum \( \bigcup X \) of \( X \) is any upper bound \( b \) such that \( (\forall c \in D)(c \text{ upper bound of } X \Rightarrow b \sqsubseteq c) \).

Notice that a lub of a set \( X \subseteq D \) may not exist or may exist but does not belong to \( X \). If it exists it is unique.

**Definition 3.14 (Complete partial order).** A complete partial order or CPO is a partially ordered set \( \langle D, \sqsubseteq \rangle \) such that:

1. \( D \) has a least member \( \bot \);

2. every directed \( X \subseteq D \) has a lub \( \bigcup X \).

When the binary relation \( \sqsubseteq \) of a partially ordered set \( \langle D, \sqsubseteq \rangle \) is not important, we shall refer simply to a partially ordered set \( D \).

A complete lattice is a partially ordered set \( D \) where each \( X \subseteq D \) has a lub.

**Definition 3.15 (Continuous functions).** Let \( D \) and \( D' \) be two CPOs and let \( f : D \rightarrow D' \).

1. \( f \) is monotonic iff \( a \sqsubseteq b \Rightarrow f(a) \sqsubseteq f(b) \) for all \( a, b \in D \);

2. \( f \) is continuous iff \( f(\bigcup X) = \bigcup \{f(x) \mid x \in X\} \) for each \( X \subseteq D \) directed, \( X \neq \emptyset \), where \( f(X) = \{f(x) \mid x \in X\} \).

A continuous function in the sense above is called also Scott-continuous function.

**Definition 3.16 (Function space CPO).** For CPOs \( \langle D, \sqsubseteq_D \rangle \) and \( \langle D', \sqsubseteq_{D'} \rangle \) let us define

\[
[D \rightarrow D'] = \{f : D \rightarrow D' \mid f \text{ continuous}\}.
\]

It is partially ordered by \( f \sqsubseteq_{[D \rightarrow D'] g \text{ iff } (\forall d \in D)(f(d) \sqsubseteq_{D'} g(d)) \).
\[ [D \to D'], \subseteq_{[D \to D']} \] is a CPO with the least element \( \perp_{[D \to D']} \) the function \( d \mapsto \perp_{D'} \). Moreover, for each \( Y \subseteq [D \to D'] \) its lub is determined by its elements: \( \bigcup Y(d) = \bigcup \{ f(d) \mid f \in Y \} \).

**Definition 3.17 (Projection).** Let \( D \) and \( D' \) be two CPOs. A projection \(( \phi, \psi )\) of \( D' \) on \( D \) is a pair of functions \( \phi \in [D \to D'] \) and \( \psi \in [D' \to D] \) such that

- \( \psi \circ \phi = 1_D \);
- \( \phi \circ \psi \subseteq 1_{D'} \)

Notice that a projection \(( \phi, \psi )\) makes \( D \) isomorphic to a subset \( \phi(D) \) of \( D' \). Moreover \( \phi(\perp_D) = \perp_{D'} \) and \( \psi(\perp_{D'}) = \perp_D \).

**Definition 3.18 (Isomorphism).** Let \(( \phi, \psi )\) be a projection of \( D' \) on \( D \) for which \( \phi \circ \psi = 1_{D'} \). We say then that \( D \) and \( D' \) are isomorphic and write \( D \cong D' \).

**The model \( D_\infty \)**

The first mathematical model of untyped \( \lambda \)-calculus was exhibited by Scott in [Sco72]. He built a class of complete lattices \( D_\infty \) such that \( D_\infty \) was isomorphic to its continuous function space \( [D_\infty \to D_\infty] \). It turned out later that the same construction can be carried out also on CPOs rather than complete lattices, which could seem better as there are much more CPOs than complete lattices.

Equality in the models \( D_\infty \) has a syntactic characterization in terms of trees: two terms \( M \) and \( N \) receive equal interpretations if and only if they have the same Böhm tree up to \( \eta \)-expansion.

The construction proposed by Scott is an inverse limit construction in the category CPO\( ^E \) of CPOs and projections. \( D_\infty \) is obtained as a limit of a chain \(( D_n, (\phi_n, \psi_n) )_{n \in \omega} \)

\[
\begin{array}{c}
D_0 \xrightarrow{\phi_0} D_1 \xrightarrow{\phi_1} D_2 \xrightarrow{\phi_2} \cdots \xrightarrow{\phi_n} D_n \xrightarrow{\phi_{n+1}} \cdots
\end{array}
\]

where each \( D_{n+1} = [D_n \to D_n] \) for \( n \geq 0 \). There are two parameters to be fixed to successfully carry out the construction: the initial CPO \( D_0 \) and the initial projection \(( \phi_0, \psi_0 )\) of \( D_1 \) on \( D_0 \). The method, indeed, yields a class of models that are usually called \( D_\infty \)-models. From this class, a particular model will be chosen: this becomes our \( D_\infty \). It is characterized by

\[
D_0 = \Sigma = \top \quad \quad (\phi_0, \psi_0) = \text{the standard projection}
\]

**Definition 3.19 (Standard projection).** Let \( D \) be a CPO. The standard projection \(( \phi_0, \psi_0 )\) of \([D \to D] \) on \( D \) is defined by \( \phi_0(x) = d \mapsto x \) and \( \psi_0(f) = f(\perp_D) \).

**Lemma 3.20.** For each projection \(( \phi, \psi )\) of \( D' \) on the CPO \( D \) there is a projection \(( \phi^*, \psi^* )\) of \([D' \to D'] \) on \([D \to D] \) where \( \phi^* \) and \( \psi^* \) are defined by \( \phi^*(f) = \phi \circ f \circ \psi \) and \( \psi^*(g) = \psi \circ g \circ \phi \).

**Proof.** For \( f \in [D \to D] \), \( (\psi^* \circ \phi^*)(f) = \psi^* (\phi \circ f \circ \psi) = \psi \circ (\phi \circ f \circ \psi) \circ \phi = f \) and for \( g \in [D' \to D'] \), \( (\phi^* \circ \psi^*)(g) = \phi^* (\psi \circ g \circ \phi) = \phi \circ (\psi \circ g \circ \phi) \circ \psi \subseteq g. \)

\( \square \)
**Definition 3.21 (The model $D_\infty$).** Let $\Sigma$ be the two-point CPO defined above and let $(\phi_0, \psi_0)$ the standard projection of $[\Sigma \to \Sigma]$ on $\Sigma$. Define then

\[
D_0 = \Sigma \\
D_{n+1} = [D_n \to D_n] \\
(\phi_{n+1}, \psi_{n+1}) = (\phi^n_n, \psi^n_n) \\
D_\infty = \lim\langle D_n, (\phi_n, \psi_n) \rangle_{n \in \omega} \\
d \subseteq D_\infty e \iff (\forall n \in \omega)(d_n \subseteq D_n \wedge e_n)
\]

Observe that the members of $D_\infty$ are infinite sequences of functions, each element of one sequence being a “finite approximation” of the continuous function denoted by the whole sequence.

**Definition 3.22 (Embedding $D_m$ into $D_n$).** From $\phi_n$ and $\psi_n$ define $\phi_{m,n} : D_m \to D_n$ for all $m, n \in \omega$ as:

\[
\phi_{m,n} = \begin{cases} 
\phi_{n-1} \circ \phi_{n-2} \circ \cdots \circ \phi_{m+1} \circ \phi_m & \text{if } m < n \\
1_{D_m} & \text{if } m = n \\
\psi_n \circ \psi_{n+1} \circ \cdots \circ \psi_{m-2} \circ \psi_{m-1} & \text{if } m > n 
\end{cases}
\]

Notice that $\phi_{m,n} \in [D_m \to D_n]$ and additionally $\phi_{n,m} \circ \phi_{m,n} = 1_{D_m}$ if $m \leq n$, $\phi_{n,m} \circ \phi_{m,n} \subseteq 1_{D_m}$ if $m > n$ and $\phi_{n,n} \circ \phi_{m,k} = \phi_{m,n}$ if $k$ is between $m$ and $n$.

**Definition 3.23 (Embedding $D_n$ into $D_\infty$).** For each $d = (d_0, d_1, d_2, \ldots) \in D_\infty$ and $n \in \omega, a \in D_n$ define

\[
\phi_{\infty,n}(d) = d_n \\
\phi_{n,\infty}(a) = (\phi_{n,0}(a), \phi_{n,1}(a), \phi_{n,2}(a), \ldots)
\]

Notice that $(\phi_{n,\infty}, \phi_{\infty,n})$ is a projection of $D_\infty$ on $D_n$.

**Definition 3.24 (Application in $D_\infty$).** For $a, b \in D_\infty$ define

\[
a \cdot b = \bigsqcup_{n \in \omega} \phi_{n,\infty}(a_{n+1}(b_n))
\]

The element $a \cdot b$ can be viewed as the luf of an increasing sequence of approximations $a_{n+1}(b_n)$.

**Proposition 3.25.** $D_\infty \cong [D_\infty \to D_\infty]$, through the functions $\Phi : D_\infty \to [D_\infty \to D_\infty]$ and $\Psi : [D_\infty \to D_\infty] \to D_\infty$ defined respectively as

\[
\Phi(d) = x \mapsto d \cdot x \\
\Psi(f) = \bigsqcup_n \phi_{n,\infty}(y \in D_n \mapsto \phi_{\infty,n}(f(\phi_{n,\infty}(y))))
\]

**Proof.** See [Bar84], Theorem 18.2.16. \qed

**Theorem 3.26.** $(D_\infty, \Phi, \Psi)$ is an environment model.

**Proof.** Recall that the interpretation of $\lambda$-terms in an environment model is:

\[
[x]_\rho = \rho(x) \\
[MN]_\rho = (\Phi([M]_\rho)([N]_\rho)) \\
[\lambda x.M]_\rho = \Psi(d \mapsto [M]_\rho[x:=d])
\]
It has to be shown that the set of continuous functions \([D_\infty \to D_\infty]\) is exactly the set of representable ones on \(D_\infty\) and that the function \(d \mapsto \([M]_{\rho[x \to d]}\) for each valuation \(\rho\), each \(d \in D_\infty\) and \(M \in \Lambda\) is continuous. These are exactly Theorem 5.4.4 and Lemma 5.4.3 of \([Bar84]\).

\(D_\infty\) satisfies some pleasant properties which allow us to call it approximable. This fact will become important when a general approximation theorem will be stated for all approximable models. The approximants for elements of \(D_\infty\) are defined as follows.

**Definition 3.27.** Given \(d \in D_\infty\) and \(n \in \omega\) let

\[
d_{(n)} = \begin{cases} \bot_{D_\infty} & \text{if } n = 0 \\ \phi_{n, \infty}(d_{(n)}) & \text{if } n > 0\end{cases}
\]

**Proposition 3.28.** The following properties for \(x, y \in D_\infty\) and \(m, n, k \in \omega\) with \(n \leq k\) hold:

1. \(d = \bigsqcup_n \{d_{(n)}\}\)
2. \((x_{(n)})_{(m)} = x_{([m \min \{n, m\}]})\)
3. \(x_{(n)} \sqsubseteq x_{(k)} \sqsubseteq x\)
4. \(x_{(0)} \cdot y = x_{(0)} = (x \cdot \bot)_{(0)}\)
5. \(x_{(n+1)} \cdot y = x_{(n+1)} \cdot y_{(n)} = (x \cdot y_{(n)})_{(n)}\)

**Proof.** See \([Bar84]\), Lemma 18.2.8 and Proposition 18.2.13.

**The model \(P^\omega\)**

The model \(P^\omega\) was introduced independently by Plotkin \([Plo72]\) and Scott \([Sco76]\). For reference see also \([Bar84]\), Chapter 18. The main idea introduced in its construction is the disaggregation of all the \(\lambda\)-definable functions in a bundle of pairs \(\langle \alpha, a \rangle\), each of which can be understood as an “elementary instruction” \([Lon83]\), giving output \(a\) whenever the input contains \(\alpha\). Each function is a set of such pairs. The intrinsic higher-order of \(\lambda\)-calculus is accomplished by coding a set of pairs as a single element. To this aim the natural numbers are used.

**Definition 3.29 (\(P^\omega\)).** \(P^\omega = \langle \{x \mid x \subseteq \omega\}, \subseteq \rangle\) is the powerset of the natural numbers, ordered by set inclusion \(\subseteq\).

**Definition 3.30 (Coding of ordered pairs and finite sets).**

- For \(n, m \in \omega\) let \((-,-) : \omega \times \omega \to \omega\) the bijection defined as

\[
(n, m) = \frac{1}{2}(n + m)(n + m + 1) + m
\]

- For \(n \in \omega\) let \(e : \omega \to P^\omega\) the bijection defined as

\[
e(n) = \{k_0, \ldots, k_{m-1}\} \text{ with } n = \sum_{k \leq m} 2^k \text{ and } k_0 < k_1 < \cdots < k_{m-1}
\]
Each choice for \((-,-)\) and \(e\) will yield a different model of \(\lambda\)-calculus. The one above is the canonical one.

\(P^\omega\) is a CPO, being clear what the notion of continuous function \(f : P^\omega \to P^\omega\) should be. The most important property which \(P^\omega\) satisfies is the following.

**Lemma 3.31.** Let \(f : P^\omega \to P^\omega\) be a function.

\[
f \text{ is continuous if and only if } f(x) = \bigcup \{ f(a) \mid a \subseteq x, \text{afinite} \}
\]

**Proof.** See [Bar84], Proposition 18.1.2. \(\square\)

The above mentioned disaggregation is then a safe way to decompose and then recompose continuous functions defined on \(P^\omega\). Moreover, \([P^\omega \to P^\omega] \cong P^\omega\).

**Definition 3.32.** Let \(\varphi : P^\omega \to [P^\omega \to P^\omega]\) and \(\psi : [P^\omega \to P^\omega] \to P^\omega\) the functions defined as follows:

- \(\varphi(u)(x) = \{ m \mid \exists e(n) \subseteq x, (n,m) \in u \}\)
- \(\psi(f) = \{ (n,m) \mid m \in f(e(n)) \}\)

**Proposition 3.33.** \(\langle P^\omega, \varphi, \psi \rangle\) is an environment model.

**Proof.** Proposition 18.1.6 of [Bar84]. \(\square\)

As in the case of \(D_\infty\), it is possible to define “approximations” of continuous functions on \(P^\omega\).

**Definition 3.34 (Projections in \(P^\omega\)).** For \(a \in P^\omega\) and \(n \in \omega\) let

\[
a_{(n)} = \{ m \in a \mid m \leq n \}
\]

**Proposition 3.35.** The following equations are valid in \(P^\omega\):

1. \(d = \bigcup_n d_{(n)}\)
2. \((d_{(n)})_{(m)} = d_{\min(n,m)}\)
3. \(\varnothing \cdot d = \lambda d. \varnothing = \varnothing = \varnothing_{(n)}\)
4. \(d_{(0)} \cdot c = d_{(0,0)} \cdot \varnothing = (d_{(0)} \cdot \varnothing)_{(0)} = (d \cdot \varnothing)_{(0)} = d_{(0)}\)
5. \(d_{(n+1)} \cdot c = d_{(n+1)} \cdot c_{(n)} = (d_{(n+1)} \cdot c_{(n)})_{(n)} \subseteq (d \cdot c_{(n)})_{(n)}\)

**Proof.** See [Bar84], Corollary 18.1.14. \(\square\)

**The model \(D_\emptyset\)**

The model \(P^\omega\) can be seen as a particular instance of a more general class of set-theoretic models of untyped \(\lambda\)-calculus which go under the name of Plotkin-Scott-Engeler algebras (PSE-algebras for short), defined in [Plo93, Sco80a, Eng81].

**Definition 3.36 (Graph model).** A PSE-algebra or graph model \(\langle U, k \rangle\) consists of an infinite set \(U\) together with a retraction \(\langle P^{\leq \omega}(U) \times U, k, k^{-1}\rangle\). It generates an applicative structure \(\langle U = P(U), \cdot \rangle\) where

\[
x \cdot y = \{ a \mid \exists u \subseteq y, u \text{finite} \& k(u, a) \in x \}
\]
The construction of $D_A$ is parametric in a non-empty set $A \neq \emptyset$:

- $B_0 = A$
- $B_{n+1} = B_n \cup \{\langle \beta, b \rangle \mid \beta \subseteq B_n \land b \in B_n\}$
- $B = \bigcup B_n$
- $D_A = \mathcal{P}(B) = \{X \mid X \subseteq B\}$

**Definition 3.37 (Application on $D_A$).** Let $\cdot : D_A \times D_A \to D_A$ be the operation defined by $d \cdot e = \{b \mid (\exists \beta \subseteq e)(\langle \beta, b \rangle \in d)\}$.

**Definition 3.38.** Let $\psi : [D_A \to D_A] \to D_A$ and $\varphi : D_A \to [D_A \to D_A]$ be the functions defined respectively by

$$\psi(f) = \{\langle \beta, b \rangle \mid b \in f(\beta)\} \cup A$$
$$\varphi(u)(x) = \{b \mid \exists \beta \subseteq x, \langle \beta, b \rangle \in u\}$$

**Proposition 3.39.** $\langle D_A, \varphi, \psi \rangle$ is an environment $\lambda$-model.

**Proof.** See Theorem 2.3 and Corollary 3.5 of [Lon83].

Sometimes the environment model $\langle D_A, \varphi, \psi \rangle$ is called “graph model” and is indicated as $(B, k)$ where $k : \mathcal{P}^{<\omega}(B) \times B \to B$ is the injective function induced by $\varphi$ and $\psi$.

**Definition 3.40 (Projections in $D_A$).** Let $b \in D_A$. Let

$$|b| = \begin{cases} 1 & \text{if } b \in A \\ |\beta| + |c| & \text{if } b = \langle \beta, c \rangle \end{cases}$$
$$|\beta| = \max\{|c| \mid c \in \beta\} + 1$$
$$b(n) = \{c \in b \mid |c| \leq n\}$$

**Proposition 3.41.** The projections on $D_A$ satisfy the following properties for $d, e \in D_A$:

1. $d = \bigsqcup d(n)$
2. $d(0) = \perp$
3. $\perp \cdot d = \perp$
4. $d(n+1) \cdot e \subseteq (d \cdot e(n))(n+1)$
5. $(d(n))(m) = d(\min\{n, m\})$

**Proof.** See Lemma 2.7 of [Lon83].

Finally let us choose $A$ as the singleton set $\{\ast\}$. The related $D_A$ construction is termed $D_\ast$. 
The completeness of environment models

All the different presentations of algebraic models we have introduced are equivalent, in the sense that, given a model in a class, there exists always a model in every other class which induces the same λ-theory. Passage from one class to another is effective and reversible. Moreover, each λ-theory has an environment model which induces exactly the identities enforced by the theory itself.

**Theorem 3.42 ([Mey82]).** Every λ-theory (extensional λ-theory) consists of precisely the equations valid in some environment model (extensional environment model).

**Proof.** See [Mey82], Section 3. □

3.2 Categorical models of λ-calculus

Dana Scott [Sc080b], to our knowledge, was the first who pointed out that category theory is more adequate than set-theory as a theory of functions. In a general theory of functions, functions of more arguments have to be considered. Cartesian closed categories allow a general treatment of functions of more arguments (in the sense of Schönfinkel) that is very natural.

It is, of course, obvious to relate λ-calculus - as a theory of functions - with category theory. It turns out [Lam80] that Cartesian closed categories are nothing more than (typed) λ-theories expressed without the aid of variables. Category theory, indeed, does not make use of variables. It expresses identities between constants (the arrows). This result extends also in the untyped case, if we look at untyped terms as terms of a universal type $U$, such that $U \Rightarrow U$ is a retract of $U$.

Scott gave also a completeness result. The theory of functions expressed by each Cartesian closed category with an universal type $U$ is a λ-theory. Conversely, given a type-free λ-theory, there is always a Cartesian closed category with an object $U$ as above in which the resulting theory of functions is exactly the starting λ-theory.

3.2.1 Definitions

Let us define what is the notion of categorical model of untyped λ-calculus.

**Definition 3.43 (Categorical model of λβ-calculus).** Let $D$ be a reflexive object $\langle (D \Rightarrow D) \triangleright D, \psi, \varphi \rangle$ in a CCC. For each term $M \in \Lambda$ with $FV(M) \subseteq \Delta = \{x_1, \ldots, x_n\}$ the interpretation $\llbracket M \rrbracket^D_\Delta : D^\Delta \rightarrow D$ is defined inductively as follows:

\[
\begin{align*}
\llbracket x \rrbracket^D_\Delta &= \pi^\Delta_x \\
\llbracket MN \rrbracket^D_\Delta &= ev^D_D \circ \langle \varphi \circ \llbracket M \rrbracket^D_\Delta, \llbracket N \rrbracket^D_\Delta \rangle \\
\llbracket \lambda x.M \rrbracket^D_\Delta &= \psi \circ \Lambda^D_D : D^\Delta \rightarrow \llbracket D \rrbracket^D_\Delta
\end{align*}
\]

Clearly, a categorical model of λβ-calculus as by Definition 3.43 is only a λ-algebra in the sense of Scott [Sc080b]. We do not imply that such a model “has enough points” in the sense of Barendregt [Bar84].

**Theorem 3.44.** Let $\langle (D \Rightarrow D) \triangleright D, \psi, \varphi \rangle$ be a reflexive object in a CCC and let $M, N \in \Lambda$ be two terms with $FV(MN) \subseteq \Delta$. Thus $M =_\beta N \Rightarrow \llbracket M \rrbracket^D_\Delta = \llbracket N \rrbracket^D_\Delta$.

**Proof.** See [Bar84], Proposition 5.5.5 or [Koy82], Theorem 3.5. □
The interpretation of Definition 3.43 validates all the axioms and rules of λ-calculus and hence also the ξ-rule as it is easy to check. If $M, N$ are terms such that $[M]_\Delta = [N]_\Delta$, then for each variable $x$ either $x \in \Delta$ and hence $\Delta, x = \Delta$ or $x \notin \Delta$ and hence $x \notin FV(M) \cup FV(N)$. This implies that $[M]_\Delta = \psi \circ \Lambda ([M]_{\Delta,x}) = \psi \circ \Lambda ([N]_{\Delta,x}) = [\lambda x.M]_\Delta$.

In a reflexive object $\langle (D \Rightarrow D) \triangleleft D, \psi, \varphi \rangle$, $\varphi \circ \psi = 1_{D \Rightarrow D}$. If we impose that also $\psi \circ \varphi = 1_D$, that is $(D \Rightarrow D) \cong D$, we obtain a categorical model of the $\lambda \beta \eta$-calculus.

**Definition 3.45 (Categorical model of the $\lambda \beta \eta$-calculus).** A categorical model of the $\lambda \beta \eta$-calculus is a reflexive object $\langle (D \Rightarrow D) \triangleleft D, \psi, \varphi \rangle$ in a CCC such that $\varphi \circ \psi = 1_D$.

Notice, in fact, that the rule $\eta$ is valid in such models:

\[
\begin{align*}
[\lambda x.M]_\Delta &= \psi \circ \Lambda ([M]_{\Delta,x}) \\
&= \psi \circ \Lambda (ev \circ (\varphi \circ [M]_{\Delta,x} \otimes [x]_{\Delta,x})) \\
&= \psi \circ \Lambda (ev \circ (\varphi \circ [M]_\Delta \otimes [x]_\Delta)) \\
&= \psi \circ \varphi \circ [M]_\Delta \quad \text{for equation } \eta_{\text{cut}} \\
&= [M]_D
\end{align*}
\]

### 3.2.2 The relation between algebraic and categorical models

Category theory aims to be more general than set-theory. This should imply (as it does) that all algebraic models of $\lambda$-calculus can be expressed as categorical models.

To each categorical $\lambda \beta$-model ($\lambda \beta \eta$-model) can be associated a $\lambda$-algebra (extensional $\lambda$-algebra) in the following way.

**Definition 3.46 (Applicative structure of a reflexive object).** Given a reflexive object $\langle (D \Rightarrow D) \triangleleft D, \psi, \varphi \rangle$ in a CCC, its associated applicative structure is $\langle [D], \cdot \rangle$, where for $x, y \in [D], x \cdot y = ev_{D,D} \circ (\varphi \circ x,y)$.

We aim to show that such an applicative structure can be made into a $\lambda$-algebra. For this purpose let us extend the interpretation of $\lambda$-terms of Definition 3.43 as follows.

**Definition 3.47.** Let $\langle (D \Rightarrow D) \triangleleft D, \psi, \varphi \rangle$ be a reflexive object in a CCC and let $\langle [D], \cdot \rangle$ its associated applicative structure.

- Let $[ \ ]$ the interpretation of Definition 3.43. For each valuation $\rho : \text{Var} \to [D]$ define $\rho^\Delta = \rho^{\{x_1, \ldots, x_n\}} = \langle \rho(x_1), \ldots, \rho(x_n) \rangle$. For each $\lambda$-term $M$ with $FV(M) = \Delta$ and each valuation $\rho$ put $[M]_\rho = [M]_\Delta \circ \rho^\Delta$.
- Put $k = [\lambda x.y.x]^D$ and $s = [\lambda x.y.z.(x)(y)(z)]^D$. Since $\lambda x.y.x$ and $\lambda x.y.z.(x)(y)(z)$ have no free variables, the interpretation is the same for each valuation $\rho$.
- Since $k$ and $s$ are defined by means of $[ \ ]^D$ we speak of a $\lambda$-algebra $\langle [D], \cdot, [k], [s] \rangle$ instead of $\langle [D], \cdot, k, s \rangle$.

**Proposition 3.48.** For each categorical model of the $\lambda \beta$-calculus (categorical model of the $\lambda \beta \eta$-calculus) $D, \triangleleft D, \psi, \varphi$ is a $\lambda$-algebra (extensional $\lambda$-algebra). If $D$ has enough points then $D$ is a $\lambda$-model (extensional $\lambda$-model).

**Proof.** See [Bar84], Theorem 5.5.6. □
The converse also holds: for each $\lambda$-algebra ($\lambda$-model) there exists a reflexive object in a CCC (a reflexive object with enough points) whose applicative structure is an isomorphic $\lambda$-algebra ($\lambda$-model). This result is due to Scott [Sco80b] and is shown in the next section.

### 3.2.3 The completeness of categorical models

Categorical models of $\lambda$-calculus enjoy a general useful property: each $\lambda$-algebra admits a categorical model which enforces exactly the same set of equations between terms.

Given a $\lambda$-algebra $A = \langle A, \cdot, k, s \rangle$, let $\langle A, \cdot, \cdot \rangle$ be the equivalent syntactical model for Theorem 3.42. Put then $b = [\lambda xyz.x(yz)]$ and, for $u, v \in A$, $u \circ v = b \cdot u \cdot v$.

**Definition 3.49.** Given a $\lambda$-algebra $A = \langle A, \cdot, k, s \rangle$, let $C_A$ the category defined as follows:

- **Objects** $\{ u \in A \mid u \circ u = u \}$
- **Arrows** $\{ f : u \to v \mid f \in A, f = v \circ f \circ u \}$
- **Identities** $I_u = u$
- **Composition** $\langle f : u \to v, g : v \to w \rangle \mapsto g \circ f : u \to w$

Such a construction also falls under the name of Karoubi envelope [Bar84]. Notice that it is an application of the only-arrows axiomatic definition of category.

**Proposition 3.50.** $C_A$ is a category.

**Proof.** Let us observe that the associativity of $\circ$ follows from its definition, since $A$ validates all the equations of $\lambda$-calculus. Moreover, if $f : u \to v$ then $f \circ u = v \circ f \circ u = v \circ f \circ u = f$ and analogously $v \circ f = f$.

**Definition 3.51 (Product and function space in $C_A$).** Let $A = \langle A, \cdot, k, s \rangle$ be a $\lambda$-algebra, let $\langle A, \cdot, \cdot \rangle$ be the equivalent syntactical model and $u, v \in A$. Let $K \equiv \lambda xy.x$, $F = \lambda xy.y$ and let, for each $a \in A$, $c_a \in \Lambda(A)$ be the constant denoting the element $a$. Let us define

\[
\begin{align*}
\text{u} \times \text{v} & = [\lambda xy.y(c_u(xK))(c_v(xF))] \\
\pi_u^{u \times v} & = [\lambda x.(c_u(xK))] \\
\pi_v^{u \times v} & = [\lambda x.c_v(xF)] \\
\langle f, g \rangle & = [\lambda xy.y(c_f(x))(c_g(x))] \\
u \Rightarrow v & = [\lambda x.c_u \circ x \circ c_v] \\
\text{ev}_{u,v} & = [\lambda x.c_u((xK)(c_v(xF)))] \\
\Lambda_{u,v} & = h \mapsto [\lambda xy.c_h(\lambda z.zxy)]
\end{align*}
\]

To verify the above equations is indeed a matter of straightforward, albeit boring, calculations. $C_A$ is then Cartesian closed. Moreover, $D = [\lambda x.x]$ is a reflexive object in $C_A$. Observe that each object $u$ in $C_A$ is a retract of $D$ with retractions $u : u \to D$ and $u : D \to u$. Hence also $D \Rightarrow D = [\lambda xy.xy] = 1$ is a retract of $D$ via $1 : (D \Rightarrow D) \to D$ and $1 : D \to (D \Rightarrow D)$.

**Theorem 3.52.** Let $A = \langle A, \cdot, k, s \rangle$ be a $\lambda$-algebra, let $\langle A, \cdot, \cdot \rangle$ be the equivalent syntactical model, let $D = [\lambda x.x]$ be a reflexive object of $C_A$ and let $D' = \langle D, \cdot, \cdot \rangle$ be its associated $\lambda$-algebra. Then $A$ and $D$ are isomorphic.

**Proof.** See [Koy82], Theorem 4.6.
3.3 General properties of models

In this section general properties of models of \( \lambda \)-calculus useful in the study of their theories are presented. For all the approximable categorical models, a general approximation theorem is proved and the theory of the models \( D_\infty, P^\omega \) and \( D_\oplus \) introduced in Section 3.1 is studied.

3.3.1 Contexts and trees

Since the syntactical characterization of the equations induced by a bunch of models of \( \lambda \)-calculus is given by means of equality of trees of terms, we need to relate the general treatment of terms allowed by contexts with the tree form of a term.

**Lemma 3.53.** Let \( M, N \in \Lambda \) be two terms.

\[
LLT(M) \subseteq LLT(N) \implies (\forall C[\ ])(LLT(C[M]) \subseteq LLT(C[N]))
\]

**Proof.** See [Bar84], Corollary 14.3.20.

As a corollary, since Böhm trees are a special case of Lévy-Longo trees, we can extend the result to the Böhm trees.

**Corollary 3.54.** Let \( M, N \in \Lambda \) be two terms.

1. \( BT(M) \subseteq BT(N) \implies (\forall C[\ ])(BT(C[M]) \subseteq BT(C[N])) \)
2. \( LLT(M) = LLT(N) \implies (\forall C[\ ])(LLT(C[M]) = LLT(C[N])) \)
3. \( BT(M) = BT(N) \implies (\forall C[\ ])(BT(C[M]) = BT(C[N])) \)

3.3.2 Böhm-out technique

The term Böhm-out technique refers to the machinery first used by Corrado Böhm in his [Böh68] to separate different \( \beta\eta \)-nf’s. The technique consists in finding a context \( C[\ ] \) capable of showing the differences of two terms. Originally, if \( M \) and \( N \) are two different \( \beta\eta \)-nf’s, the Böhm-out technique allows to find a context \( C[\ ] \) such that \( C[M] =_\beta^\eta t \) and \( C[N] =_\beta^\eta f \).

It deeply relies on the validity of the \( \eta \)-rule, and in almost every case \( C[M] \) is an \( \eta \)-expansion of a subterm \( M' \subseteq M \). Its application in the study of the theory of a model which does not validate the \( \eta \)-rule can be problematic.

**Definition 3.55.** Let \( M, N \in \Lambda \) two terms. We say that \( M \) is equivalent with \( N \), and write \( M \sim N \), if both terms are unsolvable or both terms are solvable with principal hnf’s say \( M \equiv \lambda x_1 \ldots x_n.y M_1 \ldots M_m \) and \( N \equiv \lambda x_1 \ldots x_n.y' N_1 \ldots N_m \) and \( y \equiv y' \) and \( n - m = n' - m' \).

There is not a standard terminology for \( M \sim N \). They are called equivalent in [Bar84] and [Böh68], while they are similar in [Wad76] and inseparable in [Hyl75]. Moreover, in [Hyl75] two terms \( M \) and \( N \) are similar if they are inseparable and \( n = n' \), that is they have the same number of abstractions.

**Definition 3.56.** Let \( s \in \text{Seq}, M,N \in \Lambda \) and \( A,B \in \mathcal{BT} \).

- \( A \sim_s B \) if and only if
  - \( A(s) \uparrow \) and \( B(s) \uparrow \)
- $A(s) = \lambda x_1 \ldots x_n . y$, $B(s) = \lambda x_1 \ldots x_{n'} . y$, $s$ has $m$ successors in $A$ and $m'$ successors in $B$, and $m - n = m' - n'$.

- $M \sim_s N$ if and only if $BT(M) \sim_s BT(N)$

**Lemma 3.57.** $BT(M) =_\eta BT(N) \Rightarrow (\forall s \in Seq)(BT(M) \sim_s BT(N))$.

**Proof.** By Theorem 2.22 $BT(M) =_\eta BT(N) \Rightarrow$

$$(\forall k \in \omega)(\exists A,B \in \mathfrak{S}T)(A \rightarrow^* BT(M), B \rightarrow^* BT(N), A =_k B)$$

Take $k = |s| + 1$. Then $BT(M)(s) \sim A(s) = B(s) \sim BT(N)(s)$. □

The following propositions are due to Böhm [Böh68] and were elaborated by Hyland [Hyl76] and Barendregt [Bar84], Chapter 10.

**Proposition 3.58.** Let $M, N \in \Lambda$. $M \not\equiv_s N \Rightarrow (\exists C[ \ ])(C[M] \not\equiv C[N]).$

**Proof.** See [Bar84], Proposition 10.3.13 and Lemma 10.3.4 □

**Proposition 3.59 ([Böh68]).** Let $M, N \in \Lambda$.

- If $M$ is solvable then

$M \not\equiv N \Rightarrow (\forall P \in \Lambda)(\exists C[ \ ])(C[M] =_\beta P \& C[N] \text{ unsolvable})$

- If $M, N$ are both solvable then

$M \not\equiv N \Rightarrow (\forall P, Q \in \Lambda)(\exists C[ \ ])(C[M] =_\beta P \& C[N] =_\beta Q)$

3.3.3 The approximation theorem

Models of λ-calculus sometimes allow a finitary treatment of the applicative behavior of the interpretations of λ-terms. In such a case they are called approximable. The meaning $[M]$ of a term $M$ can then be characterized by the meaning of some approximate normal forms $A(\Omega)$ of $M$. In this section we state an approximation theorem for a wide variety of categorical models of λ-calculus. We focus on the categorical models since, as we have seen, it is easy to translate algebraic models into the categorical setting.

**Definition 3.60.** A categorical model $\langle D, \varphi, \psi \rangle$ in a CCC $C$, has approximable application or is approximable if:

1. $\langle C(A, B), \sqsubseteq \rangle$ is a partial ordered set for every objects $A, B$;

2. $\forall n \in \omega \exists p_n : D \rightarrow D$ such that

   - (a) $p_n \sqsubseteq p_{n+1}$
   - (b) $\bigcup_{n \in \omega} \{p_n\} = 1_D$;

3. $\forall d, e \in C(D^k, D)$, if $d_n \triangleq p_n \circ d$:

   - (a) $d_0 = \perp$
   - (b) $d_n \sqsubseteq d_{n+1}$
   - (c) $d = \bigcup_{n \in \omega} \{d_n\}$
(d) \((d_n)_m = d_{m \in \{n, m\}}\)

(e) \(\perp \cdot d = \perp\)

(f) \(d_{n+1} \cdot e \sqsubseteq (d \cdot e_n)_{n+1}\)

(g) \(d \cdot e = \bigcup_{n \in \omega} \{d_{n+1} \cdot e_n\}\)

where \(d \cdot e\) stays for \(\text{ev}_{D, D} \circ (\varphi \circ d, e)\);

4. \(\cdot\) is conditionally continuous.

Not all models validate both rules \(\Omega_1\) and \(\Omega_2\). The models of lazy \(\lambda\)-calculus, for instance, do not validate rule \(\Omega_1\). We define then different notions of approximants to deal simultaneously with all the models of \(\lambda\)-calculus and to state a general approximation theorem.

The Böhm-tree \(BT(M)\) of a term \(M \in \Lambda(\Omega)\) is defined as

\[
BT(M) = BT(M[(\lambda x.xx)(\lambda x.xx)/\Omega])
\]

The definition of Lévy-Longo tree is extended similarly.

**Lemma 3.61.** Let \(M, N \in \Lambda(\Omega)\) be terms.

1. If \(M \rightarrow^*_{\Omega_2} N\) then \(LLT(M) = LLT(N)\)

2. If \(M \rightarrow^*_{\Omega_1, \Omega_2} N\) then \(BT(M) = BT(N)\)

**Proof.** Just observe that \(LLT(\Omega M) = BT(\Omega M) = \perp\) and \(BT(\lambda x.\Omega) = \perp\). The thesis then follows from Corollary 3.54. \(\square\)

**Definition 3.62.** For each term \(M \in \Lambda\) we define the following sets of approximate normal forms:

1. \(A^\Lambda(M) = \{A \in \Lambda(\Omega) \mid BT(A) \sqsubseteq BT(M) \text{ and } A \text{ is in } \beta \eta \Omega_1 \Omega_2 - nf\}\)

2. \(A^B(M) = \{A \in \Lambda(\Omega) \mid BT(A) \sqsubseteq BT(M) \text{ and } A \text{ is in } \beta \Omega_1 \Omega_2 - nf\}\)

3. \(A^G(M) = \{A \in \Lambda(\Omega) \mid LLT(A) \sqsubseteq LLT(M) \text{ and } A \text{ is in } \beta \Omega_2 - nf\}\)

The approximants of 1 apply only to models which validate rules \(\Omega_1, \Omega_2\) and \(\eta\). This is the case, for instance, of the model \(D_\infty\). The approximants of 2 apply to models which could not to validate the \(\eta\)-rule but still validate rules \(\Omega_1\) and \(\Omega_2\), as is the case for the model \(P^\omega\).

The approximants of 3 apply to models which validate only the \(\Omega_2\)-rule, as the model \(D_\boxtimes\) of [Lon83].

Let us observe that in [Wad76, Wad78] and [HR92] different sets of approximants are defined. Nevertheless all these different definitions allow to state the same result.

**Definition 3.63.** For each term of labelled \(\lambda\)-calculus \(M \in \Lambda(\Omega)^N\), set \(BT(M) = BT([M])\) and \(LLT(M) = LLT([M])\) and extend the definition of interpretation of terms as follows:

\[
[M^n]_\Lambda = ([M]_\Lambda)^n \\
[\Omega]_\Lambda = \perp_{D_1, 1 \rightarrow D}
\]

**Theorem 3.64 (Validity of indexed reduction).** Rules \((\Omega^l), (\Omega^r), (\beta_1), (\beta_\iota)\) are valid in each approximable categorical model \(D\). The Validity of a rule \(\alpha\) is intended in the following sense: for each \(P, Q \in \Lambda(\Omega)^N\) if \((P \rightarrow^\alpha Q)\) then \([P]_\Lambda \sqsubseteq [Q]_\Lambda\).
Proof.

\[(\Omega^0)\quad \text{by Definition 3.60(3a).}\]

\[(\Omega^n) \quad [\Omega^n]^D_{\Delta} = ([\Omega]^D_{\Delta})^n \quad \text{by Definition 3.63}\]

\[= (\perp_{D=1} D) \quad \text{by Definition 3.63}\]

\[= \perp_{D=1} D \quad \text{by Definition 3.60(3d)}\]

\[= [\Omega]^D_{\Delta} \quad \text{by Definition 3.63}\]

\[(\beta_l) \quad [((\lambda x. P)^{m+1} Q)]^D_{\Delta} \quad \text{by Definition 3.43}\]

\[= [((\lambda x. P)^{m+1})^D_{\Delta} \cdot [Q]^D_{\Delta}] \quad \text{by Definition 3.63}\]

\[= ([\lambda x. P]^D_{\Delta} \cdot [Q]^D_{\Delta})_{m+1} \quad \text{by Definition 3.60(3f)}\]

\[= ([\lambda x. P]^D_{\Delta} \cdot [Q]^D_{\Delta})_{m+1} \quad \text{by Definition 3.63}\]

\[= ([((\lambda x. P)^{m+1} Q)]^D_{\Delta} \quad \text{by Definition 3.63}\]

\[(\beta_{i,j}) \quad [(M^i)^j]^D_{\Delta} = ([M]^D_{\Delta})^j \quad \text{by Definition 3.63}\]

\[= (([M]^D_{\Delta})^j) \quad \text{by Definition 3.63}\]

\[= [M^m]^{(i,j)} \quad \text{by Definition 3.63}\]

\[\square\]

**Definition 3.65.** Let \(M^C\) be the class of all approximable categorical models of \(\lambda\beta\)-calculus \((\langle D \Rightarrow D \rangle_\Delta, \varphi, \psi)\) in a CCC \(C\). We define the following subclasses:

1. \(M^C_\Delta = \{\langle (D \Rightarrow D) \varphi, \psi \rangle \in M^C \mid \psi \circ \varphi = 1_D\}\)
2. \(M^C_\Delta \downarrow = \{\langle (D \Rightarrow D) \varphi, \psi \rangle \in M^C \mid \psi \circ \perp = \perp \text{ and } \psi \circ \varphi \neq 1_D\}\)
3. \(M^C_\Delta = \{\langle (D \Rightarrow D) \varphi, \psi \rangle \in M^C \mid \psi \circ \perp \neq \perp\}\)

**Lemma 3.66.** Let \(M, N \in \Lambda(\Omega)^0\), \(D^C \in M^C_{\Delta}, D^S \in M^C_{\Delta}, D^C \in M^C_{\Delta}\). Let \(\alpha_1 = \beta_l \beta_{i,j} \Omega^0 \Omega^0 \Omega^0 \Omega^0\) and \(\alpha_2 = \beta_l \beta_{i,j} \Omega^0 \Omega^0 \Omega^0 \Omega^0\).

1. If \(M \rightarrow^{*_{\alpha_1}} N\) then \([M]^D^C \subseteq [N]^D^C\) and \([M]^D^S \subseteq [N]^D^S\)
2. If \(M \rightarrow^{*_{\alpha_2}} N\) then \([M]^D^C \subseteq [N]^D^C\)
3. If \(M \rightarrow^{*_{\alpha_1}} N\) then \(BT(N) \subseteq BT(M)\)
4. If \(M \rightarrow^{*_{\alpha_2}} N\) then \(LLT(N) \subseteq LLT(M)\)
5. \(\forall A \in A^C(M) \quad [A]^D^C \subseteq [M]^D^C\)
6. \(\forall A \in A^S(M) \quad [A]^D^S \subseteq [M]^D^S\)
7. \(\forall A \in A^C(M) \quad [A]^D^C \subseteq [M]^D^C\)

**Proof.** 1 and 2 are direct consequences of Theorem 3.64 and Definition 3.60.4, 3 and 4 follow by observing that \(BT(\Omega)(LLT(\Omega)) = \perp \subseteq BT(M)(LLT(M))\) for each \(M\) and then concluding the thesis by Lemma 3.61 and Lemma 3.53. 5, 6 and 7 are direct consequences of 1, 2, 3 and 4. \(\square\)
Lemma 3.67. For each completely indexed term \( M \in \Lambda(\Omega)^N \) there exist terms \( N^E, N^B \) and \( N^C \in \Lambda(\Omega)^N \) such that:

1. \( [M]^D^E \subseteq [N^E]^D^E \) and \( [N^E] \in \mathcal{A}^E([M]) \)
2. \( [M]^D^B \subseteq [N^B]^D^B \) and \( [N^B] \in \mathcal{A}^B([M]) \)
3. \( [M]^D^C \subseteq [N^C]^D^C \) and \( [N^C] \in \mathcal{A}^C([M]) \)

Proof. Take \( N^E \) as the \( \Omega^n \Omega^b \beta_1 \beta_i \eta \)-normal form of \( M \) and \( N^B \) and \( N^C \) as the \( \Omega^n \Omega^b \beta_1 \beta_i \beta_j \)-normal form of \( M \). The thesis then follows from Lemma 3.66. \( \square \)

Lemma 3.68. Let \( M \in \Lambda \), \( M^* \) be a completely indexing of \( M \) and \( D \) an approximable categorical model of \( \lambda \beta \)-calculus

\[
[M]^D = \bigcup \{ [M^*]^D \}
\]

Proof. Structural induction on \( M \) using property 3.60.3c. \( \square \)

Theorem 3.69 (Approximation theorem). For each term \( M \in \Lambda \) and approximable categorical models \( D^E \in \mathcal{M}^E \), \( D^B \in \mathcal{M}^B \) and \( D^C \in \mathcal{M}^C \):

1. \( [M]^D^E = \bigcup \{ [A]^D^E \mid A \in \mathcal{A}^E(M) \} \)
2. \( [M]^D^B = \bigcup \{ [A]^D^B \mid A \in \mathcal{A}^B(M) \} \)
3. \( [M]^D^C = \bigcup \{ [A]^D^C \mid A \in \mathcal{A}^C(M) \} \)

Proof. For 1 let us observe the following:

\[
[M]^D^E = \bigcup \{ [M^*]^D^E \mid M^* \text{ a c.i. of } M \} \quad \text{by Lemma 3.68} \\
\subseteq \bigcup \{ [N]^D^E \mid N \in \Lambda(\Omega)^N, [N] \in \mathcal{A}^E(M) \} \quad \text{by Lemma 3.67} \\
\subseteq \bigcup \{ [A]^D^E \mid A \in \mathcal{A}(M) \} \\
\subseteq [M]^D \quad \text{since } [N]^D^E \subseteq [N]^D^E \quad \text{by Lemma 3.66.}
\]

Points 2 and 3 follow from similar chains. \( \square \)
II

Game semantics for Lambda Calculus
Game Semantics

“We see Game Semantics as potentially providing a very powerful unifying framework for the semantics of computation, allowing typed functional languages, concurrent processes and complexity to be handled in an integrated fashion.”

S. Abramsky [Abr95]

Abstract

The basic notions of the game semantics paradigm introduced by Abramsky et al. [AJ94a, AJM96] are presented, together with the referring categories Games, $G$, $G^*$ and $K_3(G)$ that will be utilized later. The concept of approximating strategy – new at least in this general form – is developed in the last section and the proofs of the properties satisfied by an approximating strategy are carried out.

The introduction of the game semantics paradigm for the denotational semantics of programming languages [Abr95, AJ94a, AJM96, HO00, Bla92, Joy93, Nic96] takes its origin in the need for an intensional semantics, a way of assigning meaning to a program not in terms of its external behavior (as classical denotational semantics did) nor in terms of its syntactical structure (as operational semantics did). This new semantic paradigm was intended as a bridge between classical syntactical operational semantics and classical extensional denotational semantics. The classical operational semantics, counts as principal benefit the capability of capturing the dynamical aspects of computation by defining the behavior of a machine interpreting the language. On the other side, classical denotational semantics interprets data types as domains (mathematical structured sets) and programs as suitable computable functions between domains, abstracting away from the dynamics of the computation, choosing static objects to denote programs. Nevertheless, classical denotational semantics has been very successful in providing a solid interpretation framework for a great variety of languages (i.e. functional, imperative, concurrent etc.) albeit in some cases (e.g. imperative and concurrent languages) it is not immediately apparent the appropriateness of modeling programs with functions.
There are some aspects of computation frequently arising in computer science that have not been captured in a satisfactory way by classical denotational semantics. We refer to sequentiality (the PCF problem [Pl77, Mi77, AJM96, HO00, Ni96]), computational complexity and optimality of reduction strategies. These aspects are too abstract to be convincingly expressed in an operational semantics setting and, at the same time, they are too dynamical in flavour to be reasonably caught by static denotational semantics. The new paradigm of intensional semantics intends to satisfy this need and can be considered under the slogans, “denotational semantics + dynamics” or “syntax-free operational semantics”.

In this chapter we explore the possibilities given by an intensional semantics that has gained much consideration in the last years: game semantics.

4.1 Basic definitions

4.1.1 The notion of game

The notion of game arises many times in the mathematical literature to denote a situation in which there has to be a dialogue, a communication, a dispute or, more in general, an interaction among some entities.

The history of games in mathematics and logics can be long and debatable. Wilfred Hodges dates back to the 5th century BC, with the interviews of Socrates to young men in the marketplace, the appearance of games in logic, while the logician Zermelo in 1913 signs the first published paper on games in the mathematical literature.

It is dated to 1944 the first comprehensive attempt to use games as mathematical models of a real situation. We refer to the book [vNM44] by John von Neumann and Oskar Morgenstern, where various economic conflict situations are analyzed as two or more person games. Anyway, their games, while representing very well the dynamical interaction among a number of persons, admit a naive set-theoretic definition.

More strictly related to computer science is the paper [Tur50], where the problem of the “artificial” intelligence (can machines think?) is defined by means of a “game”. Turing envisioned a well-known situation that he called imitation game. This is played by three people: a man (A), a woman (B) and a questioner (C). All of them communicate by means of a console and reside in different rooms. The questioner has to guess who of the other two is the man and who is the woman. In achieving its goal, he makes questions and receives answers. The goal of (A) is trying to confuse (C) while the goal of (B) is better that of saying the truth. If we then substitute a machine for (A) in such a game, can we expect ourselves to observe the same results or does it become easier for the questioner to answer in the right way?

While the first connection between games and meaning can be found in the thirties in Wittgenstein [Wit31], it is the latter idea of Lorenzen [Lon60] of describing a constructive proof in logic as a game which seems to have deeply influenced the latter developments [Abr94b] of denotational semantics. The basic idea of Lorenzen is to interpret a formula as a two-person game, played between the "Proponent" asserting the thesis, and the "Opponent" seeking to refute it. A proof of the formula is a winning strategy for Proponent. From the analogy between proofs and programs and between types and formulas these ideas are readily applied to the denotational semantics of programming languages.

The first ancestors of the games we shall utilize are those of Blass [Bla72] and Conway [Con76], although it is Joyal [Joy77] who first built out a category on the Conway games. Further improvements came with Andreas Blass in 1992, who introduced the game semantics.
for Linear Logic. In his [Bla92] and the successive [Bla95], he developed, with the influence of Girard [Gir87], the idea of thinking at the computation as a dispute between the program (or the system) and its environment, for the request and the supply of data. In such a framework, a type is a server from which a client (the program) can get an element (data) of that type [Bla95]. If the type is simple, the access happens in only one step. Compound types have indeed more elaborate accesses, they have access protocols, i.e. sequences of actions. The meaning of a program is then the set of protocols it is capable to execute. In other words, a program is identified by the possible interactions it may have with the different environments. This set of protocols is called in subsequent works strategy.

The last significant genetic mutation on games is that introduced by Abramsky and Jagadeesan [AJ94a] with the prominent role played by the notion of move which was only implicit and not fundamental in the Conway games. Moreover, they exhibited a category of games (two for the sake of precision) and shown that the composition of Blass strategies is not associative and hence that we cannot speak of a category of Blass games.

These ideas have been extensively developed by other authors, see for reference the papers [AJM96, AM95a, AM98, HO00, Nic96, HY97, McC96, DGFH99, KNO01, KNO09].

The games used in denotational semantics aim to model the dynamic interaction between a Program (or a System) and its Context (its Environment) during a computation. Thus they are two-person games, with one player representing the Program – the Player – and the other representing the Context – the Opponent. The choice of who is the Player and who is the Opponent is somewhat arbitrary since depends on the point of view. If Tim, Tom and Tony converse in a room, from Tim’s point of view, he is the System and Tom and Tony form the Environment, while from Tom’s point of view, he is the System and Tim and Tony form the Environment.

A computation is represented by a sequence of moves, made by the two players alternatively. What is the nature of the moves or how much essential for the whole definition they are depends on the particular application of game semantics. A general indication come from [vNM44]:

“A move is the occasion of a choice between various alternatives, to be made either by one of the players, or by some device subject to chance, under conditions precisely prescribed by the rules of the game”.

More formally, a game is a set of sequences of moves which is non-empty and prefix-closed – that is a tree. For convention it is the Opponent who moves first.

A game has three components: the players, the moves of the game and the rules of the game. The rules of a game describe what moves can be made by a player in a particular moment of the development of the game. It is natural (and is customary in game semantics literature) to define the rules of the game implicitly by restricting the set of all sequences of moves only to those that satisfy the rules.

A particular sequence of moves (a single run of the game) is termed a play of the game. The alternative term position used in [AJM96] for plays seems to be not completely adequate and consistent, since it is sometimes used in the literature [Con76] to denote an equivalence class of plays with respect to the positional equivalence relation. In a positional game, for each move m there is exactly one position ending with move m.

**Notation** In what follows, the following notions will be used. If X is a set, X* is the set of all the finite sequences of elements of X. The empty sequence – the sequence of zero
elements – is denoted by $\epsilon$. Concatenation of sequences is indicated by juxtaposition. If $a \in X$ and $s \in X^*$, $a \cdot s$ ($s \cdot a$) is the sequence obtained prefixing (postfixing) $s$ with the symbol $a$. When no confusion can arise $a \cdot s$ ($s \cdot a$) will be written more simply as $(sa)$. $s \subseteq t$ indicates that $s$ is a prefix of $t$, that is there exists a sequence $u$ such that $t = su$. Given a set $S \subseteq X^*$, $S^{\text{prefix}}$ is the non-empty prefix closure of $S$, that is the set of prefixes of elements of $S$. Notice that $S^{\text{prefix}}$ contains always at least the empty sequence, so if $S = \emptyset$, $S^{\text{prefix}} \neq \emptyset$. $S$ is prefix-closed if $S^{\text{prefix}} = S$.

If $f : X \to Y$ then $f^* : X^* \to Y^*$ is the unique monoid homomorphism extending $f$. $|s|$ is the length of a finite sequence $s$, while $s_i$ is the $i$-th element of $s$, $1 \leq i \leq |s|$. Given a set $S$ of sequences, $S^{\text{even}}$ and $S^{\text{odd}}$ denote respectively the subsets of sequences of even and odd length. If $Y \subseteq X$ and $s \in X^*$, $s \upharpoonright Y$ is the sequence obtained from $s$ by dropping all symbols not in $Y$.

**Definition 4.1 (Game [Abr95]).** A game $G$ is a structure $(M_G, \lambda_G, P_G)$ where

- $M_G$ is the set of moves of the game
- $\lambda_G : M_G \to \{P, O\}$ is a labelling function designating each move as by Player or Opponent
- $P_G \subseteq S^{\text{prefix}} M_G^{\text{alt}} \subseteq M_G^*$, where $M_G^{\text{alt}}$ is the set of alternating sequences of moves starting with an Opponent move:

$$M_G^{\text{alt}} = \{s \in M_G^* \mid \lambda_G(s_1) = O, (\forall 1 \leq i < |s|)(\lambda_G(s_i) \neq \lambda_G(s_{i+1}))\}$$

**Example 4.2 (A finite game).** Consider the following game $G$:

$$G = \langle M_G = \{a_1, a_2, b_1, b_2, b_3\}, \lambda_G = \{\{a_1, a_2\} \mapsto O, \{b_1, b_2, b_3\} \mapsto P\}, P_G = \{\epsilon, a_1, a_1 b_1, a_1 b_1 a_1, a_1 b_1 a_1 a_2, a_2 b_2, a_2 b_3\} \rangle$$

where $P_G$ can be represented as:

```
  a1  a2
 /\   /\  \\
 b1  b2  b3
```

and $\Box$ indicates that the Opponent has to move while $\blacksquare$ that it is the turn of the Player.

**Example 4.3 (An infinite game).** Of course, games can be also infinite objects as it is the case for the following example taken from [DGFH99].
\[ \mathcal{N} = \langle M, \lambda, P \rangle \]

**Definition 4.4 (Subgame).** Given a game \( G \), a subgame \( H \) of \( G \), \( H \subseteq G \) is a game such that

- \( M_H \subseteq M_G \)
- \( \lambda_H = \lambda_G \mid M_H \)
- \( P_H \subseteq P_G \)

### 4.1.2 Construction on games

Committed to develop a denotational semantics based on games we have, sooner or later, to face the need for compound games, since denotational semantics is compositional for its own nature (and this is its best feature). The standard constructions we shall present, owed much to Linear Logic, since they have been introduced to provide a model for that formal system. Our intention is, however, to model \( \lambda \)-calculus which is indeed only a little step far away. It is more convenient to define the Cartesian product of games, fundamental for the denotational semantics of \( \lambda \)-calculus, using the Kleisli construction [Mac71] onto the symmetric monoidal closed category of games \( \mathcal{G} \) rather than specifying it directly.

**Tensor product**

The tensor product \( A \otimes B \) of two games \( A \) and \( B \), is a game which engages the Player in a double competition, playing against the Opponent of \( A \) and, at the same time, against the Opponent of \( B \). However only 1-to-1 duels are admitted, so the Player faces one contender at a time. Formally:

**Definition 4.5 (Tensor product).** Given games \( A \) and \( B \) the tensor product \( A \otimes B \) is the game defined as follows:
\[M_{A \oplus B} = M_A + M_B\]
\[\lambda_{A \oplus B} = [\lambda_A, \lambda_B]\]
\[P_{A \oplus B} = \{s \in M_{A \oplus B}^\overline{\text{alt}} \mid s \mid M_A \in P_A \& s \mid M_B \in P_B\}\]

Here $+$ denotes disjoint union of sets, that is $A + B = \{\text{in}_l(a) \mid a \in A\} \cup \{\text{in}_r(b) \mid b \in B\}$, and $[-,-]$ is the usual (unique) decomposition of a function defined on disjoint unions.

The game $A \oplus B$ allows a play to proceed in both the subgames $A$ and $B$ in an interleaved fashion. A play $s$ of $A \oplus B$ is then the result of the disjoint parallel composition of two plays $s_1$ of $A$ and $s_2$ of $B$. For this reason a tensor product game is also called non-communicating parallel composition.

The alternating moves constraint is a strong one, also if this is not immediately apparent. A first consequence is that, in a tensor product game, only Opponent is allowed to switch subgame. The Player must always respond in the same subgame that Opponent have just moved in.

**Lemma 4.6 (Switching condition [Abr95]).** In any play $s \in P_{A \oplus B}$ if $s_i$ and $s_{i+1}$ are moves in different subgames then $\lambda_{A \oplus B}(s_i) = P$ and $\lambda_{A \oplus B}(s_{i+1}) = O$.

**Proof.** Define for a sequence of moves $s \ p(s)$ as the parity of $|s|$ with the correspondence even $\mapsto O$ and odd $\mapsto P$ since after a play of even parity is the turn of the Opponent and vice versa. Define the state $s^{-1} = \langle p(s \mid A), p(s \mid B) \rangle$. Initially $s^{-1} = \langle O, O \rangle$. Hence $O$ can move in either subgame. If it moves in $A$ then the state becomes $\langle P, O \rangle$ and the Player has to move in $A$. If it makes a new move the state comes back to $\langle O, O \rangle$. The same happens for an Opponent move in $B$. The situation can be convincingly depicted in the following state transition diagram:

\[\begin{array}{c}
P \quad \langle P, O \rangle \\
\langle O, O \rangle \\
O \quad \langle O, P \rangle
\end{array}\]

\[\square\]

**Definition 4.7 (Unit).** The unit game $I$ of the tensor product is the empty game $\langle \emptyset, \emptyset, \{\varepsilon\} \rangle$.

**Linear implication**

If the tensor product game gives additional work to the Player, leaving it no control on the game, the linear implication $A \rightarrow B$ of two games $A$ and $B$ marks a revenge, rising it to the role of master and leaving to the Opponents of $A$ and $B$ the role of slaves. Linear implication is intended to model the “functional space” from $A$ to $B$ and hence the Player, which is intended to compute a function, is allowed to switch freely from the output $B$ to the input $A$ and back to get all the information necessary to carry out its calculations.

**Definition 4.8 (Linear implication).** Given games $A$ and $B$ the compound game $A \rightarrow B$ is defined as follows:

\[M_{A \rightarrow B} = M_A + M_B\]
\[\lambda_{A \rightarrow B} = [\lambda_A, \lambda_B]\]
\[P_{A \rightarrow B} = \{s \in M_{A \rightarrow B}^\overline{\text{alt}} \mid s \mid M_A \in P_A \& s \mid M_B \in P_B\}\]

where $\overline{\lambda_A}$ is defined as:
\[
\overline{\lambda}_A(m) = \begin{cases} 
P & \text{if } \lambda_A(m) = O \\
O & \text{if } \lambda_A(m) = P 
\end{cases}
\]

The only difference with the definition of the tensor product is the labelling function on moves. However this is a crucial difference since it reverses the roles of Player and Opponent in the capability of switching subgames.

**Lemma 4.9 ([Abr95]).** In any play \( s \in P_{A \rightarrow B} \) if \( s_i \) and \( s_{i+1} \) are moves in different subgames then \( \lambda_{A \rightarrow B}(s_i) = O \) and \( \lambda_{A \rightarrow B}(s_{i+1}) = P \).

**Proof.** Observe that, since it is the Opponent to start, the initial move has to happen in \( B \) because the first move in \( A \) is labelled by \( P \) in \( A \rightarrow B \). As in the proof of Lemma 4.6 we can depict the situation as:

![Diagram](image)

which shows that only the Player can switch subgames. \( \square \)

The roles of Player and Opponent are interchanged in the subgame \( A \) as the definition of \( \overline{\lambda}_A \) witnesses. We can figure out this change of roles as the different characters of the System (Player) and the Environment (Opponent) in the input and output cases: on the output side the System is the producer and the Environment the consumer while in the input side the opposite happens to be.

**Exponential game**

A function can often need to use its argument more then once as is the case for \( f(x) = \lambda x.x + x \). For this purpose Linear Logic introduces modalities and the same has to be done on games. The exponential game \( !A \) consists of an infinite supply of copies of the game \( A \). Formally:

**Definition 4.10 (Exponential).** Given a game \( A \) the game \( !A \) is defined by:

- \( M_{!A} = \omega \times M_A = \sum_{i \in \omega} M_A \)
- \( \lambda_{!A} = \langle i, a \rangle \mapsto \lambda_A(a) \)
- \( P_{!A} = \{ s \in M_{!*A} \mid (\forall i \in \omega)(s \upharpoonright \{ i \} \times M_A \in P_A) \} \)

### 4.1.3 Strategies

A strategy is a set of rules which tells the Player which move to make at a given instant of the development of a game. A natural question which arises is what is the next move determined by. Different answers to this question give rise to the various implementations of game semantics. In particular:

- if the answer is *everything* then we obtain the *history-sensitive* strategies that form the category \( G^* \) (Definition 4.33) we shall utilize;
\begin{itemize}
  \item if the answer is \textit{the last move} then we obtain the \textit{history-free} strategies that form the category \( \mathcal{G} \) (Definition 4.21) in which we carry out an extensive study of models of \( \lambda \)-calculus;
  \item if the answer is \textit{the current position} then we are referring to Conway games \[\text{Con76}\] where, unfortunately, the composition of strategies is problematic;
  \item if the answer is \textit{the current view} then we obtain the \textit{innocent} strategies of Hyland and Ong \[\text{HO00}\] and Nickau \[\text{Nic96}\], studied deeply in \[\text{McC96}\].
\end{itemize}

In each case, a strategy is given implicitly as a subset of the plays of a game.

\textbf{Definition 4.11.} \textit{A strategy for the Player in a game} \( G \) \textit{is a non-empty set} \( \sigma \subseteq (P_G)^{\text{even}} \) \textit{of plays of even length such that, termed} \( \operatorname{dom}(\sigma) = \{ t \in (P_G)^{\text{odd}} \mid (\exists a)(ta \in \sigma) \} \), \( \overline{\sigma} = \sigma \cup \operatorname{dom}(\sigma) \) \textit{is prefix-closed}.

Moreover, a strategy \( \sigma \) for a game \( G \) will be said \textit{deterministic} if whenever \( sab, sac \in \sigma \) then \( b = c \).

\textbf{Set-theoretic description of strategies}

A strategy \( \sigma \) can be seen as a generalization of a relation \( S \) on the set of moves. A relation can be understood as a list of stimulus-response items. If \( \langle a, b \rangle \in S \), then we can think that a response \( b \) can happen in occasion of a stimulus \( a \).

On the other hand, if \( s \equiv a_1b_1 \ldots a_nb_n \in \sigma \) then we can figure out that the following situation is expressed:

\begin{itemize}
  \item if the Opponent initially does \( a_1 \) then respond with \( b_1 \);
  \item if the Opponent then does \( a_2 \) then respond with \( b_2 \);
  \item \ldots
  \item if the Opponent then does \( a_n \) then respond with \( b_n \).
\end{itemize}

Hence, quoting from Abramsky, \textit{strategies are relations extended in time}, since they describe repeated interactions of the type stimulus/response.

Usually we can have different possible responses to a single stimulus. A deterministic strategy generalizes, then, the notion of single-valued relation, that is the notion of function. Deterministic strategies are then \textit{partial functions extended in time}.

\textbf{History-free strategies}

History-free strategies suggest the next move to make analyzing only the last move made by the opposite player. They deserve special attention, since they induce and are induced by partial functions on the moves of a game.

\textbf{Definition 4.12 (History-free strategy).} \textit{A strategy} \( \sigma \) \textit{for a game} \( G \) \textit{is history-free if it satisfies the following properties:}

1. \( sab, tac \in \sigma \Rightarrow b = c \)
2. \( sab, t \in \sigma, ta \in P_G \Rightarrow tab \in \sigma \)
Given a game $G$, let $M_G^P$ and $M_G^O$ be the disjoint subsets of moves labelled respectively as Player and Opponent. Given an history-free strategy $\sigma$ for the game $G$, let us define $f_\sigma : M_G^O \to M_G^P$ by

$$f_\sigma(a) \geq b \iff (\exists s)(sab \in \sigma)$$

where $f_\sigma(a) \geq b$ indicates that $f_\sigma(a)$ is defined and equal to $b$. If $f : M_G^O \to M_G^P$ is a partial function define $\text{traces}(f) \subseteq M_G^{\text{all}}$ inductively by

$$\text{traces}(f) = \{\epsilon\} \cup \{sab \mid s \in \text{traces}(f), sa \in P_G, f(a) \geq b\}$$

The function $f$ induces the strategy $\sigma_f$ if $\text{traces}(f) \subseteq P_G$. Notice that

$$f_{\sigma_f} \subseteq f \quad \sigma_{\sigma_f} = \sigma$$

and hence there is always a canonical least partial function inducing a history-free strategy.

**Example 4.13.** Consider the boolean game $\mathbb{B}$ defined by

$$\mathbb{B} = \langle\{*, t, f\}, \{* \mapsto O, \{t, f\} \mapsto P\}, \{\epsilon, *, t, *f\}\rangle$$

There are three possible (history-free) strategies for the Player:

$$\bot = \{\epsilon\} \quad \text{true} = \{\epsilon, *t\} \quad \text{false} = \{\epsilon, *f\}$$

**Copy-cat strategies**

Game semantics we are just presenting is born as a denotational semantics for Linear Logic. This, of course, has some implications. The only axiom contemplated in Linear Logic is the tautology $A \vdash A$. It corresponds - under the isomorphism of Curry-Howard - to the so called copy-cat strategy $1_A$ which is the basic building block of the category of games $\mathcal{G}$, being the identity morphism. The strategy $1_A$ for the game $A \rightarrow A$ (where $A$ is a given game) tells the Player to repeat the move made by the Opponent in one occurrence of the game $A$ to the other occurrence. It depends only on the last move made by the Opponent, so $1_A$ is history-free. It proceeds as follows:

$$A_1 \rightarrow A_2$$

$$m_1$$

$$m_2$$

$$\vdots$$

We need to distinguish the two occurrences of the game $A$ in $A \rightarrow A$ for ease of reference. Formally:

$$1_A = \{s \in P_{A_1 \rightarrow A_2}^{\text{even}} \mid s \mid A_1 = s \mid A_2\}$$
where \( s \models A_1 \) is an abbreviation for \( s \models M_{A_1} \). If we have the temptation to look at the condition \( s \models A_1 = s \models A_2 \) as very liberal, and maybe too weak, to determine the resulting \( s \), we have to dismiss it, since the alternating moves constraint and the prefix-closed condition for a strategy strongly limits the cases. It is clear that if \( s \models A_1 = m_1m_2 \cdots m_k = s \models A_2 \) then \( s = m_1m_1m_2m_2 \cdots m_km_k \) is the only possibility.

As an illustration of the power of the copy-cat strategy, let us consider the possibility of engaging a double chess challenge, with no time constraints, against Kramnik and Kasparov with the certainty of winning one game (or, at least, with the certainty of drawing). We only have to play as white against Kasparov and as black against Kramnik (or vice versa, of course).

When Kramnik makes his starting move, we reply this move on the board we are playing against Kasparov. We wait for Kasparov to reply and then copy that move to the game against Kramnik and so on. This is what the copy-cat strategy tells us. If Kramnik, for instance, wins the game then there are us who win the game against Kasparov and vice versa. It can be argued that we are nothing else that a buffer to connect Kramnik and Kasparov, and it is true, but this is the role of the variable in \( \lambda \)-calculus: repeating to the System what the Environment assigns to it.

The application strategy

In Linear Logic, as in all other logical systems, theorems are formed by repeatedly applying inference rules to the axioms. If the copy-cat strategies are the axioms in game semantics, the application strategy is the equivalent of *modus ponens* rule. An application strategy is a concatenation of two copy-cat strategies:

\[
ev_{A,B} = \{ s \in P((A_1 \rightarrow s = A_2) \rightarrow B_2) \mid s \models A_1 = s \models A_2 \& s \models B_1 = s \models B_2 \}\]

that is the application strategy \( \ev_{A,B} \) is the way of “applying” a strategy for the functional game \( A \rightarrow B \) to a strategy for an argument \( A \) by connecting the two occurrences of the games \( A \) and \( B \). This application strategy can be illustrated as follows:

\[
((A \rightarrow B) \otimes A) \rightarrow B
\]

\[
b_1 \quad \text{(req. for output)}
\]

\[
a_1 \quad \text{(req. for input)}
\]

\[
\vdots \quad \text{(input data)}
\]

\[
\vdots \quad \text{(output data)}
\]

The initial request for output of the application function is copied to the output side of the function argument; the function’s argument request for input is copied to a request for
Composition of strategies

If we want to build a category of games, we have to find a way to compose strategies. The meaning of such a composition has to be clear: if $\sigma$ tells the Player how to move in a game $A \rightarrow B$ and $\tau$ tells him how to move in a game $B \rightarrow C$ then the compound strategy $\tau \circ \sigma$ has to tell how to move in the game $A \rightarrow C$. What comes very natural for the Player, is to use $\tau$ to move after the Opponent in $C$ and use $\sigma$ for the moves in $A$. However also $B$ has to be taken in consideration since either $\sigma$ and $\tau$ may propose moves in $B$.

For the structure of the game $A \rightarrow C$ the Opponent has to move first in $C$ with a move, say, $c_1$. If $\tau$ indicates to respond with a move in $C$, say $c_2$, then this is the move which $\tau \circ \sigma$ indicates to answer to $c_1$. If $\tau$ indicates to respond with a move in $B$, say $b_1$, then such a move made by the Player according to $\tau$ for the game $B \rightarrow C$ is termed as an Opponent move in the game $A \rightarrow B$. Hence immediately the Player can apply, to this move, the strategy $\sigma$ to go on. If $\sigma$ indicates to respond with a move $a_1$ in $A$ then this is also the response of $\tau \circ \sigma$ to the move $c_1$. If $\sigma$ response to $b_1$ is a move in $B$, say $b_2$, then this Player move for the game $A \rightarrow B$ is an Opponent move for the game $B \rightarrow C$ and the Player goes on by applying strategy $\tau$ to $b_2$. Continuing in this way we obtain a uniquely determined sequence $c_1 b_1 b_2 \ldots b_k \ldots$. If the sequence ends with a move in $A$ or $C$ then this is the response of $\tau \circ \sigma$ to $c_1$, with the internal dialogue between $\sigma$ and $\tau$ in $B$ being hidden from the Environment. Notice that the internal dialogue can go on forever; in such a case the strategy $\tau \circ \sigma$ does not respond to $c_1$. Notice, further, that $\tau \circ \sigma$ may not to respond to $c_1$, also because $\sigma$ or $\tau$, at some point, has no answer to suggest to the Player.

This explanation would aim to clarify the slogan of strategy composition as “parallel composition + hiding”: $\sigma$ and $\tau$ are run in parallel and their internal chattering is hidden from the Environment.

**Definition 4.14 (Composition of strategies).** Given two strategies $\sigma : A \rightarrow B$ and $\tau : B \rightarrow C$, the strategy $\tau \circ \sigma : A \rightarrow C$ is defined by

$$\tau \circ \sigma = \{ s \mid A, C \mid s \in (M_A + M_B + M_C)^* \& \ s \mid A, B \in \overline{\sigma}, s \mid B, C \in \overline{\tau}\}^{even}$$

where $s \mid A, B$ is the sequence obtained from $s$ by erasing all the symbols not in $A$ or $B$ and $\overline{\sigma}$ is defined in Definition 4.11.

This definition is well posed because there is only one $s \in (M_A + M_B + M_C)^*$ such that $s \mid A, B \in \overline{\sigma}$ and $s \mid B, C \in \overline{\tau}$. Moreover the composition is associative.

### 4.2 Categories of games

This section is devoted to the introduction of the categories of games of interest for the study which will be carried out later. The referring category is $\mathcal{G}$, the category of games and history-free strategies, introduced by Abramsky, Jagadeesan and Malacaria in [AJM96]. This category is symmetric monoidal closed, equipped with a co-monad, and, hence, by standard categorical constructions [Mac71], it is possible to build on it a Cartesian closed category of games $\mathcal{K}_c(\mathcal{G})$. For our purposes, we are committed to introduce also the category $\mathcal{G}^*$ of games and history-sensitive strategies to correctly interpret the “approximating” strategies of Section 4.3.
4.2.1 The category Games

Games and deterministic strategies (Definitions 4.1 and 4.11) with the composition given by Definition 4.14 and the copy-cat strategies as identities constitute the category Games. Such a category, however, is not suitable to model λ-calculus and other programming languages, when we try to find models that are fully complete. For this purpose we have to introduce additional ingredients on Games. The category Games will be rebuilt through the Geometry of Interaction construction from another perspective in Chapter 7.

4.2.2 The category \( \mathcal{G} \)

The main novelties introduced in [AJM96] with respect to the standard definition of games (Definition 4.1) are the labelling function, which decorates a move not only with “P” or “O” for a Player or an Opponent move, but also with “Q” or “A” if the move is a question or an answer and the equivalence relation on plays \( \approx \). The category \( \mathcal{G} \) aimed to be a model for the language PCF [Plo97], which is a typed language, and where it is important to distinguish a request of data (question) from a supply of it (answer). This is not the case for \( \lambda \)-calculus where there is no clear notion of output but only an “observable” termination of the computation. We use the category \( \mathcal{G} \) for historical reasons and to provide a wider framework for our results.

The equivalence relation \( \approx \) is introduced to identify strategies for an exponential game \( !A \) that differ only at plays that are the same up to a permutation of the indexes. Differentiating such strategies causes problems for the composition of strategies.

Definition 4.15 (Games in \( \mathcal{G} \)). A game \( G = (M_G, \lambda_G, P_G, \approx_G) \) is a structure where

- \( M_G \) is the set of moves of the game
- \( \lambda_G : M_G \rightarrow \{P, O\} \times \{Q, A\} \) is a labelling function designating each move as by Player or Opponent and as Question or Answer. We can decompose \( \lambda_G \) into the functions \( \lambda_G^{OP} : M_G \rightarrow \{O, P\} \) and \( \lambda_G^{QA} : M_G \rightarrow \{Q, A\} \) and put \( \lambda_G = (\lambda_G^{OP}, \lambda_G^{QA}) \). We denote with \( \lambda_G \) the function \( (\lambda_G^{OP}, \lambda_G^{QA}) \) where \( \lambda_G^{OP} \) is defined in Definition 4.8.
- \( P_G \subseteq \text{pref} M_G^\circ \subseteq M_G^* \), where \( M_G^\circ \) is the set of sequences of moves \( s \in M_G^* \) satisfying:
  1. \( \lambda_G^{OP}(s_1) = O \);
  2. \( (\forall 1 \leq i < |s|)(\lambda_G^{OP}(s_i) \neq \lambda_G^{OP}(s_{i+1})) \);
  3. \( (\forall t \subseteq s)(|t \parallel M_G^1| \leq |t \parallel M_G^2|) \)

where \( M_G^1 \) and \( M_G^2 \) denote the subsets of game moves labelled respectively as Answers and Questions
- \( \approx_G \) is an equivalence relation on \( P_G \) which satisfies the following properties:
  1. \( s \approx_G s' \Rightarrow |s| = |s'| \)
  2. \( \text{sa} \approx_G s'a' \Rightarrow s \approx_G s' \)
  3. \( s \approx_G s' \& \text{sa} \in P_G \Rightarrow (\exists a')(sa \approx_G s'a') \)

We need then to extend the constructions on games to accommodate question and answers and the equivalence relation on plays.
Definition 4.16 (Tensor product in \( \mathcal{G} \)). Given two games \( A \) and \( B \) the tensor product \( A \otimes B \) is the game defined as follows:

- \( M_{A \otimes B} = M_A + M_B \)
- \( \lambda_{A \otimes B} = [\lambda_A, \lambda_B] \)
- \( P_{A \otimes B} \subseteq M_{A \otimes B}^\oplus \) where \( M_{A \otimes B}^\oplus \) is the set of plays, \( s \in M_{A \otimes B}^\oplus \), that satisfy the following conditions:
  - \( s \mid A \in P_A \) & \( s \mid B \in P_B \)
  - every answer in \( s \) must be in the same component as the corresponding question
- \( s \approx_{A \otimes B} s' \iff \)
  - \( s \mid A \approx_A s' \mid A \)
  - \( s \mid B \approx_B s' \mid B \)
  - \( (\forall i)(s_i \in M_A \iff s'_i \in M_A) \)

Definition 4.17 (Unit of the tensor product in \( \mathcal{G} \)). The unit element for the tensor product is given by the empty game \( I = (\emptyset, \emptyset, \{\epsilon\}, \{(\epsilon, \epsilon)\}) \).

Definition 4.18 (Linear implication in \( \mathcal{G} \)). Given games \( A \) and \( B \) the compound game \( A \Rightarrow B \) is defined as the tensor product but for the condition \( \lambda_{A \Rightarrow B} = [\lambda_A, \lambda_B] \).

Definition 4.19 (Exponential in \( \mathcal{G} \)). Given a game \( A \) the game \( !A \) is defined by:

- \( M_{!A} = \omega \times M_A = \sum_{i \in \omega} M_A \)
- \( \lambda_{!A}((i, a)) = \lambda_A(a) \)
- \( P_{!A} \subseteq M_{!A}^\oplus \) is the set of plays, \( s \), that satisfy the following conditions:
  - \( (\forall i \in \omega)(s \mid \{i\} \times M_A \in P_A) \)
  - every answer in \( s \) is in the same index as the corresponding question
- \( s \approx_{!A} s' \iff \exists \) a permutation of indexes \( \alpha \in S(\omega) \) such that:
  - \( \pi_1^*(s) = \alpha^*(\pi_1^*(s')) \)
  - \( (\forall i \in \omega)(\pi_2^*(s \mid \alpha(i)) \approx \pi_2^*(s \mid i)) \)

where \( \pi_1 \) and \( \pi_2 \) are the projections of \( \omega \times M_A \) and \( s \mid i \) is an abbreviation of \( s \mid A_i \).

The equivalence relation \( \approx \) on plays says when two different runs of a game have to be considered essentially the same. Since strategies are sets of plays, it is natural to extend the equivalence relation on strategies. When two strategies have to be considered essentially the same? The correct answer is when, in equivalent runs of a game, they suggest moves which bring to equivalent runs. This means also that if, in equivalent runs of a game, one strategy proposes a response, an equivalent strategy has necessarily to propose a response.

Definition 4.20 (Equivalence of strategies). Let \( \sigma, \tau \) be strategies. \( \sigma \approx \tau \) if and only if

1. \( \text{sa} \in \sigma, s'a'b' \in \tau, \text{sa} \approx_A s'a' \Rightarrow \text{sa} \approx_A s'a'b' \)
2. \( s \in \sigma, s' \in \tau, sa \approx_A s'a' \Rightarrow (\exists b)(sab \in \sigma) \iff (\exists b')(s'a'b' \in \tau) \)

Notice that the relation \( \approx \) is symmetric by definition and obviously transitive but it is not, in general, reflexive. The following game presents a counterexample.

Let \( A \) be the game defined as

\[
A = \langle \{a_1, a_2, b_1, b_2, b_3\}, \{a_1, a_2\} \mapsto O, \{b_1, b_2, b_3\} \mapsto P, P_A, \approx_A \rangle
\]

where \( P_A \) is defined as:

\[
\text{and } s \approx_A s' \iff |s| = |s'|. \text{ Consider the strategy } \sigma \text{ defined as:}
\]

\[
\text{It is not the case that } \sigma \approx \sigma \text{ since } a_1b_1, a_2b_1 \in \sigma, a_1b_1a_1 \approx a_2b_2a_2 \text{ but } a_1b_1a_1b_1 \in \sigma \text{ and } a_2b_2a_2b_2 \notin \sigma \text{ violating condition 2 of Definition 4.20.}
\]

If a strategy \( \sigma \) for a game \( A \) is such that \( \sigma \approx \sigma \) we write \( \sigma : A \) and denote with \([\sigma]\) the equivalence class containing \( \sigma \). Notice that \( 1_A \approx 1_A \) for each object \( A \).

**Definition 4.21 (The category \( \mathcal{G} \)).** The category \( \mathcal{G} \) has:

- **Objects:** Games (Definition 4.15)
- **Morphisms:** \([\sigma] : A \rightarrow B\) the equivalence classes of history-free \( \sigma : A \twoheadrightarrow B \).

The identity for each game \( A \) is given by the (equivalence class) of the copy-cat strategy \( 1_A \) (Section 4.1.3) and composition is given by the extension on equivalence classes of the composition of strategies of Definition 4.14.
As already mentioned, $\mathcal{G}$ is not Cartesian closed as we would want. However it is symmetric monoidal closed, a property which allows to transform it easily in a Cartesian closed category. In the following it will be shown the symmetric monoidal closed structure of $\mathcal{G}$.

**Definition 4.22.** Given two strategies $\sigma : A \rightarrow B$ and $\tau : A' \rightarrow B'$ let $\sigma \otimes \tau : A \otimes A' \rightarrow B \otimes B'$ be the strategy defined as:

$$\sigma \otimes \tau = \{ s \in P_{A \otimes A' \rightarrow B \otimes B'} \mid s \mid A, B \in \sigma, s \mid A', B' \in \tau \}$$

**Proposition 4.23.** $\otimes : \mathcal{G} \times \mathcal{G} \rightarrow \mathcal{G}$ of definitions 4.5 and 4.22 is a functor.

**Proof.** Observe the following equations for $A, A', B, B', C, C'$ objects in $\mathcal{G}$ and $\sigma : A \rightarrow B$, $\sigma' : B \rightarrow C$, $\tau : A' \rightarrow B'$ and $\tau' : B' \rightarrow C'$:

$$1_A \otimes 1_B = \begin{cases} \{s \mid s \in P_{A_1 \otimes B_1 \rightarrow A_2 \otimes B_2}^\text{even} \mid s \mid A_1, A_2 \in 1_A \& s \mid B_1, B_2 \in 1_B \} \\
\{s \mid s \in P_{A_1 \otimes B_1 \rightarrow A_2 \otimes A_2}^\text{even} \mid s \mid A_1 = s \mid A_2 \& s \mid B_1 = s \mid B_2 \} \\
\{s \mid s \in P_{A_1 \otimes A_1 \rightarrow A_2 \otimes A_2}^\text{even} \mid s \mid A_1 \otimes A_1 = s \mid A_2 \otimes A_2 \}
\end{cases} = 1_{A \otimes B}$$

$$(\sigma' \otimes \tau') \circ (\sigma \otimes \tau) = \begin{cases} \{s \mid A \otimes A', C \otimes C' \mid s \in M^*, s \mid A \otimes A', B \otimes B' \in \sigma \otimes \tau, s \mid B \otimes B', C \otimes C' \in \sigma' \otimes \tau' \} \\
\{s \mid A \otimes A', C \otimes C' \mid s \in M^*, s \mid A, B \in \sigma, s \mid A', B' \in \tau, s \mid B, C \in \sigma', s \mid B', C' \in \tau' \} \\
\{s \mid A \otimes A', C \otimes C' \mid s \in M^*, s \mid A, C \in \sigma' \circ \sigma, s \mid A', C' \in \tau ' \circ \tau \} \\
(\sigma' \circ \sigma) \otimes (\tau' \circ \tau)
\end{cases}$$

where $M^* = (M_A + M_{A'} + M_B + M_{B'} + M_C + M_{C'})^*$.

**Definition 4.24.** Given games $A, B, C$ in $\mathcal{G}$ let us introduce the following isomorphisms as sets of plays $\{ s \in P \mid s \mid X_1 = s \mid X_2 \}$ defined parametrically in $P$ and $X$:

- $\text{assoc}_{A, B, C} : (A \otimes B) \otimes C \rightarrow A \otimes (B \otimes C)$
  - $P = P_{(A \otimes B) \otimes C \rightarrow A \otimes (B \otimes C)}^\text{even}$
  - $X \in \{A, B, C\}$

- $\text{symm}_{A, B} : A \otimes B \rightarrow B \otimes A$
  - $P = P_{(A \otimes B) \rightarrow (B \otimes A)}^\text{even}$
  - $X \in \{A, B\}$

- $\text{unitl}_{A} : I \otimes A \rightarrow A$
  - $P = P_{(I \otimes A_1) \rightarrow A_2}^\text{even}$
  - $X \equiv A$

- $\text{unitr}_{A} : A \otimes I \rightarrow A$
  - $P = P_{(A_1 \otimes I) \rightarrow A_2}^\text{even}$
  - $X \equiv A$
Definition 4.25 (Closed structure). Given games $A, B, C$ of $\mathcal{G}$ let us define
\[ \Lambda : \mathcal{G}(A \otimes B, C) \to \mathcal{G}(A, B \multimap C) \]
as $\Lambda(\sigma) = \{ \alpha^x(s) \mid s \in \sigma \}$ where $\alpha : (M_A + M_B) + M_C \to M_A + (M_B + M_C)$ is the canonical isomorphism in $\text{Set}$.

Proposition 4.26. $\mathcal{G}$ is a symmetric monoidal closed category.

Proof. See [AJ94a], Section 3.5. \hfill \Box

4.2.3 The category $K_1(\mathcal{G})$

As is standard in category theory (see Theorem 1.37), we can obtain a Cartesian closed category from a symmetric monoidal closed one provided we can exhibit a co-monad over it. This is the route which will be taken.

Definition 4.27. Let $A, B$ be games of $\mathcal{G}$ and $\sigma : A \multimap B$ be a strategy. Let us extend the constructor $!$ (Definition 4.10) to morphisms and define the following strategies:

\[
\begin{align*}
!\sigma : !A & \to !B & = & \{ s \in P^{\text{even}}_{!A \to !B} \mid s \mid i \in \sigma \} \\
der^j_A : !A & \to A & = & \{ s \in P^{\text{even}}_{!A \to A} \mid s \mid (!A)_{i} = s \mid A \} \\
\delta^p_A : !A & \to !(\cdot)A & = & \{ s \in P^{\text{even}}_{!A \to !(\cdot)A} \mid s \mid (!A)_{p(i,j)} = s \mid (!A)_{j} \} \\
& & & \text{where } p : \omega \times \omega \to \omega \text{ is a pairing function}
\end{align*}
\]

Notice that the choice of a particular instance of the game $A$ in $!A$ - the $i$-th copy - does not play any significant role in the morphism $[\text{der}^i_A]$ of $\mathcal{G}$ as the following lemma shows.

Lemma 4.28. For each game $A$ of $\mathcal{G}$, for each $i, j \in \omega$, $\text{der}^i_A \approx \text{der}^j_A$.

Proof. Notice that in the occurrence of $!A$, in the game $!A \multimap A$, only one component is used by $\text{der}^i_A$ (the $i$-th component) and by $\text{der}^j_A$ (the $j$-th component). Consider the following permutation:

\[ \alpha(k) = \begin{cases} 
  j & \text{if } k = i \\
  k & \text{otherwise}
\end{cases} \]

If $s, s' \in P^{\text{even}}_{!A_1 \rightarrow A_2}$ are plays such that $s \in \text{der}^i_A$ and $s' \in \text{der}^j_A$ and $sa \approx (!A_1 \to A_2) s'a'$ then, for the properties of $\approx$, also $s \approx (!A_1 \to A_2) s'$. Moreover, if there exists $b$ such that $sab \in \text{der}^i_A$ then, for the properties of $\approx$, there exists $b'$ such that $sab \approx (!A_1 \to A_2) s'a'b'$. It must be the case that $s'a'b' \notin \text{der}^j_A$. This completes the proof, since the roles of $\text{der}^i_A$ and $\text{der}^j_A$ can be reversed and if $s'a'b'' \in \text{der}^j_A$ then, by the determinacy of a strategy, $b' = b''$ and properties 1 and 2 of Definition 4.20 are satisfied.

$s'a'b' \in \text{der}^j_A$ follows by a case analysis on where the last move $b$ in $sab$ happens to be made. There are two cases. Notice, in fact, that, since $b$ is a Player move, it has to be made in a component game of $!A_1 \multimap A_2$ different from the one where the previous move $a$ has to be made because, by definition of $\text{der}^i_A$, each even length play $s \in \text{der}^i_A$ has to satisfy the condition $s \mid (!A_1)_{j} = s \mid A_2$ and this cannot happen if $a$ and $b$ are made in the same component. For the same reason it must be the case that $b = a$. The two arising cases are the following:
• \( b \in !A_1 \) and \( a \in !A_2 \). In this case \((sab) \upharpoonright !A_1 = (s \upharpoonright !A_1)b\) with \( b = a \) and \( s \upharpoonright !A_1 \approx !A_1, s' \upharpoonright !A_1 \) under the permutation \( \alpha \). For the properties of \( \approx \) there exists \( b' \) such that \((s \upharpoonright !A_1)b \approx !A_1, (s' \upharpoonright !A_1)b'\). If \( a' \) is a valid Opponent move in \( !A_2 \), for the reverse role of Player and Opponent in \( !A_1 \) in the compound game \( !A_1 \to !A_2 \), \( a' \) is a valid Player move in \( !A_1 \) and hence \( b' = a' \) for the determinacy requirement. But \( s'd'b = s'a'a' \in \der_A^X \) by definition.

• \( b \in !A_2 \) and \( a \in !A_1 \). Just symmetric to the previous case.

We shall write then simply \( \der_A \) with no ambiguity. In the same way the pairing function \( p : \omega \times \omega \to \omega \) (a bijective function) in the definition of \( \delta_A^0 \) is not important.

**Lemma 4.29.** For each game \( A \) of \( \mathcal{G} \), for each couple of pairing functions \( p,q : \omega \times \omega \to \omega \), \( \delta_A^0 \approx \delta_A^0 \).

**Proof.** The proof is very similar to that of Lemma 4.28. Let us only define the permutation \( \alpha \in S(\omega) \) as \( \alpha(k) = q(\pi_1(p^{-1}(k)),\pi_2(p^{-1}(k))) \) where \( \pi_1((a,b)) = a \) and \( \pi_2((a,b)) = b \).

**Proposition 4.30.** The structure \( \langle !A, (\der_A, [\delta]) \rangle \) is a co-monad.

**Proof.** The basic equations to verify (see Definition 4.27) are the following:

• \( \der_A \circ [\delta_A] = [1_A] \). We shall show that \( \der_A \circ \delta_A^0 \approx 1_A \) with \( p \) a fixed pairing function and \( i \in \omega \). Given \( s \in P_{\text{even}}^{!A_1 \to !A_2 \to !A_3} \), \( s | (\upharpoonright !A_1, !A_3) \in \der_A \circ \delta_A \) if and only if \( s \upharpoonright !A_1, !A_3 \in \der_A \) and \( s \upharpoonright !A_2 \in \delta_A \). The first condition says that \( s \upharpoonright !A_1 \approx !A_3 \), and \( s \upharpoonright !A_2 \approx !A_3 \), but \( i \) is fixed and then it is the same as \( s \upharpoonright !A_1 = s \upharpoonright !A_2 \). The second condition says that \( s \upharpoonright !A_1 = s \upharpoonright !A_3 \) and hence \( s \upharpoonright !A_3 \), that is \( s \in 1_A \). For the converse, if \( s \in 1_A \) then there is a unique \( s' \in P_{\text{even}}^{!A_1 \to !A_2 \to !A_3} \) such that \( s' \upharpoonright !A_1 !A_2 \in \delta_A \) and \( s' \upharpoonright !A_2 \approx !A_3 \) defined by \( s' \upharpoonright !A_1 = s' \upharpoonright !A_2 \) and \( s' \upharpoonright !A_3 \) is the only possibility.

• \( \langle [\der_A], [\delta_A] \rangle = [1_A] \). It is very similar to the previous case. It suffices to choose an \( i \in \omega \) and show that \( \der_A \circ \delta_A^0 \approx 1_A \). Observe that \( \der_A \circ !A \to !A \) uses only the \( i \)-th components of the compound game \( !A \), that is that \( \der_A : !A \to !A \).

• \( ![\der_A, \delta_A] = [\delta_A, \der_A] \). Just observe that \( ![A \to B] \cong !A \to !B \). The isomorphic strategy \( e \circ ![A \to B] \cong ![A \to B] \) is induced by the function \( f_e : M_{[A \to B]} + M_{[A \to B]} \to M_{[A \to B]} + M_{[A \to B]} \) defined by

\[
    f_e(m) = \begin{cases} 
        \left[ n_1((i, n_i(b))) \right] & \text{if } m = n_1((i, n_i(b))) \\
        \left[ n_1((i, n_i(b))) \right] & \text{if } m = n_1((i, n_i(b))) \\
        \left[ n_1((i, b)) \right] & \text{if } m = n_1((i, n_i(b))) \\
        \left[ n_1((i, a)) \right] & \text{if } m = n_1((i, n_i(b))) \\
    \end{cases}
\]

Hence \( ![\delta_A] \cong \delta_A \) and \( ![\delta_A] \approx \delta_A \approx \delta_A \cdot \delta_A \).

**Definition 4.31 (The category \( K_l(\mathcal{G}) \)).** The category \( K_l(\mathcal{G}) \) has:

**Objects:** Games (Definition 4.15)

**Morphisms:** \( [\sigma] : A \to B \) the equivalence classes of history-free \( \sigma : !A \to !B \).
The identity for each game $A$ is given by the (equivalence class) of the strategy $\text{der}_A$ (Definition 4.27) and the composition of two strategies $\sigma : !A \rightarrow B$ and $\tau : !B \rightarrow C$ is given by $\tau \circ (\text{!} \sigma \circ \delta_A)$.

The announced Cartesian product in $K_1(\mathcal{G})$ is defined as follows:

**Definition 4.32 (Cartesian product).** Given games $A$ and $B$ the Cartesian product $A \& B$ is the game defined as follows:

$$
M_{A \& B} = M_A + M_B \\
\lambda_{A \& B} = [\lambda_A, \lambda_B] \\
P_{A \& B} = P_A + P_B \\
\approx_{A \& B} = \approx_A + \approx_B
$$

Given the strategies:

- $\text{fst}_{A,B} : A \& B \rightarrow A = \{ s \in \text{\text{even}}_{(A_1 \& B)\rightarrow A_1} \mid s \upharpoonright A_1 = s | A_2 \text{ and } s | B = \epsilon \}$
- $\text{snd}_{A,B} : A \& B \rightarrow B = \{ s \in \text{\text{even}}_{(A \& B_1)\rightarrow B_1} \mid s \upharpoonright B_1 = s | B_2 \text{ and } s | A = \epsilon \}$

the projections are defined as:

$$
\pi_{A,B}^A : ! (A \& B) \rightarrow A = \text{\text{fst}}_{A,B} \circ \text{der}_{(A \& B)} \\
\pi_{A,B}^B : ! (A \& B) \rightarrow B = \text{\text{snd}}_{A,B} \circ \text{der}_{(A \& B)}
$$

and, given a game $C$ and strategies $\sigma : !C \rightarrow A$ and $\tau : !C \rightarrow B$, the pairing strategy $\langle \sigma, \tau \rangle : !C \rightarrow A \& B$ is defined as

$$
\langle \sigma, \tau \rangle = \text{\text{der}}_{(A \& B)} \circ e_{A,B} \circ ! (\sigma \otimes \tau) \circ (\delta_C \otimes \delta_C) \circ \text{\text{con}}_C
$$

where $e_{A,B} : !(A \& B) \cong !(A \otimes B)$ is the isomorphic strategy of [AJM96], Proposition 2.8.1 and $\text{\text{con}}_C : !C_1 \rightarrow !C_2 \otimes !C_3$ is induced by the function $f_{\text{\text{con}}_C} : M^O_{C_2 \otimes C_3} + M^P_{C_1} \rightarrow M^P_{(C_2 \otimes C_3) C_1} + M^O_{C_1}$ defined by

$$
f_{\text{\text{con}}_C}(m) = \left\{
\begin{array}{ll}
\text{in}_r((r(i),c)) & \text{if } m = \text{in}_l(\text{in}_l((i,c))) \\
\text{in}_r((s^{-1}(i),c)) & \text{if } m = \text{in}_l(\text{in}_r((i,c))) \\
\text{in}_l(\text{in}_r((s^{-1}(i),c))) & \text{if } m = \text{in}_r((i,c)) \text{ and } s^{-1}(i) \text{ is defined} \\
\text{in}_l(\text{in}_r((s^{-1}(i),c))) & \text{if } m = \text{in}_r((i,c)) \text{ and } s^{-1}(i) \text{ is defined}
\end{array}
\right.
$$

where, given a tagging function $c : \omega + \omega \rightarrow \omega$, that is a bijection, $r(i) = c(\text{in}_l((i,c)))$ and $s(i) = c(\text{in}_r((i,c)))$.

### 4.2.4 The category $\mathcal{G}^*$

The category $\mathcal{G}^*$ is the supercategory of $\mathcal{G}$ in which general strategies rather then history-free ones are considered.

**Definition 4.33 (The category $\mathcal{G}^*$).** The category $\mathcal{G}^*$ has:

- **Objects:** Games (Definition 4.15)
- **Morphisms:** $[\sigma] : A \rightarrow B$ the equivalence classes of $\sigma : A \rightarrow B$.

The identity for each game $A$ is given by the (equivalence class) of the copy-cat strategy $1_A$ (Section 4.1.3) and composition is given by the extension on equivalence classes of the composition of strategies of Definition 4.14.
Such a relaxation from \( \mathcal{G} \) is necessary to accommodate the approximating strategies that will be introduced in the next section.

These strategies aim to represent finite approximate normal forms of terms of \( \lambda \)-calculus and do not belong, in general, to \( \mathcal{G} \) since they could not be history-free. They enjoy, anyway, a bunch of pleasant properties and allow to investigate the whole structure of the interpretation of \( \lambda \)-terms in \( \mathcal{G} \).

The analysis of approximate normal forms of terms is a well known tool in the study of \( \lambda \)-calculus models. In the standard categories of domains, the interpretation of these truncated terms fall out in the category itself – and usually it is said that these categories have too much points to provide fully abstract models for theories of \( \lambda \)-calculus – while, in the case of \( \mathcal{G} \), there are not so many points and we need further external support.

### 4.3 Approximating strategies.

An approximating strategy is a restricted development of a strategy. The sequences of moves which fill out the approximating strategy are the sequences of moves of the approximated strategy up to a given length. Approximating strategies will be used to prove that the interpretation of a term, through the game semantics paradigm, in the category \( \mathcal{G} \), is the least upper bound of the interpretations of its “approximate normal forms”.

**Definition 4.34.** Let \( D \) be a game. We indicate with \( D^n \) the subgame of \( D \) where \( P_{D^n} = \{ s \in P_D \mid |s| \leq n \} \).

**Lemma 4.35.** For each pair of games \( A \) and \( B \) \( (A \rightarrow B)^{n+1} \preceq A^n \rightarrow B^{n+1} \).

**Proof.** Let \( s \in P_{(A \rightarrow B)^{n+1}} \) be a play. We obviously have that \( |s| \leq n + 1 \), and, since the first move of \( s \) has to be made in \( B \), we have also that \( |s| \leq n \).

**Lemma 4.36.** For each pair of games \( A \) and \( B \)

\[
ev_{A,B}[(A^n \rightarrow B^m) \otimes A \rightarrow B) = ev_{A,B}[(A \rightarrow B) \otimes A^n \rightarrow B^m]
\]

**Proof.** \( ev_{A,B}[(A^n \rightarrow B^m) \otimes A \rightarrow B) = \)

\[
\{ s \in P_{(A^n \rightarrow B^m) \otimes A \rightarrow B) \mid s \mid A^n = s \mid A_2 \land s \mid B^m = s \mid B_2 \}
\]

\[
= \{ s \in P_{(A \rightarrow B) \otimes A \rightarrow B) \mid s \mid A_1 = s \mid A_2 \land s \mid B_1 = s \mid B_2 \}
\]

\[
= ev_{A,B}[(A \rightarrow B) \otimes A^n \rightarrow B^m].
\]

Given a strategy \( \sigma : A \), an approximating strategy for \( \sigma \) can be seen as a strategy \( \tau \) for a subgame \( A' \) of \( A \).

**Definition 4.37 (Approximating strategy).** Let \( A \) be a game and \( \sigma \) a strategy for \( A \).

1. Let \( A' \) be a subgame of \( A \). We write \( \sigma|A' \) for the strategy \( \{ s \in \sigma \mid s \in P_{A'} \} \).

2. Let \( \sigma : A \rightarrow B \) be a strategy. We indicate with \( \sigma^n \) the history-sensitive strategy \( \sigma|(A \rightarrow B^n) \) and with \( [\sigma^n] \) the equivalence class \( [\sigma^n] \).

Observe that if \( \sigma \approx \tau \) then \( \sigma^n \approx \tau^n \), since equivalent plays have the same length. Thus we can write \( [\sigma^n] \) with no ambiguity. In general the strategy \( \sigma^n \) can be history-sensitive also if the strategy \( \sigma \) is history-free. This is because \( \sigma^n \) can reply to a move \( a \) of the Opponent in some point of the development of the game and does not reply to it in points represented by longer plays. In order to accommodate and freely use the strategies \( \sigma^n \) we introduced the
category $\mathcal{G}^n$ of games and history-sensitive strategies. Such strategies will be used to prove an approximation theorem along the same line of the works [HR92, Hyl76, Wad78]. In these works the approximation of a semantical point is obtained through a series of projection functions. Here we use a different approach that, in the context of games, is simpler and more direct.

**Proposition 4.38.** For each pair of games $A$ and $B$ and strategy $\sigma : A \rightarrow B$, the following properties hold:

1. $\sigma^0 = \{\epsilon\}$
2. $\sigma^n \subseteq \sigma^{n+1}$
3. $\bigcup_{n \in \omega} \{\sigma^n\} = \sigma$
4. $\sigma^n = \sigma^{\min \{m,n\}}$

*Proof.* The property 1 follows from the fact that the first move in the game $A \rightarrow B$ has to be in $B$. The other proofs are immediate. □

The following lemmas establish a connection between retracts and approximation strategies.

**Lemma 4.39.** For each retract $\langle B \triangleleft A, [\psi], [\varphi] \rangle$ in the category $\mathcal{G}$, $\varphi : A \rightarrow B$ does not contain any play $sab$ with $a, b \in M_B$.

*Proof.* By contradiction. Suppose there exists a play $sab \in \varphi$ with $a, b \in M_B$. Let $s \mid B = b_1 \ldots b_n$. The play $b_1b_1b_2b_2 \ldots b_nb_n \in P_{B \rightarrow B}$ belongs to the strategy $1_B$, therefore there exists an equivalent play $t$ in the strategy $\varphi \circ \psi \approx 1_B$. Since we are considering history-free strategies, from $t \mid B \approx s \mid B$, and $sab \in \varphi$, it follows that $tab$ belongs to $\varphi \circ \psi$, and this is in contradiction with the fact that $\varphi \circ \psi \approx 1_B$, since the sequence $tab$ cannot be equivalent to a copy-cat sequence of moves. □

**Lemma 4.40.** For each retract $\langle B \triangleleft A, [\psi], [\varphi] \rangle$ in $\mathcal{G}$, for each strategy $\sigma : C \rightarrow A$ and $n \in \omega$ we have: $\varphi \circ \sigma^n \subseteq (\varphi \circ \sigma)^n$.

*Proof.* By the previous lemma, for any position $s \in \varphi : A \rightarrow B$, we have $|s \mid B| \leq |s \mid A|$, and using the definition of composition, one readily prove the thesis. □

Finally, the relevant properties of approximations of strategies interpreting terms of $\lambda$-calculus are given.

**Proposition 4.41.** Let $A$ be a game, $\langle (D \Rightarrow D) \triangleleft D, [\psi], [\varphi] \rangle$ be a reflexive object in the Cartesian closed category of games $K_1(\mathcal{G})$ and $\sigma, \tau : A \Rightarrow D$ be two strategies. Then we have

1. $\sigma^0 \cdot \tau = \epsilon_{A \Rightarrow D}$
2. $\sigma^{n+1} \cdot \tau \subseteq (\sigma \cdot \tau^n)^{n+1}$

*Proof.* The following chains of relations hold:
\[
\sigma^0 \cdot \tau = ev_{D,D} \circ \langle (\varphi \circ \sigma^0), \tau \rangle \\
\subseteq ev_{D,D} \circ \langle (\varphi \circ \sigma)^0, \tau \rangle \\
= ev_{D,D} \circ \langle \epsilon_{A \Rightarrow (D \Rightarrow D)}, \tau \rangle \\
= \epsilon_{A \Rightarrow D} \\
\text{by Lemma 4.40 and definition of } ev \\
\sigma^{n+1} \cdot \tau = ev_{D,D} \circ \langle (\varphi \circ \sigma^{n+1}), \tau \rangle \\
\subseteq ev_{D,D} \circ \langle (\varphi \circ \sigma)^{n+1}, \tau \rangle \\
= (ev_{D,D}|(D \Rightarrow D)^{n+1} \times D \Rightarrow D) \\
\circ \langle (\varphi \circ \sigma), \tau \rangle \\
\subseteq (ev_{D,D}|(D^n \Rightarrow D^{n+1}) \times D \Rightarrow D) \\
\circ \langle (\varphi \circ \sigma), \tau \rangle \\
= (ev_{D,D} \circ \langle (\varphi \circ \sigma), \tau^n \rangle)^{n+1} \\
= (\sigma \cdot \tau^n)^{n+1}. \\
\]

\[\square\]
5

Game Lambda Theories

“The lion roaring behind the door, turned out, when that door was opened, to be a little, domesticated cat.”

Allan Bloom, The Closing of the American Mind

Abstract

The game semantics paradigm introduced in Chapter 4 is now applied to untyped λ-calculus. Different models are built using ad hoc techniques. The λ-theories that have got a model in the category $\mathcal{G}$ of games and history-free strategies are characterized completely. These are the theory $\mathcal{H}^*$ – the maximal sensible theory –; $\mathcal{B}$ – the theory which equates two terms if and only if they have the same Böhm tree – and the theory $\mathcal{L}$ which equates two terms if and only if they have the same Lévy-Longo tree.

Game semantics, as introduced in [AJ94a, AJM96], has been very fruitful in providing fully-abstract models for many different programming languages, capturing very well different features of the computation [AHM98, AM97, AM98]. In this chapter we shall show that, in the category $\mathcal{G}$ of games and history-free strategies introduced by Abramsky et al. (but similar results also apply to other categories [KNO01, KNO99]), only a restricted class of λ-theories have a model at all. This class, indeed, has only three elements: $\mathcal{H}^*$ – the maximal sensible theory, $\mathcal{B}$ – the theory which identifies two terms if and only if they have the same Böhm tree and $\mathcal{L}$ – the theory which identifies two terms if and only if they have the same Lévy-Longo tree.

This result has to be compared with other semantical settings for λ-calculus as, for instance, the category of complete partial orders and continuous functions, where an extraordinary rich class of different theories have a model. Evidently, the category $\mathcal{G}$ has a strong bias towards head reduction and weak head reduction, since all the models it allows to build are semi-sensible and that many of them are even sensible. Moreover, all models in $\mathcal{G}$ are approximable, i.e. have a finitary character which can be revealed by analyzing the structure of the strategies interpreting λ-terms.
Abramsky and McCusker [AM95b] showed that recursive objects in the category \( \mathcal{G} \) can be obtained by standard categorical constructions as initial solutions of projective systems [SP82]. For untyped \( \lambda \)-calculus, such initial solutions originate always trivial models, i.e. one-point models. In [DGFH99] a methodology is developed to obtain also non-initial solutions of projective systems. The standard methodology proposed by Abramsky et al. [AJM96, AM95a] consists in building a model candidate that becomes a proper model after it has been pruned by an extensional collapse, which allows to throw away the information not really needed. It is worth mentioning that the models we built do not need such a treatment. The extensional collapse, however, can be seen as a tool for modelling coarser theories. An example is provided in [AM95a].

The main result we are able to achieve is Theorem 5.24 that says that it is possible to partition all the models of untyped \( \lambda \)-calculus in \( \mathcal{G} \) in three different classes where all the models in a class induce the same theory.

The same result is presented also in [DGF00], where the proof follows a different route.

**Definition 5.1.** Let \( \mathcal{D} \) be the class of all reflexive objects \( \langle (D \Rightarrow D) \triangleright D, [\psi], [\varphi] \rangle \) in the category \( K_1(\mathcal{G}) \). We define the following subclasses:

1. \( \mathcal{D}^c = \{ (D \Rightarrow D) \triangleright D, [\psi], [\varphi] \} \in \mathcal{D} \mid \psi \circ \varphi \approx 1_D \}
2. \( \mathcal{D}^r = \{ (D \Rightarrow D) \triangleright D, [\psi], [\varphi] \} \in \mathcal{D} \mid \psi \circ \epsilon_I \Rightarrow (D \Rightarrow D) = \epsilon_I \Rightarrow D \text{ and } \psi \circ \varphi \not\approx 1_D \}
3. \( \mathcal{D}^c = \{ (D \Rightarrow D) \triangleright D, [\psi], [\varphi] \} \in \mathcal{D} \mid \psi \circ \epsilon_I \Rightarrow (D \Rightarrow D) \not\approx \epsilon_I \Rightarrow D \}

**Theorem 6.24.** The theory induced by a categorical model \( D \in \mathcal{D} \) (Definition 5.1) of untyped \( \lambda \)-calculus in \( K_1(\mathcal{G}) \) is either

1. \( \mathcal{H}^* \), the theory induced by the canonical \( D_\infty \) model of Scott [Sco72, Bar84] and [Wad78], if \( D \in \mathcal{D}^c \);
2. \( \mathcal{B} \), the theory which identifies two terms iff they have the same Böhm tree, if \( D \in \mathcal{D}^r \);
3. \( \mathcal{L} \) the theory which identifies two terms iff they have the same Lévy-Longo tree if \( D \in \mathcal{D}^c \).

The proof of the theorem occupies Section 5.2.

### 5.1 Some categorical game models for \( \lambda \)-calculus

The category \( \mathcal{G} \) allows for the construction of recursive objects [AM95b], i.e. objects which are the fixed point of some functor. Abramsky and McCusker also exhibit a method to construct such objects as initial fixed points of suitable functors \( \Phi : \mathcal{G}^{op} \times \mathcal{G} \rightarrow \mathcal{G} \). These are obtained as colimits of an \( \omega \)-chain \( \langle D_n, \langle \eta_n, p_n \rangle \rangle \) obtained by successive applications of the functor \( \Phi \) to the initial game object \( I \), in which, for each \( n \in \omega \), \( D_n \leq D_{n+1} \). The colimit exists when \( \Phi \) satisfies some continuity conditions.

This method is suitable to obtain only initial fixed points and hence it is not practicable to obtain non-trivial extensional models of untyped \( \lambda \)-calculus in \( \mathcal{G} \). In fact, each initial fixed point of the functor \( \text{Fun} : \mathcal{G} \rightarrow \mathcal{G} \) defined by \( \text{Fun}(D) = !D \rightarrow D \), used to build a model of untyped \( \lambda \)-calculus, is isomorphic to the empty game \( I \) since \( !I \rightarrow I \cong I \). The choice of another functor – the Scott’s trick – for instance \( \text{Fun}(D) = A \& !D \rightarrow D \), is, of course, another, albeit indirect, solution.
In [DGFH99] the above method is extended to deal also with non-initial fixed points, and, hence, allowing for the construction of non-trivial extensional game models of untyped \(\lambda\)-calculus. The extended method differs from the original one in requiring a more liberal relation between \(D_n\) and \(D_{n+1}\): it suffices that \(D_n\) would be isomorphic to a subgame of \(D_{n+1}\). To be able to build limits under this weaker constraint we are forced, however, to introduce \(G^c\), the category of games and embeddings, which is a sub-category of \(G\).

**Definition 5.2 (Embeddings).** Let \(A\) and \(B\) be two games in \(G\). An embedding \(f : A \rightarrow B\) is a total injective function \(f : M_A \rightarrow M_B\) such that:

- \(\lambda_A = \lambda_B \circ f\)
- \(f^*(P_A) = P_B \cap (f(A))^*\)
- \(s \approx_A s'\) iff \(f^*(s) \approx_B f^*(s')\)

In the above we have used the notation \(f^*\) to denote the natural extension of \(f\) both to sequences and sets of sequences. It is immediate to see that the composition of two embeddings is an embedding, that the identity function is an embedding, and then that we have built, using the associativity of function composition, a category.

**Definition 5.3 (The category \(G^c\)).** The category of games \(G^c\) has as objects games and as morphisms embeddings.

The main reason to introduce yet another category is that, in \(G^c\), all colimits exist, as we shall show. The operation is similar to the one realized by Scott [Sco72] and Smith and Plotkin [SP82] in the category \(\text{CPO}\), to assure the existence of limits of suitable contravariant functors, the new category of CPOs and embedding-projection pairs \(\text{CPO}^E\) has been introduced.

**Proposition 5.4.** The category \(G^c\) is co-complete.

**Proof.** Given a chain \((D_n, f_n)\) with \(f_n : D_n \rightarrow D_{n+1}\) put \(M_{\{D_n, f_n\}} = \bigcup_{n \in \omega} M_{D_n}\). Define on \(M_{\{D_n, f_n\}}\) the following relation. For each \(n \in \omega\), for each \(a \in D_n\) and \(b \in D_{n+1}\)

\[ a \sim b \text{ if and only if } f_n(a) = b \]

Let \(\equiv\) be the smallest equivalence relation on \(M_{\{D_n, f_n\}}\) containing \(\sim\). The (co)limit game of the chain \((D_n, f_n)\) is then the game \(D_{\infty}\) defined by:

\[
\begin{align*}
M_{D_{\infty}} & = (\bigcup_{n \in \omega} M_{D_n})/\equiv \\
\lambda_{D_{\infty}} & = a \in D_n \mapsto \lambda_{D_n}(a) \\
P_{D_{\infty}} & = \bigcup_{n \in \omega} \{(a_1)\equiv \cdots \equiv (a_p)\equiv \mid a_1a_2\cdots a_p \in P_{D_n}\} \\
\approx_{D_{\infty}} & = \{(a_1)\equiv \cdots \equiv (a_p)\equiv, (a'_1)\equiv \cdots \equiv (a'_p)\equiv \mid (a_1 \cdots a_p, a'_1 \cdots a'_p) \in \approx_{D_n}\} \\
\end{align*}
\]

The colimit functions \(\mu_n : D_n \rightarrow D_{\infty}\) are defined by \(\mu_n(a) = [a]\equiv\). Observe that for each \(n \in \omega\) the diagram

\[
\begin{array}{ccc}
D_{\infty} & \xrightarrow{\mu_{n+1}} & D_{n+1} \\
\mu_n \downarrow & & \downarrow \mu_n \\
D_n & \xrightarrow{f_n} & D_{n+1}
\end{array}
\]

commutes. In fact, for \( a \in D_n \), \( \mu_{n+1}(f_n(a)) = [a] = \mu_n(a) \) because \( a \equiv f_n(a) \). Moreover, if \( \langle D', \nu_n \rangle \) is another object such that, for each \( n \in \omega \),

\[
\begin{array}{c}
D' \\
\downarrow \nu_n \\
D_n \\
\uparrow f_n \\
D_{n+1}
\end{array}
\]

commutes, then there exists a unique \( \varphi : D_\infty \rightarrow D' \) such that \( \varphi \circ \mu_n = \nu_n \) for each \( n \in \omega \). For \( a \in D_n \) put \( \varphi([a]_\equiv) = \nu_n(a) \). This definition, of course, implies \( \varphi \circ \mu_n = \nu_n \). Observe that if \( \varphi' \) is another such function then from \( \varphi' (\mu_n(a)) = \nu_n(a) \) we get \( \varphi'([a]_\equiv) = \nu_n(a) \), that is \( \varphi' = \varphi \).

Several functors in \( \mathcal{G} \) can be also defined in \( \mathcal{G}^e \). A sufficient condition for this to happen is the following.

**Definition 5.5 (E-extensible functors).** Let \((\mathcal{G}^e)^i\) be the subcategory of \( \mathcal{G}^e \) with isomorphisms as arrows.

1. A functor \( F : \mathcal{G} \rightarrow \mathcal{G} \) is \( e \)-extensible if it is monotone w.r.t. \( \leq \) and if there exists a functor \( F^e : (\mathcal{G}^e)^i \rightarrow (\mathcal{G}^e)^i \) which acts on objects as \( F \).

2. Let \( G : \mathcal{G} \rightarrow \mathcal{G} \) be an \( e \)-extensible functor. The functor \( G^e : \mathcal{G}^e \rightarrow \mathcal{G}^e \) is defined as follows:
   
   - \( G^e(A) = G(A) \)
   - given an embedding \( f : A \rightarrow B \), let \( f(A) \) be the image of \( A \) under \( f \), let \( f' : A \rightarrow f(A) \) be the canonical surjective embedding obtained restricting \( f \) and \( i : G(f(A)) \rightarrow G(B) \) be the inclusion function; then put \( G^e(f) = i \circ G^e(f') \).

One can easily see that the functors \&, \( \odot \), \( \rightarrow \), \( ! \) are \( e \)-extensible. Notice that the functor \((\rightarrow)^e\) is covariant in each of its arguments. In particular, given games \( A \) and \( B \) with move sets \( M_A \) and \( M_B \), the action of these functors on these move sets is the following:

- \( M_{A \odot B} = M_{A \rightarrow B} = M_{A \& B} = M_A + M_B \);
- \( M_{! A} = \sum M_A = \omega \times M_A \).

while the action on the embeddings is the following. Let \( f : A \rightarrow B \) and \( g : A' \rightarrow B' \) then:

- \( f \odot g = f \& g = [f, g] \);
- \( f \rightarrow g = [f, g^{-1}] \);
- \( !f = \langle n, a \rangle \mapsto \langle n, f(a) \rangle \).

Observe that the category \( \mathcal{G}^e \) is isomorphic to a subcategory of \( \mathcal{G} \) and to a subcategory of \( \mathcal{G}^{op} \). In fact, each embedding \( f : A \rightarrow B \) in \( \mathcal{G}^e \) induces two morphisms \( f^+ : A \rightarrow B \) and \( f_- : B \rightarrow A \) in \( \mathcal{G} \) defined as follows.

**Definition 5.6.** Given an embedding \( f : A \rightarrow B \), put \( f^+ = \{ t \in P_{A \rightarrow B} \mid t \in s_f \} \) and \( f_- = \{ t' \in P_{B \rightarrow A} \mid t' \in s_f \} \) where \( s_f \) is the least set satisfying:

\[
s_f = \{ \epsilon \} \cup \{ t a f(a) \mid t \in s_f, a \in M_A \} \cup \{ t' f(a) a \mid t' \in s_f, a \in M_A \}
\]
Moreover, each e-extensible functor \( F \), continuous w.r.t. to \( \leq \), has a fixed point.

**Theorem 5.7 (Existence of fixed points).** Let \( F : \mathcal{G} \to \mathcal{G} \) be an e-extensible functor which is continuous w.r.t. \( \leq \), let \( D \) be a game and let \( f : D \to F(D) \) be an embedding. Let \( \langle D_\infty, \mu_n \rangle \) be the colimit of the chain \( \langle (F^n)D, (F^n)f \rangle \). The game \( D_\infty \) is the fixed point of the functor \( F^e \).

**Proof.** Since \( \mathcal{G} \) is co-complete, if \( \langle F(D_\infty), F(\mu_n) \rangle \) is a colimit then \( D_\infty \cong F(D_\infty) \). But if \( F \) is continuous then also \( F^e \) is continuous (and hence also co-continuous) and then it preserves colimits. \( \square \)

Using the methodology and results above, we present some models of untyped \( \lambda \)-calculus in the category \( \mathcal{G} \).

**Example 5.8 (Extensional game models).** Let \( \text{Fun} : \mathcal{G} \to \mathcal{G} \) be the functor defined by \( \text{Fun}(D) = D \to D \) and for \( f : A \to B, \text{Fun}(f) = f \to f \). Let \( D^e_\infty, D^o_\infty \) and \( D^s_\infty \) respectively be the colimits of the chains obtained by iterating the functor \( \text{Fun} \) on the following starting games and initial embeddings:

\[
D^e_0 &= \{ * \mapsto \text{in}_{r}(*), \{ \epsilon, \epsilon \}, \{ * \mapsto \text{in}_{r}(*), \{ \epsilon, \epsilon \}, \{ * \mapsto \text{in}_{r}(*), \{ \epsilon, \epsilon \}) \}
\]

\[
f^e_0 &= \{ * \mapsto \text{in}_{r}(*), \{ \epsilon, \epsilon \}, \{ * \mapsto \text{in}_{r}(*), \{ \epsilon, \epsilon \}, \{ * \mapsto \text{in}_{r}(*), \{ \epsilon, \epsilon \}) \}
\]

\[
D^o_0 &= \{ * \mapsto \text{in}_{r}(*), \{ \epsilon, \epsilon \}, \{ * \mapsto \text{in}_{r}(*), \{ \epsilon, \epsilon \}, \{ * \mapsto \text{in}_{r}(*), \{ \epsilon, \epsilon \}) \}
\]

\[
f^o_0 &= \{ * \mapsto \text{in}_{r}(*), \{ \epsilon, \epsilon \}, \{ * \mapsto \text{in}_{r}(*), \{ \epsilon, \epsilon \}, \{ * \mapsto \text{in}_{r}(*), \{ \epsilon, \epsilon \}) \}
\]

\[
D^s_0 &= \{ * \mapsto \text{in}_{r}(*), \{ \epsilon, \epsilon \}, \{ * \mapsto \text{in}_{r}(*), \{ \epsilon, \epsilon \}, \{ * \mapsto \text{in}_{r}(*), \{ \epsilon, \epsilon \}) \}
\]

\[
f^s_0 &= \{ * \mapsto \text{in}_{r}(*), \{ \epsilon, \epsilon \}, \{ * \mapsto \text{in}_{r}(*), \{ \epsilon, \epsilon \}, \{ * \mapsto \text{in}_{r}(*), \{ \epsilon, \epsilon \}) \}
\]

The models \( D^e_\infty, D^o_\infty \) and \( D^s_\infty \) are all extensional and, hence, they belong to the class \( D^e \).

**Example 5.9 (Scott’s trick).** Let \( A_N = (N, \lambda n.OQ, \{ \epsilon \} \cup N, id) \). The model \( D_N \) is the one naturally induced by the least fixed point of the functor \( F(D) = N \to N \) where the following chain holds for every \( n \in N \): \( D_{n+1}^N \cong \text{!}(D_n^N \to A_N) \& \text{!(}D_{n+1}^N \to A_N) \cong 1 \text{!}(D_n^N \to A_N) \& \text{!(}D_{n+1}^N \to A_N) \cong D_{n+1}^N \). Hence we have \( D_{n+1}^N \cong D_n^N \to A_N \cong 1 \text{!}(D_n^N \to A_N) \cong D_{n+1}^N \). One can easily see that \( D_N \) is a \( \lambda \)-model since any bijection \( p : N + N \to N \), induces an isomorphism between \( A_N \) and \( A_N \).

**Example 5.10 (Non-extensional game models).** Let \( \text{Fun}^B_A(D) = 1 \text{!}(D \to D) \times A \) for an arbitrary game \( A \). Let \( D_B^B \) be the colimit to the chain obtained iterating \( \text{Fun}^B_A \) on the initial empty game \( I \) and the unique embedding \( f : I \to A \). It is immediate to see that \( D_B^B \in D_B \).

**Example 5.11 (Lazy game models).** Let \( A \) be a game. Let \( A_\perp \) \( [AM95b] \) be the game defined as:

\[
M_{A_\perp} = M_A \cup \{ o \mapsto \bullet \}
\]

\[
\lambda_{A_\perp} = \lambda_A \cup \{ o \mapsto OQ, \bullet \mapsto PA \}
\]

\[
P_{A_\perp} = \{ e \} \cup \{ o \cdot s \mid s \in PA \}
\]

\[
\approx_{A_\perp} = \{ (e, e), (o, o) \} \cup \{ (o \cdot s, o \cdot o) \mid (s, t) \in \approx_{A} \}
\]

Let \( \text{Fun}^C \) be the functor defined by \( \text{Fun}^C(D) = 1 \text{!}(D \to D) \perp A \) for an arbitrary game \( A \). It is immediate to see that \( D_C \), obtained as the colimit of the chain arisen by iterating \( \text{Fun}^C \) on the initial game \( I \), belongs to the class \( D_C \).
5.2 The theories of the game models

Proposition 4.38 states a general property of game models which have strong consequences. In particular, the very fact that each game model \( D \in D \) is approximable implies a kind of rigidity of the game semantics, providing a strong bias towards head reduction. As we shall show, in the category \( G \), history-free strategies interpreting a \( \lambda \)-term precisely reflect the Böhm or Lévy-Longo structure of the term.

**Lemma 5.12.** Let \( M, N \in \Lambda \) and let \( D \) be a (non-trivial) approximable model of \( \lambda \)-calculus. \( M \not\equiv N \Rightarrow \langle [M] \rangle^D \not= \langle [N] \rangle^D \).

*Proof.* By Proposition 3.59 and Theorem 3.69. \( \Box \)

**Lemma 5.13.** For each model \( D \in D^C \), \( \lambda \)-terms \( M, N \), if \( M \) and \( N \) are both unsolvable but of different order then \( \langle [M] \rangle^D \not= \langle [N] \rangle^D \).

*Proof.* By the Approximation Theorem 3.69, for each unsolvable term \( P \) of order 0, \( \langle [P] \rangle^D = \epsilon_{\not\equiv D} \) since \( \Omega \) is the only approximant of \( P \). For a term \( Q \) unsolvable of order 1 \( \langle [Q] \rangle^D \not= \epsilon_{\not\equiv D} \) since

\[
\langle [Q] \rangle^D = \psi \circ \Lambda ([\langle [Q] \rangle^D]) = \psi \circ \epsilon_{\not\equiv D} \not= \epsilon_{\not\equiv D}
\]

by definition of \( D^C \) since \( Q \) is unsolvable of order 0. We can conclude that \( \langle [P] \rangle^D \not= \epsilon_{\not\equiv D} \) for \( P \) unsolvable of order \( n > 0 \) since \( \lambda x. \Omega \in A^C (P) \) and \( \langle [\lambda x. \Omega] \rangle^D \not= \epsilon_{\not\equiv D} \). If \( M \) and \( N \) are unsolvable of different order, say respectively \( n \) and \( m \) with \( n < m \), then the context \( C \) is such that \( C[M] \) is unsolvable of order 0 and \( C[N] \) unsolvable of order \( m \) and hence the thesis follows. \( \Box \)

**Lemma 5.14.** For each non-extensional game model \( D \in D^B \cup D^C \), for each variable \( x \) and \( \lambda \)-term \( M \equiv \lambda y. M' \) we have that \( \langle [x] \rangle^D \not= \langle [\lambda y. M'] \rangle^D \).

*Proof.* By contradiction: \( \langle [x] \rangle^D = 1_\Delta \) and \( \langle [\lambda y. M'] \rangle^D = \psi \circ \tau \) for some suitable strategy \( \tau : D \Rightarrow (D \Rightarrow D) \). If \( \psi \circ \tau = 1_\Delta \), this would mean that the strategy \( \psi \) has a left (by definition of retract) and a right inverse, which, by categorical arguments, need to coincide, contradicting the hypothesis that \( \psi \circ \varphi \not= 1_\Delta \). \( \Box \)

**Lemma 5.15.** Let \( M, N \) solvable \( \lambda \)-terms such that \( M = \lambda x_1 \ldots x_n. y M_1 \ldots M_m \), \( N = \lambda x_1 \ldots x_m. y N_1 \ldots N_m \) and let \( D \in D^B \cup D^C \). If \( n \not= n' \) then \( \langle [M] \rangle^D \not= \langle [N] \rangle^D \).

*Proof.* Suppose \( n < n' \), it is not difficult to find a context \( C \) s.t. \( C[M] = x \) and \( C[N] = \lambda y. N' \). The thesis then follows from Lemma 5.14. \( \Box \)

**Lemma 5.16.** Let \( D \in D \) be a game \( \lambda \)-model, and let \( M \equiv x M_1 \ldots M_m \) and \( N \equiv x N_1 \ldots N_m \) be two \( \lambda \)-terms. If \( \langle [M] \rangle^D = \langle [N] \rangle^D \) then \( (\forall 1 \leq i \leq m)(\langle [M_i] \rangle^D = \langle [N_i] \rangle^D) \).

*Proof.* Suppose by contradiction that there exists \( 1 \leq i \leq m \) such that \( \langle [M_i] \rangle^D \not= \langle [N_i] \rangle^D \). Then there exists a play \( s \in [M_i] \) such that \( s \not\equiv t \) for each \( t \in [N_i] \). By definition of the interpretation of \( \lambda \)-terms, it is possible to calculate that the strategy \( \langle [M] \rangle^D : !D_x \otimes !D_{x_1} \otimes \cdots \otimes !D_{x_m} \Rightarrow D \) replies to the initial question of the Opponent repeating the question on a particular copy of \( D_x \). Let us suppose that the Opponent follows in \( !D_x \) a strategy which consists in, if interrogated on the \( j \)-th copy, of replaying the moves of the Player in the strategy \( \langle [M_j] \rangle^D \). Let us call \( s' \in [M] \) the play which originates by replaying the initial question of the Opponent on the \( j \)-th copy of \( !D_x \) and following the play \( s \) in the strategy \( \langle [M_i] \rangle^D \). The
strategy \( [N]^D \) contains a play \( t' \) equivalent to \( s' \), only if \( [N]^D \) contains a play \( t \) equivalent to \( s \), but this is negated by hypothesis. Therefore \( [M]^D \neq [N]^D \).

5.2.1 The theory of models \( \mathcal{D}^L \)

**Proposition 5.17.** For each model \( D \in \mathcal{D}^L \), \( \lambda \)-terms \( M, N \)

\[
LLT(M) = LLT(N) \Rightarrow [M]^D = [N]^D
\]

**Proof.** If \( LLT(M) = LLT(N) \) then \( A^L(M) = A^L(N) \) and then the thesis follows from Theorem 3.69.

**Proposition 5.18.** For each model \( D \in \mathcal{D}^L \), \( \lambda \)-terms \( M, N \)

\[
[M]^D = [N]^D \Rightarrow LLT(M) = LLT(N)
\]

**Proof.** We prove the converse. If \( LLT(M) \neq LLT(N) \) then there exist \( k \in \omega \) such that \( LLT^k(M) \neq LLT^k(N) \). We show that \( [M]^D \neq [N]^D \) by induction on \( k \).

1. If \( k = 0 \) then one of the following cases need to occur:
   
   (a) The two terms are not similar. In this case from Lemma 5.12 one readily obtains the thesis.
   
   (b) The two terms are both unsolvable but of different order. In this case from Lemma 5.13 the thesis follows.
   
   (c) The two terms are similar but with a different number of abstractions. In this case Lemma 5.15 applies.

2. If \( k = l + 1 \), then \( M \) and \( N \) are equivalent, solvable, with the same number of lambda abstractions. Suppose that their principal hnf's are respectively \( \lambda x_1 \ldots x_n y M_1 \ldots M_m \) and \( \lambda x_1 \ldots x_n y N_1 \ldots N_m \). There exists then \( i \) such that \( LLT^i(M_i) \neq LLT^i(N_i) \). By induction hypothesis \( [M_i]^D \neq [N_i]^D \), and then from Lemma 5.16 \( [M]^D \neq [N]^D \).

**Theorem 5.19.** For each \( D \in \mathcal{D}^L \) \( D \models \mathcal{L} \) that is, for \( M, N \in \Lambda \),

\[
[M]^D = [N]^D \Leftrightarrow LLT(M) = LLT(N)
\]

**Proof.** By Propositions 5.17 and 5.18.

5.2.2 The theory of models \( \mathcal{D}^G \)

The equational theory \( \mathcal{B} \), induced by the models \( D \in \mathcal{D}^G \), is the theory induced also by the models \( P^w \) [Plo72, Plo93, Sco87, Bar84], \( T^w \) [Plo78, BL80] and the filter models [BCDC83, CDCHL84, Ale90, Ron82]. In the game semantics setting, a model whose theory is \( \mathcal{B} \) is built in [KNO99] in the category of games and innocent strategies.

**Proposition 5.20.** For each model \( D \in \mathcal{D}^G \), \( \lambda \)-terms \( M, N \)

\[
BT(M) = BT(N) \Rightarrow [M]^D = [N]^D
\]

**Proof.** By Theorem 3.69.
Proposition 5.21. For each model \( D \in D^S \), \( \lambda \)-terms \( M, N \)
\[
[M]^D = [N]^D \Rightarrow BT(M) = BT(N)
\]

Proof. The proof is exactly the same as the proof of Proposition 5.18 where the case 1b is dropped since it cannot occur for Böhm trees. \( \square \)

Theorem 5.22. For each \( D \in D^S \) \( D \models \mathcal{B} \) that is, for \( M, N \in \Lambda \),
\[
[M]^D = [N]^D \Leftrightarrow BT(M) = BT(N)
\]

Proof. By Propositions 5.20 and 5.21. \( \square \)

5.2.3 The theory of models \( D^C \)

In [DGF99] it is shown that all models in the class \( D^C \) induce the same theory which is \( \mathcal{H}^* \), the maximal sensible theory. In that work, by introducing suitable categorical techniques, a game model \( D^* \in D^C \) is built and its equational theory is shown to be exactly \( \mathcal{H}^* \). Then, all the possible models \( D \in D^C \) are shown to be applicative isomorphic to \( D^* \). In [KNO01] a game lambda model inducing the theory \( \mathcal{H}^* \) is built in the category of games and innocent strategies.

In this section we exhibit a more general result: each extensional approximable model of untyped \( \lambda \)-calculus enforces the theory \( \mathcal{H}^* \).

Theorem 5.23. For each \( D \in D^C \) \( D \models \mathcal{H}^* \).

Proof. Using Theorem 3.64 and standard techniques it can be shown that the structure of the approximants \( A^C(M) \) coincide precisely with those of Scott’s \( D_{\infty} \) model as in the continuous case. Moreover, their corresponding Böhm trees satisfy the \( \infty\eta \)-equivalence. The argument for the continuous case then applies also in the game semantics setting. \( \square \)

5.2.4 The main result

We are now able to state the main result of the thesis.

Theorem 5.24. The theory induced by a categorical model \( D \in D \) (Definition 5.1) of untyped \( \lambda \)-calculus in \( K_1(G) \) is either

1. \( \mathcal{H}^* \), the theory induced by the canonical \( D_{\infty} \) model of Scott [Sco72, Bar84] and [Wad78], if \( D \in D^C \);
2. \( \mathcal{B} \), the theory which identifies two terms iff they have the same Böhm tree, if \( D \in D^S \);
3. \( \mathcal{L} \) the theory which identifies two terms iff they have the same Lévy-Longo tree if \( D \in D^L \).

Proof. By Theorems 5.19, 5.22 and 5.23. \( \square \)
Further developments
A Finitary Logical Description of Game Semantics

He thought he saw a Garden-Door
That opened with a key;
He looked again, and found it was
A Double Rule of Three.
“And all its mystery,” he said,
“Is clear as day to me!”
Lewis Carroll, Alice in Wonderland

Abstract

Following the intuitions of Stone [Sto36] resumed later by Johnstone [Joh82] and Abramsky [Abr91], we take advantage of a duality existing in standard denotational semantics to give a finitary logical description of the game semantics for untyped \( \lambda \)-calculus introduced in chapters 4 and 5, in the style of [BCDC83, CDCHL84, Ale90]. A proof of the coincidence of the two presentations is given.

Domain theory, the mathematical theory of computation introduced by Scott [Sco70] as a foundation for denotational semantics, has been extensively studied since its appearance in 1970. It was originally presented as a model theory for computation, but it was immediately evident the effective character of domains construction and, hence, a finitary, set-theoretic presentation followed [Sco82].

The two different presentations may be connected through the mathematical theory of Stone duality [Sto36, Joh82, Abr91]. The denotational semantics of a programming language in domain theory can be given in two forms: a program (term) can be interpreted by some element of a particular domain, or by a set of properties enjoyed by that program. The Stone-duality theorems tell us that these two alternative descriptions are equivalent, establishing a junction between semantics (space of points) and logics (lattices of properties). In this setting, the properties of a term are usually called types and a set of rules, that allow to
derive the properties satisfied by a term, is called type assignment system. In the literature
type assignment systems are mainly used in the semantics of lambda calculi, but, in principle,
they can be used as an alternative description of any kind of denotational semantics, based
on domain theory.

Games, as presented in Chapter 4, reflect, since the definition, their finitary mathematical
nature. A type assignment system in the context of game semantics, provides a more concrete
and intuitive account of the interpretation of terms given by game models. In fact, standard
game semantics is given using a categorical definition; to derive a concrete definition from the
categorical one can be a tiring task. In the game semantics, the plays of a game are particular
instances of a computation. If a play $s$ belongs to the strategy $\sigma$ interpreting a $\lambda$-term $M$,
it means that the term $M$, in a suitable environment, can perform the computation steps
described by $s$. The underlying idea of the type assignment system we shall introduce, is
to consider plays as the relevant properties (types) of programs. This approach is fruitful
because the set of instances of the computation that a program $M$ can perform, describes
completely the strategy interpreting $M$.

The very fact that game models of untyped $\lambda$-calculus presented in this thesis are indeed
categorical models has many considerable consequences. First of all, a finitary description of
the semantics of $\lambda$-calculus must reflect the finitary nature of the models built in the chosen
category of games. In particular, a type assignment system will be introduced for all the
reflexive objects which are obtained as categorical inverse limit constructions.

Secondly, the non-concrete nature of the category of games $\mathcal{G}$ implies a defined shape of
the assignment rules. The type system we shall introduce allows to derive judgments in the form

$$ \vdash_{\Delta} M : s $$

where $M$ is a $\lambda$-term whose free variables are in the list $\Delta = \{x_1, \ldots, x_n\}$ and $s$ is a play in
the game model under consideration.

It is worth noting that the environment never appears in a judgment, since it is included
in the play itself. In fact, a play in the strategy $\sigma$ contains also the moves describing the
interaction of $M$ with the environment. The traditional form of the judgment for the type
assignment system, that is $\Gamma \vdash M : s$, with $\Gamma$ a description of the environment, must be
abandoned because, using the above traditional form of judgment, a term is described by
an extensional function from the values of the free variables of the term to the value of the
term itself, but this description is not sound for the category $\mathcal{G}$.

### 6.1 The type assignment system

The main purpose of the type assignment system we are going to introduce, is the finitary
characterization of the game semantics of untyped $\lambda$-terms. Before introducing the rules of
the system, it is convenient to define a suitable notation for the moves of a game categorical
$\lambda$-model. The game models we shall take in consideration are obtained as categorical limit
constructions, and hence are characterized by a starting game – which will be indicated
generically as $D_0$ – and by a starting embedding – $f_0 : D_0 \hookrightarrow F(D_0)$, where the iterating
functor is $F(D) = \uparrow D \Rightarrow D$.

Moves will be indicated by sequences of labels ending with a move $a$ belonging to the
starting game $D_0$. The sequence of labels in each move will identify the necessary unfoldings
to make on the model to obtain the correct shape of the game where the move resides. We
use juxtaposition to indicate sequence concatenation. Let $a, b$ be moves of games $A$ and $B$
respectively, and \( i \in \omega \). We indicate with \( ia \) and \( rb \) the moves \( i_m(a) \) and \( i_n(b) \) in the game \( A \rightarrow B \), with \( ia \) the move \((i, a)\) of the game \(!A\) and, if \( i < n \), we indicate with \( ia \) also the obvious move in the game \((A \& \ldots \& A)\).

For what concerns plays, we will use the symbol \( \cdot \) to indicate concatenation of plays. The use of two different symbolisms for concatenation, one for moves and one for plays, allows to omit parenthesis.

In order to present the type assignment system we need to define the following (partial) functions on sequences of moves. Remember that \( D^n \) indicates the game \((D \& \ldots \& D)\) and that \( D \Rightarrow D \) is the game \(!D \rightarrow D\).

**Definition 6.1.** Given a sequence \( s \) of moves of a game \( D \) (not necessarily giving rise to a valid play), let \( \Delta = \{x_1, \ldots, x_n\} \) be a list of variables with \( n \geq 1\):

1. the sequences \( cc^i_\Delta(s) \) and \( cc_\Delta(s) \) of moves of the game \( D^n \Rightarrow D \), are defined as follows:
   
   \[ \begin{align*}
   (a) & \quad cc^i_\Delta(e) = cc_\Delta(e) = \epsilon, \\
   (b) & \quad cc^i_\Delta(a \cdot s') = ra \cdot 0i_\Delta \cdot cc^i_\Delta(s'), \\
   (c) & \quad cc_\Delta(a \cdot s') = 0i_\Delta \cdot ra \cdot cc_\Delta(s');
   \end{align*} \]

2. Let \( \lambda \) be the function from moves in the game \( D^{n+1} \Rightarrow D \) to moves in \( D^n \Rightarrow (D \Rightarrow D) \) defined as follows:

\[ \lambda(a) = \begin{cases} 
   a & \text{if } a = lja' \text{ and } i \leq n \\
   rtja' & \text{if } a = l(j(n + 1)a') \\
   ra & \text{if } a = ra'
\end{cases} \]

Let \( \lambda^* \) be the extension of \( \lambda \) to plays.

3. Let \( s \) be a sequence of moves of the game \( D^n \Rightarrow (D \Rightarrow D) \) and \( t_1, \ldots, t_m \) be sequences of moves in the game \( D^n \Rightarrow D \) and let \( j \leq n \). We indicate with \( (s|t_1| \ldots |t_m|j) \) a sequence of the game \( D^n \Rightarrow D \), defined as follows:

\[ \begin{align*}
   (a) & \quad (\epsilon| \ldots |\epsilon,i) = \epsilon; \\
   (b) & \quad (rra \cdot s|t_1| \ldots |t_m,0) = ra \cdot s|t_1| \ldots |t_m,0; \\
   (c) & \quad (rtia \cdot s|t_1| \ldots |ra \cdot t'_i| \ldots |t_m,0) = (s|t_1| \ldots |t'_i| \ldots |t_m,0; \\
   (d) & \quad (rtia \cdot s|t_1| \ldots |ra \cdot t'_i| \ldots |t_m,i) = (s|t_1| \ldots |t'_i| \ldots |t_m,0; \\
   (e) & \quad (lhja \cdot s|t_1| \ldots |t_m,0) = (2h + 1)ja \cdot (s|t_1| \ldots |t_m,0; \\
   (f) & \quad (s|t_1| \ldots |lhja \cdot t'_i| \ldots |t_m,i) = lh(2p(i,j))a \cdot (s|t_1| \ldots |t'_i| \ldots |t_m,i)
\end{align*} \]

where \( p : \omega \times \omega \rightarrow \omega \) is a pairing function, that is a bijection between \( \omega \times \omega \) and \( \omega \). Take, for example, the function \( \langle x, y \rangle \mapsto (2x + 1)2^y \). Observe that the sequence \( (s|t_1| \ldots |t_m|j) \) is not always defined. If it is defined we write \( (s|t_1| \ldots |t_m|j) \downarrow \).

Observe that if \( s \) is a valid play in the game \( D \), then \( cc^*_\Delta(s) \) is a valid play in the game \( D^n \Rightarrow D \). Moreover one can easily prove that \( cc^*_\Delta(s) \) is a play of the “copy-cat” strategy on the \( i^{th} \) argument ([AJ94a]), and that every play of the copy-cat strategy on the \( i^{th} \) argument can be obtained in this way.
It is not difficult to observe that the function \( \lambda \) defines an isomorphism between the games \( D^{n+1} \Rightarrow D \) and \( D^n \Rightarrow (D \Rightarrow D) \).

The play \((s|t_1| \ldots |t_m,0)\) is the result, as defined by game semantics, of the interaction between the play \(s\), seen as the instance of the computation of a function, and the plays \(t_1, \ldots, t_m\), seen as several instances of the computation of the argument of the function. Since a function can interrogate its argument several times, more than one instance of the computation for the argument is necessary. The instance of computation \(s, t_1, \ldots, t_m\) can also interact with an environment represented by the game \(D^n\). In the expression \((s|t_1| \ldots |t_m, i)\) the index \(i\) is necessary to indicate in which instance of the computation the next move has to be made.

Rule 3b considers the case where \(s\) has to move and makes a move \(a\) on the result component; in this case \(a\) appears also on the result of the interaction. Rule 3c considers the case where \(s\) has to move and makes a move \(a\) on the \(i^{th}\) copy of the argument. If \(t_i\) can make the same move \(a\), then there is an interaction between function and argument, the two instances of \(a\) are eliminated and the control is passed to the \(i^{th}\) copy of the argument. If \(s\) and \(t_i\) do not agree, it means that there cannot be any interaction between these particular instances of the computation. Rule 3d is the dual of rule 3c; it considers the case where \(t_i\) has to move and makes a move \(a\). If \(s\) can make the same move \(a\) then there is an interaction between function and argument, the two instances of \(a\) are cancelled and the control is given to the function. Rule 3e considers the case where \(s\) has to move and makes a move \(a\) in correspondence of the \(n^{th}\) copy on the \(j^{th}\) variable of the environment. In this case, the move \(a\) appears also on the result of the interaction, but in a different copy of the environment. Rule 3f is the dual of rule 3e, and considers the case where the \(i^{th}\) copy of the argument is in control.

**Definition 6.2 (The type assignment system).** Let \(M, N\) be \(\lambda\)-terms, let \(D\) be a game categorical model and let \(\Delta\) be a list of variables. The type assignment system has the following rules given in a natural deduction system style:

\[
\frac{s \in P_D}{\vdash_{\Delta} x_i : cc^{-}_{\Delta}(s)} \quad (var)
\]

\[
\frac{\vdash_{\Delta} x : M \quad s}{\vdash_{\Delta} \lambda x : M : \lambda^{s} s} \quad (\lambda)
\]

\[
\frac{\vdash_{\Delta} M : s \quad \vdash_{\Delta} N : t_1 \cdots \vdash_{\Delta} N : t_m}{\vdash_{\Delta} MN : (s|t_1| \ldots |t_m, 0)} \quad (app)
\]

the side condition of the rule (app) being that \((s|t_1| \ldots |t_m, 0)\)

### 6.2 Equivalence of semantics

We shall show now that game denotational semantics and the semantics given by the type assignment system are equivalent. We show, in particular, that the set of plays assigned to a term \(M\) by the type assignment system constitute a strategy, the same strategy which interprets the term \(M\) as defined by the categorical denotational semantics.
Theorem 6.3. Given a game categorical model \( D \), for each \( \lambda \)-term \( M \) whose free variables are among the list \( \Delta = \{ x_1, \ldots, x_n \} \), \([M]_\Delta^D = \{ s \mid \vdash \Delta M : s \} \).

Proof. By induction on the structure of \( M \).

- \( M \equiv \lambda x. P \). In this case \([x]_\Delta^D = \pi^\Delta_1 \). By the categorical definition of \( \pi^\Delta_1 : D^n \Rightarrow D \), it is immediate to observe that \( \pi^\Delta_1 \) is the “copy-cat” strategy on the \( i \)th argument, that is the same strategy defined by the type assignment system.

- \( M \equiv \lambda x. P \). In this case \([\lambda x. P]_\Delta^D = \psi \circ \Lambda([P]_\Delta^D, x) \). In our sequence notation for moves, we use the same sequence to denote a move \( d \) in \( D \) and the equivalent (unfolded) move in \( D \Rightarrow D \). It follows that the strategies \( \psi \circ \Lambda([P]_\Delta^D, x) \) and \( \Lambda([P]_\Delta^D, x) \) are denoted in the same way.

It is not difficult to prove that the isomorphism between the game \( D^{n+1} \Rightarrow D \) and the game \( D^n \Rightarrow (D \Rightarrow D) \) defined by the function \( \Lambda \) coincides with the one defined by the function \( \lambda^* \).

- \( M \equiv PQ \). In this case we have: \([PQ]_\Delta^D = ev \circ ((\varphi \circ [P]_\Delta^D), [Q]_\Delta^D) \). By the same arguments used in the previous point, in our notation the strategies \( \varphi \circ [P]_\Delta^D \) and \([P]_\Delta^D \) are denoted in the same way. The remaining part of the proof can be obtained by explicating the categorical definition of ev.

\[ \square \]
7

Geometry of Interaction and other stories

“... diminution of infinity is not the implementation of a clever, but artificial algorithm.”

J.-Y. Girard [Gir95]

Abstract

Game semantics can be seen as a programming languages semantics paradigm which lives in a more general semantical framework: the Geometry of Interaction, introduced by Girard [Gir88a, Gir88b, Gir95]. In this chapter, a survey of some aspects of the semantics of untyped \( \lambda \)-calculus connected with the Geometry of Interaction is exposed. The general notion of linear combinatory algebra as a model of untyped \( \lambda \)-calculus and a general categorical method to build such structures [Abr97] are introduced. The notions involved are those of traced monoidal category and weak linear category. The game semantics (as presented in Chapter 4) is then described as an application of this general method to the category \text{Rel}.

7.1 Linear combinatory algebras

Linear combinatory algebras were introduced by Abramsky [Abr97] with the explicit intent of clarifying what the Geometry of Interaction paradigm of Girard [Gir89] should be able to model. He was interested, in particular, in the kind of models of untyped \( \lambda \)-calculus obtainable through a Geometry of Interaction construction. The study carried out suggested an extended notion of applicative structure which naturally arises in such a context: the linear combinatory algebras.

Definition 7.1 (Linear combinatory algebra). A linear combinatory algebra \( A \) is a structure \( \langle A, \cdot, ! \rangle \), where \( \langle A, \cdot \rangle \) is an applicative structure, \( ! : A \to A \) is an injective operator and there exist elements \( B, C, I, K, W, D, \delta, F \) in \( A \) such that:
\[
\begin{align*}
B \cdot x \cdot y \cdot z &= x \cdot (y \cdot z) \\
C \cdot x \cdot y \cdot z &= (x \cdot z) \cdot y \\
K \cdot x \cdot (!y) &= x \\
W \cdot x \cdot (!y) &= x \cdot (y \cdot (!y)) \\
F \cdot (!x \cdot (!y)) &= !(x \cdot y) \\
D \cdot (!x \cdot y) &= x \cdot y \\
\delta \cdot (!x) &= !(!x) \\
I \cdot x &= x
\end{align*}
\]

A linear combinatory algebra \( \langle A, \cdot, ! \rangle \) can be easily made a model of untyped \( \lambda \)-calculus. In fact, it is possible to derive from it a standard combinatory algebra \( \langle A, \cdot_s, K_s, S_s \rangle \) (see [AHPS98], [Hag00] or [Gir87]) defining for each \( a, b \in A \):

\[
\begin{align*}
a \cdot_s b & \triangleq a \cdot (!b) \\
K_s & \triangleq D \cdot K \\
S_s & \triangleq B_s \cdot_s (B_s \cdot_s (B_s \cdot_s W_s) \cdot_s C_s) \cdot_s (B_s \cdot_s B_s) \\
B_s & \triangleq C \cdot (B \cdot (B \cdot B) \cdot (D \cdot I)) \cdot (C \cdot ((B \cdot B) \cdot F) \cdot \delta) \\
C_s & \triangleq D \cdot C \\
W_s & \triangleq D \cdot W
\end{align*}
\]

**Proposition 7.2.** \( \langle A, \cdot_s, K_s, S_s \rangle \) is a combinatory algebra.

**Proof.** A straight verification from the definitions above. \( \square \)

### 7.1.1 A case study: \( Pfn(\omega, \omega) \)

An example of linear combinatory algebra could clarify where did the combinators \( B, C, I, K, W, D, \delta, F \) come from. Consider the structure \( \langle Pfn(\omega, \omega), \cdot, ! \rangle \) where \( Pfn(\omega, \omega) \) is the set of partial endofunctions on the natural numbers. There are two retractions: \( \langle \omega \times \omega < \omega, t, t^{-1} \rangle \) and \( \langle \omega \times \omega < \omega, p, p^{-1} \rangle \), which allow, in the spirit of the Geometry of Interaction, to define the application \( - \cdot - \) and the operator \(!\) . Notice that \( t \) and \( p \) are total injective functions.

**Application**

The application acts as follows. \( \alpha \) is transformed, through the tagging functions \( t, t^{-1} \) in a 2-input, 2-output box \( t^{-1} \circ \alpha \circ t \). Since \( t : \omega + \omega \to \omega \) is defined on a disjoint union (categorical co-product) it can be written in the unique form \([t_1, t_2] \) with \( t_1 \) that applies on arguments \( n = \text{in}_l(n') \) and \( t_2 \) on arguments \( n = \text{in}_r(n') \). The function \( t^{-1} \circ \alpha \circ t \) can then be decomposed in four disjoint partial functions:

\[
\begin{align*}
\alpha_{1,1} & \triangleq t_1^{-1} \circ \alpha \circ t_1 \\
\alpha_{2,1} & \triangleq t_1^{-1} \circ \alpha \circ t_2 \\
\alpha_{1,2} & \triangleq t_2^{-1} \circ \alpha \circ t_1 \\
\alpha_{2,2} & \triangleq t_2^{-1} \circ \alpha \circ t_2
\end{align*}
\]
The application $\alpha \cdot \beta$ is then defined (in terms of the graph of the resulting function) as follows:

$$\alpha \cdot \beta \triangleq \alpha_{1,1} \cup \alpha_{2,1} \circ \beta \circ (\alpha_{2,2} \circ \beta)^* \circ \alpha_{1,2}$$

and can be depicted as:

![Diagram](image)

We can figure out the dynamic aspects of this “function application” by thinking of a token who enters the box $t^{-1} \circ \alpha \circ t$ at the only external input (the left one). It is then an element $n = \text{in}_1(n')$. One of two cases needs to occur: either $n' \in t^{-1}_1(\omega)$ or $n' \in t^{-1}_2(\omega)$. In the first case the function $\alpha_{1,1}$ applies, the value $\alpha_{1,1}(n')$ is presented to the output and the computation ends.

If $n' \in t^{-1}_2(\omega)$, the function $\alpha_{1,2}$ applies and the value $n'' = \alpha_{1,2}(n')$ is presented to the box of the function $\beta$. Then the value $\beta(n'')$ is fed again to $t^{-1} \circ \alpha \circ t$. This time it is an element $\text{in}_r(m)$ and, then, one of $\alpha_{2,1}$ and $\alpha_{2,2}$ has to apply. The first case puts the transformed token to the output halting the computation. The second one feeds again the token to $\beta$ and a new token is presented again to $t^{-1} \circ \alpha \circ t$ on the right input.

Notice that the token could not reach the output link for two reasons: one of the $\alpha_{i,j}$ ($i, j = 1, 2$) is not defined on an input or the loop $(\alpha_{2,2} \circ \beta)^*$ never ends. The two cases are different only from an intensional point of view, since the external behavior of $\alpha \cdot \beta$ is the same in the two cases.

**Combinators**

The combinator $\textbf{I}$, can be obtained from the function $\text{twist} : \omega + \omega \rightarrow \omega + \omega$
analytically defined by

$$\text{twist}(n) = \begin{cases} in_r(n') & \text{if } n = in_1(n') \\ in_1(n'') & \text{if } n = in_r(n'') \end{cases}$$

as

$$I \triangleq t \circ \text{twist} \circ t^{-1}$$

Observe that $I \cdot x$ can be represented as

and hence, by “yanking the string straight” this is the same as

For this reason, the Geometry of Interaction universe, is sometimes referred as the \textit{algebra of strings}.

A little more complex is the definition of the combinator $B$. Giving $f, g \in Pf \langle \omega, \omega \rangle$, put $f \otimes g : \omega + \omega \to \omega + \omega \triangleq [in_1 \circ f, in_r \circ g]$. $B$ can be defined as a 6-input, 6-output box (it acts on three arguments) from the following switching function $\pi$
as
\[ B \triangleq t \circ (t \otimes 1_\omega) \circ (t \otimes t) \circ \pi \circ (t^{-1} \otimes t^{-1} \otimes t^{-1}) \circ (t^{-1} \otimes 1_\omega) \circ t^{-1} \]

Similarly the combinator \( C \) can be defined. For an exhaustive analytical and diagrammatical definition of all the combinators refer to [AHPS98].

Combinators \( B, C \) and \( I \), however, give the so called BCI-algebras, which give only linear functional completeness, that is they are models only for linear \( \lambda \)-calculus (\( \lambda I \)-calculus). To obtain the full functional completeness we need to define combinators (as \( K \)) that can ignore completely an argument or combinators (as \( S \) in standard combinatory algebras or \( W \)) that use their arguments more than once.

**The operator !**

For the considerations above, it is necessary to introduce the operator \(!\), whose purpose is to make available zero or multiple copies of an argument. The operator \(!\) is defined on \( Pf n(\omega, \omega) \) as:

\[ !\alpha \triangleq p \circ (1_\omega \times \alpha) \circ p^{-1} \]

At this point all the remaining combinators can be defined. As an example we present the box which defines \( K \):

\[ K \]

Observe that \( K \cdot x \) is
and \((K \cdot x) \cdot y\) is

that with “yanking the string straight” (and deleting the dangling links) becomes

7.2 The general Geometry of Interaction construction

This section faces the problem of building linear combinatory algebras in an effective way. Abramsky [Abr96, Abr97] developed a method to obtain such structures starting from a (strict) traced monoidal category. A similar construction was proposed also in [JSV96].

7.2.1 Weak linear categories

The natural setting for linear combinatory algebras are the weak linear categories, where each reflexive object gives rise to a linear combinatory algebra.

**Definition 7.3 (Weak linear category [Abr97]).** A weak linear category \(\langle C, ! \rangle\) (WLC for short) consists of a symmetric monoidal closed category \(C\), a strong symmetric monoidal functor \(!: C \to C\) and the following monoidal natural transformations:

- \(\text{der}: ! \to I[I]\)
- \(\delta: ! \to !!\)
- \(\text{con}: I \to ! \odot !\)
- \(\text{weak}: ! \to I\)

**Definition 7.4 (Reflexive object in a WLC).** A reflexive object in a WLC is an object \(V\) such that \(\langle V \to V \cdot V, r, s \rangle, \langle V \cdot V, p, q \rangle\) and \(\langle I \cdot V, u, v \rangle\).

**Definition 7.5.** Let \(V\) a reflexive object in a WLC \(C\). Let \(f, g \in C(I, V)\). Define the following operations \(\bullet: C(I, V) \times C(I, V) \to C(I, V)\) and \(!: C(I, V) \to C(I, V)\) as follows:
\[
\begin{align*}
    f \bullet g & \triangleq ev_{V,V} \circ ((s \circ f) \otimes g) \\
    !f & \triangleq p \circ (!f) \circ \varphi_I
\end{align*}
\]
where \( \varphi_I : I \to !I \) is the arrow defined by the symmetric monoidal functor \(!\).

**Proposition 7.6.** \( \langle C(I,V), \bullet, ! \rangle \) is a linear combinatory algebra.

**Proof.** See Theorem 3.5 of [AHPS98]. \( \Box \)

### 7.2.2 The Geometry of Interaction (GoI) construction

The general construction of LCAs proposed by Abramsky [Abr97] focuses on a special class of WLCs, the traced symmetrical monoidal categories which originate a GoI situation. The construction clarifies the relation between “trace” and “computation”.

**Definition 7.7 (GoI situation [AHPS98]).** A GoI situation is a triple \( \langle C,T,U \rangle \) where

- \( C \) is a traced symmetric monoidal category;
- \( T : C \to C \) is a symmetric monoidal functor with the following retractions:
  1. \( \langle T \circ T \triangleleft T, e, e' \rangle \)
  2. \( \langle T \triangleright T, d, d' \rangle \)
  3. \( \langle T \otimes T \triangleleft T, c, c' \rangle \)
  4. \( \langle T \triangleright T, w, w' \rangle \)
- \( U \) is an object of \( C \) with the following retractions:
  1. \( \langle U \otimes U \triangleleft U, r, s \rangle \)
  2. \( \langle I \triangleleft U, u, v \rangle \)
  3. \( \langle TU \triangleright U, p, q \rangle \)

**Definition 7.8 (GoI construction [Abr96, JSV96]).** Given a traced symmetric monoidal category \( \langle C, \otimes, I, \alpha, \lambda, \beta, \gamma \rangle \) the category \( \mathcal{G}(C) \) is obtained as follows:

- **Objects:** \( (A^+, A^-) \) where \( A^+, A^- \in \text{Obj}(C) \)
- **Arrows:** \( f : (A^+, A^-) \to (B^+, B^-) \) where \( f \in C(A^+ \otimes B^-, A^- \otimes B^+) \)
- **Identity:** \( 1_{(A^+, A^-)} \triangleq \gamma_{A^+, A^-} \)
- **Composition:** \( g \circ f \triangleq \tau_{A^+ \otimes B^+, A^- \otimes C^+}(\beta \circ (f \otimes g) \circ \alpha) \)
  where
  \[
  \alpha = (1_{A^+} \otimes 1_{B^-} \otimes \gamma_{C^- \otimes B^-}) \circ (1_{A^+} \otimes \gamma_{C^- \otimes B^-} \otimes 1_{B^+})
  \]
  \[
  \beta = (1_{A^-} \otimes 1_{C^+} \otimes \gamma_{B^+ \otimes B^-}) \circ (1_{A^-} \otimes \gamma_{B^+ \otimes C^+} \otimes 1_{B^-}) \circ (1_{A^-} \otimes 1_{B^+} \otimes \gamma_{B^- \otimes C^+})
  \]
- The tensor structure of \( \mathcal{G}(C) \) is defined as follows. Given objects \( (A^+, A^-) \) and \( (B^+, B^-) \) of \( \mathcal{G}(C) \) and arrows \( f : (A^+, A^-) \to (B^+, B^-) \) and \( g : (C^+, C^-) \to (D^+, D^-) \):
  \[
  (A^+, A^-) \otimes (B^+, B^-) \triangleq (A^+ \otimes B^+, A^- \otimes B^-)
  \]
  \[
  f \otimes g \triangleq (1_{A^-} \otimes \gamma_{B^+ \otimes C^- \otimes 1_{D^+}}) \circ (f \otimes g) \circ (1_{A^+} \otimes \gamma_{C^+ \otimes B^- \otimes 1_{D^-}})
  \]
  \[
  I_{\mathcal{G}} \triangleq (I, I)
  \]
Definition 7.9. Let \( \langle C, T, U \rangle \) be a GoI situation. Define on the set \( G(C)(I_G, V) \) where \( V \triangleq (U, U) \) the operations \( \bullet : G(C)(I_G, V) \times G(C)(I_G, V) \rightarrow G(C)(I_G, V) \) and \( ! : G(C)(I_G, V) \rightarrow G(C)(I_G, V) \) in the following way:

\[
f \bullet g \triangleq \text{Tr}_{U,U}((I_U \otimes g) \circ (s \circ f \circ r))
\]

\[
!f \triangleq p \circ T(f) \circ q
\]

Theorem 7.10. Let \( \langle C, T, U \rangle \) be a GoI situation. The structure

\[
\langle G(C)(I_G, V), \bullet, ! \rangle
\]

is a linear combinatory algebra.

Proof. See [AHPS98].

\[\square\]

7.3 Graph models

7.3.1 Linear graph models

In chapter 3, a general \( \lambda \)-calculus model construction, which was called graph model \( \langle U, k \rangle \), was introduced. It consists of an infinite set \( U \) together with a retraction \( \langle P^{<\omega}(U) \times U \triangleq U, k, k^{-1} \rangle \). It generates an applicative structure \( \langle U = P(U), \cdot \rangle \) where

\[
x \cdot y = \{ a \mid \exists u \subseteq y, u \text{ finite } & k(u, a) \in x \}
\]

A similar construction is the linear graph model [Abr97].

Definition 7.11 (Linear graph model). A linear graph model \( \langle S, p, r^- \rangle \) consists of a set \( S \) together with two retractions \( \langle S \times S \triangleq S, p, p^{-1} \rangle \) and \( \langle P^{<\omega}(S) \triangleq S, r^- \rangle \).

It originates a linear combinatory algebra \( \langle S = P(S), \ast, ! \rangle \) where

\[
\begin{align*}
\alpha \ast \beta &= \{ y \mid \exists x.p(x, y) \in \alpha \& x \in \beta \} \\
!\alpha &= \{ r^u \mid u \text{ finite, } u \subseteq \alpha \}
\end{align*}
\]

Proposition 7.12. Given a linear graph model \( \langle S, p, r^- \rangle \), \( \langle P(S), \ast, ! \rangle \) is a linear combinatory algebra.

Proof. Example: \( W \triangleq \{ \langle r^u, \langle r^v, x \rangle \rangle, \langle r^u \cup v^\uparrow, x \rangle \} \).

\[\square\]

7.3.2 Linear and standard graph models

It is of interest to investigate the relation between standard graph models – which yield a large class of \( \lambda \)-calculus models – and linear graph models.

Proposition 7.13. For each standard graph model \( \langle U, k \rangle \) there exists a linear graph model \( \langle S, p, r^- \rangle \) such that \( \langle U, \cdot \rangle \cong \langle S, \ast \rangle \).

Proof. Since there is a retraction \( \langle P^{<\omega}(U) \times U \triangleq U, k, k^{-1} \rangle \), there exists always a retraction \( \langle P^{<\omega}(U) \triangleq U, \sigma, \sigma^{-1} \rangle \). Put then \( S = U, p = k \circ (\sigma^{-1} \times 1_U) \) and \( r^- = \sigma \).

\[
x \cdot y &= \{ a \mid \exists u \subseteq y, u \text{ finite } & k(u, a) \in x \} \\
&= \{ a \mid \exists u \subseteq y, u \text{ finite } & k(\sigma^{-1}(\sigma(u)), a) \in x \} \\
&= \{ a \mid \exists b \in !y \& k(\sigma^{-1}(b), a) \in x \} \\
&= x \ast (\{ y \}) \\
&= x \ast s y
\]

\[\square\]
Each λ-theory induced by a standard graph model can then be accomplished in a linear graph model fashion. Moreover also the converse holds.

**Proposition 7.14.** For each linear graph model \( \langle S, p, \tau \rangle \) there exists a standard graph model \( \langle U, k \rangle \) such that \( \langle U, \cdot \rangle \cong \langle S, \ast \rangle \).

**Proof.** The same as above by putting \( k = p \circ (\tau \cdot \cdot \cdot 1_U) \).

### 7.3.3 Some linear combinatory algebras arising from GoI construction

The GoI construction introduced before, allows to build a linear combinatory algebra from a traced symmetric monoidal category. It is natural to ask if all the LCAs can be obtained through a GoI construction. This question is very important to clarify what the Geometry of Interaction allows to build, since LCAs derived from linear graph models are isomorphic to combinatory algebras derived from standard graph models.

We shall focus on the category \( \text{Rel} \) equipped with the tensor product given by the Cartesian product of sets \( \times \), its unit given by the singleton set \( \{*\} \) and the trace operator defined by

\[
\text{Tr}_{U,U}^f := \{ (p(x,x'), p(y,y')) \mid (\langle x,x' \rangle, \langle y,y' \rangle) \in f \}
\]

Let us remind that the bifunctor \( \times : \text{Rel} \times \text{Rel} \to \text{Rel} \) is a right adjoint to itself, that is \( \text{Rel}(A \times B, C) \cong \text{Rel}(A, B \times C) \). Moreover, if \( U \) is a set such that \( U \times U \) is a retract of \( U \) via \( p, p^{-1} \) the functor \( \times \) yields a GoI situation \( \langle \text{Rel}, U \times -, U \rangle \).

**Definition 7.15.** Given the GoI situation \( \langle \text{Rel}, U \times -, U \rangle \), let us consider the linear graph model \( \langle (U \times U), \tau, \tau \rangle \) with the retractions \( (U \times U) \times (U \times U) \triangleleft (U \times U), \tau, \tau^{-1} \) and \( (\mathcal{P}^\omega(U \times U) \triangleleft (U \times U), \tau, \tau^{-1}, \cdot, \cdot^{-1}) \). The LCA it induces is \( \langle \mathcal{P}(U \times U), \cdot, \cdot \rangle \) where, for \( \alpha, \beta \in \mathcal{P}(U \times U) \),

\[
\alpha \cdot \beta = \{ (a, b) \in U \times U \mid \exists (c, d) \in \beta, \tau((a, b), (c, d)) \in \alpha, (c, d) \in \beta \}
\]

The GoI situation \( \langle \text{Rel}, U \times -, U \rangle \) induces, through the general construction presented above, the LCA \( \langle \mathcal{P}(U \times U), \cdot, \cdot \rangle \). To visualize how the application \( \cdot \) acts, it is useful to remember that \( \alpha \cdot \beta \) is realized as

![Diagram](attachment://diagram.png)

where \( p \) is the retraction between \( U \times U \) and \( U \), that is

\[
\begin{align*}
\alpha \cdot \beta &= \{ (a, b) \in U \times U \mid \exists (c, d) \in \beta, (a, d), (b, c) \in p^{-1} \circ \alpha \circ p \} \\
&= \{ (a, b) \in U \times U \mid \exists (c, d) \in \beta, p(a, d), p(b, c) \in \alpha \}
\end{align*}
\]
It is then clear that, if we have the LCA \( \langle \mathcal{P}(U \times U), \cdot, ! \rangle \) induced by a GoI construction we can always build an isomorphic structure \( \langle \mathcal{P}(U \times U), \cdot, ! \rangle \) induced by a linear graph model \( \langle U \times U, \tau, \Gamma \rangle \) by putting \( \tau((a,b), (c,d)) = \langle p(a, d), p(b, c) \rangle \). It is an open problem if also the converse holds.

### 7.4 Game semantics

In this section it will be shown how it is possible to obtain the category Games of Chapter 4 as a Geometry of Interaction construction. To this aim we need the specification properties construction of [AGN96].

#### 7.4.1 The category of resumptions

The category to work on is the category \( \mathcal{R} \) of sets and resumptions [Abr96]. Given sets \( X \) and \( Y \), the space of resumptions with inputs in \( X \) and outputs in \( Y \) is any solution to the recursive specification

\[
R = X \rightarrow (Y \times R)
\]

This equation was firstly introduced and solved by Milner [Mil75] in a category of domains. Abramsky [Abr96] pointed out that it could be solved also in the category \( \text{Set} \).

**Definition 7.16 (The category of resumptions \( \mathcal{R} \) [Abr96]).** The category of resumptions \( \mathcal{R} \) is defined as follows:

**Objects:** Sets

**Arrows:** \( \mathcal{R}(X, Y) = X \rightarrow (Y \times \mathcal{R}(X, Y)) \)

**Composition:** \((g \circ f)(x) = \begin{cases} \langle z, g' \circ f' \rangle & \text{if } f(x) = \langle y, f' \rangle, g(y) = \langle z, g' \rangle \\ \text{undefined} & \text{otherwise} \end{cases} \)

**Identity:** \(1_X(x) = \langle x, 1_X \rangle \)

**Definition 7.17 (Symmetric monoidal structure on \( \mathcal{R} \) [Abr96]).**

\( \otimes \) is defined on \( \mathcal{R} \) as follows: for sets \( X \) and \( Y \) pose

\[
X \otimes Y \triangleq X + Y
\]

If \( f \in \mathcal{R}(X, Y) \) and \( g \in \mathcal{R}(X', Y') \), \( f \otimes g \in \mathcal{R}(X \otimes X', Y \otimes Y') \) is defined by:

\[
(f \otimes g)(\text{in}_L(x)) = \begin{cases} \langle \text{in}_L(y), f' \otimes g \rangle & \text{if } f(x) = \langle y, f' \rangle \\ \text{undefined} & \text{otherwise} \end{cases}
\]

\[
(f \otimes g)(\text{in}_R(x')) = \begin{cases} \langle \text{in}_R(y'), f \otimes g' \rangle & \text{if } g(x') = \langle y', g' \rangle \\ \text{undefined} & \text{otherwise} \end{cases}
\]

The unit is the empty set \( I \triangleq \emptyset \). The natural isomorphisms \( \lambda_A \in \mathcal{R}(I \otimes A, A) \), \( \rho_A \in \mathcal{R}(A \otimes I, A) \) and \( \alpha_{A,B,C} \in \mathcal{R}((A \otimes B) \otimes C, A \otimes (B \otimes C)) \) are defined as expected:
\[ \alpha_{A,B,C}(\text{in}_1(\text{in}_1(a))) = \langle \text{in}_1(a), \alpha_{A,B,C} \rangle \]
\[ \alpha_{A,B,C}(\text{in}_1(\text{in}_r(b))) = \langle \text{in}_r(\text{in}_1(b)), \alpha_{A,B,C} \rangle \]
\[ \alpha_{A,B,C}(\text{in}_r(c)) = \langle \text{in}_r(\text{in}_r(c)), \alpha_{A,B,C} \rangle \]
\[ \lambda_A(\text{in}_r(a)) = \langle a, \lambda_A \rangle \]
\[ \rho_A(\text{in}_1(a)) = \langle a, \rho_A \rangle \]

**Definition 7.18 (Trace on \( \mathcal{R} \) [Abr96]).** A trace on \( \mathcal{R} \),
\[ \text{Tr}^U_{X,Y} : \mathcal{R}(X \otimes U, Y \otimes U) \to \mathcal{R}(X, Y) \]
for each \( X, Y, U \) is defined as follows:
\[ \text{Tr}^U_{X,Y}(f)(x) = \begin{cases} 
\langle y, f' \rangle & \text{if } \exists k. f(x) = \langle u_0, f_0 \rangle, \\
f_0(u_0) = \langle u_1, f_1 \rangle, \\
\vdots & \\
f_k(u_k) = \langle y, f' \rangle \\
\text{undefined otherwise}
\end{cases} \]

### 7.4.2 Specification structures

*Specification structures* have been introduced in [AGN96] to make possible to build a tower of categories
\[ C_0 \rightleftharpoons C_1 \rightleftharpoons C_2 \rightleftharpoons \cdots \rightleftharpoons C_k \]
where, for each \( n < k \), \( C_{n+1} \) is fully and faithfully embeddable in \( C_n \), and where \( C_0 \) is a category with structure suitable for modelling a computational situation, but, possibly, only carrying a rudimentary notion of “type”. \( C_0 \) is then successively refined with richer kinds of properties for its objects (types).

**Definition 7.19 (Specification structure).** Let \( C \) be a category. A specification structure \( S \) over \( C \) is defined by:

- for each object \( A \) of \( C \) there is a set \( PA \) of properties over \( A \);
- for each pair of objects \( A, B \) of \( C \) there is a relation \( R_{A,B} \subseteq PA \times C(A,B) \times PB \) which satisfies the following properties for \( f : A \to B, g : B \to C, \varphi \in PA, \psi \in PB, \theta \in PC \):
  \[ \varphi[1_A] \varphi \]
  \[ \varphi[f] \psi, \psi[g] \theta \Rightarrow \varphi[g \circ f] \theta \]

where, of course, \( R_{A,B}(\varphi, f, \psi) \) is written in the Hoare form \( \varphi[f] \psi \).

Provided we have a specification structure over a category, we can build a new category whose objects exhibit the properties introduced by the specification structure.

**Definition 7.20.** Given a specification structure \( S \) over a category \( C \), the category \( C_S \) is defined as follows:

- **Objects:** \( \langle A, \varphi \rangle \) with \( A \in \text{Obj}(C) \) and \( \varphi \in PA \)
- **Arrows:** \( f : \langle A, \varphi \rangle \to \langle B, \psi \rangle \) are arrows \( f : A \to B \) of \( C \) s.t. \( \varphi[f] \psi \)
Composition and identities are inherited from \( \mathcal{C} \). The properties of the relation \( R_{A, B} \) for every \( A, B \) assures that \( \mathcal{C}_S \) is indeed a category.

If the category \( \mathcal{C} \) has additional structure – for instance it is monoidal – it is in general easy to extend the specification structure with other notions in such a way that the original structure of \( \mathcal{C} \) is lifted to \( \mathcal{C}_S \).

If \( \mathcal{C} \) is a monoidal category, we need an element \( U \in PI \), and, for each pair of objects \( A \) and \( B \), an operation \( \otimes_{A, B} : PA \times PB \to P(A \otimes B) \), such that, for \( f : A \to B \), \( f' : A' \to B' \) and properties \( \varphi, \varphi', \psi, \psi' \), \( \theta \) we have:

\[
\varphi[f] \psi, \varphi'[f'] \psi' = \varphi \otimes \varphi'[f \otimes f'] \psi \otimes \psi' \\
(\varphi \otimes \psi) \otimes \theta(\alpha_{A, B, C}) \varphi \otimes (\psi \otimes \theta) \\
U \otimes \varphi(\lambda_A) \varphi \\
\varphi \otimes U(\rho_A) \varphi
\]

If \( \mathcal{C} \) is symmetric monoidal closed, we further need for each pair of objects \( A \) and \( B \), an operation \( \to_{A, B} : PA \times PB \to P(\to A \to B) \), such that, for each \( f : A \otimes B \to C \) and \( \varphi \in PA, \psi \in PB \) and \( \theta \in PC \):

\[
\varphi \otimes \psi \gamma_{A, B} \psi \otimes \varphi \\
(\varphi \to \psi) \otimes \varphi(e_{A, B}) \psi \\
\varphi \otimes \psi[f] \theta \Rightarrow \varphi(A(f)) \psi \to \varphi
\]

7.4.3 Finding again the category Games

(Section 4.4.2.1). The application of the construction of Definition 7.8 to the category \( \mathcal{R} \) of sets and resumptions originates a category \( \mathcal{G}(\mathcal{R}) \) which can be interpreted as a rudimentary category of games, in the sense of Chapter 4. Each object \( (A^+, A^-) \) can be thought of a partition of a set of moves \( A \triangleq A^+ \cup A^- \), where \( A^+ \) is the set of moves labelled as Player and \( A^- \) the set of moves labelled as Opponent. Each morphism \( f : (A^+, A^-) \to (B^+, B^-) \) is a resumption \( f' \) with inputs in \( A^+ \otimes B^- = A^+ + B^- \) and outputs in \( A^- \otimes B^+ = A^- + B^+ \), which has to be intended as a strategy for the Player, in the form of a partial function from Opponent moves to Player moves, extended in time. However, there are no restrictions on what \( f \) could do, while in the introduction of games we wanted to pose limits to what a play has to be, defining explicitly the set of admissible plays in the definition of game itself. In other words, to obtain the category Games from \( \mathcal{G}(\mathcal{R}) \), we have to introduce the set of valid plays (the game tree) as an additional property for the objects of \( \mathcal{G}(\mathcal{R}) \), obtaining in this way objects of the shape \( (A, P_A) \) with an implicit labelling function \( \lambda_A : A \to \{P, O\} \) defined by \( \{in_l(a^+) \mapsto P, in_r(a^-) \mapsto O\} \).

**Definition 7.21.** Let \( S \) be the specification structure over the category \( \mathcal{G}(\mathcal{R}) \) defined as follows:

- for each object \( A = (A^+, A^-) \) let \( P_A \) be a non-empty, prefix-closed subset of \( M_A^{\text{alt}} \), which is the set of alternating sequences of moves from \( A^- \) and \( A^+ \), starting with a move of \( A^- \);
- for each pair of objects \( A, B \) there are operators \( \otimes_{A, B} \) and \( \to_{A, B} \) specified as in Definitions 4.5 and 4.8 which satisfy the needed properties;
- for each pair of objects \( A = (A^+, A^-) \) and \( B = (B^+, B^-) \),

\[
R_{A, B} \triangleq \{[\varphi]f[\psi] \mid f \in \mathcal{G}(\mathcal{R})(A, B), \text{Plays}(f) \subseteq \varphi \to \psi\} 
\]
where for each resumption \( f \) we can define its set of plays as:

\[
\text{Plays}(f) = \{ x_1 y_1 \ldots x_k y_k \mid f(x_1) = (y_1, f_1), \ldots, f_{k-1}(x_k) = (y_k, f_k) \}
\]

**Theorem 7.22.** \( \mathcal{G}(\mathcal{H})_S \cong \text{Games} \).

**Proof.** Each object of \( \mathcal{G}(\mathcal{H})_S \) is an object of Games and vice versa. For each arrow \( f \in \mathcal{G}(\mathcal{H})_S(A, B) \) there is a corresponding strategy given by \( \text{Plays}(f) \) in Games\((A, B)\) and, as shown in Chapter 4, each strategy in Games is representable as a partial function extended in time. \( \square \)
Conclusions

“How can you do new math problems with an old math mind?”

Charlie Brown

In this thesis we have shown that the application of the intensional semantical paradigm of games and history-free strategies to untyped $\lambda$-calculus reveals that game semantics is strongly connected to head and weak head reduction. It is very difficult to escape from this cage: the strategy that interprets a term in a categorical game model describes exactly the tree form of the term, that depends only on the global properties of the model. If the model is extensional, all terms with same Böhm tree up to $\eta$-conversion are identified. If the model does not validate the $\eta$-rule all terms with same Böhm tree are still identified. If the model is able to distinguish unsolvable terms with different order, all terms with same Lévy-Longo tree receive the same interpretation.

Related work

The game semantics of untyped $\lambda$-calculus is then more rigid than topological semantics or naive set-theoretic models. This rigidity seems to be inherent in the structure of the strategies. In a recent work [DCIVZ98], it has been shown that the adjoining to pure $\lambda$-calculus of a construct, which aim is to represent a non-deterministic choice, allows for the internal discrimination of two terms with different Böhm-trees. Analogously, the ability of game semantics to discriminate terms with different Böhm trees resides on the nature of strategies. Every strategy has to consider, at each stage, each possible move of the Opponent, encapsulating, in this way, each possible choice operated by the environment.

An overcoming of the rigidity of game semantics can be achieved by altering the strong constraint that players should alternate. In [FMAF99], a new category is introduced where players should alternate only in a round of the game, that is, in two consecutive moves starting in an odd position. More specifically, in every sequence of moves of the game, two consecutive odd and even position moves can not be played by the same player but there is freedom in the choice of the player starting the next round. In this way a model for a standard CCS-like process algebra can be built.

A successful attempt to discard the original rigidity of game semantics is considered also in [DG00], where a fully-abstract model for pure lazy $\lambda$-calculus (without constants) is built. For this purpose a new category $g_M$ of monotonic games is introduced. In this category, moves of the games are questions or answers ordered by a notion of strength. A question $a$ is stronger than a question $b$ if it asks for more information and the definition extends similarly to the answers. Asking that a play should evolve with stronger and stronger moves, restricts

\footnote{From PEANUTS® by Charles Schulz © 1964 United Feature Syndicate Inc.}
the way a strategy can behave, allowing to obtain, in this way, a fully-abstract model for lazy $\lambda$-calculus.

A similar, but weaker, result is obtained in [AM95a], where a fully-abstract model for the lazy $\lambda$-calculus, with a test convergence constant $\Phi$, is built in the category $\mathcal{E}$ of games and history-free strategies obtained as an extensional collapse of $\mathcal{G}$. The main difference between [DG00] and [AM95a] resides on the fact that the model in [DG00] is fully-abstract for pure lazy $\lambda$-calculus while the model in [AM95a] needs to extend the original calculus to achieve a full-definability result.

The extensional collapse is a tool to model coarser operational theories than the three presented in this thesis. Given the Sierpinski game $\Sigma$, which has only two strategies, denoted as $\bot$ and $\top$, with $\bot \subseteq \top$, the intrinsic preorder on the strategies of a game $A$ is defined as

$$\sigma \leq_A \tau \iff (\forall \alpha : A \rightarrow \Sigma)(\alpha \circ \sigma = \top \Rightarrow \alpha \circ \tau = \top)$$

Denoting with $\simeq_A$ the equivalence relation associated with the preorder $\leq_A$, the category $\mathcal{E}$ has as objects games and as morphisms from $A \rightarrow B$ the equivalence class of strategies for $A \rightarrow B$ under $\simeq$. This technique was used also in standard denotational semantics, where quotients of domains were introduced with the same purpose.

A different road has been taken by Ong et al. in [KNO01, KNO99], where universal models (models where each element is the denotation of some term) for the theories $\mathcal{H}^*$ and $\mathcal{B}$, are introduced in the slightly different setting of games and innocent strategies of Hyland and Ong [HO00] and Nickau [Nic96]. The key idea in these works is that the innocent strategies definable by untyped $\lambda$-terms are, what they called, effectively almost-everywhere copy-cat (EAC). An EAC strategy is constrained to behave as a copy-cat strategy but for the response to finitely many moves made by the Opponent. The model $\mathcal{P}_{EAC}$ obtained as a global section of a reflexive object in the category $\mathcal{A}_{EAC}$ of arenas and innocent EAC strategies is shown to enforce the theory $\mathcal{H}^*$. A natural extension of the EAC strategies are the EXAC (explicitly and effectively almost-everywhere copy-cat) strategies, which allow to build the universal model $\mathcal{M}$, which induces the theory $\mathcal{B}$.

Open problems

A challenging theme, in our opinion, is the definition of the syntactical constructs which allow to define exactly all the “semantical points” of our models. It would be interesting to build universal models in $\mathcal{G}$. Notice that the models $D^B_{\omega}$, $D^C_{\omega}$, $D^B_{\infty}$ (Example 5.5.8), $D^B$ (Example 5.5.10) and $D^C$ (Example 5.5.11) clearly are not universal. To see this in the case $e.g.$ of $D^B_{\infty}$, let us consider the following strategy $\sigma : \downin D^B_{\infty} \rightarrow D^B_{\infty}$

$$\sigma \circ_i \downin D^B_{\infty} \rightarrow D^B_{\infty} \cong D^B_{\infty},$$

where the subscript in the second move indicates which component of the exponential game it is in. Such a strategy is not the denotation of any term of $\lambda$-calculus. To see why this is the case, let us remind the behavior of a strategy interpreting a $\lambda$-term in a categorical model of $\lambda\beta$-calculus in $\mathcal{G}$. The initial question by the Opponent is a request of data: the value of the term. If the term is unsolvable the Player does not answer, otherwise it asks for the value of the head variable. The Opponent has then to provide this value and usually it has to ask for the value of some arguments the head variable is applied to.
The strategy $\sigma$ depicted above, answers the initial move $\circ$ of the Opponent interrogating ($\circ_1$) the head variable of the term and never answering to a following (possible) question by the Opponent asking for the value of an argument. Since the number of arguments of the head variable could be any number, $\sigma$ can be only seen as defining the behavior of the term $\lambda x.x\Omega\Omega\cdots\Omega\cdots$ which is a term of infinite $\lambda$-calculus.

In Chapter 7, the main problem which is still unresolved is whether each linear combinatory algebra can be obtained through a Geometry of Interaction construction. We have shown that, in the category $\mathbf{Rel}$, each linear combinatory algebra obtained through a Geometry of Interaction construction can be also obtained from a linear graph model and that there is not a clear motivation for the converse to hold. The validity of the converse implication would be very important to clarify what the general Geometry of Interaction construction can give us.

In [Bar84], Chapter 5, a combinatory complete applicative structure $\mathcal{A} = \langle A, \cdot \rangle$ is said a categorical $\lambda$-model ($\lambda$-algebra, combinatory algebra) if there is a unique expansion $\langle A, \cdot, k, s \rangle$ making $\mathcal{A}$ into a $\lambda$-model ($\lambda$-algebra, combinatory algebra). We conjecture that, in the category $\mathcal{G}$, each combinatory complete applicative structure is categorical.

In Chapter 6, a type assignment system is introduced to give a finitary logical description of the game semantics of untyped $\lambda$-calculus. The type system presented there, concerns each categorical model of the untyped $\lambda$-calculus $D$ obtained as an inverse limit construction, and uses this fact to give structured names to the moves of the model game. An extension of the type system to a generic model has to be done.
Bibliography


Bibliography


Bibliography


Bibliography


