Functorial Operational Semantics

and its Denotational Dual

Ψ

Daniele Turi

PhD Thesis
Free University, Amsterdam
June 1996

Author's e-mail address: dt@dcs.ed.ac.uk

VRIJE UNIVERSITEIT

Functorial Operational Semantics

and its Denotational Dual

ACADEMISCH PROEFSCHRIFT

ter verkrijging van de graad van doctor aan
de Vrije Universiteit te Amsterdam,
op gezag van de rector magnificus
prof.dr E. Boeker,
in het openbaar te verdedigen
ten overstaan van de promotiecommissie
van de faculteit der wiskunde en informatica
op donderdag 6 juni 1996 te 15.45 uur
in het hoofdgebouw van de universiteit, De Boelelaan 1105

door

Daniele Turi

geboren te Salerno, Italië

Promotor: prof.dr J.W. de Bakker
Copromotoren: dr B.P.F. Jacobs
dr J.J.M.M. Rutten
Referent: dr A.M. Pitts

Research supported by:

- The "Stichting Informatica Onderzoek in Nederland"
  (Project: *Non-well-founded sets and semantics of programming languages*)
- The CWI, Amsterdam
- The Laboratory for Foundations of Computer Science, Edinburgh
- The European Institute in the Logical Foundations of Computer Science (*EuroFOCS*)
- The European Community SCIENCE programme
  (Project: *Mathematical Structures in Semantics of Concurrency*)

a Lucia D'Agosto, per la sua fiducia fatta di universali

e

alla memoria di Giovanni Turi – e della sua filosofia

met dank aan mijn paranimfen Maurizio Gabbrielli en Roos Vogel

Preface

The notion of 'functorial operational semantics' introduced in this thesis is a categorical formulation (and generalization) of 'well-behaved' structural operational semantics based on labelled transition systems. This notion has several desirable properties (such as congruence of the associated strong bisimilarity, and existence of a dual denotational semantics) and it subsumes existing, concrete schemes (such as GSOS) for guaranteeing such good behaviour – at least in the case of languages extending 'basic process algebra'. All this is achieved via use of the category theory of monads and comonads. The thesis also contains a coalgebraic treatment of the theory of non-well-founded sets which simplifies and improves some aspects of Peter Aczel's original presentation.

Non-well-founded sets have played an important rôle in the development of the whole thesis: by working within Jan Rutten and Jaco de Bakker's project 'non-well-founded sets and programming languages semantics', I have had the opportunity of distilling the mathematical foundations for the main contribution of the thesis, the introduction of the functorial approach to operational semantics.

Most of the research presented here has been conducted at the CWI, in Amsterdam. I can hardly imagine a better place to work on a thesis: the serene atmosphere, the international contacts, the superb library, the efficient organization, and the building itself, with quiet, balanced rooms, have made of this institute an ideal place for conducting pure research.

Jaco de Bakker's department at the CWI is part of EuroFOCS, the European institute in the logical foundations of computer science. This has offered me the opportunity of spending six, most profitable months at LFCS, Edinburgh, visiting Gordon Plotkin, one of whose many contributions to the theory of computer science has been the introduction of the *structural* approach to operational semantics.

When, in the early 80's, it was introduced, the novelty of structural operational semantics was that of bringing the mathematics of (structural) induction in the operational description of the behaviour of programming languages, providing a powerful formal tool for reasoning about programs. The present *functorial* approach can be seen as one step further in that direction: based on a suitable interplay between inductive and (dual) coinductive principles, it provides a mathematical definition and treatment of 'well-behaved' structural operational semantics.

The contact with Gordon Plotkin has been crucial both for this thesis and for my general development. Particularly vivid in my memory is the image of a beautiful February of two years ago, when, during some discussions with him, the blackboard looked like self-drawing; the last picture he drew, with "algebras over coalgebras", has been decisive for formulating the notion of functorial operational semantics.

The development of this notion, in Edinburgh, has been influenced by exciting discussions with Marcelo Fiore and Alex Simpson. More generally, Marcelo has been precious for my whole research activity.

Conceived, for the functorial part, in Edinburgh, this thesis has been written in Amsterdam. Thanks to very frequent reviewing sessions with my supervisors, Jaco de Bakker, Bart Jacobs, and Jan Rutten, the writing has rapidly converged to its final form, in a natural and serene rhythm.

Jaco, one of the pioneers of the mathematical approach to the semantics of programming languages which inspires this thesis, has granted me the room to develop the mathematics I felt most suitable, free from any prejudice. Almost without realizing it, I have written a much more thorough thesis than I had imagined, thanks to his gentle, but steady influence.

Jan, who brought me to the CWI, has collaborated to the development of coalgebraic methods in semantics which has been the basis for the research presented here. Bart, with his secure knowledge of category theory, has been a constant source of suggestions, corrections, and improvements. His limpid mind has always been available for discussions. Like Jan, he has shown great interest in and has collaborated to the foundational work on coalgebras.

The last step in the preparation of this thesis, the refereeing process, is due to Andy Pitts, who has been very sympathetic to the problems tackled and the methods used in this thesis. In this preface, I have used many expressions plundered from his precise summarizing words.

The 'palaestra' for my early scientific development has been the 'Amsterdam Concurrency Group' led by Jaco and including Marcello Bonsangue, Frank de Boer, Franck van Breugel, Arie de Bruin, Joost Kok, Erik de Vink, and Herbert Wiklicky. Nostalgically, I remember the first three-sessions talk I gave there, a promising winter of four years ago.

Marcello "kamergeno(o)t" Bonsangue, together with Franck room-mate in the beloved M335, has shared these early developments and my growing interest in category theory. He is one of the extraordinarily many Italians who, from Catuscia Palamidessi on, have been at the CWI over the years. One of the persons who are most 'responsible' for this Italian 'colonization' is Krzysztof Apt; he was also the supervisor of my "tesi di laurea" for the University of Pisa, in my 'prehistorical' time at the CWI.

Also at LFCS I have been surrounded by Italians or Italian speakers. One of them, Pietro 'everywhere' Di Gianantonio, has also been my colleague at the CWI and in the European SCIENCE project 'Mathematical Structures in Semantics of Concurrency'. This project has been an important forum for discussions to me; apart from the CWI, the sites involved have been the university of Koblenz (Lutz Priese), Mannheim (Mila Majster-Cederbaum), Pisa (Ugo Montanari), and Udine (Furio Honsell), and the IRISA-INRIA of Rennes, where, in particular, I have had

fruitful contacts with Eric Badouel and Philippe Darondeau.

At the CWI, I have enjoyed discussions with Fer-Jan de Vries, Tim Fernando, and Femke van Raamsdonk, the efficient secretarial support by Mieke Bruné and Marja Hegt, the technical support by the Computer Help Information Desk, and the outstanding library service. My visit to Edinburgh has been arranged thanks to George Cleland and Monika Lekuse's help at LFCS.

Most of the economic support for this thesis has been provided by the "Stichting Informatica Onderzoek in Nederland" of the Dutch organization for scientific research (NWO); my grant has been handled in a particularly friendly way by Richard Kellermann Deibel and Virginie Meijer-Mes. The remainder of the support has come from the SCIENCE project and from EuroFOCS.

I have tried to write this thesis in the most unassuming way, trying to communicate m-my p-personal experience of discovering, through elementary problems, the beauty and necessity of the universals of category theory, a discovery which has turned my mathematical activity into a "fröhliche Wissenschaft".

Daniele Turi – Amsterdam, April 1996

Contents

Introduction

> "It is all very well to aim for a more 'abstract' and a 'cleaner' approach to semantics, but if the plan is to be any good, the operational aspects cannot be completely ignored. The reason is obvious: in the end the program still must be run on a machine – a machine which does not possess the benefit of 'abstract' human understanding, a machine that must operate with finite configurations. Therefore, a mathematical semantics, which will represent the first major segment of the complete, rigourous definition of a programming language, must lead naturally to an operational simulation of the abstract entities, which – if done properly – will establish the practicality of the language, and which is necessary for a full presentation."
>
> Dana Scott, *Outline of a Mathematical Theory of Computation*

> "Many modern programming languages are inconsistent with standard mathematical foundations. The task of finding sound interpretations for what it is that computer scientists do strikes this writer as, perhaps, the highest type of applied mathematics. It is akin to the process that has been going on throughout the 20th Century with respect to physics. The interaction between the mathematicians and the practitioners in each case has resulted in the growth of both subjects."
>
> Peter Freyd, *Computer Science Contradicts Mathematics*
> lecture at the Int'l Conf. on Category Theory
> held in Como, Italy, July 1990 (see [Fre91])

The *operational semantics* of a programming language accounts for a formal description of the *behaviour* of the programs, specifying the way programs should be executed and the kind of behaviour which should be observable. The operational semantics is usually contrasted with the *mathematical* interpretation of the programs called *denotational semantics*.

This thesis presents a new mathematical approach to the semantics of programming languages aimed at bridging the gap between the operational and the denotational aspects of semantics. This is based on a suitable interplay between the standard *induction principle* which pervades modern mathematics, and the dual '*coinduction principle*' which has led to *non-standard* mathematical foundations.

In order to introduce coinduction as the dual of induction, it is convenient to move from the traditional presentation of induction in the language of *set theory* to a presentation in the language of *category theory*. The primitive notions of category theory are those of composition and equality of abstract functions called *arrows*, like the notions of membership and equality of those abstract collections called sets are the primitives notions of set theory. Now, every statement expressible in the language of category theory can be straightforwardly *dualized* by 'reversing the arrows'. (*Duality principle*.)

Induction. In set theory, mathematical induction is based on the notion of a *well-founded* relation, that is, a relation R such that, for every set x, there is *no* infinitely descending chain

$$\ldots R\, x_2\, R\, x_1\, R\, x_0 = x$$

For instance, one can perform induction on the set $N = \{0,\, 1 = s(0),\, 2 = s^2(0),\, \ldots\}$ of natural numbers by using the well-foundedness of the order relation on them

$$0 < s(0) < s^2(0) < \cdots < s^n(0) = n$$

as follows.

> *Recursion Theorem.* Given a set X, an element $e \in X$ and a function $g : X \to X$, there *exists* a *unique* function $f : N \to X$ from the set of natural numbers to the given set such that
>
> $$f(0) = e \qquad \text{and} \qquad f(s(n)) = g(f(n))$$
>
> for all numbers $n \in N$.

The value e of the function f at (the least element) 0 (wrt the order relation) is the 'base' of the induction and g defines the 'inductive step'.

The fact that standard mathematical constructions are inductive is mirrored by the common assumption that the axioms of set theory include the *axiom of foundation* which postulates that the set-membership relation '$\in$' is well-founded: for every set x, there exists no infinitely descending chain

$$\ldots \in x_2 \in x_1 \in x_0 = x$$

The axiom of foundation allows an inductive (idealized) construction of sets starting from the empty set (the base) and recursively applying the power-set operator mapping a set to the set of its subsets. The induction is on those generalized natural numbers which are the ordinal numbers.

In this thesis, an equivalent categorical formulation of the foundation axiom is given which allows for a straightforward dualization. This is best illustrated starting from the above recursion theorem:

The recursion theorem can be taken as the *definition* of natural numbers. That is, every set N with a distinguished element $0 \in N$ and a unary operation $s : N \to N$ such that the recursion theorem holds, is *isomorphic* to the natural numbers. (See, eg, [Mac86, Chapter 2].) As pointed out by Lawvere, the existence/uniqueness statement of the recursion theorem asserts the **universal property** characterizing the natural numbers: *initiality*. This property underlies induction, not only on the natural numbers, but in general.

Category Theory. The mathematical study of universal properties is called category theory. It is based on an abstract notion of function called *arrow*

$$f : X \to Y$$

which formally is a triple: name (f), domain (X), and codomain (Y).

A category is a collection of arrows with a *composition* operation '$\circ$' which obeys generalized monoidal laws: any two arrows $f : X \to Y$ and $g : Y \to Z$ which 'match' in the sense that the codomain of f is the same as the domain of g can be composed

$$\begin{tikzcd} X \arrow[r, "f"] \arrow[rr, bend right, "g \circ f"'] & Y \arrow[r, "g"] & Z \end{tikzcd}$$

to form the arrow $g \circ f : X \to Z$; the composition of arrows is associative, ie $f \circ (g \circ h) = (f \circ g) \circ h$; the domains and codomains of the arrows are called the *objects* of the category and for every object X there exists an *identity arrow* $\mathsf{id}_X : X \to X$ which is both a left and a right unit for the composition, ie $\mathsf{id}_Y \circ f = f = f \circ \mathsf{id}_X$.

The archetypal category is **Set**, having sets as objects and functions as arrows. However, it is very misleading (especially at the beginning!) to try and understand the *universals* of category theory in terms of **Set**.

The most elementary universal property which an object of a category can enjoy is initiality: an object X is initial in a category if, for every object Y of the category, there *exists* an arrow $f : X \to Y$ from X to Y and, moreover, this arrow is *unique*.

The basic way of understanding the natural numbers as an initial object is by regarding them as an object $\langle N, 0, s \rangle$ in the category having as objects triples $\langle X, e, t \rangle$, where X is a set with a distinguished element $e \in X$ and a function $t : X \to X$ on it. The arrows $f : \langle X, e, t \rangle \to \langle X', e', t' \rangle$ of the category are functions $f : X \to X'$ such that

$$f(e) = e' \qquad \text{and} \qquad f(t(x)) = t'(f(x))$$

(It is easy to verify that the above objects and arrows form a category with composition and identities as in **Set**.) Then the recursion theorem says exactly that the triple $\langle N, 0, s \rangle$ is initial in this category. (Notice that in the category **Set** the initial object is the trivial empty set.) Conversely, since initial objects, like all universals, are unique up to isomorphism, the initial object of this category defines the natural numbers up to isomorphism.

Next, a series of abstractions is necessary in order to generalize this specific form of initiality.

Firstly, notice that the element $e \in X$ of a set X can be written as a function from the one-element set $1 = \{*\}$ to the set X; that is, one can identify a function $e : 1 \to X$ from the one-element set 1 to a set X with its value $e(*) \in X$ at the unique element $*$ of 1. Then the recursion theorem amounts to having an object $1 \xrightarrow{0} N \xrightarrow{s} N$ such that for every object $1 \xrightarrow{e} X \xrightarrow{g} X$, there exists a unique function $f : N \to X$ with

$$f \circ 0 = e \qquad \text{and} \qquad f \circ s = g \circ f$$

Diagrammatically, using dashed arrows to denote arrows given by universal properties, one has that the following diagram commutes.

$$\begin{tikzcd}
1 \arrow[r, "0"] \arrow[d, equal] & N \arrow[r, "s"] \arrow[d, dashed, "f"] & N \arrow[d, dashed, "f"] \\
1 \arrow[r, "e"'] & X \arrow[r, "g"'] & X
\end{tikzcd}$$

Secondly, every pair of functions with the same codomain (thus, eg, $e : 1 \to X$ and $g : X \to X$) can be made into a single arrow with as domain the disjoint union of the domains. This holds in general in every category with *coproducts*: given two objects X and Y in a category, their coproduct, if it exists, is an object $X + Y$ with two arrows $\mathsf{inl}_X : X \to X + Y$ and $\mathsf{inl}_Y : Y \to X + Y$ which is *universal* in the sense that for every pair of arrows $f : X \to Z$ and $f : Y \to Z$ there *exists* a *unique* arrow $[f, g] : X + Y \to Z$, making the following diagram commute.

$$\begin{tikzcd}
X \arrow[r, "\mathsf{inl}_X"] \arrow[dr, "f"'] & X + Y \arrow[d, dashed, "{[f,g]}"] & Y \arrow[l, "\mathsf{inr}_Y"'] \arrow[dl, "g"] \\
& Z &
\end{tikzcd}$$

(The dual of the coproduct $X + Y$ is the product $X \times Y$: its *projections* $\mathsf{fst}_X : X \times Y \to X$ and $\mathsf{snd}_Y : X \times Y \to Y$ are universal among all pairs of arrows $f : Z \to X$ and $g : Z \to Y$.)

In **Set** the disjoint union, together with the corresponding injection functions, is a coproduct. Hence, one can write $[e, g] : 1 + X \to X$ instead of $1 \xrightarrow{e} X \xrightarrow{g} X$. Correspondingly, the initiality of the natural numbers can be expressed by saying that for every function $h : 1 + X \to X$ there exists a unique arrow $f : N \to X$ such that the following diagram commutes.

$$\begin{tikzcd}
1+N \arrow[r, "1+f"] \arrow[d, "{[0,s]}"'] & 1+X \arrow[d, "h"] \\
N \arrow[r, dashed, "f"'] & X
\end{tikzcd}$$

The arrow $1 + f : 1 + N \to 1 + X$ is defined by universality:

$$1 + f = [\mathsf{inl}_1 \circ \mathsf{id}_1,\ \mathsf{inr}_X \circ f] = [\mathsf{inl}_1,\ \mathsf{inr}_X \circ f] : 1 + N \to 1 + X$$

Thus the operation $X \mapsto 1 + X$ on objects extends to an operation $f \mapsto 1 + f$ on arrows: this defines a *functor* from **Set** to **Set**.

Functors are arrows between categories (regarded as objects!). A general criterion for forming a category from a collection of objects is to take as arrows the 'homomorphisms', that is, the morphisms which preserve the structure of the objects. Now, the structure of a category is given by composition and identities, and functors preserve it: a functor $F : \mathbf{C} \to \mathbf{D}$ from a category $\mathbf{C}$ to a category $\mathbf{D}$ maps every object X of $\mathbf{C}$ to an object FX of $\mathbf{D}$ and every arrow $f : X \to Y$ of $\mathbf{C}$ to an arrow $Ff : FX \to FY$ of $\mathbf{D}$ in such a way that

$$F(\mathsf{id}_X) = \mathsf{id}_{FX} \qquad \text{and} \qquad F(g \circ f) = Fg \circ Ff$$

The composition of functors can be then defined 'pointwise'.

Universal definitions are always functorial. For instance, given two functions $f : X \to Y$ and $g : Y \to Z$ one defines $1 + f : 1 + X \to 1 + Y$ by

$$\begin{tikzcd}
1 \arrow[r, "\mathsf{inl}_1"] \arrow[d, equal] & 1+X \arrow[d, dashed, "{1+f = [\mathsf{inl}_1,\ \mathsf{inr}_Y \circ f]}"] & X \arrow[l, "\mathsf{inr}_X"'] \arrow[d, "f"] \\
1 \arrow[r, "\mathsf{inl}_1"'] & 1+Y & Y \arrow[l, "\mathsf{inr}_Y"]
\end{tikzcd}$$

and then $1 + (g \circ f)$ is, by uniqueness, necessarily equal to $(1 + g) \circ (1 + f)$.

Algebras and Coalgebras. The third step of abstraction is now to move from the above (endo) functor $FX = 1 + X$ on **Set** to arbitrary endofunctors $F : \mathbf{C} \to \mathbf{C}$ and, correspondingly, to consider initial objects in categories of structures $h : FX \to X$ rather than $h : 1 + X \to X$.

Given an endofunctor $F : \mathbf{C} \to \mathbf{C}$ on a category $\mathbf{C}$ one can form the category of F-*algebras* having as objects pairs $\langle X, h\rangle$ with X an object and $h : FX \to X$ an arrow of $\mathbf{C}$. An arrow $f : \langle X, h\rangle \to \langle X', h'\rangle$ between F-algebras is an arrow $f : X \to X'$ between their 'carriers' such that

$$\begin{array}{ccc} FX & \xrightarrow{Ff} & FX' \\ {\scriptstyle h}\downarrow & & \downarrow{\scriptstyle h'} \\ X & \xrightarrow[f]{} & X' \end{array}$$

commutes, that is, $f \circ h = h' \circ Ff$. Therefore, the natural numbers can also be understood as the initial algebra of the endofunctor $FX = 1 + X$ on **Set**. Similarly, the axiom of foundation can be understood as postulating the initiality of an algebra as follows.

Form the *class* (ie large set) V of all sets, namely the *universe of sets*. This class is a (strict) fixed point $V = \mathcal{P}_S V$ of the operator $\mathcal{P}_S$ mapping a class (ie a possibly large set) to the class of all its (small) subsets. This operator can be extended to an endofunctor $\mathcal{P}_S : \mathbf{SET} \to \mathbf{SET}$ on the (superlarge!) category **SET** of classes and class-functions. Thus the identity function given by the equality $\mathcal{P}_S V = V$ can be seen as an algebra structure of this endofunctor.

Now, it is shown in this thesis that the axiom of foundation is equivalent to postulating that 'the universe $\mathcal{P}_S V = V$ is an initial $\mathcal{P}_S$-algebra'. This gives the formal link between initiality and (generalized) induction (on well-founded relations). Most importantly, in this form the foundation axiom is easily dualized:

The dual of the notion of initiality is the notion of *finality*: an object X is *final* (or *terminal*) in a category when *from* every object of the category there is a unique arrow *to* X. And the dual of the notion of an algebra of an endofunctor F on a category C is the notion of an F-*coalgebra*, that is, a pair $\langle X, k\rangle$ with X an object and $k : X \to FX$ an arrow of $\mathbf{C}$; the arrows $f : \langle X, k\rangle \to \langle X', k'\rangle$ between coalgebras are those arrows $f : X \to X'$ between their carriers such that

$$\begin{array}{ccc} X & \xrightarrow{f} & X' \\ {\scriptstyle k}\downarrow & & \downarrow{\scriptstyle k'} \\ FX & \xrightarrow[Ff]{} & FX' \end{array}$$

commutes, ie $Ff \circ k = k' \circ f$. Therefore, the dual of foundation amounts to postulating that 'the universe $V = \mathcal{P}_S V$ is a final $\mathcal{P}_S$-coalgebra', which, as shown in this thesis, is equivalent to Peter Aczel's '*anti-foundation axiom*' yielding *non-well-founded sets*.

Coinduction with non-well-founded sets.

> "The original stimulus for my own interest in the notion of a non-well-founded set came from a reading of the work of Robin Milner in connection with his development of a mathematical theory of concurrent processes. This topic in theoretical computer science is one of a number of such topics that are generating exciting new ideas and intuitions that are in need of suitable mathematical expression."
>
> Peter Aczel, *Non-Well-Founded Sets*

Aczel's theory of non-well-founded sets was driven by the quest for a set-theoretic foundation for the (abstract) semantics of Milner's *Calculus of Communicating Systems* (CCS). In CCS, the behaviour of a program t is given by the set

$$\{t \xrightarrow{a_i} t_i\}$$

of *transitions* $t \xrightarrow{a_i} t_i$ which the program can perform, producing an observable *action* a_i and becoming t_i. The *non-deterministic* nature of the calculus is expressed by the fact that a program t can *choose* among a set of transitions.

The meaning $[\![t]\!]$ of a program t should abstract from the name of the programs involved in the transitions and focus to the actions which can be performed, together with the choices which can be made. It should then be the following 'coinductively' defined set.

$$[\![t]\!]^{@} = \{< a, [\![t']\!]^{@} > \mid t \xrightarrow{a} t'\}$$

(The superscript '@' is used in this thesis to denote coinductive definitions in general; its dual is the superscript '#' used for inductive definitions.) Now, in general, the transition relation is not well-founded, since, for instance, cyclic programs $t \xrightarrow{a} t$ are allowed. Therefore, the above meaning $[\![t]\!]^{@}$ can be a non-well-founded set.

Traditionally, this 'problem' has been overcome by imposing either an order or a metric on the transition relation and then defining $[\![t]\!]^{@}$ as a suitable *limit*. (See, eg, [Win93] for the order-theoretic and [BV96] for the metric-theoretic approach.) Aczel, instead, chose to look for new foundations allowing for non-well-founded sets and then replaced the foundation axiom by the anti-foundation axiom [Acz88]. But one does not need to resort to non-standard foundations: as already clear in [Acz88], coinductive definitions can be founded on final coalgebras and these exist also in the standard category of ordinary sets (and in many other categories).

What the anti-foundation axiom gives is the non-standard fact that the greatest (strict) fixed point

$$gfp(F) = F(gfp(F))$$

of an endofunctor F on **SET** is a final F-coalgebra, provided F satisfies some mild conditions. This theorem [Acz88, "Special Final Coalgebra Theorem"] is the 'dual' of the standard fact (holding also without anti-foundation) that the least fixed points of most endofunctors on **SET** are initial algebras.

In particular, the special final coalgebra theorem holds for the endofunctor mapping a class X to the class $\mathcal{P}_S(A \times X)$ having as elements (small) sets of pairs $<a, x>$, with $a \in A$ and $x \in X$. Now, the behaviour of CCS programs can be seen as a coalgebra of this endofunctor by taking for A the set *Act* of actions performable by the programs, for X the set *Prog* of programs, and for coalgebra structure the function $[\![\text{-}]\!] : Prog \to \mathcal{P}_S(Act \times Prog)$ defined for every program $t \in Prog$ as follows.

$$[\![t]\!] = \{< a, t' > \mid t \xrightarrow{a} t'\}$$

Then the function $[\![\text{-}]\!]^{@}$ mapping a program to its abstract meaning can be defined as the *coinductive extension* of this coalgebra structure, that is, as the unique coalgebra arrow from the coalgebra of programs to the greatest fixed point of the '*behaviour endofunctor*'

$$BX = \mathcal{P}_S(Act \times X)$$

which, by the special final coalgebra theorem, is a final coalgebra:

$$\begin{tikzcd} Prog \arrow[r, dashed, "{[\![\text{-}]\!]^{@}}"] \arrow[d, "{[\![\text{-}]\!]}"'] & gfp(B) \arrow[d, equal] \\ B(Prog) \arrow[r, "{B([\![\text{-}]\!]^{@})}"'] & B(gfp(B)) \end{tikzcd}$$

That is, for every program $t \in P$, $[\![t]\!]^{@} = \{< a, [\![t']\!]^{@} > \mid t \xrightarrow{a} t'\}$.

The special final coalgebra theorem is stated in terms of the "Solution Lemma" [Acz88]. The final coalgebra presentation of anti-foundation introduced in this thesis makes the solution lemma (and its equivalence with anti-foundation) trivial. Correspondingly, the 'uniformity on maps' condition – which an endofunctor has to satisfy in order for the special final coalgebra theorem to hold – can be formulated in a more transparent way than in [Acz88].

Structural Operational Semantics. The operational semantics of CCS, that is, the definition of the transition relation between CCS programs, is given using Gordon Plotkin's *structural* approach to *operational semantics* [Plo81b]. In structural operational semantics both the programs and their behaviour are defined by induction on the basic program constructs – the *structure* of the programs. In particular, the behaviour of the programs is defined as the least transition relation closed under some conditional operational rules.

Since its inception, the structural approach has rapidly become the predominant approach to operational semantics. The two main reasons are that (i) it is *universal*, in the sense that all existing languages can be described this way, and (ii) it comes with a *structural induction principle* for reasoning about programs.

In this thesis, a mathematical theory of 'well-behaved' operational semantics is introduced which arises from a suitable interplay between the inductive (ie algebraic) aspects of the structural approach and the coinductive (ie coalgebraic) aspects present in Aczel's work on CCS.

Let us focus on the inductive aspects first. In the structural approach, programs are inductively defined in terms of some basic constructs $\sigma \in \Sigma$ from a *signature* Σ. Every signature can be seen as an endofunctor mapping a set X to the coproduct

$$\Sigma X = \coprod_{\sigma} X^{\mathrm{arity}(\sigma)}$$

indexed by the constructs σ of the language. The programs form then the (unique up to isomorphism) initial algebra of this endofunctor. In particular, by taking as constructs a constant (arity $= 0$) and a unary operator (arity $= 1$) one obtains the equivalence between the natural numbers (as inductively defined from zero and successor) and the initial algebra of the endofunctor $X \mapsto 1 + X$.

The initial Σ-algebra gives the set of *closed* programs, that is, programs without variables. In order to adjoin variables from a set *Var* it is sufficient to take the initial algebra of the endofunctor

$$X \mapsto \mathit{Var} + \Sigma X$$

(In particular, if *Var* is empty then one gets back the original Σ.) This initial algebra is also called the *free Σ-algebra over Var*.

It is worthwhile to make one more step of abstraction and introduce the notion of a *monad*.

Monads. Given a signature Σ, let $X \mapsto TX$ be the operation mapping a set X, regarded as a set *Var* of variables, to the free Σ-algebra over X (ie the initial $(X + \Sigma)$-algebra). By universality, this operation extends to an endofunctor $T : \mathbf{Set} \to \mathbf{Set}$ on $\mathbf{Set}$. This endofunctor T comes equipped with two 'operations': the 'insertion-of-the-variables' $\eta_X : X \to TX$ and a 'multiplication' $\mu_X : T^2X \to TX$ for plugging programs into contexts. These operations are 'natural' in X and the triple $T{=}{<}T, \eta, \mu{>}$ is a *monad* on $\mathbf{Set}$.

In general, a monad $T =< T, \eta, \mu >$ on a category $\mathbf{C}$ can be understood as a monoid in a category of endofunctors on $\mathbf{C}$, the 'operation' μ being the associative multiplication of the monoid and η its unit.

The notion of a monad is one of the most general mathematical notions. For instance, *every* algebraic theory, that is, every set of operations satisfying equational laws, can be seen as a monad; thus the monoid laws of the monad do subsume all possible algebraic laws! And algebraic theories are only a minor source of monads. In fact, every 'canonical' construction between two categories gives rise to a monad: the free Σ-algebra construction from $\mathbf{Set}$ to the category of Σ-algebras is one such canonical construction.

Next, there is a notion of a T-algebra which subsumes the notion of an algebra and, in particular, of a Σ-algebra. (Σ-algebras can be understood as algebras in which the operators (of the signature) are not subject to any law.) In particular, the monad T freely generated by a signature Σ is such that its category of algebras is isomorphic to the category of Σ-algebras. Therefore, the *syntax* of a programming language can be identified with a monad, the *syntactical monad* T freely generated by the program constructs Σ.

Now that the *syntax* is understood as a monad T and the *behaviour* as an endofunctor B whose coalgebras can be regarded as operational models (eg $BX = \mathcal{P}(Act \times X)$) the new notion of a 'functorial operational semantics' can be introduced.

Functorial Operational Semantics.

> A *functorial operational semantics* for a syntax T and a behaviour B is a monad Φ which 'lifts' the syntactical monad T to the coalgebras of the behaviour endofunctor B.

The *operational monad* Φ inherits the operations η and μ of the syntactical monad T; as a functor it maps a coalgebra structure $k : X \to BX$ to a structure $\Phi k : TX \to BTX$ which can be seen as the *operational model* on the set of programs TX given by the semantics Φ starting from the 'assumptions' $k : X \to BX$.

There are many possible liftings Φ of the same syntax T, each giving a different operational interpretation of the programs corresponding to T.

The novelty of this approach to operational semantics is that it captures in terms of abstract notions of *syntax* and *behaviour* the essence of 'well-behaved' operational semantics.

A condition which a well-behaved operational semantics should satisfy is *compositionality*: To every behaviour B there corresponds a notion of *observational equivalence* called B-bisimulation [AM89] (which for the behaviour $BX = \mathcal{P}(Act \times X)$ corresponds to Park and Milner's (strong) bisimulation – the finest notion of observational equivalence for transition relations); if this observational equivalence is a congruence wrt the constructs of the syntax, then the operational semantics is compositional. This means that programs with the same observable behaviour can be interchanged in any context without affecting the overall observable behaviour. Now, as shown in this thesis, *every* functorial operational semantics enjoys the property of being compositional.

Previous general results on compositional operational semantics stem from the theory of concurrent processes: the operational semantics is then assumed to be structural and the behaviour is fixed to be $BX = \mathcal{P}(Act \times X)$ (ie the notion of observational equivalence is (strong) bisimulation). The compositionality is ensured by imposing some restrictions on the *syntactic* format of the operational rules. Several formats have been proposed [dS85, BIM88, GV92, Gro93] and one of the most general is '*GSOS*' [BIM88], suitable to model most of the imperative or concurrent languages, including Milner's CCS.

Another result in this thesis is that every set $\mathcal{R}$ of GSOS rules defines an 'action' of the syntactical monad T on the composite endofunctor BT; in turn, this action *induces* a functorial operational semantics observationally equivalent to the operational semantics induced by the rules $\mathcal{R}$. Hence the syntactic restrictions making GSOS well-behaved are explained mathematically in terms of abstract notions of syntax and behaviour.

Denotational Semantics. A more general way of understanding the compositionality (and 'well-behaviour') of an operational semantics is in terms of 'denotational models'. Given a syntactical monad T, a *denotational model* for the corresponding language is simply a T-algebra; if the monad T is freely generated by a signature Σ, then this is the same as a Σ-algebra, that is, a set and a 'denotation' on this set of each program construct in Σ.

(More structured denotational models can be obtained by 'interpreting' the syntactical monad T in categories of structured objects like partial orders or metric spaces, rather than simply sets.)

The unique algebra arrow from the initial algebra of programs to the denotational model gives an inductive interpretation mapping programs to elements of the model. (This is the well-known *initial algebra semantics* approach of the 'ADJ group' – cf, eg, [GTW78].) This interpretation is by definition compositional, but one has to establish its *adequacy*:

> A denotational model is *adequate* wrt an operational semantics if it determines the operational behaviour of the programs up to observational equivalence.

It is at this point that the coalgebraic (ie coinductive) aspects of the functorial approach to operational semantics start playing a rôle: one of the pleasing properties of functorial operational semantics is that they (canonically) coinduce adequate denotational models. In order to understand this property, let us first look at coinduction in the category of ordinary (ie well-founded) sets.

Coinduction with ordinary sets. One of the properties of Aczel's coinductive semantics for CCS is that it maps two programs to the same set if and only if they are observationally equivalent:

$$[\![t_1]\!]^{@} = [\![t_2]\!]^{@} \iff t_1 \sim t_2$$

That is, the coinductive extension of the operational model $[\![\text{-}]\!] : \mathit{Prog} \to B(\mathit{Prog})$ does preserve B-bisimulation and, conversely, it can be 'pulled back' to form the largest B-bisimulation relation.

The above is a property which holds in general for every coinductive extension of coalgebras of endofunctors B preserving categorical (weak) *pullbacks*, where the endofunctor B can be on any category. Therefore:

> One does not need to work with non-well-founded sets: all one needs is that there exists a final coalgebra (hence coinduction) for B. In particular, one can work in the category of *ordinary* sets.

If anti-foundation is not assumed, then one cannot apply the special final coalgebra theorem in order to obtain final coalgebras from greatest (strict) fixed points. (While initial algebras can still be obtained as least fixed points.) There are several categorical methods to obtain final coalgebras though. One is a simple generalization of the standard greatest fixed point construction (à la Tarski) but it does not hold for endofunctors like the power-set functor.

There is also a problem of size: the structure of a final coalgebra is an isomorphism, that is, if $\widehat{B}$ is the carrier of a final B-coalgebra then its structure is an isomorphism

$$\varphi : \widehat{B} \cong B\widehat{B}$$

(This fact, in its dual version for initial algebras, is known as "Lambek's lemma".) Therefore, there is no final coalgebra for the endofunctor $BX = \mathcal{P}(Act \times X)$ or just $\mathcal{P}$, because there is no set isomorphic to the set of its subsets.

Aczel overcomes this problem by moving to the superlarge category of classes and considering the endofunctor $\mathcal{P}_S$ mapping a class to the class of its (*small*) subsets. Another solution, adopted here, consists in taking the *finite* power-set endofunctor mapping a set X to the set $\mathcal{P}_{\mathrm{fi}}(X)$ of its finite subsets.

In general for establishing the existence of a final object in a category one can use categorical theorems like the "Special Adjoint Functor Theorem". As shown in [Bar93] this applies also to the coalgebras of endofunctors like the finite power-set and the corresponding behaviour

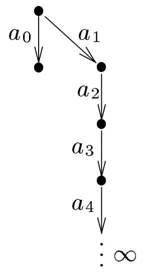
$$BX = \mathcal{P}_{\mathrm{fi}}(Act \times X)$$

In particular, since CCS programs have only a *finite* degree of non-determinism, that is, each program can choose only among a finite set of transitions, the operational model of CCS is a coalgebra of this behaviour; its coinductive extension $[\![\text{-}]\!]^{@} : \mathit{Prog} \to \widehat{B}$ yields a semantics in the ordinary category of sets which is 'almost' the same as Aczel's one. The difference is in the fact that the final coalgebra structure is an isomorphism $\varphi : \widehat{B} \cong B\widehat{B}$ rather than an equality $\widehat{B} = B\widehat{B}$. Correspondingly, one has, for every program t,

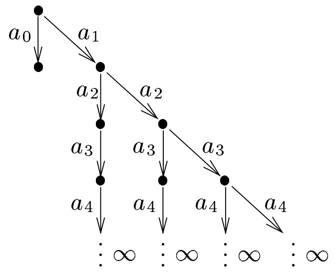
$$[\![t]\!]^{@} = \varphi^{-1}\{<a, [\![t']\!]^{@}> \mid t \xrightarrow{a} t'\}$$

(In the sequel, for simplicity, the isomorphism φ is omitted.) This is the *final coalgebra semantics* corresponding to the operational model $[\![\text{-}]\!] : \mathit{Prog} \to B(\mathit{Prog})$.

Concretely, the final coalgebra for the behaviour $BX = \mathcal{P}_{\mathrm{fi}}(Act \times X)$ is the set of rooted, finitely branching trees, with branches labelled by the actions $a \in Act$, *quotiented* by the (largest) bisimulation relation. These (equivalence classes of) trees can be seen as the *abstract global behaviours* corresponding to $BX = \mathcal{P}_{\text{fi}}(Act \times X)$: the root of a tree τ is the starting point of an abstract computation c with behaviour B; the branching structure records the alternatives of the computation c and the labels of the branches are its observable actions; the quotient modulo bisimulation is needed in order to identify trees like

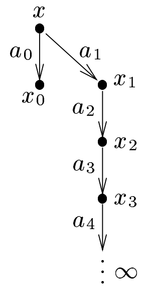


and

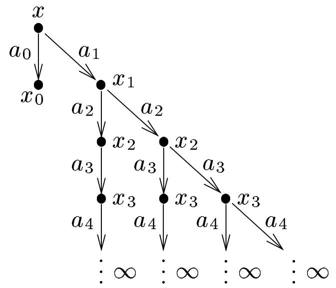


Notice that branches can be of infinite depth.

The fact that the nodes have no name reflects the abstractness of these global behaviours. This can be seen as a special case of the global behaviours observable with a set of 'states' X, which is obtained by labelling the nodes of the trees by elements x of X and, correspondingly, taking the quotient wrt a subtler form of bisimulation which takes into account the name of these states. For instance:



$\sim$



By putting $X = 1$, that is, by using the same label for all nodes, one gets back the abstract global behaviours.

Observational Comonads. The above operation $X \mapsto DX$ mapping a set X to the set of its global behaviours can be understood as a *cofree construction*, dual to the free construction of a monad from a signature. In general, given an endofunctor B on a category (with products) $\mathbf{C}$ a *cofree* B-coalgebra over an object X, if it exists, is the final coalgebra of the product endofunctor $X \times B$ mapping an object X' to the product $X \times BX'$. This generates a *comonad* $D{=}{<}D, \varepsilon, \delta{>}$, that is, an endofunctor $D : \mathbf{C} \to \mathbf{C}$ together with two 'operations' $\varepsilon_X : DX \to X$ and $\delta_X : DX \to D^2X$ 'natural' in X which make D a comonoid in a category of endofunctors on $\mathbf{C}$.

Comonads cofreely generated by behaviour endofunctors are called here *observational comonads*. Correspondingly, of the three conditions (implicitly) arisen so far which make of an endofunctor B a behaviour endofunctor, namely

1. the coalgebras of B have a computational interpretation as operational models,
2. B has a final coalgebra (hence coinduction),
3. B preserves weak pullbacks (hence coinduction can be 'pulled back' to B-bisimulation),

the second has to be generalized by requiring the existence of a final coalgebra of the product endofunctor $X \times B$ for every object X. Correspondingly, the category $\mathbf{C}$ should have finite products (including a final object 1). Since in every category $1 \times X \cong X$ holds, one has that the final coalgebra is isomorphic to the cofree coalgebra over 1.

As mentioned above, in the specific case of the behaviour $BX = \mathcal{P}_{\mathrm{fi}}(Act \times X)$, the value of the observational comonad D at a set X is a set of (equivalence classes of) rooted trees with nodes labelled by 'states' $x \in X$. The operations of the observational comonad $D = < D, \varepsilon, \delta >$ permit to visit these trees: the 'counit' ε is the operation which extracts the label of the root of a tree and the 'comultiplication' δ gives the remaining part of the tree.

One can form a category of *D-coalgebras* and, like for Σ-algebras and the algebras of the corresponding freely generated monad T, one can prove that if D is cofreely generated by an endofunctor B then this category is isomorphic to the category of B-coalgebras. Therefore, a functorial operational semantics can be seen as a lifting Φ of the syntactical monad T to the coalgebras of the observational comonad D. In this form, the notion of a functorial operational semantics can be readily dualized as follows.

Functorial Denotational Semantics.

> A *functorial denotational semantics* for a syntax T and a (global, observable) behaviour D is a comonad Ψ which 'lifts' the observational comonad D to the algebras of the syntactical monad T.

The *denotational comonad* Ψ inherits the operations ε and δ of the observational comonad D. In terms of Σ-algebras, the endofunctor Ψ maps a structure $h : \Sigma X \to X$ to a structure $\Psi h : \Sigma DX \to DX$ which can be seen as the *denotational model* on the set of global behaviours DX given by the semantics Ψ starting from the 'assumptions' $k : \Sigma X \to X$.

Operational is Denotational. Now, the abstract property showing that functorial operational semantics are well-behaved is that there is a one-to-one correspondence between operational monads Φ and denotational comonads Ψ (over the same syntax and behaviour). Symbolically:

$$
\begin{array}{ccc}
\Phi & \begin{tikzcd} \mathbf{C}_D \arrow[r, "\Phi"] \arrow[d, "U_D"'] & \mathbf{C}_D \arrow[d, "U_D"] \\ \mathbf{C} \arrow[r, "T"'] & \mathbf{C} \end{tikzcd} & \Psi^{\#} \\
\big\downarrow & \overline{\overline{\hspace{6em}}} & \big\uparrow \\
\Phi^{@} & \begin{tikzcd} \mathbf{C}^T \arrow[r, "\Psi"] \arrow[d, "U^T"'] & \mathbf{C}^T \arrow[d, "U^T"] \\ \mathbf{C} \arrow[r, "D"'] & \mathbf{C} \end{tikzcd} & \Psi
\end{array}
$$

The category $\mathbf{C}_D$ is the category of coalgebras of a comonad D over $\mathbf{C}$ and the 'forgetful' functor $U_D : \mathbf{C}_D \to \mathbf{C}$ forgets the coalgebra structure mapping a coalgebra to its carrier. The dual holds for $\mathbf{C}^T$ and $U^T : \mathbf{C}^T \to \mathbf{C}$.

The mapping $\Phi \mapsto \Phi^{@}$ is defined by coinduction. In particular, the value of the comonad $\Phi^{@}$ at the (trivial) final T-algebra is the coinductive extension of the coalgebra structure obtained by applying the given operational monad Φ to the final D-coalgebra. The resulting T-algebra is the 'canonical' denotational model coinduced by the operational semantics Φ.

The essence of the above coinductive construction was already presented in [RT94], but there the assumption was needed that observational equivalence be a congruence (hence compositionality had already to be known) and, in order to ensure this fact, the operational semantics was assumed to be à la GSOS. Instead, here the functoriality of Φ ensures that the construction can always take place. Moreover, the fact that the mapping $\Phi \mapsto \Phi^{@}$ is a bijection immediately gives that $\Phi^{@}$ is *adequate* wrt Φ, that is, one can recover the operational semantics from the denotational one. Compositionality becomes here a corollary.

The bijection 'operational $\longleftrightarrow$ denotational' can be used also in the reverse direction. The mapping $\Psi \mapsto \Psi^{\#}$ gives an *inductive* construction of *operational* models from denotational ones. This is a new principle which had been forecasted in [RT94]. It is used here to show that *basic process algebra* – the 'minimal' language corresponding to the behaviour $BX = \mathcal{P}_{\mathrm{fi}}(Act \times X)$ – is functorial. This is an important result because the proof given here that GSOS is functorial is based on the (mild) assumption that every set of GSOS rules embeds basic process algebra. Correspondingly, the syntactical monad is assumed to correspond to an algebra containing an associative, commutative, and absorptive binary operator of non-deterministic choice. (This is one example of the advantage of working with the T-algebras rather than with algebras of a signature.)

Φ-algebras are $\Phi^@$-coalgebras. Another way of understanding the above adequacy result is by considering the category of algebras of the operational monad Φ. It is shown in this thesis that the category of Φ-algebras is the same as the category of coalgebras of its coinduced denotational comonad $\Phi^@$. One can take this category as the category of *models of* Φ: its objects carry both a T-algebra and a D-coalgebra structure which are suitably related via Φ. (Thus a Φ-model carries both a denotational and an operational structure.) The arrows of the category are those which preserve both the algebraic and the coalgebraic structure.

The category of Φ-models has both an initial and a final object: the initial Φ-model is the initial algebra of closed programs corresponding to the syntactical monad T, together with the operational model obtained by applying Φ to the (trivial) initial D-coalgebra; dually, the final Φ-model is the final coalgebra of abstract global behaviours corresponding to the observational comonad D, together with the denotational model obtained by applying $\Phi^@$ to the (trivial) final T-algebra.

Now, the (both by initiality and finality) unique arrow from the initial to the final Φ-model is a mapping going from the closed program $T0$ to the abstract global behaviours $D1$ and it necessarily is both an initial algebra semantics and a final coalgebra semantics. This is the categorical formulation of adequacy.

Interestingly, if Φ is the operational monad corresponding to a set of GSOS rules, then the notion of a Φ-model cuts down to the notion of a *GSOS-model* independently introduced by Alex Simpson in [Sim95].

Adjunctions subsume induction and coinduction. It should be stressed that, categorically, induction and coinduction are just two instances of the same notion, namely the one of an *adjunction*:

If the forgetful functor mapping the algebras of an arbitrary monad T to their carriers has a *left adjoint*, then the T-algebras come with an induction principle; the monad T itself is defined by this adjunction. Dually, if the forgetful functor mapping the coalgebras of a comonad D to their carriers has a *right adjoint*, then the D-coalgebras come with a coinduction principle.

Every 'canonical' construction between two categories defines an adjunction and every adjunction defines both a monad and a comonad. It is in this sense that canonical constructions give rise to monads (and comonads).

Sets like recursive processes. Finally, one remark on the title of the part of this thesis dedicated to non-well-founded sets.

It is shown in this thesis that recursive programs can be seen as coalgebras having as carrier the set of variables involved in the recursion. As a consequence, no (explicit) binding operator (like the operator "fix" in GSOS) is needed and the solution of a recursive program is (a *recursive process*) defined by coinduction. This subsumes standard fixed point methods like least fixed points in categories of complete partial orders [Plo76] or unique fixed points in categories of complete metric spaces [Niv79, BZ82].

Now, the same method is used here to treat (and trivialize!) the "Solution Lemma" [Acz88] for defining non-well-founded sets as solution of recursive equations involving exclusively (variables and) well-founded sets.

Historical Notes. The study of adequate denotational models for structural operational semantics has been, from [BZ82] on, the central topic of Jaco de Bakker's Amsterdam school of semantics based on the use of metric spaces. (See [BR92, BV96] for overviews.) The present functorial approach harvests the fruits of that work.

The main mathematical tool available in (complete) metric spaces is "Banach's theorem" ensuring the existence of unique fixed points of 'contracting' functions. Like coalgebraic finality, Banach's theorem, especially in its higher-order form, can be used both for dealing with coinductive *definitions* and for *proving* adequacy results. (Cf [KR90].)

In particular, Banach's theorem is used in [Rut90] for coinductively deriving denotational models from structural operational semantics. The assumption is that the operational rules are 'well-behaved' in the sense that they are in (a sub-format of) the GSOS format [BIM88] and this implies that the coinduced models are adequate. (A precursor of this method is presented in [Bad87], which, in turn, has been inspired by [DG87].)

A considerable improvement of the above method is achieved in [Rut92] by treating the semantic domain of abstract global behaviours (ie the set of *processes*) as a transition system and subsequently applying the operational rules to it, that is, by treating "processes as terms". Coinduction is dealt there by means of non-well-founded sets and of the corresponding solution lemma; the operational rules are in the "tyft/tyxt" format of [GV92], a more general format than the positive GSOS used in [Rut90].

An explicit use of the finality of the greatest fixed point of the endofunctor $BX = \mathcal{P}_S(Act \times X)$ (under the anti-foundation axiom) is made in [Acz88] for coinductively defining a *denotational* model for CCS. That example has led the author of this thesis to try and understand the mathematics behind the "processes as terms" method in terms of an interplay between algebraic and coalgebraic aspects. The article [RT94] contains preliminary results in this sense, but the actual derivation of models, although formulated coalgebraically, still relies there on the use of 'well-behaved' structural operational rules à la GSOS and on regarding the final coalgebra (ie the abstract global behaviours) as a transition system.

The abstraction step from well-behaved transition systems to operational monads has come only after Gordon Plotkin's suggestion of working with algebras *over* coalgebras rather that with algebras *and* coalgebras: that has proved to be the extra 'dimension' needed for formulating the present functorial approach to operational semantics.

Algebraic Compactness. Another way of looking at initial algebras and final coalgebras of endofunctors F is as *data types*: the initial F-algebra is the *inductive* data type corresponding to the 'type constructor' F, while the final F-coalgebra is the *coinductive* one. For instance, the type constructor $FX = 1 + X$ yields, in **Set**, the natural numbers N as inductive data type and the 'extended natural numbers' $N \cup \{\infty\}$ as coinductive one.

Studies on coinductive types in **Set** date back at least to [AM80]. A more recent view, put forward by Peter Freyd in [Fre91], is that data types should be defined in *algebraically compact categories*, that is, in categories where endofunctors have both initial algebras and final coalgebras which, moreover, do *coincide* in the sense that they are 'canonically isomorphic'. (See also [Fre90, Fre92].)

The archetypal example of an algebraically compact category is the category **pCpo** of complete partial orders and partial 'Scott-continuous' functions: regarded as an 'order-enriched' category, it has as endofunctors the 'locally continuous' ones, which, as shown in [SP82], make it algebraically compact indeed. (See [Bar92] for more examples.)

Instead, algebraic compactness fails in the category of sets, no matter whether ordinary or non-well-founded sets are considered. The absence of algebraic compactness in **Set** motivated Peter Freyd's remark on the need for non-standard mathematical foundations in computer science quoted at the beginning of this introduction.

Algebraic compactness is one of the axioms of Fiore and Plotkin's *axiomatic domain theory* [FP92, FP94, Fio96] which aims at isolating the abstract properties which a category should satisfy for hosting interpretations of programming languages. In particular, the semantic domain of a language – in the present setting the final coalgebra of the behaviour – should 'live' in such a category, typically **pCpo**. In contrast, the operational model of a language should carry only the structure imposed by syntax and behaviour and thus live in a simpler category, typically **Set**. This raises the problem of how to extend/lift a functorial operational semantics from an unstructured category like **Set** to a category of domains like **pCpo**.

Towards a mathematical operational semantics

> "The motivation for trying to formulate a mathematical theory of computation is to give *mathematical semantics* for high-level computer languages. The word 'mathematical' is to be contrasted in this context with some such term as 'operational'."
>
> Dana Scott, *Outline of a Mathematical Theory of Computation*

The present functorial approach shows that 'operational' and 'mathematical' are no longer necessarily contrasting attributes for a semantics. This is achieved by defining operational semantics in terms of abstract, mathematical notions of syntax and behaviour. Yet, considerable work remains to be done before this conceptual achievement will be of any 'practical' relevance.

Firstly, the examples of behaviour considered here are all minor variations of the endofunctor $BX = \mathcal{P}_{\mathrm{fi}}(Act \times X)$, with (strong) bisimulation as the corresponding observational equivalence. Among the other behaviours which can be described functorially and will be treated in future work there are those for side effects, for probabilistic computation, for *trace equivalence*, and for applicative languages like the *untyped lambda calculus*.

The first two behaviours are similar to the one for bisimulation, while a treatment of trace equivalence and of the lambda calculus require, for different reasons, the ability of extending or lifting an operational monad from **Set** to a more structured category, namely **pCpo** for the lambda calculus [Plo85] and the category of semi-lattices and join-preserving functions for trace equivalence [HP79]. Preliminary results on a coalgebraic treatment of trace equivalence and of the lambda calculus are presented in [TJ93, RT94].

Secondly, a more refined notion of syntactical monads is needed in order to deal with typed terms and with higher-order terms as introduced, eg, by *variable binding* in the *lambda calculus* and in many imperative and concurrent languages. For typed terms one can easily adapt the above approach using multi-sorted algebras. (Categorically, it means to deal with a power of **Set**.) For higher-order terms the plan is to consider signatures on variable sets (*presheaves*) rather than simple sets. Correspondingly, one has for a function(al) not an arity but a list of numbers. The length of the list is the number of arguments; the i-th number is the number of variables the function(al) binds at its i-th argument. (This notion of signature is considered, for semantics, in [Acz80], and, for syntax, in [Plo90]. Associated ideas are the work on higher-order rewriting [Klo80], and the work on higher-order algebra [Mei92].)

Thirdly, the above adequacy result should be strengthened by dealing also with *non-termination*: when, like in the untyped lambda calculus, programs might not terminate, adequacy imposes further requirements. For example, by using partial functions for the denotational semantics, the interpretation of a term should be undefined if and only if it does not terminate. This property is hard to verify and much work has been devoted to introduce methods for simplifying this kind of proofs. (See, eg, [Pit94b].) Therefore, a 'meta' adequacy result would be of a great relevance. (A related point still to be investigated is whether there exist some extra conditions which make a functorial operational semantics *fully-abstract*, but this is much harder a result to obtain.)

Finally, the present functorial approach seems closely related to Eugenio Moggi's monadic approach to operational semantics [Mog91]. His examples of *computational monads* do all qualify as *behaviours* and it would be interesting to incorporate their extra monadic structure in this functorial framework. As a result, a general notion of operational semantics for computational monads and a corresponding adequacy theorem could be obtained.

Synopsis

This thesis is divided in five parts: the first four parts are devoted to the functorial approach to operational semantics, while Part V (*Sets like Recursive Processes*) is a new presentation of Peter Aczel's theory of non-well-founded sets.

In Part I, after some preliminaries, the definition of functorial operational semantics is introduced. As an example, a simple deterministic language is treated with $BX = 1 + Act \times X$ as behaviour. Final coalgebras and recursive programs are also treated.

In Part II, the general properties of functorial operational semantics are illustrated. In Section 6 it is shown that every operational monad coinduces an adequate denotational model. This construction is explained in Section 7 in terms of the notion of functorial denotational semantics, dual to the operational one: every operational monad Φ coinduces a denotational comonad $\Phi^{@}$. This is the *basic property* of the functorial approach to operational semantics.

Section 8 shows that the mapping $\Phi \mapsto \Phi^{@}$ is a bijection between operational monads and denotational comonads, which implies that $\Phi^{@}$ is always adequate wrt Φ. This adequacy result is rephrased in Section 9, where it is shown that the algebras of an operational monad Φ are the same as the coalgebras of its coinduced comonad $\Phi^{@}$. The category of Φ-algebras (alias $\Phi^{@}$-coalgebras) is then taken as the category of Φ-*models*, and the unique arrow from the initial to the final Φ-model is both the initial algebra and the final coalgebra semantics corresponding to Φ.

Part III is dedicated to the non-deterministic behaviour $BX = \check{\mathcal{P}}(1 + Act \times X)$. Correspondingly, the simple deterministic language used as example in the two previous parts can be enriched with a non-deterministic choice construct à la CCS. In Section 10, following [HP79, Plo81a], the (non-empty) finite power-set $\check{\mathcal{P}}$ is introduced as the semi-lattice monad. Next, a functorial *denotational* semantics is 'naturally' associated to the behaviour $BX = \check{\mathcal{P}}(1 + Act \times X)$ and its induced operational semantics is shown to be *basic process algebra* [BW90]. This is used in Section 11 to prove that GSOS is functorial, under the mild assumption that GSOS embeds basic process algebra.

In Section 12, the observational equivalences corresponding to (arbitrary) behaviours B are treated using the notion of a relation lifting to a 'B-bisimulation' introduced in [AM89], which, for $BX = \check{\mathcal{P}}(1 + Act \times X)$, cuts down to Park and Milner's notion of a bisimulation. If the endofunctor B preserves (weak) pullbacks, then every coinductive definition of type B can be 'pulled back' to a relation lifting to a B-bisimulation, which fact is useful to reason about coinductively defined entities. Here it is shown that, as a corollary of adequacy, for every functorial operational semantics, bisimulation (wrt to the behaviour B) is a congruence (wrt the syntax T).

Section 13 treats the construction of cofree coalgebras for the finite power-set functor $\mathcal{P}_{\mathrm{fi}}$ and for the behaviour $BX = \check{\mathcal{P}}(1 + Act \times X)$. It is based on material in [AM89] and [Bar93].

Part IV consists of a technical summary (with proofs) of the first three parts phrased in terms of adjunctions rather than in terms of induction and coinduction.

Basic Universal Constructions

Category theory is the mathematical study of *universal* entities: an entity x is universal among a family $\mathcal{F}$ of entities if all entities of $\mathcal{F}$ can be 'reduced' to x. Formally, this can be expressed in a very general form by considering the family of arrows determined by a functor $F : \mathbf{C} \to \mathbf{D}$ and an object Y of the codomain category $\mathbf{D}$ of F. The family of entities is the set

$$\mathcal{F} = \{f : FX \to Y \in \mathbf{D} \mid X \in \mathbf{C}\}$$

of *arrows from* F *to* Y. (Alternatively, the dual case of arrows *from* Y *to* F can also be considered.)

The universal among the arrows of $\mathcal{F}$ (if it exists!) is an arrow $\varepsilon_Y : FGY \to Y$ such that, for every $f : FX \to Y$, there *exists* a *unique* arrow $f^\flat : X \to GY$ such that f factorizes through ε_Y as follows:

$$f = \varepsilon_Y \circ F(f^\flat)$$

Diagrammatically:

$$\begin{tikzcd}
\mathbf{C} \arrow[r, "F"] & \mathbf{D} & \\
X \arrow[d, dashed, "f^\flat"'] & FX \arrow[d, "F(f^\flat)"'] \arrow[dr, "f"] & \\
GY & FGY \arrow[r, "\varepsilon_Y"'] & Y
\end{tikzcd}$$

The object GY is unique up to isomorphism and so is the arrow ε_Y (in a suitable sense).

Particularly interesting is the case when a universal arrow from F to Y exists for every object Y of $\mathbf{D}$: then, by universality, the operation $Y \mapsto GY$ extends to a functor $G : \mathbf{D} \to \mathbf{C}$ by putting, for every $k : Y \to Y'$ in $\mathbf{D}$,

$$\begin{tikzcd}
GY \arrow[d, dashed, "Gk \,=\,"', "(k \circ \varepsilon_X)^\flat"] & FGY \arrow[r, "\varepsilon_Y"] \arrow[d, "FGk"'] & Y \arrow[d, "k"] \\
GY' & FGY' \arrow[r, "\varepsilon_{Y'}"'] & Y'
\end{tikzcd}$$

Moreover, one can check that, in this case, the arrow

$$\eta_X = (\mathsf{id}_{FX})^\flat : X \to GFX$$

obtained by 'reducing' the identity on FX to ε_{FX}, is a universal arrow from X to G, for every object X of $\mathbf{C}$:

$$\begin{tikzcd} \mathbf{C} \arrow[r, shift left, "F"] & \mathbf{D} \arrow[l, shift left, "G"] \end{tikzcd}$$

$$\begin{tikzcd} X \arrow[r, "\eta_X"] \arrow[dr, "g"'] & GFX \arrow[d, "G(g^\sharp)"] & & FX \arrow[d, dashed, "g^\sharp"] \\ & GY & & Y \end{tikzcd}$$

Dually, a universal arrow from X to a functor $G : \mathbf{D} \to \mathbf{C}$ for every object X of $\mathbf{C}$, defines a functor $F : \mathbf{C} \to \mathbf{D}$ and a universal arrow from F to Y, for every Y in $\mathbf{C}$. There is thus a hidden symmetry behind the notion of a universal arrow, a symmetry which is captured by the notion of an 'adjunction'.

Formally, an *adjunction from a category* $\mathbf{C}$ *to a category* $\mathbf{D}$ is given by a pair of functors $F : \mathbf{C} \to \mathbf{D}$ and $G : \mathbf{D} \to \mathbf{C}$ in opposite direction and by a 'natural' bijection between the arrows of type $FX \to Y$ and those of type $X \to GY$, for every X in $\mathbf{C}$ and Y in $\mathbf{D}$:

$$\begin{array}{ccc} f & FX \longrightarrow Y & g^\sharp \\ \downarrow & \overline{\overline{\qquad\qquad\qquad}} & \uparrow \\ f^\flat & X \longrightarrow GY & g \end{array}$$

The *naturality* of the mapping $f \mapsto f^\flat$ amounts to the fact that it is 'well-behaved' wrt both pre- and post-composition; that is, for all arrows $h : X' \to X$ in $\mathbf{C}$ and $k : Y \to Y'$ in $\mathbf{D}$, the following two equations hold.

$$(f \circ Fh)^\flat = f^\flat \circ h \qquad (k \circ f)^\flat = Gk \circ f^\flat$$

By duality, this is equivalent to the following.

$$(g \circ h)^\sharp = g^\sharp \circ Fh \qquad (Gk \circ g)^\sharp = k \circ g^\sharp$$

One usually writes the above adjunction as

$$F \dashv G$$

and says that G is a *right adjoint* for F; dually, F is a *left adjoint* for G. Correspondingly, $f^\flat$ is the *right adjunct of* f and $g^\sharp$ is the *left adjunct of* g.

Now, if there exists a universal arrow $\eta_X : X \to GFX$ from every object X of a category $\mathbf{C}$ to a functor $G : \mathbf{D} \to \mathbf{C}$, then G has a left adjoint, the functor F which, by universality, extends the operation $X \mapsto FX$. (And the dual holds for universal arrows from F to the objects of $\mathbf{D}$.) Conversely, every adjunction determines two families of universal arrows

$$\{\eta_X = (\mathsf{id}_{FX})^\flat : X \to GFX\}_{X \in \mathbf{C}} \qquad \{\varepsilon_Y = (\mathsf{id}_{GY})^\sharp : FGY \to Y\}_{Y \in \mathbf{D}}$$

(See, eg, [Mac71, §IV.1, Theorems 1 and 2].)

The description of an adjunction in terms of universal arrows is *procedurally* very important for the actual 'construction' of adjunctions. Usually, one has a simple functor at hand, like an inclusion functor or a a functor forgetting some structure, and one investigates the problem of the existence of a right or left adjoint to it: if this problem can be solved then the result can be a complex construction. For instance, the left adjoint of the forgetful functor from a category of algebras to sets maps a set to the free algebra over it. (Adjoints, like all universals, are unique up to isomorphism, thus one can speak of *the* left adjoint of a functor.) The advantage is that a complex construction is reduced to the notion of an adjoint to a simple construction and, moreover, in this form, the same result can be understood in different categories. For instance, one can consider algebras over complete partial orders rather than over sets and the left adjoint to the corresponding forgetful functor gives the free algebras over cpos rather than over sets. Similarly, various topological completions like the one of metric spaces can all be understood as left adjoints of inclusion functors. In general, every 'canonical' construction *arises* from an adjunction.

The family $\{\varepsilon_Y : FGY \to Y\}_{Y \in \mathbf{D}}$ of universal arrows determined by an adjunction has the property that, for all arrows $k : Y \to Y'$ in $\mathbf{D}$, the following diagram commutes.

$$\begin{tikzcd} FGY \arrow[r, "\varepsilon_Y"] \arrow[d, "FGk"'] & Y \arrow[d, "k"] \\ FGY' \arrow[r, "\varepsilon_{Y'}"'] & Y' \end{tikzcd}$$

(And similarly for the family $\{\eta_X : X \to GFX\}_{X \in \mathbf{C}}$.) This gives a 'natural transformation' from the composite functor FG on $\mathbf{D}$ to the identity functor $I_{\mathbf{D}}$.

In general, given two functors $F_1, F_2 : \mathbf{E} \to \mathbf{D}$, a *natural transformation*

$$\vartheta : F_1 \Rightarrow F_2$$

from F_1 to F_2 is a family $\{\vartheta_Y : F_1X \to F_2X \in \mathbf{D} \mid X \in \mathbf{E}\}$ of arrows of $\mathbf{D}$ indexed by the objects of $\mathbf{E}$ such that, for every arrow $f : X \to X'$ in $\mathbf{E}$ the square in the following diagram commutes.

$$
\begin{tikzcd}
Y \arrow[d, "f"] & F_1Y \arrow[r, "\vartheta_Y"] \arrow[d, "F_1 f"'] & F_2Y \arrow[d, "F_2 f"] \\
Y' & F_1Y' \arrow[r, "\vartheta_{Y'}"'] & F_2Y'
\end{tikzcd}
$$

For every two categories $\mathbf{D}$ and $\mathbf{E}$ one can form the *functor category* $\mathbf{D}^{\mathbf{E}}$ having as objects the functors from $\mathbf{E}$ to $\mathbf{D}$ and as arrows the natural transformations between them. Identities and composition are obtained 'pointwise'. Thus: natural transformations are arrows between functors, which, in turn, are arrows between categories.

One usually omits the subscript under the identity functors and writes

$$\eta : I \Rightarrow GF \qquad \text{and} \qquad \varepsilon : FG \Rightarrow I$$

for the two natural transformations defined by an adjunction $F \dashv G$; these are the *unit* and the *counit* of the adjunction, respectively.

Initial and final objects can be described in terms of adjunctions as follows. Consider the trivial category $\mathbf{1}$ with only one object and one (identity) arrow. From every category $\mathbf{C}$ there is a unique functor

$$\mathbf{C} \to \mathbf{1}$$

to $\mathbf{1}$. Now, this functor has a left adjoint if and only if $\mathbf{C}$ has an initial object: this left adjoint maps the unique object of $\mathbf{1}$ to the initial object of $\mathbf{C}$; the counit of the adjunction at an object X of $\mathbf{C}$ gives the unique arrow from the initial object to X. Dually, the functor $\mathbf{C} \to \mathbf{1}$ has a right adjoint if and only if $\mathbf{C}$ has a final object and the unit of the adjunction gives the unique arrows to this final object.

Also coproducts and products can be described in terms of adjunctions. Consider the *product category* $\mathbf{C} \times \mathbf{C}$ having as objects and arrows pairs $<X, X'>$ of objects and pairs $< f, f' >$ of arrows of $\mathbf{C}$, with componentwise composition. There is a *diagonal functor*

$$\Delta : \mathbf{C} \to \mathbf{C} \times \mathbf{C} \qquad X \mapsto <X, X> \qquad f \mapsto <f, f>$$

'duplicating' the objects and the arrows of $\mathbf{C}$. This diagonal functor has a left adjoint if and only if $\mathbf{C}$ has (binary) coproducts; this left adjoint maps a pair $<X, Y>$ of objects of $\mathbf{C}$ to their *binary coproduct* $X + Y$ and the value of the unit at $<X, Y>$ is the corresponding pair of *injections* $<\mathsf{inl}_X, \mathsf{inr}_Y>$. Dually, the right adjoint, if it exists, gives *binary products* and the counit gives the corresponding *projections*.

The above binary product and coproduct adjunctions are instances of the following. Consider an arbitrary *small category* J, that is, a category with a (small) set of objects and a (small) set of arrows. (Counterexample: **Set** is not small.) Next, take the functor category

$$\mathbf{C}^J$$

having as objects the functors from J to $\mathbf{C}$ and as arrows the natural transformations between them. By putting J in $\mathbf{C}^J$ equal to the empty category $\mathbf{0}$ with no objects one obtains a category isomorphic to $\mathbf{1}$; similarly, by putting J equal to the category

$$\cdot \;\; \cdot$$

with two objects and no arrows other than the identities, one obtains a category isomorphic to $\mathbf{C} \times \mathbf{C}$:

$$\mathbf{C}^{\mathbf{0}} \cong \mathbf{1} \qquad \mathbf{C}^{\cdot\;\cdot} \cong \mathbf{C} \times \mathbf{C}$$

Correspondingly, the two functors $\mathbf{C} \to \mathbf{1}$ and $\Delta : \mathbf{C} \to \mathbf{C} \times \mathbf{C}$ can be seen as instances of a general notion of a diagonal functor

$$\Delta : \mathbf{C} \to \mathbf{C}^J$$

This diagonal functor maps an object X of $\mathbf{C}$ to a functor from J to $\mathbf{C}$ which, in turn, maps every object of J to X and all arrows of J to the identity on X. The left adjoint to this Δ give the 'colimits' of functors $D : J \to \mathbf{C}$ and the right adjoint gives the 'limits'. Thus initial objects and coproducts on the one hand and final objects and products on the other hand are, respectively, special cases of colimits and limits, which are the most common form of universals.

As an example, consider the category J with three objects and, apart from the identities, two arrows connecting one object to the other two:

$$\cdot \leftarrow \cdot \rightarrow \cdot$$

A functor $D : J \to \mathbf{C}$ from J to $\mathbf{C}$ can be seen as a *diagram* D in $\mathbf{C}$ of 'shape' J:

$$\begin{tikzcd} D: & Y_1 & Y_0 \arrow[l, "f"'] \arrow[r, "g"] & Y_2 \end{tikzcd}$$

A natural transformation $\vartheta : D \Rightarrow \Delta X$ from such a diagram D to the constant diagram $\Delta X : J \to \mathbf{C}$ obtained by applying Δ to an object X of $\mathbf{C}$

$$\begin{tikzcd} D: & Y_1 \arrow[d, "\vartheta_1"'] & Y_0 \arrow[l, "f"'] \arrow[r, "g"] \arrow[d, "\vartheta_0"'] & Y_2 \arrow[d, "\vartheta_2"] \\ \Delta X: & X \arrow[r, equal] & X \arrow[r, equal] & X \end{tikzcd}$$

can be collapsed into a '*cocone*' over D having X as 'vertex':

$$\begin{tikzcd} D: & Y_1 \arrow[dr, "\vartheta_1"'] & Y_0 \arrow[l, "f"'] \arrow[r, "g"] \arrow[d, "\vartheta_0"] & Y_2 \arrow[dl, "\vartheta_2"] \\ & & X & \end{tikzcd}$$

that is, a family of arrows from the objects of the diagram D to X making everything in sight commute. (Notice that the middle arrow $\nu_0 : Y_0 \to X$ is superfluous because it factorizes (both) as $\nu_1 \circ f$ (and as $\nu_2 \circ g$).)

The *colimit of the diagram* D is then the universal cocone over D, that is, a cocone $\nu : D \Rightarrow \mathrm{Colim}D$ such that every cocone over D factorizes uniquely through it:

$$\begin{tikzcd}
D: & Y_1 \arrow[dr, "\nu_1"] \arrow[ddr, bend right, "\vartheta_1"'] & Y_0 \arrow[l, "f"'] \arrow[r, "g"] & Y_2 \arrow[dl, "\nu_2"'] \arrow[ddl, bend left, "\vartheta_2"] \\
& & \mathrm{Colim}\,D \arrow[d, dashed] & \\
& & X &
\end{tikzcd}$$

The existence and uniqueness of the '*mediating arrow*' from the colimit of a diagram D to the vertex X of any cocone over D expresses the universal property of the colimit.

In general, the left adjoint of the diagonal functor $\Delta : \mathbf{C} \to \mathbf{C}^J$, if it exists, maps diagrams of shape J to their colimit object; the unit of the adjunction gives the corresponding (universal) colimiting cocone.

The study of colimits can be reduced to the study of initial objects and '*pushouts*', the latter being colimits of diagrams of shape $J = \cdot \leftarrow \cdot \rightarrow \cdot$. Indeed, the colimit of any (small) diagram can be expressed in terms of combinations of (generalized) pushouts and initial objects. For instance, the coproduct $X + Y$ is isomorphic to the pushout of the diagram

$$\begin{tikzcd}
X & 0 \arrow[l, dashed] \arrow[r, dashed] & Y
\end{tikzcd}$$

where 0 is the initial object. Alternatively, (small) colimits can also be described in terms of (generalized) coproducts and '*coequalizers*', the latter being colimits of diagrams of shape

$$J \;=\; \cdot \;\rightrightarrows\; \cdot$$

A generalized coproduct is obtained by generalizing the two objects and no arrows category $J = \cdot\,\cdot$ to a category with a (small) set I of objects and no arrows. One writes then

$$\coprod_I X_i$$

for the corresponding coproduct. (And, similarly, binary pushouts can be generalized by taking (small) sets of arrows with the same domain.)

By duality, *limits* are right adjoints to diagonal functors and the counit gives the *limiting cones* over diagrams $D : J \to \mathbf{C}$, that is, the universal among the cones

$\nu : \Delta X \to D$. Products are limits with $J = \cdot\ \cdot$, while the dual of coequalizers and pushouts are limits of

$$J = \cdot \rightrightarrows \cdot$$

and

$$J = \cdot \to \cdot \leftarrow \cdot$$

and are called *equalizers* and *pullbacks*, respectively. All limits can be described with products and equalizers only, as well as with final objects and pullbacks only.

Notice that equalizers are 'left-cancellable' in the sense that, given an equalizer $m : Y \to Z$ and two parallel arrrows $f, g : X \to Y$, if $m \circ f = m \circ g$ then $f = g$; in general, left-cancellable arrows are called *monic arrows*. Dually, coequalizers are *epi*, ie 'right-cancellable'. In **Set** the epi and the monic arrows are the surjective and the injective functions, respectively.

Some final notational remarks. The (standard) notation for pullbacks and pushouts is

$$\begin{tikzcd} \cdot \arrow[r] \arrow[d] \arrow[dr, phantom, very near start, "\lrcorner"] & Y_2 \arrow[d, "g"] \\ Y_1 \arrow[r, "f"'] & Y_0 \end{tikzcd} \qquad \text{and} \qquad \begin{tikzcd} Y_0 \arrow[r, "g"] \arrow[d, "f"'] & Y_2 \arrow[d] \\ Y_1 \arrow[r] \arrow[ur, phantom, very near end, "\ulcorner"] & \cdot \end{tikzcd}$$

respectively. Also, it is useful to introduce a special (non-standard) notation for the injection arrows $\mathsf{inl}_X : X \to X + Y$ and $\mathsf{inr}_Y : Y \to X + Y$ into a coproduct, namely by adding a triangle to their 'tails':

$$\begin{tikzcd} X \arrow[r, tail] \arrow[dr, "f"'] & X + Y \arrow[d, dashed, "{[f,g]}"] & Y \arrow[l, tail] \arrow[dl, "g"] \\ & Z & \end{tikzcd}$$

Thus the above '*copair*' $[f, g] : X + Y \to Z$ of $f : X \to Z$ and $g : Y \to Z$ is the universal mediating arrow from the coproduct $X + Y$ to the vertex Z of the cocone formed by f and g over X and Y.

Notes. The standard textbook of category theory is [Mac71], whose first six chapters include the basic category theory used in this thesis; a useful summary (with examples and exercises) of those chapters can be found in Part 0 of [LS86].

For an alternative, vivid presentation of category theory see [FS90]. Computer scientists might want to consult also [Poi92] and [Cro93].

For the philosophical import of category theory (and of the notion of adjointness in particular) one can consult [Law69] and other Lawvere's writings, which are rich of stimulating connections between disparate fields.