Università degli Studi di Milano - Bicocca

Facoltà di Scienze Matematiche, Fisiche e Naturali

Corso di Laurea in Informatica

INDUCTIVE AND COINDUCTIVE TECHNIQUES IN THE OPERATIONAL ANALYSIS OF FUNCTIONAL PROGRAMS: AN INTRODUCTION

Relatore:
Prof. Felice Cardone
Correlatore:
Prof. Simonetta Ronchi Della Rocca

Laureando:
Marco Gaboardi
matricola
053489

ANNO ACCADEMICO 2003-2004

A man who is always asking "Is what I do worth while?" and "Am I the right person to do it?" will always be ineffective himself and a discouragement to others. He must shut his eyes a little and think a little more of his subject and himself than they deserve. This is not too difficult: it is harder not to make his subject and himself ridiculous by shutting his eyes too tightly.

Godfrey Harold Hardy: "A Mathematician's Apology".

Abstract

Many works have shown that *operational semantics* is a useful framework for the formal study of programs properties. Our investigation takes as object of study a call-by-name variant of FPC, a functional language with higher order functions and recursive types.
We show that by using Plotkin's *structural operational semantics* it is possible to specify formal semantics for FPC for both convergent and divergent programs. Structural operational semantics can be specified by means of rules, therefore it allows to use induction and coinduction principles, to define objects and to reason about their properties.
Two approaches are well known in giving structural operational semantics for programming languages: *big-step* and *small-step*. For a language like FPC they can be related by showing that they capture the same meaning. Furthermore, we show that it is possible to translate a proof in the big-step formalism in a proof in the small-step by adding some information to rules.
Each formal theory of programming languages must analyze *equality* in-depth. In an operational setting, Morris's style *contextual equivalence* is the widely accepted natural notion of equivalence for programs. Many properties of contextual equivalence can be proved by using simple proof principles. In particular, we prove two interesting properties, *syntactic continuity* and *rational completeness*. They can thought of as the syntactical counterparts of well-known notions in *domain theory*.
Unfortunately, contextual equivalence is not straightforward to prove. Nevertheless it is coinductive in nature, for this reason it is interesting to find different formulations which permit to coinductively reason about program equality. We show that two classical programs equivalence notions, named *experimental equivalence* and *bisimilarity* respectively, coincide with contextual equiavalence for FPC. Finally we conclude by showing an example of how the theory developed in this work can be used in programming practice.

Acknowledgments

First and foremost thanks to Prof. Felice Cardone. It was he that first introduced me to operational semantics and showed me how its theory is interesting. During last year he helped and encouraged me to realize the present work. Prof. Cardone patiently answered to all my questions providing me with interesting relevant material. He gave me suggestions about all the parts of this thesis and most of all he corrected many of my infinite errors.

Also, I'm grateful to Prof. Simonetta Ronchi della Rocca, who gave me useful suggestions concerning this work and its presentation. Her critiques helped me to clarify some ideas.

I would like to thank all my friends (they are too many to be named, but they surely know who they are!) that spent their time in (sometimes crazy) conversations with me; their influence permeate this work as well as all my ideas.

Most of all, I wish to thank my parents, my brother and my sister for their support through the years. They could not even realize how much my vision depends on what I have learned from them.

Contents

Chapter 1

Introduction

> Among those philosophers who have carried out semantical analyses and thought about suitable tools for this work, beginning with Plato and Aristotle and, in a more technical way on the basis of modern logic, with C. S. Peirce and Frege, a great majority accepted abstract entities. This does, of course, not prove the case. After all, semantics in the technical sense is still in the initial phases of its development, and we must be prepared for possible fundamental changes in methods. Let us therefore admit that the nominalistic critics may possibly be right. But if so, they will have to offer better arguments than they have so far. Appeal to ontological insight will not carry much weight. The critics will have to show that it is possible to construct a semantical method which avoids all references to abstract entities and achieves by simpler means essentially the same results as the other methods.
>
> Rudolf Carnap: "Empiricism, Semantics, and Ontology".

Specifying a *formal semantics* for programming languages means to clarify what does each program means; this must be the first step in any analysis of any language. There are different approaches in defining the formal semantics of a programming language. Historically three of them emerged: *denotational*, *operational*, *axiomatic*.

Giving an *operational semantics* to a programming language means to specify the meaning of the language by the description of how each *construct* is executed by an abstract mathematical machine. Since in origin it was considered a low-level reasoning, in the first years of "semantics research" it attracted only marginally the attention of the computer theorists. Nevertheless different people carried out studies based on this approach adding new insights. The development of the research in this field culminates in the 80s in the works [Plotkin, 1981] and [Kahn, 1987]. Thanks to that developments nowadays operational approach is considered an easy and quite natural approach to give meanings to real programming languages.

Traditionally denotational semantics was preferred to operational semantics because it permits to deal with data structure by familiar set-theoretical tools. The denotational approach originates from ideas of Scott and Strachey

[Scott and Strachey, 1971], and considers two programs to be equivalent when they denote the same object in some abstract *domain*. Unfortunately, as noted by Scott, usually this *domains* are too rich, in the sense that there exist elements of the domain which are not denotation of any sequential programs. This is well known as the *full abstraction* problem.
An interesting feature of operational semantics is that in order to use it one needs only to introduce some elementary mathematics; in contrast, denotational semantics needs a complex mathematical machinery even for basic definitions. Hence one can hope to use only operational semantics to reason about programs, but in order to make this possible one must show that operational semantics *achieves by simpler means essentially the same results* as denotational semantics.
This is nowaday far to be proved, nevertheless in the first years of the 90s, the idea emerged that many basic structures of domain theory can be built using operational tools alone: this can be considered a *bottom-up* approach to the solution of full abstraction problem. Furthermore, many of the advanced results provided by domain theory, because of their abstract nature, are far from being used in the practice of programming; this does not mean that they are not useful in general.
Hence, for the above reasons, we can state that operational semantics *achieve by simpler means essentially the same "basic" results* of denotational semantics.

A further step in the analysis of a programming language is to provide techniques that can be used to reason about the properties of programs. *Inductive definitions* and *induction* as a proof principle are well established techniques in the analysis of programs properties, in particular they are useful in dealing with finite objects. An interesting feature of some functional programming languages is that they provide also infinite and circular object, hence in order to study this kind of objects inductive techniques need an extension. Less known techniques are *coinductive definitions* and *coinduction* proof principle, which are the formal duals of inductive definitions and induction respectively. They originally emerged in the fields of concurrency, from ideas of Park [Park, 1981] and Milner [Milner, 1989], and only in the 90s they come to be studied in the field of functional sequential languages.
An interesting property of programming languages which must be investigated, in order to make any useful analysis, is the equivalence of programs. In fact every formal semantics gives origin to a natural notion of program equivalence. Morris's style *Contextual equivalence* [Morris, 1968] is the more natural notion of equivalence for functional programs in an operational settings. It is in nature coinductive but its formulation makes hard the job of proving equalities of programs. For this reason in the last years different formulations of equivalence which can be proved to agree with contextual equivalence for some languages emerged. These formulations come equipped with natural coinductive proof principle which make easy to prove equalities of programs.

1.1 This work

This work is intended to survey the different techniques which can be used in the operational analysis of functional programming language. We introduce

the main techniques and we show how they can be related by following three principles.
The first principle which leads our investigations is to "be as operational as possible". This means that we try to prove the results only with the few instruments which come associated with our definitions. For this reason, sometimes our proofs are long and strongly based on the syntax; we usually prefer not to make use of higher-level methods even if they are clearer. This principle is justified by the fact that this work can be considered as a first step in the analysis of the base of operational semantics, where as an exercise we prove all what can be proved with the limited tools at our disposal.
Another principle that we follow is "exploit induction and coinduction". In each inductive or coinductive definition, where it is not immediately clear, we emphasise how they can be inductively or coinductively formulated. This can be useful to understand some proofs better, because associated with each (co)inductive definition we have a (co)inductive principle which is useful to prove properties about the objects under investigations.
Finally we follow the principle "be concrete". We could have chosen to analyze abstract languages which follow a structured semantics, nevertheless we chose to take as base for our investigations a concrete language. This in the spirit of a future abstraction which originates from a concrete example. We chose FPC[1] because it possesses all the interesting features that a functional language must possess to be a real programming language.
All the result presented in this thesis were already investigated by other authors. We think that the originality of this works is based in the methods of the investigations, in the reformulation of some interesting concepts and in the form of presentation. We think that our work can be considered as a good introduction to the field of operational semantics for functional programming language. Furthermore we believe that several remarks in the work need to be expanded in future works.
Let us now consider the structure of the thesis in more detail.

Chapter 2 We present induction and coinduction from a set-theoretic point of view. We introduce the notions of partial orders, lattices and complete lattices. We show how a set of rules induces a monotone operator on a complete lattice. We show how we can define sets inductively and coinductively as least fixed points and greatest fixed points respectively of monotone operators on complete lattices. We investigate some properties of monotone operators with consequences on inductively and coinductively defined sets.

Chapter 3 We briefly introduce FPC and its syntax.

Chapter 4 We give both static and dynamic semantics for FPC following the principles of Plotkin's *Structural Operational Semantics*. We give dynamic semantics both for convergent terms and for divergent ones. We present dynamic semantics through two different relation, one "big-step" which capture the entire computations from a term possibly to a "result" and another, "small-step",

[1] 'FPC' is an acronym for 'Fixed Point Calculus', a language traditionally attributed to [Plotkin, 1985], albeit the language described in those lectures was a metalanguage for the category of predomanis and partial functions.

which captures single steps of computation. We give an inductive characterization of convergence and a coinductive characterization of divergence. Finally we prove that semantics through big-step semantics is equivalent to that defined as the transitive and reflexive closure of the small-step relation.

Chapter 5 We demonstrate a method to translate proofs through big-step semantics in proofs through small-step semantics, which is inspired by works of Ibraheem and Schmidt [Ibraheem and Schmidt, 2000]. To apply this method we need to extend proofs by big-step semantics by adding some syntactical informations. The method applies both to convergent and divergent proofs.

Chapter 6 We introduce a general theory of contexts, useful to formulate a theory of program equivalence based on the definition of *contextual equivalence*. Contextual equivalence is the natural notion of program equivalence. We prove, in a pure operational settings, different properties of FPC with respect to our definition of contextual equivalence. In particular we conclude this chapter by proving *syntactic continuity* and *rational completeness* properties.

Chapter 7 We analyze other programs equivalence notions, in particular we introduce *experimental equivalence* and *bisimilarity* which were already studied by different authors. Furthermore we show that these two formulations coincide with the formulation of contextual equivalence given in the previous chapter; this allows to reason coinductively about program equivalence.

Chapter 8 We show how we can use FPC to define the usual constructions which can be defined in a functional programming language with recursive types. In particular we define *Booleans Natural numbers* and *Lists*. We reformulate the coinductive proof principle associated with these constructions and as an exercise we prove a restricted version of *Take Lemma* [Bird and Wadler, 1988].

Chapter 9 We review the results and discuss further developments.

1.2 Related work

In the past, the main contributions in the theory of operational approach have followed two strictly related directions: the first was oriented to develop methods to make operational approach clear, general and powerful; the second to study the limits of the formal theory of programs obtained by applying the methods of the first.
The main developments in the former direction take origins in the 60s and 70s by ideas of Landin [Landin, 1964] an of the IBM Vienna Group, (see [Jones, 2003] for a retrospective account of that period), and reached in the 80s a good and complete formalization in the works [Plotkin, 1981] and [Kahn, 1987].
In the latter direction the situation is not so easy to explain, the different developments deriving from different motivations and sometimes the same things being discovered indipendently in different fields. We now try to summarize the main works.

Milner in [Milner, 1977] proved a result well known as "context lemma" which firstly stated the coincidense of *contextual equivalence* with what we call *experimental equivalence*, for a form of PCF based on combinators [2].
Howe in [Howe, 1989] gave an important method, refined in [Howe, 1996], to prove congruence property for a class of languages with a particular style of operational semantics. Howe's method can be easily extended to prove that a specific formulation of *bisimilarity* coincides with contextual equivalence.
Bloom in [Bloom, 1990] and Jim and Meyer in [Jim and Meyer, 1991] studied the relations beetween operational and denotational semantics for a class of PCF-like languages, in particular they investigated relations beetween Milner's context lemma and the full abstraction problem.
Smith, in [Smith, 1991] and with Mason and Talcott in [Mason et al., 1996], showed how it is possible to derive, for different versions of untyped *lambda*-calculus with arithmetic constants, the basic notions of domain theory in an operational settings. Mason and Talcott in [Mason and Talcott, 1991] had already shown that operational techniques can be widely applied to complex languages with continuations, first-class objects and a global state.
Gordon in [Gordon, 1995] used a version of Howe's method based on *CCS-style* labelled transition system to prove that usual CCS-style bisimilarity, for a version of FPC with ground types, coincides with contextual equivalence. Furthermore in the same work he studied various refinements of bisimilarity.
Pitts in [Pitts, 1994] made a survey of operational techniques and showed how they can be related to the denotational ones; in [Pitts, 1995] he showed how to reason coinductively about program equivalence. In [Pitts and Stark, 1998] it is shown how to extend operational techniques to a functional language with local state and, in [Pitts, 2002], how operational semantics can be used to reason about program equivalence for a fragment of ML.
Birkedal and Harper in [Birkedal and Harper, 1999] applied operational techniques to prove *relational* properties of semantics for a language with recursive types. Sands in [Sands, 1997] and in elsewhere studies in an operational setting an *improvement* version of induction.
Operational semantics was already studied in the field of process algebra, see [Aceto et al., 1999] for a complete survey, and from a categorical point of view, see [Turi, 1996], [Turi and Plotkin, 1997] and [Power, 2003].
Finally, many interesting works about *higher order operational techniques in semantics* were collected in [Gordon and Pitts, 1998], [Gordon et al., 1998],[Gordon and Pitts, 1999] and [Jeffrey, 2000].

[2] PCF(= Programming language for Computable Functions) is a simply typed λ-calculus with arithmetic constants and recursion introduced by Plotkin as the term language of Scott's Logic for Computable Functions, and may be seen as the core of the functional programming language ML.

Chapter 2

Induction and Coinduction

> The turkey found that, on his first morning at the turkey farm, that he was fed at 9 a.m. Being a good inductivist turkey he did not jump to conclusions. He waited until he collected a large number of observations that he was fed at 9 a.m. and made these observations under a wide range of circumstances, on Wednesdays, on Thursdays, on cold days, on warm days. Each day he added another observation statement to his list. Finally he was satisfied that he had collected a number of observation statements to inductively infer that "I am always fed at 9 a.m.". However on the morning of Christmas eve he was not fed but instead had his throat cut.
>
> A version of the famous original Bertrand Russell's tale

Inductive Definitions and the *Induction proof principle* are well known and extensively studied techniques in Logic, Mathematics and Theoretical Computer Science. Recent researche has shown the need to extend these techniques, especially in the study of infinite and circular objects.
The appropriate way of formulating such extensions emerged originally in the field of concurrent programming languages from ideas of Park in [Park, 1981] and Milner in his *Calculus of Communicating Systems*, see [Milner, 1989]. These ideas converged into techniques now well known as *Coinductive definitions* and the *Coinduction proof principle* which are the formal duals of inductive definitions and induction respectively.
We are interested in (co)inductive definitions and (co)induction for the central role they play in the semantics of functional programming languages. Nevertheless, we try to introduce them in as simple framework as possible. In the sequel we only assume the knowledge of very basic notions of set theory.

2.1 Ordered structures

We consider a set of "everything in the world" and call this the *universal set* $\mathcal{U}$. We need some kind of structure on our universal set.

Definition 2.1.1. *A **partial order** on a set $P \subseteq \mathcal{U}$ is a binary relation $\leq_P \subseteq P \times P$ with the following properties:*

reflexive $\forall\ X\ \in\ P\ :(X \leq_P X)$

transitive $\forall\ X,Y,Z\ \in\ P\ :(X \leq_P Y\ \&\ Y \leq_P Z\ \Longrightarrow X \leq_P Z)$

antisymmetric $\forall\ X,Y\ \in\ P\ :(X \leq_P Y\ \&\ Y \leq_P X\ \Longrightarrow X = Y)$

Definition 2.1.2. *A **partially ordered set** is a set $P \subseteq \mathcal{U}$ with a partial order $\leq_P \subseteq P \times P$.*

We usually write $\leq_P$ as $\leq$ when it is clear the set where it is defined, we usually use the shorthand *poset* for partially ordered set.
An interesting example of poset, which will be central in the sequel, is the powerset $\mathcal{P}(X)$ for any $X \subseteq \mathcal{U}$ with $\subseteq$ as partial order. We write $\langle \mathcal{P}(X), \subseteq \rangle$ when we want to stress that we consider this poset structure.
An interesting class of partial orders is the following.

Definition 2.1.3. *Let $P \subseteq \mathcal{U}$ be a poset with partial order $\leq_P \subseteq P \times P$.*
*An **ascending ω-chain** is a subset $C = \{c_0, c_1, \ldots, c_m, \ldots\}$ of P such that:*

$$c_0 \leq_P c_1 \leq_P \cdots \leq_P c_m \leq_P \cdots$$

*A **descending ω-chain** is a subset $C = \{c_0, c_1, \ldots, c_m, \ldots\}$ of P such that:*

$$\cdots \leq_P c_m \leq_P \cdots \leq_P c_1 \leq_P c_0$$

We are interested in relations beetween partially ordered sets. An interesting class of relations beetween posets is the class of maps which preserve the order structure.

Definition 2.1.4. *A map $\phi : P \to Q$ where $P, Q \subseteq \mathcal{U}$ are posets is **monotone** if and only if for all $x, y \in P$ if $x \leq_P y$ then $\phi(x) \leq_Q \phi(y)$.*

When we define a poset P we have for free the *dual poset* $P^{\mathcal{D}}$.

Definition 2.1.5. *Given a poset P we can define its **dual poset** $P^{\mathcal{D}}$ by defining $x \leq y$ to hold in $P^{\mathcal{D}}$ if and only if $y \leq x$ holds in P*

Some special elements for partially ordered sets are the following.

Definition 2.1.6. *An element x is an **upper bound** for a subset $S \subseteq P$ where P is a poset if and only if for all $s \in S\ :\ (s \leq x)$.*

Definition 2.1.7. *An element x is the **least upper bound** for a subset $S \subseteq P$ where P is a poset if and only if*

(i) $\forall s \in S\ :\ (s \leq x)$

(ii) $\forall\ p \in P\ :\ (\forall\ s \in S\ :\ (s \leq p) \Longrightarrow (x \leq p))$

It is easy to verify that if such element exists it is unique. We usually write the least upper bound of two elements x, y as $x \vee y$ and the least upper bound of a set S as $\bigvee S$. Sometimes we use **lub** as shorthand for least upper bound, other used shorthands are *sup* or *join*.

Definition 2.1.8. *An element x is a **lower bound** for a subset $S \subseteq P$ where P is a poset if and only if $\forall s \in S\ :\ (x \leq s)$.*

Definition 2.1.9. *An element x is the* ***greatest lower bound*** *for a subset $S \subseteq P$ where P is a poset if and only if*

(i) $\forall s \in S \ : (x \leq s)$

(ii) $\forall p \in P \ : \ (\forall s \in S \ : \ (p \leq s) \Longrightarrow p \leq x)$

It is easy to verify that if greatest lower bound exists then it is unique. We usually write the greatest lower bound of two elements x, y as $x \wedge y$ and the greatest lower bound of a set S as $\bigwedge S$. Sometimes we use **glb** as shorthand for gratest lower bound, other used shorthands are *inf* and *meet*.
It is clear that lower bound is the dual notion of upper bound and hence *glb* is the dual notion of *lub*.
More interesting ordered structures than posets are *lattices* and *complete lattices*.

Definition 2.1.10. *A poset P is a* ***lattice*** *if and only if for all $x, y \in P$ $x \vee y$ and $x \wedge y$ exist.*

Definition 2.1.11. *A poset P is a* ***complete lattice*** *if and only if for all $S \subseteq P$ $\bigvee S$ and $\bigwedge S$ exist.*

We have seen that for all $X \subseteq \mathcal{U}$, $\langle \mathcal{P}(X), \subseteq \rangle$ is a poset. Moreover it is also a complete lattice if we define glb and lub as:

$$\bigvee \{X_i \in \mathcal{P}(X) \mid i \in I\} = \bigcup_{i \in I} X_i$$
$$\bigwedge \{X_i \in \mathcal{P}(X) \mid i \in I\} = \bigcap_{i \in I} X_i$$

We write $\langle \mathcal{P}(X), \subseteq, \bigcup, \bigcap \rangle$ to stress that we consider $\mathcal{P}(X)$ with complete lattice structure.
In a complete lattice P the elements $\bigvee P$ and $\bigwedge P$ are respectively the greatest and the least elements of P. We sometimes call them respectively *top* and *bottom*.
It is easy to prove the following lemmas.

Lemma 2.1.12. *Let P be a complete lattice then also its dual $P^{\mathcal{D}}$ is a complete lattice.*

Lemma 2.1.13. *Let P be a finite lattice then P is complete.*

Lemma 2.1.14. *Let P be a poset, then $\bigvee S$ exists for all $S \subseteq P$ if and only if $\bigwedge S$ exists for all $S \subseteq P$. Hence a poset P is a complete lattice if and only if for all $S \subseteq P$, $\bigvee S$ exists.*

Proof. For all $S \subseteq P$ we have that its glb can be expressed as a lub and dually its lub can be expressed as a glb, as follow :

$$\bigwedge S = \bigvee \{p \in P | \forall s \in S \ (p \leq s)\}$$
$$\bigvee S = \bigwedge \{p \in P | \forall s \in S \ (s \leq p)\}$$

hence the existence of the glb for all sets imply the existence of lub and vice versa. □

An interesting property of maps on complete lattices is *continuity*.

Definition 2.1.15. *A map $\phi : P \to Q$ where $P, Q \subseteq \mathcal{U}$ are complete lattice is **continuous** if and only if for each ascending ω-chain $S \subseteq P$:*

$$\phi(\bigvee S) = \bigvee \phi(S)$$

Definition 2.1.16. *A map $\phi : P \to Q$ where $P, Q \subseteq \mathcal{U}$ are complete lattice is **cocontinuous** if and only if for each descending ω-chain $S \subseteq P$:*

$$\phi(\bigwedge S) = \bigwedge \phi(S)$$

2.2 Fixed points

We call **operator** any map ϕ from a poset S into itself. Operators on poset are interesting because them permit to isolate subsets of the poset which posses very useful properties.

Definition 2.2.1. *Given a poset P and an operator $\phi : P \to P$, the set of **prefixed points** of ϕ is defined as:*

$$Prefix_\phi \stackrel{\text{def}}{=} \{p \in P | \phi(p) \leq p\}$$

*if $x \in Prefix_\phi$ is such that $\forall p \in Prefix_\phi (x \leq p)$ then x is the **least prefixed point**.*

Definition 2.2.2. *Given a poset P and an operator $\phi : P \to P$, the set of **postfixed point** of ϕ is defined as:*

$$Postfix_\phi \stackrel{\text{def}}{=} \{p \in P | p \leq \phi(p)\}$$

*if $x \in Postfix_\phi$ is such that $\forall p \in Postfix_\phi (p \leq x)$ then x is the **greatest postfixed point**.*

Definition 2.2.3. *Given a poset P and an operator $\phi : P \to P$, the set of **fixed point** of ϕ is defined as:*

$$Fix_\phi \stackrel{\text{def}}{=} \{p \in P | p = \phi(p)\}$$

*if $x \in Fix_\phi$ is such that $\forall p \in Fix_\phi (p \leq x)$ then x is the **greatest fixed point**. Dually, if $x \in Fix_\phi$ is such that $\forall p \in Fix_\phi (p \leq x)$ then x is the **least fixed point**.*

From these definitions it is clear that $Fix = Prefix \cap Postfix$.
Fixed points have central roles in computer science. The next theorem states the existence of fixed points for monotone operators on complete lattices. This is an interesting version of the original theorem due to Knaster and Tarski [Tarski, 1955].

Theorem 2.2.4 (Knaster-Tarski). *Let P be a complete lattice and $\phi : P \to P$ a monotone operator then Fix_ϕ is a complete lattice.*

Proof. For any $F \subseteq Fix_\phi$ define:

$$H_F = \{p \in P |\ \forall x \in F\ (x \leq p)\ \&\ \phi(p) \leq p\}$$
$$H'_F = \{p \in P |\ \forall x \in F\ (p \leq x)\ \&\ p \leq \phi(p)\}$$

since P is a complete lattice $\bigwedge H_F$ and $\bigvee H'_F$ exists. We prove that:

$$(i)\ \bigvee F = \bigwedge H_F$$
$$(ii)\ \bigwedge F = \bigvee H'_F$$

and then for the conclusion we only need to show that $\bigwedge H_F$ and $\bigvee H_F$ are fixed points.

(i) By definition of H_F we have $\forall x \in F, (x \leq p)$ where $p \in H_F$. Hence all $x \in F$ are lower bounds of H_F and by definition of glb $\forall x \in F\ (x \leq \bigwedge H_F)$. By definition of H_F we have that any $y \in Fix_\phi$ such that $\forall x \in F\ (x \leq y)$ is in H_F. Since $\bigwedge H_F$ is a lower bound of H_F we have $\bigwedge H_F \leq y$.

(ii) By definition of H'_F we have that $\forall x \in F, (p \leq x)$ where $p \in H'_F$. Hence all $x \in F$ are upper bounds of H'_F and by definition of lub $\forall x \in F\ (\bigvee H'_F \leq x)$. By definition of H'_F we have that any $y \in Fix_\phi$ such that $\forall x \in F\ (y \leq x)$ is in H'_F. Since $\bigvee H'_F$ is an upper bound of H'_F we have $y \leq \bigvee H'_F$.

We now prove that $\bigwedge H_F$ is a fixed point. Since ϕ is monotone and by definition of glb, $\forall p \in H_F$ we have $\phi(\bigwedge H_F) \leq \phi(p) \leq p$ and by transitivity $\phi(\bigwedge H_F) \leq p$. Hence $\phi(\bigwedge H_F)$ is a lower bound of H_F but then $\phi(\bigwedge H_F) \leq \bigwedge H_F$ by definition of glb. Moreover $\forall x \in F\ (x \leq \bigwedge H_F)$, since ϕ is monotone we have $\forall x \in F\ (\phi(x) \leq \phi(\bigwedge H_F))$ and by fixed point property of each $x \in F$ we have $\forall x \in F\ (x \leq \phi(\bigwedge H_F))$, but we have already shown that for any such element $\bigwedge H_F \leq \phi(\bigwedge H_F)$ hence the conclusion.
Finally we need to prove that $\bigvee H'_F$ is a fixed point. Since ϕ is monotone and by definition of lub, $\forall p \in H'_F$ we have $p \leq \phi(p) \leq \phi(\bigvee H'_F)$ and by transitivity $p \leq \phi(\bigvee H'_F)$. Hence $\phi(\bigvee H'_F)$ is an upper bound of H'_F but then $\bigvee H'_F \leq \phi(\bigvee H'_F)$ by definition of lub. Moreover $\forall x \in F\ (\bigvee H'_F \leq x)$, since ϕ is monotone we have $\forall x \in F\ (\phi(\bigvee H'_F) \leq \phi(x))$ and by fixed point property of each $x \in F$ we have $\forall x \in F\ (\phi(\bigvee H'_F) \leq x)$, but we have already shown that for any such element $\phi(\bigvee H'_F) \leq \bigvee H'_F$ hence the conclusion.

□

An immediate consequence of Knaster-Tarski theorem is the following corollary.

Corollary 2.2.5. *Let P be a complete lattice, $\phi : P \to P$ a monotone operator on P and L its complete lattice of fixed point. Then $\mu(\phi)$ and $\nu(\phi)$ defined as:*

$$\mu(\phi) \stackrel{\text{def}}{=} \bigwedge Prefix_\phi$$
$$\nu(\phi) \stackrel{\text{def}}{=} \bigvee Postfix_\phi$$

are respectively the least (pre)fixed point and the greatest (post)fixed point.

Proof. By Knaster-Tarski theorem and by lemma 2.1.14 it is easy to verify:

$$\mu(\phi) \stackrel{\text{def}}{=} \bigwedge Prefix_\phi = \bigwedge\{x \in P | \phi(x) \leq x\} = \bigwedge_P H_\emptyset = \bigvee_L \emptyset = \bigwedge_P L$$

$$\nu(\phi) \stackrel{\text{def}}{=} \bigvee Postfix_\phi = \bigvee\{x \in P | x \leq \phi(x)\} = \bigvee_P H_\emptyset = \bigwedge_L \emptyset = \bigvee_P L$$

□

The proof of the Knaster-Tarski theorem is not constructive but the last corollary give us two foundamental elements of the complete lattice of fixed points of a monotone operator over a complete lattice. The importance of these element will become clear in the study of the complete lattice $\langle \mathcal{P}(X), \subseteq, \bigcup, \bigcap \rangle$ for each $X \subseteq \mathcal{U}$.

2.3 Inductive and coinductive definitions by set of rules

An inductively generated subset of the universe can be described abstractly by means of *rules*.

Definition 2.3.1. *A **set of rules** on $X \subseteq \mathcal{U}$ is a subset $\mathcal{R} \subseteq \mathcal{P}(X) \times X$:*

$$\mathcal{R} = \{(S, x) | S \subseteq X, x \in X\}$$

For each rule *$r = (S, x) \in \mathcal{R}$, the set S is called the set of* premisses *of r and x is called the* conclusion *of the rule r. A set of rules $\mathcal{R}$ is **finitary** if for all $(S, x) \in \mathcal{R}$, S is finite. A set of rules $\mathcal{R}$ is **deterministic** if for all $r, s \in \mathcal{R}$ such that, $r = (S, x), s = (S', x)$ then $S = S'$.*

We sometimes write a rule $r = (S, x)$ as $r = (\{x_1, \ldots, x_n, \ldots\}, x)$ if $S = \{x_1, \ldots, x_n, \ldots\}$. If $r = (\{x_1, \ldots, x_n\}, x)$ is a rule of a finitary set of rules we usually write it as

$$\frac{x_1 \ \cdots \ x_n}{x}$$

Definition 2.3.2. *Given a set of rules $\mathcal{R}$ on $X \subseteq \mathcal{U}$ the **operator induced** by $\mathcal{R}$, $\phi_\mathcal{R} : \mathcal{P}(X) \to \mathcal{P}(X)$ is defined as:*

$$\phi_\mathcal{R}(S) = \{x \in X \mid \exists S' \subseteq S \text{ such that } (S', x) \in \mathcal{R}\}$$

It is easy to verify that for each set of rules $\mathcal{R}$ the operator $\phi_\mathcal{R}$ induced by $\mathcal{R}$ is monotone if we consider it as an operator on the complete lattice $\langle \mathcal{P}(X), \subseteq, \bigcup, \bigcap \rangle$.
In particular we have the following.

Lemma 2.3.3. *Given $\langle \mathcal{P}(X), \subseteq, \bigcup, \bigcap \rangle$ and $\phi_\mathcal{R} : \mathcal{P}(X) \to \mathcal{P}(X)$ the operator induced by a set of rules $\mathcal{R}$ on $X \subseteq \mathcal{U}$. If $\mathcal{R}$ is* finitary *then $\phi_\mathcal{R}$ is* continuous*.*

Proof. We need to prove that for all $\{Y_1, \ldots, Y_n, \ldots\} \subseteq \mathcal{P}(X)$ such that $Y_1 \subseteq Y_2 \subseteq \cdots \subseteq Y_n \subseteq \cdots$, we have:

$$\phi_\mathcal{R}(\bigcup_{i \in \mathbb{N}} Y_i) \ = \ \bigcup_{i \in \mathbb{N}} \phi_\mathcal{R}(Y_i)$$

Given an ascending ω-chain

$$Y_1 \subseteq Y_2 \subseteq \cdots \subseteq Y_n \subseteq \cdots$$

since $\phi_{\mathcal{R}}$ is monotone we have:

$$\phi_{\mathcal{R}}(Y_1) \subseteq \phi_{\mathcal{R}}(Y_2) \subseteq \cdots \subseteq \phi_{\mathcal{R}}(Y_n) \subseteq \cdots$$

but for all Y_i we have $\phi_{\mathcal{R}}(Y_i) \subseteq \phi_{\mathcal{R}}(\bigcup_{i\in\mathbb{N}} Y_i)$ and then $\bigcup_{i\in\mathbb{N}} \phi_{\mathcal{R}}(Y_i) \subseteq \phi_{\mathcal{R}}(\bigcup_{i\in\mathbb{N}} Y_i)$.
For each $(S, x) \in \mathcal{R}$ we know that S is finite and so if $S \in \bigcup_{i\in\mathbb{N}} Y_i$ then exists finite Y_j such that $S \subseteq Y_j$ and $Y_j \subseteq \bigcup_{i\in\mathbb{N}} Y_i$. But hence $x \in \phi_{\mathcal{R}}(Y_j)$ and so $x \in \bigcup_{i\in\mathbb{N}} \phi_{\mathcal{R}}(Y_i)$ from which follows $\phi_{\mathcal{R}}(\bigcup_{i\in\mathbb{N}} Y_i) \subseteq \bigcup_{i\in\mathbb{N}} \phi_{\mathcal{R}}(Y_i)$. □

Lemma 2.3.4. *Given $\langle \mathcal{P}(X), \subseteq, \bigcup, \bigcap \rangle$ and $\phi_{\mathcal{R}} : \mathcal{P}(X) \to \mathcal{P}(X)$ the operator induced by a set of rules $\mathcal{R}$ on $X \subseteq \mathcal{U}$. If $\mathcal{R}$ is* deterministic *then $\phi_{\mathcal{R}}$ is* cocontinuous.

Proof. We need to prove that for all $\{Y_1, \ldots, Y_n, \ldots\} \subseteq \mathcal{P}(X)$ such that $\cdots \subseteq Y_n \subseteq \cdots \subseteq Y_2 \subseteq Y_1$, we have:

$$\phi_{\mathcal{R}}(\bigcap_{i\in\mathbb{N}} Y_i) \;=\; \bigcap_{i\in\mathbb{N}} \phi_{\mathcal{R}}(Y_i)$$

Given a descending ω-chain

$$\cdots \subseteq Y_n \subseteq \cdots \subseteq Y_2 \subseteq Y_1$$

since $\phi_{\mathcal{R}}$ is monotone we have:

$$\cdots \subseteq \phi_{\mathcal{R}}(Y_n) \subseteq \cdots \subseteq \phi_{\mathcal{R}}(Y_2) \subseteq \phi_{\mathcal{R}}(Y_1)$$

but for all Y_i we have $\phi_{\mathcal{R}}(\bigcap_{i\in\mathbb{N}} Y_i) \subseteq \phi_{\mathcal{R}}(Y_i)$ and then $\phi_{\mathcal{R}}(\bigcap_{i\in\mathbb{N}} Y_i) \subseteq \bigcap_{i\in\mathbb{N}} \phi_{\mathcal{R}}(Y_i)$.
If $x \in \bigcap_{i\in\mathbb{N}} \phi_{\mathcal{R}(Y_i)}$ then for all $n \in \mathbb{N}$ we have $x \in \phi_{\mathcal{R}}(Y_n)$ and since $\mathcal{R}$ is deterministic $\forall n\ S \subseteq Y_n$. So we have $S \in \bigcap_{i\in\mathbb{N}} Y_i$ and then $x \in \phi_{\mathcal{R}}(\bigcap_{i\in\mathbb{N}}(Y_i))$. □

Another important observation, which is easy to verify, is that any monotone operator $\phi : \mathcal{P}(X) \to \mathcal{P}(X)$ can be written in the form $\phi_{\mathcal{R}}$ for some rule set $\mathcal{R}$ on X.
We now restate some useful notions of the previous section in this framework.

Definition 2.3.5. *A set $Y \subseteq X$ is:*

- ***$\phi_{\mathcal{R}}$-closed*** *if and only if* $\phi_{\mathcal{R}}(Y) \subseteq Y$
- ***$\phi_{\mathcal{R}}$-dense*** *if and only if* $Y \subseteq \phi_{\mathcal{R}}(Y)$

where $\phi_{\mathcal{R}} : \mathcal{P}(X) \to \mathcal{P}(X)$ is the operator induced by the set of rules $\mathcal{R}$ on $X \subseteq \mathcal{U}$.

Since the order relation of $\langle \mathcal{P}(X), \subseteq, \bigcup, \bigcap \rangle$ is $\subseteq$ and remembering the definitions of prefixed and postfixed points we have the following lemma.

Lemma 2.3.6. *Given $\langle \mathcal{P}(X), \subseteq, \bigcup, \bigcap \rangle$ and $\phi_{\mathcal{R}} : \mathcal{P}(X) \to \mathcal{P}(X)$ the operator induced by the set of rules $\mathcal{R}$ on $X \subseteq \mathcal{U}$, a set $Y \subseteq X$ is:*

- ***$\phi_{\mathcal{R}}$-closed*** *if and only if* $Y \in Prefix_{\phi_{\mathcal{R}}}$
- ***$\phi_{\mathcal{R}}$-dense*** *if and only if* $Y \in Postfix_{\phi_{\mathcal{R}}}$

From the Knaster-Tarski theorem it follows that we can give the next definitions.

Definition 2.3.7. *Given* $\langle \mathcal{P}(X), \subseteq, \bigcup, \bigcap \rangle$ *and* $\phi_{\mathcal{R}} : \mathcal{P}(X) \to \mathcal{P}(X)$ *the operator induced by the set of rules* $\mathcal{R}$ *on* $X \subseteq \mathcal{U}$*. The sets* $\mu X.\phi_{\mathcal{R}}(X)$ *and* $\nu X.\phi_{\mathcal{R}}(X)$ *defined as:*

$$\mu X.\phi_{\mathcal{R}}(X) \stackrel{\text{def}}{=} \bigcap \{Y \subseteq X \mid \phi_{\mathcal{R}}(Y) \subseteq Y\}$$
$$\nu X.\phi_{\mathcal{R}}(X) \stackrel{\text{def}}{=} \bigcup \{Y \subseteq X \mid Y \subseteq \phi_{\mathcal{R}}(Y)\}$$

are the ***set inductively defined*** *by* $\phi_{\mathcal{R}}$ *and the* ***set coinductively defined*** *by* $\phi_{\mathcal{R}}$*, respectively.*

Hence a corollary of Knaster-Tarski theorem applied to $\langle \mathcal{P}(X), \subseteq, \bigcup, \bigcap \rangle$ is the following.

Corollary 2.3.8. *For each* $\phi_{\mathcal{R}}$ *on* $\langle \mathcal{P}(X), \subseteq, \bigcup, \bigcap \rangle$*:*

- $\mu X.\phi_{\mathcal{R}}(X)$ *is the least fixed point of* $\phi_{\mathcal{R}}$
- $\nu X.\phi_{\mathcal{R}}(X)$ *is the gratest fixed point of* $\phi_{\mathcal{R}}$

furthermore them are respectively the ***least $\phi_{\mathcal{R}}$-closed*** *and the* ***greatest $\phi_{\mathcal{R}}$-dense*** *sets.*

By definition of set (co)inductively defined follows that associated with $\mu X.\phi_{\mathcal{R}}(X)$ and $\nu X.\phi_{\mathcal{R}}(X)$ we have two dual foundamental proof principles.

Corollary 2.3.9.

Induction $\mu X.\phi_{\mathcal{R}}(X) \subseteq X$ *if* $\phi_{\mathcal{R}}(X) \subseteq X$

Coinduction $X \subseteq \nu X.\phi_{\mathcal{R}}(X)$ *if* $X \subseteq \phi_{\mathcal{R}}(X)$

Since the set of natural number $\mathbb{N}$ can be easily defined inductively we have that the well known *mathematical induction* is its associated proof principle.

Example of an inductively defined set A simple but interesting example of inductive definition is the definition of *syntax* of *simple untyped λ-calculus*. Given a countable infinite set of *variables* $Var = \{x, y, \ldots\}$ ranged over by x, y and three special characters λ, (,) we define:

$$\lambda^* \stackrel{\text{def}}{=} \{x, y, \ldots, \lambda, (,)\}^*$$

as the set of all ordered sequences of symbols of the set $Var \cup \{\lambda, (,)\}$. We use M, N to range over λ^*. Given $\mathcal{R}$, the set of rule rules generated by the following rule schemata.

$$\frac{x \in Var}{x \in X} \qquad \frac{M \in X}{(\lambda x.M) \in X} \qquad \frac{M \in X \qquad N \in X}{(MN) \in X}$$

we have that $\mathcal{R}$ induces a monotone operator $\phi_{\mathcal{R}} : \mathcal{P}(\lambda^*) \to \mathcal{P}(\lambda^*)$ such that for each $X \in \mathcal{P}(\lambda^*)$:

$$\phi_{\mathcal{R}}(X) = \{M \in \ \lambda^* | \ \exists \ S \subseteq X \text{ such that } \frac{S}{x} \in \mathcal{R}\}$$

which can be written as:

$$\phi_{\mathcal{R}}(X) = \{x | \ x \in Var\} \cup \{(\lambda x.M) | \ x \in Var \ \& \ M \in X\} \cup \{(MN) | \ M, N \in X\}$$

We have that the set inductively defined by $\phi_{\mathcal{R}}$:

$$\Lambda \ \stackrel{\text{def}}{=} \ \mu X.\phi_{\mathcal{R}}(X) = \bigcap \{X \in \lambda^* | \ \phi_{\mathcal{R}}(X) \subseteq X\}$$

is the set of term of simple untyped λ-calculus, usually called λ-terms. We will see in the next section that Λ as some other inductively defined set can be defined through an alternative costructive method.
Associated with the definition of the syntax of this language we have a proof principle.

$$\forall \ X \in \lambda^* \ (\phi_{\mathcal{R}}(X) \subseteq X \Longrightarrow \Lambda \subseteq X)$$

If we interpret each set X as the set of element of λ^* which satisfy a predicate P on λ^*, we can read the induction principle as:

> To show that all λ-term satisfy the predicate P it suffices to show that the property P is preserved under application of the rules of $\mathcal{R}$.

A predicate which satisfy induction principle is usually referred to as **inductive predicate**.
It is possible to generalize our example to all *recursively enumerable sets*. The proof principle associated with the inductive definition of syntax of a language is usually referred in literature as **structural induction**. The term *structural* emphasizes that to prove properties about the language you must prove that the properties are preserved by the structure of syntax of the language.
It is well known that a language like λ-calculus can be defined as the grammar defined by a production in *Backus-Naur form* (BFN). For our example we have that Λ is the set of terms defined by the following grammar:

$$M \ ::= \ x \mid (\lambda x.M) \mid (MN)$$

This means that grammars defined via BNF have inductive nature. In the next section we use BNF as another kind of definition without explicitate their inductive nature, but we think that it will be clear when we use inductive definitions.

Example of a set coinductively defined An example of a coinductively defined set is the set of all *binary streams*. We define formally the set of all *real streams* $\mathbb{R}^\omega$ as:

$$\mathbb{R}^\omega \ \stackrel{\text{def}}{=} \ \{\sigma \mid \sigma : \{0, 1, 2, \ldots\} \to \mathbb{R}\}$$

On $\mathbb{R}^\omega$ we can define two useful functions. The function *head*, $hd : \mathbb{R}^\omega \to \mathbb{R}$ defined as:

$$hd(\sigma) \ \stackrel{\text{def}}{=} \ \sigma(0)$$

and the function *tail*, $tl : \mathbb{R}^\omega \to \mathbb{R}^\omega$ defined as:

$$tl(\sigma) \stackrel{\text{def}}{=} \sigma'$$

where for all n the function σ' is defined as $\sigma'(n) = \sigma(n+1)$.
We call *observations* the functions *head* and *tail*. We can now define the set $\{0,1\}^\omega$ of *binary streams* through the set of rules $\mathcal{R}$ generated by the following schemata.

$$\frac{hd(\sigma) = 0 \qquad tl(\sigma) \in X}{\sigma \in X} \qquad \frac{hd(\sigma) = 1 \qquad tl(\sigma) \in X}{\sigma \in X}$$

we have that $\mathcal{R}$ induce a monotone operator $\phi_{\mathcal{R}} : \mathcal{P}(\mathbb{R}^\omega) \to \mathcal{P}(\mathbb{R}^\omega)$ such that for each $X \in \mathcal{P}(\mathbb{R}^\omega)$:

$$\phi_{\mathcal{R}}(X) = \{\sigma |\ hd(\sigma) = 0 \ \&\ tl(\sigma) \in X\} \cup \{\sigma |\ hd(\sigma) = 1 \ \&\ tl(\sigma) \in X\}$$

The set coinductively defined by $\phi_{\mathcal{R}}$ is the following

$$\{0,1\}^\omega \stackrel{\text{def}}{=} \nu X.\phi_{\mathcal{R}}(X) = \bigcup \{X \in \mathbb{R}^\omega |\ X \subseteq \phi_{\mathcal{R}}(X)\}$$

We will see in the next section that $\{0,1\}^\omega$ as some other coinductively defined set can be defined through an alternative construction method.
Associated with $\{0,1\}^\omega$ we have a proof principle.

$$\forall\ X \in \mathbb{R}^\omega\ (X \subseteq \phi_{\mathcal{R}}(X) \Longrightarrow X \subseteq \{0,1\}^\omega)$$

Each set $X \subseteq \mathbb{R}^\omega$ can be thought of as a set of elements which satisfy a predicate on $\mathbb{R}^\omega$. We can now interpret the coinduction proof principle as:

> To show that all the elements satisfy the predicate P are elements of $\{0,1\}^\omega$ it suffices to show that the property P is preserved by the rules of $\mathcal{R}$.

A set of elements which satisfies this property is often called **invariant**. For an in-depth discussion on why they can be called invariants see [Jacobs, 1997]. The use of coinduction in the theory of streams is a recent field of research. For an extensive treatment of this and related examples see [Rutten, 2001].

2.4 Constructive definitions by set of rules

As shown clearly by Winskel in [Winskel, 1993] and Pitts in [Pitts, 1994] if $\phi_{\mathcal{R}}$ is a monotone operator induced by a *finitary* set of rules we can give an alternative constructive characterization of $\mu X.\phi_{\mathcal{R}}(X)$ through two iteration schema of $\phi_{\mathcal{R}}$ which originates from Klenee in [Kleene, 1952]. The same can be done for $\nu X.\phi_{\mathcal{R}}(X)$ if $\phi_{\mathcal{R}}$ is a monotone operator induced by a *deterministic* set of rules.
Given a monotone operator $\phi_{\mathcal{R}} : \mathcal{P}(X) \to \mathcal{P}(X)$ induced by a set of rules $\mathcal{R}$ on X and a set $S \subseteq X$, the sets defined by recursion as:

$$\begin{array}{lcl} \phi^0_{\mathcal{R}}(S) & \stackrel{\text{def}}{=} & S \\ \phi^{n+1}_{\mathcal{R}}(S) & \stackrel{\text{def}}{=} & \phi_{\mathcal{R}}(\phi^n_{\mathcal{R}}(S)) \end{array}$$

are respectively the *n-th iterates* $\phi_{\mathcal{R}}$ with *base* S.
In particular, if we take $S = \emptyset$ which is the bottom of $\langle \mathcal{P}(X), \subseteq, \bigcup, \bigcap \rangle$ we have that $\phi^0_{\mathcal{R}}(\emptyset) = \emptyset \subseteq \phi_{\mathcal{R}}(\emptyset)$ and since $\phi_{\mathcal{R}}$ is monotonic we have the following chain of inclusions:

$$\phi^0_{\mathcal{R}}(\emptyset) \subseteq \phi^1_{\mathcal{R}}(\emptyset) \subseteq \cdots \subseteq \phi^n_{\mathcal{R}}(\emptyset) \subseteq \cdots$$

and dually if we take $S = X$ as base, which is the top of $\langle \mathcal{P}(X), \subseteq, \bigcup, \bigcap \rangle$, we have:

$$\cdots \subseteq \phi^n_{\mathcal{R}}(X) \subseteq \cdots \subseteq \phi^1_{\mathcal{R}}(X) \subseteq \phi^0_{\mathcal{R}}(X)$$

Now we can prove the following lemma.

Lemma 2.4.1. *Given a monotone operator $\phi_{\mathcal{R}} : \mathcal{P}(X) \to \mathcal{P}(X)$ induced by a set of rules $\mathcal{R}$:*

(i) if $\mathcal{R}$ is finitary then $\mu X.\phi_{\mathcal{R}}(X) = \bigcup_{n \in \mathbb{N}} \phi^n_{\mathcal{R}}(\emptyset)$

(ii) if $\mathcal{R}$ is deterministic then $\nu X.\phi_{\mathcal{R}}(X) = \bigcap_{n \in \mathbb{N}} \phi^n_{\mathcal{R}}(X)$

Proof. We sometimes omit to write the superscript ($n \in \mathbb{N}$) in $\bigcap \phi^n_{\mathcal{R}}(X)$ and $\bigcup \phi^n_{\mathcal{R}}(\emptyset)$ only for clarity of notation.

(i) By lemma 2.3.4 we have that $\phi(\bigcup \phi^n_{\mathcal{R}}(X)) = \bigcup \phi^n_{\mathcal{R}}(X)$ hence it is a fixed point. By least fixed point property of $\mu X.\phi_{\mathcal{R}}(X)$ we only need to show that $\bigcup \phi^n_{\mathcal{R}}(\emptyset) \subseteq \mu X.\phi_{\mathcal{R}}(X)$.
We prove this by induction on n. Obviously $\emptyset \subseteq \mu X.\phi_{\mathcal{R}}(X)$, then assume $\phi^n_{\mathcal{R}}(\emptyset) \subseteq \mu X.\phi_{\mathcal{R}}(X)$, by monotonicity $\phi_{\mathcal{R}}(\phi^n_{\mathcal{R}}(\emptyset)) \subseteq \phi_{\mathcal{R}}(\mu X.\phi_{\mathcal{R}}(X))$ and then $\phi^{n+1}_{\mathcal{R}}(\emptyset) \subseteq \phi_{\mathcal{R}}(\mu X.\phi_{\mathcal{R}}(X)) \subseteq \mu X.\phi_{\mathcal{R}}(X)$. So we have $\bigcup \phi^n_{\mathcal{R}}(\emptyset) \subseteq \mu X.\phi_{\mathcal{R}}(X)$

(ii) By lemma 2.3.4 we have that $\phi(\bigcap \phi^n_{\mathcal{R}}(X)) = \bigcap \phi^n_{\mathcal{R}}(X)$ hence it is a fixed point.By greatest fixed point property of $\nu X.\phi_{\mathcal{R}}(X)$ we only need to show that $\nu X.\phi_{\mathcal{R}}(X) \subseteq \bigcap \phi^n_{\mathcal{R}}(\emptyset)$.
We prove this by induction on n. Obviously $\nu X.\phi_{\mathcal{R}}(X) \subseteq X$, then assume $\nu X.\phi_{\mathcal{R}}(X) \subseteq \phi^n_{\mathcal{R}}(X)$, by monotonicity $\phi_{\mathcal{R}}(\nu X.\phi_{\mathcal{R}}(X)) \subseteq \phi_{\mathcal{R}}(\phi^n_{\mathcal{R}}(X))$ and then $\nu X.\phi_{\mathcal{R}}(X) \subseteq \phi_{\mathcal{R}}(\nu X.\phi_{\mathcal{R}}(X)) \subseteq \phi^{n+1}_{\mathcal{R}}(X) \subseteq \phi^n_{\mathcal{R}}(X)$ so by induction $\forall n \in \mathbb{N}\ \nu X.\phi_{\mathcal{R}}(X) \subseteq \phi^n_{\mathcal{R}}(X)$ and then $\nu X.\phi_{\mathcal{R}}(X) \subseteq \bigcap \phi^n_{\mathcal{R}}(X)$

□

We remark that by lemma 2.4.1 we have that $\mu X.\phi_{\mathcal{R}}(X)$ is the least upper bound of the chain:

$$\phi^0_{\mathcal{R}}(\emptyset) \subseteq \phi^1_{\mathcal{R}}(\emptyset) \subseteq \cdots \subseteq \phi^n_{\mathcal{R}}(\emptyset) \subseteq \cdots$$

and dually $\nu X.\phi_{\mathcal{R}}(X)$ is the greatest lower bound of the chain:

$$\cdots \subseteq \phi^n_{\mathcal{R}}(X) \subseteq \cdots \subseteq \phi^1_{\mathcal{R}}(X) \subseteq \phi^0_{\mathcal{R}}(X)$$

Lemma 2.4.1 works only because the rule set is finitary and deterministic in the first and the second case, respectively. The result can be generalized to non finitary rule set iterating up to transfinite ordinals, see Aczel [Aczel, 1977]. Another general constructive version of Knaster-Tarski theorem is in [Cousot and Cousot, 1979] due to Cousot and Cousot.

2.5 Complete lattices of binary relations

We now specialize our definitions to an interesting example which will be useful in the sequel. We take, as complete lattice, the powerset of $X \times X$ where $X \subseteq \mathcal{U}$. The complete lattice $\langle \mathcal{P}(X \times X), \subseteq, \bigcup, \bigcap \rangle$ is the complete lattice of *binary relations* on X. From now on we do not write the subscript in ϕ.

Definition 2.5.1. *Given the complete lattice of binary relations on $X \subseteq \mathcal{U}$, $\langle \mathcal{P}(X \times X), \subseteq, \bigcup, \bigcap \rangle$ and a monotone operator $\phi : \mathcal{P}(X \times X) \to \mathcal{P}(X \times X)$ we have that:*

- $\mathcal{R} \in \mathcal{P}(X \times X)$ *is a* ***ϕ-congruence*** *if and only if* $\phi(\mathcal{R}) \subseteq \mathcal{R}$
- $\mathcal{R} \in \mathcal{P}(X \times X)$ *is a* ***ϕ-bisimulation*** *if and only if* $\mathcal{R} \subseteq \phi(\mathcal{R})$

Definition 2.5.2. *Given the complete lattice of binary relations on $X \subseteq \mathcal{U}$, $\langle \mathcal{P}(X \times X), \subseteq, \bigcup, \bigcap \rangle$ and a monotone operator $\phi : \mathcal{P}(X \times X) \to \mathcal{P}(X \times X)$ we have that:*

- $\mathcal{R} \in \mathcal{P}(X \times X)$ *is a* ***ϕ-equality*** *if and only if* $\mathcal{R} = \mu\mathcal{R}.\phi(\mathcal{R})$
- $\mathcal{R} \in \mathcal{P}(X \times X)$ *is a* ***ϕ-bisimilarity*** *if and only if* $\mathcal{R} = \nu\mathcal{R}.\phi(\mathcal{R})$

By the last definition we have that the ϕ-equality is contained in all the ϕ-congruences and dually the ϕ-bisimilarity contains all the ϕ-bisimulations. Associated with these definition we have the following proof principles.

Definition 2.5.3. *Given the complete lattice of binary relations on $X \subseteq \mathcal{U}$, $\langle \mathcal{P}(X \times X), \subseteq, \bigcup, \bigcap \rangle$ and a monotone operator $\phi : \mathcal{P}(X \times X) \to \mathcal{P}(X \times X)$ we have that:*

ϕ-induction *if $(x, y) \in \mathcal{R}$ and $\mathcal{R}$ is a ϕ-congruence then $(x, y) \in \mu\mathcal{R}.\phi(\mathcal{R})$*

ϕ-coinduction *if $(x, y) \in \mathcal{R}$ and $\mathcal{R}$ is a ϕ-bisimulation then $(x, y) \in \nu\mathcal{R}.\phi(\mathcal{R})$*

In the sequel we will study some notions of program equivalence. Hence we are interested in some additional properties of operator induced by set of rules, which permits to verify that a relation is an equivalence relation.
If we take the *equality relation* on X, $Eq \in \mathcal{P}(X \times X)$ which is defined as:

$$Eq \stackrel{\text{def}}{=} \{(x, x) \mid x \in X\}$$

we can isolate a first class of monotone operators. The following terminology is due to Crole, see [Crole, 2003].

Definition 2.5.4. *Given the complete lattice of binary relations on $X \subseteq \mathcal{U}$, $\langle \mathcal{P}(X \times X), \subseteq, \bigcup, \bigcap \rangle$ an operator $\phi : \mathcal{P}(X \times X) \to \mathcal{P}(X \times X)$ is* ***pre-extensional*** *if and only if:*

- $\phi(Eq) = Eq$
- $\phi(\mathcal{R}) \circ \phi(\mathcal{R}') \subseteq \phi(\mathcal{R} \circ \mathcal{R}')$

The following lemma show why we have isolated this class of monotone operator.

Lemma 2.5.5. *Given the complete lattice of binary relations on $X \subseteq \mathcal{U}$, $\langle \mathcal{P}(X \times X), \subseteq, \bigcup, \bigcap \rangle$ and a pre-extensional monotone operator $\phi : \mathcal{P}(X \times X) \to \mathcal{P}(X \times X)$. Then $\nu\mathcal{R}.\phi(\mathcal{R})$ is a preorder.*

Proof.

Reflexivity By definition of extensional operator we have $\phi(Eq) = Eq$ and since $\nu\mathcal{R}.\phi(\mathcal{R})$ is the greatest fixed point we have $Eq \subseteq \nu\mathcal{R}.\phi(\mathcal{R})$.

Transitivity By fixed point property $\nu\mathcal{R}.\phi(\mathcal{R}) \circ \nu\mathcal{R}.\phi(\mathcal{R})$ is equal to $\phi(\nu\mathcal{R}.\phi(\mathcal{R})) \circ \phi(\nu\mathcal{R}.\phi(\mathcal{R}))$ and by definition of pre-extensional operator $\nu\mathcal{R}.\phi(\mathcal{R}) \circ \nu\mathcal{R}.\phi(\mathcal{R}) \subseteq \phi(\nu\mathcal{R}.\phi(\mathcal{R}) \circ \nu\mathcal{R}.\phi(\mathcal{R}))$ hence $\nu\mathcal{R}.\phi(\mathcal{R}) \circ \nu\mathcal{R}.\phi(\mathcal{R})$ is a ϕ-bisimulation and by coinduction we have the conclusion.

□

Now remembering that the opposite $\mathcal{R}^{op}$ of a relation $\mathcal{R} \in \mathcal{P}(X \times X)$ is defined as:

$$\mathcal{R}^{op} \stackrel{\text{def}}{=} \{(y, x) \mid (x, y) \in \mathcal{R}\}$$

we can isolates another useful set of monotone operator.

Definition 2.5.6. *Given the complete lattice of binary relations on $X \subseteq \mathcal{U}$, $\langle \mathcal{P}(X \times X), \subseteq, \bigcup, \bigcap \rangle$ a pre-extensional operator $\phi : \mathcal{P}(X \times X) \to \mathcal{P}(X \times X)$ is **extensional** if and only if:*

- $\phi(\mathcal{R})^{op} \subseteq \phi(\mathcal{R}^{op})$

The following lemma show that the set of extensional monotone operator is the one we are looking for.

Lemma 2.5.7. *Given the complete lattice of binary relations on $X \subseteq \mathcal{U}$, $\langle \mathcal{P}(X \times X), \subseteq, \bigcup, \bigcap \rangle$ and an extensional monotone operator $\phi : \mathcal{P}(X \times X) \to \mathcal{P}(X \times X)$. Then $\nu\mathcal{R}.\phi(\mathcal{R})$ is an equivalence relation.*

Proof. Since ϕ is pre-extensional we only need to prove simmetry property.

Simmetry By fixed point property $\nu\mathcal{R}.\phi(\mathcal{R})^{op} = \phi(\nu\mathcal{R}.\phi(\mathcal{R}))^{op}$, since ϕ is extensional $\nu\mathcal{R}.\phi(\mathcal{R})^{op} = \phi(\nu\mathcal{R}.\phi(\mathcal{R}))^{op} \subseteq \phi(\nu\mathcal{R}.\phi(\mathcal{R})^{op})$ and by coinduction $\nu\mathcal{R}.\phi(\mathcal{R})^{op} \subseteq \nu\mathcal{R}.\phi(\mathcal{R})$.

□

Finally we have that an extensional operator can corresponds exactly with equality relation.

Definition 2.5.8. *Given the complete lattice of binary relations on $X \subseteq \mathcal{U}$, $\langle \mathcal{P}(X \times X), \subseteq, \bigcup, \bigcap \rangle$ an extensional operator $\phi : \mathcal{P}(X \times X) \to \mathcal{P}(X \times X)$ is **fully extensional** if and only if:*

$$\nu\mathcal{R}.\phi(\mathcal{R}) = Eq$$

equality is the largest bisimulation.

2.6 Possible future developments

We have seen so far that the least and greatest fixed points of a monotone operator on a complete lattices are respectively the set inductively and coinductively defined by the operator. It is now natural to consider other intermediate fixed point. It will be interesting to know if they can be useful in any computational situation. If so it is possible to try to characterize them in any particular way, to use them to define special sets which have associated proof principles.
An interesting question is about the existence of properties of monotone operators which permit to characterize the least(greatest) fixed point of an operator ϕ as the greatest(least) fixed point of another monotone operator ϕ' and find a general method to prove that the two definitions coincide. In particular we think that can be useful, treat the case when $\phi = \phi'$ hence we have one and only one fixed point and we can use as proof principle either induction and coinduction.
Finally we think that it is possible to find and usefully characterize, analogously to extensional operators, a class of monotone operators on the complete lattices of binary relation which assure that the least fixed point is an equivalence relation.

2.7 Guide to references

We have tried to introduce (co)inductive definitions and (co)induction in a general framework. This to be a base for the developments of the next sections. Our presentation follows in spirit those given by Gordon [Gordon, 1995], Pitts [Pitts, 1994] and more recently by Crole [Crole, 2003].
For a recent and simple account of ordered structures and fixed points see Davey and Prestley [Davey and Priestley, 2002].
The main reference for inductive definition is the foundamental paper [Aczel, 1977] by Aczel. Another interesting introduction to inductive and coinductive definitions is [Cousot and Cousot, 1992] by the Cousots.
In [Winskel, 1993] (chapter 3 and 4) Winskel explains in depth how sets of rules give rise to inductive definition of operational semantics, with associated proof principle of *rule induction*.
Induction and Coinduction are well studied in the general framework of *Categories*, which seems to be the more natural way of study these notions. We have chosen not to use the machinery of category theory only because its use does not add anything to our exposition. Nevertheless we remark that all the concepts we will introduce can be restated in the category theoretic framework. For an introduction to induction and coinduction and applications in category theory see Jacobs and Rutten in [Jacobs and Rutten, 1996], Crole in [Crole, 1998] and Aczel [Aczel, 1995].

Chapter 3

The FPC Language

> Language is a labyrinth of paths. You approach from one side and know your way about; you approach the same place from another side and no longer know your way about.
>
> Ludwig Wittgenstein. "Philosophical Investigations"

FPC is a functional deterministic language with higher order functions and with recursive types. FPC was first introduced by Plotkin in [Plotkin, 1985] and then many authors studied different versions of FPC.
Winskel in [Winskel, 1993] gives both operational and denotational semantics for an *eager* and a *lazy* version of FPC, Fiore in [Fiore, 1994] studies FPC in the framework of a denotational model based on partial maps, Gordon in [Gordon, 1995] studies *bisimilarity* for FPC and McCusker in [McCusker, 1996b] gives a fully abstract model for FPC through game semantics.
We study a **call-by-name** extension of FPC which is not trivial in the sense that it contains potentially infinite data structures built by means of recursive types.

3.1 Syntax

The **syntax** of a functional programming language specifies the structure of all possible terms of the language. Since for languages of this kind the terms are *programs*, the syntax specifies the structure of all possible programs.
We are interested in *abstract* syntax. This means that we are not concerned with the concrete details of how terms are written down as a linear sequences of symbols, e.g lexical analysis, but only with their parse tree.

Definition 3.1.1. *The set Term of raw (or untyped)* ***FPC terms*** *is given by the abstract syntax trees generated by the following grammar specified via* BNF*:*

M	::=	x	*variables*
	|	$\lambda x.M$	*lambda abstraction*
	|	(MM)	*function application*
	|	$\langle M, M\rangle$	*pairing*
	|	$\boldsymbol{fst}(M)$	*first projection*
	|	$\boldsymbol{snd}(M)$	*second projection*
	|	$\boldsymbol{inl}_{\tau_1,\tau_2}(M)$	*left injection*
	|	$\boldsymbol{inr}_{\tau_1,\tau_2}(M)$	*right injection*
	|	$\boldsymbol{case}\ M\ \boldsymbol{of}\ \{\boldsymbol{inl}_{\tau_1,\tau_2}(x_1).M \mid \boldsymbol{inr}_{\tau_1,\tau_2}(x_2).M\}$	*sum conditional*
	|	$\boldsymbol{abs}_{\mu T.\tau}(M)$	*type abstraction*
	|	$\boldsymbol{rep}(M)$	*type representation*

where $x \in Var$ is a fixed, infinite set of variables.

The above definition uses a fixed set of basic constructors. To make things easier to read we have tried to keep notations as intuitive as possible, for example we wrote (M_1M_2) for the well known function $app(M_1, M_2)$.
The informal meaning of the constructors is explained as follows. $\lambda x.M$ is a name for the function mapping x to M where M can depend on x and (MN) is the result of applying function M to the argument N. $\langle M, N\rangle$ is the ordered pair with first and second components M and N, respectively, $\mathbf{fst}(M)$ is the first component of the pair M and $\mathbf{snd}$(M) is the second component of the pair M.
The meaning of $\mathbf{case}\ M\ \mathbf{of}\ \{\mathbf{inl}_{\tau_1,\tau_2}(x_1).M_1 \mid \mathbf{inr}_{\tau_1,\tau_2}(x_2).M_2\}$ is the same of M_1 with x_1 replaced by M or M_2 with x_2 replaced by M according on whether the term M is in the left or right component of a sum, $\mathbf{inl}_{\tau_1,\tau_2}(M)$ is the mapping of a term M into the corresponding left component of a sum and analogusly $\mathbf{inr}_{\tau_1,\tau_2}(M)$ is the mapping of a term M into the corresponding right component of a sum. Finally $\mathbf{abs}_{\mu T.\tau}(M)$ and $\mathbf{rep}(M)$ are the obvious coercions from a recursive type to its unfolding and vice-versa.
We study a *call-by-name* version of FPC; at the syntactic level this differs from a *call-by-value* version only for the absence of a convergent term. We shall see that the greatest differences will be at the semantic level.
In order to work with abstract syntax we need some assumptions on the structure of terms. We assume that function application associates to the left (e.g. MNP means $(MN)P$) and λ bindings extends as far to the right as possible (e.g. $\lambda x.MN$ means $(\lambda x.MN)$ not $(\lambda x.M)N$).
We remark that we built the terms of the language starting from a fixed infinite set of variables and that function definitions and type abstractions are handled anonymously, rather than through an explicit environment-based mechanism binding identifiers to their definitions.
Some versions of FPC make available an explicit basic constructor for recursively defined terms, and a term which represents divergent computations, see for example [Winskel, 1993]. We have decided not to do so but we note that it is possible, as in type-free lambda calculus, to define a derived costructors for recursively defined terms as

$$\mathbf{Y} \stackrel{\text{def}}{=} \lambda f.(\lambda x.f(\mathbf{rep}(x)x))(\mathbf{abs}_{\mu T.\tau}(\lambda x.f(\mathbf{rep}(x)x)))$$

When $\mathbf{Y}$ is applied to a term $\lambda x.F$ we write:

$$\mathbf{rec}\ x.F \stackrel{\text{def}}{=} \mathbf{Y}\lambda x.F$$

Similarly we can define a divergent terms as:

$$\Omega \stackrel{\text{def}}{=} (\lambda x.\mathbf{rep}(x)x)(\mathbf{abs}_{\mu T.\tau}(\lambda x.\mathbf{rep}(x)x))$$

We will see in the sequel that these definition are consistent with our operational semantics.

Chapter 4

Structural Operational Semantics

> In logic, there are no morals. Everyone is at liberty to build his own logic, i.e. his own form of language, as he wishes. All that is required of him is that, if he wishes to discuss it, he must state his methods clearly, and give syntactical rules instead of philosophical arguments.
>
> Rudolf Carnap. "The formal syntax of Language"

Specifying the semantics for a programming language corresponds to defining the mapping from the syntactical constructs of the language into their *meaning* in an appropiate formal model. There are several approaches to the specification of semantics for a programming language, depending on which formal model is chosen to represent the meaning. Historically three of them emerged: *denotational*, *operational*, *axiomatic*. In this work we are concerned with the operational kind of semantics.

Operational semantics specifies the meaning of language through the description of how an abstract machine executes its constructs. The term *operational* is due to the fact that this method of specifying semantics is based on syntactic transformations of programs and simple operations on data.

Operational Semantics, because of its simplicity, emerged early in computer science, the same idea of Turing machine derives from an operational setting. In the '60 and '70 many computer scientists improved this theory, see [Jones, 2003], with the definitions of new abstract machines to study computations. With the theory emerged, also, the need for a formalization of the concepts involved in order to make this a general theory and not only a bunch of different concepts and techniques.

In 1981 in a course at Aarhus University [Plotkin, 1981], Plotkin proposed a new useful formalization of how to specify operational semantics which is now known as **Structural Operational Semantics** (SOS). For an historical account of how this idea emerged see [Plotkin, 2003].

A structural operational semantics is usually given by rules which specify in a syntax-directed manner, how programs must be executed. We quote from [Plotkin, 2003] the ideas which led Plotkin to SOS:

> The first idea,[...], was that structural operational semantics was intended as being like an abstract machine but without all the complex machinery in the configurations, just the minimum needed to explain the semantical aspects of the programming language constructs.[...] The second idea was that the rules should be syntax-directed; this is reflected in the title of the Aarhus notes: the operational semantics is structural, not, as some took it, structured.

The term *structural* refers to the fact that the application of the rules is driven by the abstract syntax of the terms. We follow Plotkin's intuition and we call syntax the abstract syntax tree of the language instead of concrete syntax.
The rules of structural operational semantics define a relation on the terms of the language (or some *configuration* which does involve such terms). In [Plotkin, 1981] Plotkin proposed many example of operational semantics based on a single step relation, this approches is called *small-step* in opposition to *big-step* which was also suggested in the same work by Plotkin but was really applied only a few years later by Kahn in [Kahn, 1987].
By an analysis of abstract-syntax of a language we can obtain many different interesting properties, particularly we can divide the properties in two class: *context-sensitive* and *context-free* properties.
We call **static semantics** the context-sensitive aspects of abstract-syntax; the word *static* emphasizes that in order to to deduce context-sensitive properties of a program we do not need to compute but only to analyze its structure and the context. Analogously we call **dynamic semantic** the context-free aspects of the abstract syntax; in opposition the word *dynamic* emphasizes that the context-free properties relate to the dynamical behaviour of the language.
The terms context-free and context-sensitive can cause some misunderstandings, so we remark that here they are referred to semantical properties and not to properties of the grammar of language. We hope that it is fairly clear that no context-free grammar by itself would suffice to capture exactly well-formed phrases of FPC.

4.1 Static Semantics

Static semantics isolates which abstract syntax trees of the language are well-formed. Static semantics can be further be decomposed into two parts: *variable scope* and *rules of typing*. Them specify how to interpret variables, and how to discern the meaningful expressions. Our presentation of variable scope partly follows in spirit the one in [Pitts, 1994] and the presentation of typing rules partly follows the one in [Pitts, 1995].

Variable scoping

We are interesting in terms where consistent renamings of the variables don't affect their meanings. In FPC λ and **case** are said to be **binding operators**; in particular if we have $\lambda x.M$, and **case** M **of** $\{\mathbf{inl}_{\tau_1,\tau_2}(x_1).M_1 \mid \mathbf{inr}_{\tau_1,\tau_2}(x_2).M_2\}$ the **bound** variables are x and x_1, x_2 respectively.
A variable x is **free** in a term M if it is not bound by a binding operator. More precisely we can define the set of free variables as follows.

Definition 4.1.1. *The set $FV(M)$ of **free variables** of M is the set inductively defined as:*

$$
\begin{array}{lcl}
FV(x) & = & x \\
FV(\lambda x.M) & = & FV(M) - \{x\} \\
FV(M_1 M_2) & = & FV(M_1) \cup FV(M_2) \\
FV(\langle M_1, M_2 \rangle) & = & FV(M_1) \cup FV(M_2) \\
FV(\boldsymbol{fst}(M)) & = & FV(\boldsymbol{snd}(M)) = FV(M) \\
FV(\boldsymbol{inl}_{\tau_1,\tau_2}(M)) & = & FV(\boldsymbol{inr}_{\tau_1,\tau_2}(M)) = FV(M) \\
FV(\boldsymbol{abs}_{\mu T.\tau}(M)) & = & FV(\boldsymbol{rep}(M)) = FV(M) \\
\multicolumn{3}{l}{FV(\boldsymbol{case}\ M\ \boldsymbol{of}\ \{\boldsymbol{inl}_{\tau_1,\tau_2}(x_1).M_1 \mid \boldsymbol{inr}_{\tau_1,\tau_2}(x_2).M_2\})} \\
\multicolumn{3}{l}{= FV(M) \cup (FV(M_1) - \{x_1\}) \cup (FV(M_2) - \{x_2\})}
\end{array}
$$

Analogously we can define the set of $Bound(M)$ of **bound variables**. A term M with $FV(M) = \emptyset$ is a **closed** term, otherwise it is an **open** term.
Now bound variables names are inessential. For this reason we treat bound variables as "nameless". This could be done literally by adopting *De Bruijn's nameless terms* but we prefer to use an easier notation at the cost of introduce new notions.

Definition 4.1.2. *Two terms M, N are said **α-equivalent**, written $M \equiv_\alpha N$ if they are syntactically identical up to renaming of bound variables.*

To give a more formal definition of α-equivalence we need term substitution.

Definition 4.1.3. *The **term substitution** of a term N for a variable x in a term M, written $M[N/x]$, is inductively defined as follows:*

$$
\begin{array}{lcll}
x[N/x] & = & N & \\
y[N/x] & = & y & \text{where } y \not\equiv x \\
(\lambda y.M)[N/x] & = & \lambda y.(M[N/x]) & \text{where } y \not\equiv x \text{ and } y \notin FV(N) \\
(M_1 M_2)[N/x] & = & (M_1[N/x])(M_2[N/x]) & \\
\langle M_1, M_2 \rangle[N/x] & = & \langle M_1[N/x], M_2[N/x] \rangle & \\
\boldsymbol{fst}(M)[N/x] & = & \boldsymbol{fst}(M[N/x]) & \\
\boldsymbol{snd}(M)[N/x] & = & \boldsymbol{snd}(M[N/x]) & \\
\boldsymbol{inl}_{\tau_1,\tau_2}(M)[N/x] & = & \boldsymbol{inl}_{\tau_1,\tau_2}(M[N/x]) & \\
\boldsymbol{inr}_{\tau_1,\tau_2}(M)[N/x] & = & \boldsymbol{inr}_{\tau_1,\tau_2}(M[N/x]) & \\
\boldsymbol{abs}_{\mu T.\tau}(M)[N/x] & = & \boldsymbol{abs}_{\mu T.\tau}(M[N/x])) & \\
\boldsymbol{rep}(M)[N/x] & = & \boldsymbol{rep}(M[N/x])) &
\end{array}
$$

$$
\begin{array}{l}
\boldsymbol{case}\ M\ \boldsymbol{of}\ \{\boldsymbol{inl}_{\tau_1,\tau_2}(x_1).M_1 \mid \boldsymbol{inr}_{\tau_1,\tau_2}(x_2).M_2\}[N/x] \\
= \boldsymbol{case}\ M[N/x]\ \boldsymbol{of}\ \{\boldsymbol{inl}_{\tau_1,\tau_2}(x_1).M_1[N/x] \mid \boldsymbol{inr}_{\tau_1,\tau_2}(x_2).M_2[N/x]\} \\
\text{where } x \not\equiv x_1,\ x \not\equiv x_2 \text{ and } x_1, x_2 \notin FV(N)
\end{array}
$$

Note that with this definition, substitution is a partial operation. We have defined single term substitution, but when we need to perform several substitutions we use a notation for *multiple substitution*. We write $M[N_1/x_1, \cdots, N_n/x_n]$ or $M[\vec{N}/\vec{x}]$ for substitution of N_1 for x_1, ..., N_n for x_n in M, we think it is clear how to perform multiple substitution as a composition of single term substitutions.
We can now define inductively α-equivalence. The operator $\phi_\alpha : \mathcal{P}(Term \times Term) \to \mathcal{P}(Term \times Term)$ induced by the set of instances of rules in table 4.1

is defined as:

$$\phi_\alpha(X) \stackrel{\text{def}}{=} \{(M,N) \mid \exists \ \frac{M_1 \equiv_\alpha M_1' \cdots M_n \equiv_\alpha M_n'}{M \equiv_\alpha N}, (n \geq 0) \\ \& \ (M_1, M_1'), \cdots, (M_n, M_n') \in X\}$$

It is easy to verify that ϕ_α is monotonic. Hence it posseses a least fixed point $\mu X.\phi_\alpha(X)$, since rules of table 4.1 are all *finitary* it can be constructively defined with respect to $\subseteq$ and with $\emptyset$ as basis.

Definition 4.1.4. ***α-equivalence*** *is the relation* $\equiv_\alpha \subseteq \ Term \times Term$ *inductively defined as:*

$$\equiv_\alpha \ = \ \mu X.\phi_\alpha(X)$$

With this definition we have obtained the inessentiality of bound variables. For example $\lambda x.x \equiv_\alpha \lambda y.y$. From now on we shall consider FPC terms up to renaming of bound variables. In the sequel we will not make a notational distinction between a FPC syntax tree and the α-equivalence class of the term it determines. This requires that all definitions which involve terms are invariant under α-equivalence.
An example of that is substitution. We have seen that our definition of substitution is a partial operation on *Term* but if we identify syntax trees with the α-equivalence classes of terms they determines, it becomes a total operation. For example the substitution $(\lambda x.xz)[zx/z]$ which is undefined because of the capture of the free variable x can be performed if we change $\lambda x.xz$ with an α-equivalent term $\lambda y.yz$. They identify the same term, so the change doesn't affect the meaning and we have $(\lambda y.yz)[zx/z] \equiv \lambda y.y(zx)$.
It is easy to prove by induction the following lemma which states the invariance of substitution under α-equivalence:

Lemma 4.1.5. *Let* $M \equiv_\alpha M'$, $x \in FV(M)$ *and* $N \equiv_\alpha N'$ *then :*

$$M[N/x] \equiv_\alpha M'[N'/x]$$

So our definition of substitution preserves α-equivalence in the sense that substitution is *capture avoiding*, free variable of N cannot be bound in $M[N/x]$. In the sequel we write *Term* for the set of α-equivalence class of terms. Furthermore we write simply $M \equiv N$ for syntactical equivalence up to α-equivalence.

Typing rules

FPC is a typed language: this means that all expressions of FPC can assume values only in a fixed range (depending on their type) during computations. We have chosen to leave out type informations in the definition of the language, but they are implicitly or explicitly present in a valid expression. The set *Type* of **FPC-types** is given by the following grammar:

$\tau ::=$	T	type variable
$\mid$	$\tau \to \tau'$	function type
$\mid$	$\tau \times \tau'$	product type
$\mid$	$\tau + \tau'$	sum type
$\mid$	$\mu T.\tau$	recursive type

$$\frac{}{M \equiv_\alpha M} \quad (\alpha\text{-reflexivity})$$

$$\frac{N \equiv_\alpha M}{M \equiv_\alpha N} \quad (\alpha\text{-simmetry})$$

$$\frac{M \equiv_\alpha M' \quad M' \equiv_\alpha N}{M \equiv_\alpha N} \quad (\alpha\text{-transitivity})$$

$$\frac{y \notin FV(M)}{\lambda x.M \equiv_\alpha \lambda y.M[y/x]} \quad (\alpha\text{-lambda})$$

$$\frac{}{\begin{array}{l}\mathbf{case}\ M\ \mathbf{of}\ \{\mathbf{inl}_{\tau_1,\tau_2}(x).M_1[x/x_1] \mid \mathbf{inr}_{\tau_1,\tau_2}(y).M_2[y/x_2]\} \\ \quad \equiv_\alpha \mathbf{case}\ M\ \mathbf{of}\ \{\mathbf{inl}_{\tau_1,\tau_2}(x_1).M_1 \mid \mathbf{inr}_{\tau_1,\tau_2}(x_2).M_2\}\end{array}} \quad (\alpha\text{-case})$$

$$\frac{M \equiv_\alpha M' \quad N \equiv_\alpha N'}{MN \equiv_\alpha M'N'} \quad (\alpha\text{-app})$$

$$\frac{M_1 \equiv_\alpha M_1' \quad M_2 \equiv_\alpha M_2'}{\langle M_1, M_2\rangle \equiv_\alpha \langle M_1', M_2'\rangle} \quad (\alpha\text{-pairs})$$

$$\frac{M \equiv_\alpha M'}{\mathbf{fst}(M) \equiv_\alpha \mathbf{fst}(M')} \quad (\alpha\text{-fst})$$

$$\frac{M \equiv_\alpha M'}{\mathbf{snd}(M) \equiv_\alpha \mathbf{snd}(M')} \quad (\alpha\text{-snd})$$

$$\frac{M \equiv_\alpha M'}{\mathbf{inl}_{\tau_1,\tau_2}(M) \equiv_\alpha \mathbf{inl}_{\tau_1,\tau_2}(M')} \quad (\alpha\text{-inl})$$

$$\frac{M \equiv_\alpha M'}{\mathbf{inr}_{\tau_1,\tau_2}(M) \equiv_\alpha \mathbf{inr}_{\tau_1,\tau_2}(M')} \quad (\alpha\text{-inr})$$

$$\frac{M \equiv_\alpha M'}{\mathbf{rep}(M) \equiv_\alpha \mathbf{rep}(M')} \quad (\alpha\text{-rep})$$

$$\frac{M \equiv_\alpha M'}{\mathbf{abs}_{\mu T.\tau}(M)\rangle \equiv_\alpha \mathbf{abs}_{\mu T.\tau}(M')} \quad (\alpha\text{-abs})$$

$$\frac{M \equiv_\alpha M' \quad x \in FV(M)}{\lambda x.M \equiv_\alpha \lambda x.M'} \quad (\alpha\text{-cong-lambda})$$

$$\frac{M \equiv_\alpha M' \quad M_1 \equiv_\alpha M_1' \quad x_1 \in FV(M_1) \quad M_2 \equiv_\alpha M_2' \quad x_2 \in FV(M_2)}{\begin{array}{l}\mathbf{case}\ M\ \mathbf{of}\ \{\mathbf{inl}_{\tau_1,\tau_2}(x_1).M_1 \mid \mathbf{inr}_{\tau_1,\tau_2}(x_2).M_2\} \equiv_\alpha \\ \quad \mathbf{case}\ M'\ \mathbf{of}\ \{\mathbf{inl}_{\tau_1,\tau_2}(x_1).M_1' \mid \mathbf{inr}_{\tau_1,\tau_2}(x_2).M_2'\}\end{array}} \quad (\alpha\text{-cong-case})$$

Table 4.1: α-equivalence rules

$$\frac{}{\Gamma \vdash x : \tau} \quad \text{if } x \text{ is defined in } \Gamma \text{ with type } \tau \qquad (\vdash \text{Var})$$

$$\frac{\Gamma, x : \tau \vdash M : \tau_1}{\Gamma \vdash \lambda x.M : \tau \to \tau_1} \qquad (\vdash \text{lam})$$

$$\frac{\Gamma \vdash M : \tau \to \tau_1 \qquad \Gamma \vdash M_1 : \tau}{\Gamma \vdash M M_1 : \tau_1} \qquad (\vdash \text{app})$$

$$\frac{\Gamma \vdash M_1 : \tau \qquad \Gamma \vdash M_2 : \tau_1}{\Gamma \vdash \langle M_1, M_2 \rangle : \tau \times \tau_1} \qquad (\vdash \text{pair})$$

$$\frac{\Gamma \vdash M : \tau \times \tau_1}{\Gamma \vdash \mathbf{fst}(M) : \tau} \qquad (\vdash \text{fst})$$

$$\frac{\Gamma \vdash M : \tau \times \tau_1}{\Gamma \vdash \mathbf{snd}(M) : \tau_1} \qquad (\vdash \text{snd})$$

$$\frac{\Gamma \vdash M : \tau_1}{\Gamma \vdash \mathbf{inl}_{\tau_1,\tau_2}(M) : \tau_1 + \tau_2} \qquad (\vdash \text{inl})$$

$$\frac{\Gamma \vdash M : \tau_2}{\Gamma \vdash \mathbf{inr}_{\tau_1,\tau_2}(M) : \tau_1 + \tau_2} \qquad (\vdash \text{inr})$$

$$\frac{\Gamma \vdash M : \tau_1 + \tau_2 \qquad \Gamma, x_1 : \tau_1 \vdash M_1 : \tau \qquad \Gamma, x_2 : \tau_2 \vdash M_2 : \tau}{\Gamma \vdash \mathbf{case}\ M\ \mathbf{of}\ \{\mathbf{inl}_{\tau_1,\tau_2}(x_1).M_1 \mid \mathbf{inr}_{\tau_1,\tau_2}(x_2).M_2\} : \tau} \qquad (\vdash \text{case})$$

$$\frac{\Gamma \vdash M : \tau[\mu T.\tau/T]}{\Gamma \vdash \mathbf{abs}_{\mu T.\tau}(M) : \mu T.\tau} \qquad (\vdash \text{abs})$$

$$\frac{\Gamma \vdash M : \mu T.\tau}{\Gamma \vdash \mathbf{rep}(M) : \tau[\mu T.\tau/T]} \qquad (\vdash \text{rep})$$

Table 4.2: FPC typing rules

We have no *ground type* instead we have a fixed infinite set of type variables. We can build *derived types* by using type constructors; in particular $\rightarrow$ denotes the function type constructor, $\times$ the cartesian product, $+$ the separated sum and μT the recursive type constructor.
Our choice of ground type and type constructors depend from the fact that we are interested in a call-by-name version of FPC.
μT is a *binding* constructor, we can define the set of *free type variables* for a type τ as the set of variables of τ which are not in the scope of a binder. As usual *open types* are types with occurrences of some free type variables and analogously *closed types* are types without free variables occurrences.
An FPC **typing judgement** takes the form

$$\Gamma \vdash M : \tau$$

where Γ is a finite function from variables to types and $M : \tau$ is a *type assertion*. The set $TEnv$ of all finite partial function from variable to types defined as:

$$TEnv = \{\Gamma \mid \Gamma : Var \rightharpoonup Type\}$$

and can be seen as the set of all possible *type environments*, hence we read a type judgement $\Gamma \vdash M : \tau$ as "in the environment Γ, M is a term of type τ". We want to isolate those term which are *well formed*, this means that we consider a term only if it can be assigned a type.
Now we are able to define inductively the *type assignment relation*: fixed $\Gamma \in TEnv$ define the operator $\phi_\Gamma : \mathcal{P}(Term \times Type) \rightarrow \mathcal{P}(Term \times Type)$ induced by the set of instances of typing rules in table 4.2 as:

$$\begin{aligned}\phi_\Gamma(X) \stackrel{\text{def}}{=} \{(M,\tau) \mid \exists\ & \frac{\Gamma_1 \vdash M_1 : \tau_1 \cdots \Gamma_n \vdash M_n : \tau_n}{\Gamma \vdash M : \tau}, (n \geq 0) \\ & \&\ (M_1,\tau_1), \cdots, (M_n,\tau_n) \in X\}\end{aligned}$$

We can easily see that this operator is monotonic, hence it possesses a least fixed point $\mu X.\phi_\Gamma(X)$ and since rules of table 4.2 are all *finitary* we can define it constructively with respect to $\subseteq$ and with $\emptyset$ as basis.

Definition 4.1.6. *For each $\Gamma \in TEnv$, the **type assignment relation for Γ** is the relation $\Theta_\Gamma \subseteq Term \times Type$ inductively defined as:*

$$\Theta_\Gamma = \mu X.\phi_\Gamma(X)$$

We finally can define

Definition 4.1.7. ***Type assignment relation** is the relation $\Theta \subseteq TEnv \times Term \times Type$ defined as:*

$$\Theta = \{(\Gamma, M, \tau) \mid \Gamma \in TEnv\ \&\ (M,\tau) \in \Theta_\Gamma\}$$

We say that a type judgement $\Gamma \vdash M : \tau$ is **valid** if and only if $(\Gamma, M, \tau) \in \Theta$. When we write $(\Gamma \vdash M : \tau) \in \Theta$ we mean that $(\Gamma, M, \tau) \in \Theta$ and in particular that $(M,\tau) \in \Theta_\Gamma$.
We write $dom(\Gamma)$ when we want to refer to the domain of the function $\Gamma : Var \rightharpoonup Type$ and $\Gamma, x : \tau$ for the extension of the function Γ by mapping x to τ, with the assumption that $x \notin dom(\Gamma)$. If $dom(\Gamma) = \{x_i \mid 1 \leq i \leq n\}$ and for all

i we have $\Gamma(x_i) = \tau_i$ than we can write $\Gamma \vdash M : \tau$ as $x_1 : \tau_1, \ldots, x_n : \tau_n \vdash M : \tau$. Similarly, given $\Gamma, \Gamma' \in TEnv$ we write Γ, Γ' for the union of the partial functions Γ and Γ' under the assumption that $dom(\Gamma) \cap dom(\Gamma') = \emptyset$.
We can now state some useful properties of the type assignment relation.

Lemma 4.1.8 (Inversion).

1. *If* $\Gamma \vdash x : \tau$ *then* $\Gamma(x) = \tau$
2. *If* $\Gamma \vdash \lambda x.M : \tau \to \tau_1$ *then* $\Gamma, x : \tau \vdash M : \tau_1$
3. *If* $\Gamma \vdash MN : \tau_1$ *then there exists* τ *such that* $\Gamma \vdash M : \tau \to \tau_1$ *and* $\Gamma \vdash N : \tau$
4. *If* $\Gamma \vdash \langle M_1, M_2 \rangle : \tau \times \tau_1$ *then* $\Gamma \vdash M_1 : \tau$ *and* $\Gamma \vdash M_2 : \tau_1$
5. *If* $\Gamma \vdash \mathbf{fst}(M) : \tau$ *then there exists* τ_1 *such that* $\Gamma \vdash M : \tau \times \tau_1$
6. *If* $\Gamma \vdash \mathbf{snd}(M) : \tau$ *then there exists* τ_1 *such that* $\Gamma \vdash M : \tau_1 \times \tau$
7. *If* $\Gamma \vdash \mathbf{inl}_{\tau_1,\tau_2}(M) : \tau_1 + \tau_2$ *then* $\Gamma \vdash M : \tau_1$
8. *If* $\Gamma \vdash \mathbf{inr}_{\tau_1,\tau_2}(M) : \tau_1 + \tau_2$ *then* $\Gamma \vdash M : \tau_2$
9. *If* $\Gamma \vdash \mathbf{case}\ M\ \mathbf{of}\ \{\mathbf{inl}_{\tau_1,\tau_2}(x_1).M_1 \mid \mathbf{inr}_{\tau_1,\tau_2}(x_2).M_2\} : \tau$ *then* $\Gamma \vdash M : \tau_1 + \tau_2$, $\Gamma, x_1 : \tau_1 \vdash M_1 : \tau$ *and* $\Gamma, x_2 : \tau_2 \vdash M_2 : \tau$
10. *If* $\Gamma \vdash \mathbf{abs}_{\mu T.\tau}(M) : \mu T.\tau$ *then* $\Gamma \vdash M : \tau[\mu T.\tau/T]$
11. *If* $\Gamma \vdash \mathbf{rep}(M) : \tau[\mu T.\tau/T]$ *then* $\Gamma \vdash M : \mu T.\tau$

Proof. All cases are obvious because in each case exactly one rule applies. □

Lemma 4.1.9.

(i) *If* $\Gamma \vdash M : \tau$ *then* $FV(M) \subseteq dom(\Gamma)$

(ii) *If* $\Gamma \vdash M : \tau$ *then for any* $\Gamma' \in TEnv$ *such that* $dom(\Gamma) \subseteq dom(\Gamma')$ *and for each* $x \in (dom(\Gamma') - dom(\Gamma))$, $x \notin Bound(M)$ *we have* $\Gamma' \vdash M : \tau$

(iii) *If* $\Gamma, \Gamma' \vdash M : \tau$ *and* $FV(M) \subseteq dom(\Gamma)$ *then* $\Gamma \vdash M : \tau$

Proof.

(i) We prove it by rule induction on the derivation of $\Gamma \vdash M : \tau$. We show only the case $\lambda x.M$, the others are similar.
If $\Gamma \vdash \lambda x.M : \tau \to \tau_1$ then by lemma 4.1.8 $\Gamma, x : \tau_1 \vdash M : \tau$. By induction hypothesis $FV(M) \subseteq dom(\Gamma) \cup \{x\}$, we know that $FV(\lambda x.M) = FV(M) - \{x\} \subseteq (dom(\Gamma) \cup \{x\}) - \{x\}$, hence $FV(\lambda x.M) \subseteq dom(\Gamma)$

(ii) We prove it by rule induction on the derivation of $\Gamma \vdash M : \tau$. We show only the case $\lambda x.M$, the others are similar.
If $\Gamma \vdash \lambda x.M : \tau_1 \to \tau$ then by lemma 4.1.8 we have $\Gamma, x : \tau_1 \vdash M : \tau$. By induction hypothesis for each Γ' such that $dom(\Gamma) \cup \{x\} \subseteq dom(\Gamma')$ and for each $x \in (dom(\Gamma) - dom(\Gamma'))$ then $x \notin Bound(M)$, we have $\Gamma' \vdash M : \tau$ but in particular if we write Γ_1 for the function Γ' restricted to $dom(\Gamma') - \{x\}$ we have $\Gamma_1, x : \tau_1 \vdash M : \tau$ and by ($\vdash$ lam) $\Gamma_1 \vdash \lambda x.M : \tau_1 \to \tau$.

(iii) We prove it by rule induction on the derivation of $\Gamma \vdash M : \tau$. We show only the case $\lambda x.M$, the others are similar.
If $\Gamma, \Gamma' \vdash \lambda x.M : \tau \to \tau'$ then by lemma 4.1.8 we have $\Gamma, \Gamma', x : \tau \vdash M : \tau'$. Since $FV(\lambda x.M) = FV(M) - \{x\}$ and $FV(\lambda x.M) \subseteq dom(\Gamma)$ we have $FV(M) \subseteq dom(\Gamma) \cup \{x\}$ and so by induction hypothesis $\Gamma, x : \tau \vdash M : \tau'$ and by ($\vdash$ lam) $\Gamma \vdash \lambda x.M : \tau \to \tau'$

□

It follows from (i) that if $\emptyset \vdash M : \tau$ then M is a closed term.

Lemma 4.1.10.

(i) *If* $\Gamma \vdash M : \tau$ *and* $\Gamma \vdash M : \tau_1$ *then* $\tau = \tau_1$

(ii) *If* $\Gamma, x : \tau_1 \vdash M : \tau$ *and* $\Gamma \vdash N : \tau_1$ *then* $\Gamma \vdash M[N/x] : \tau$

(iii) *If* $\Gamma \vdash N_1 : \tau_1, \ldots, \Gamma \vdash N_n : \tau_n$ *and* $\Gamma, x_1 : \tau_1, \ldots, x_n : \tau_n \vdash M : \tau$ *then* $\Gamma \vdash M[N_1/x_1, \cdots, N_n/x_n] : \tau$

Proof.

(i) We prove it by rule induction on the derivation of $\Gamma \vdash M : \tau$. We show only the case (MN), the others are similar.
Suppose $\Gamma \vdash MN : \tau$ and $\Gamma \vdash MN : \tau_1$, hence by lemma 4.1.8 we have that $\Gamma \vdash M : \alpha \to \tau$ and $\Gamma \vdash N : \alpha$ for the former and $\Gamma \vdash M : \alpha_1 \to \tau_1$ and $\Gamma \vdash N : \alpha_1$ for the latter. By induction hypothesis $\alpha \to \tau = \alpha_1 \to \tau_1$ and $\alpha = \alpha_1$ and hence $\tau = \tau_1$.

(ii) We prove it by rule induction on the derivation of $\Gamma, x : \tau_1 \vdash M : \tau$. We show only the case $\lambda y.M$, the others are similar.
Suppose $\Gamma, x : \tau_1 \vdash \lambda y.M : \tau \to \tau'$, hence by lemma 4.1.8 we have that $\Gamma, x : \tau_1, y : \tau \vdash M : \tau'$ and $y \notin dom(\Gamma \cup \{x\})$. By induction hypothesis for N such that $\Gamma \vdash N : \tau_1$ we have $\Gamma, y : \tau \vdash M[N/x] : \tau'$ and ($\vdash$ lam) give us $\Gamma \vdash \lambda y.M[N/x] : \tau \to \tau'$. Finally the definition of substitution $\Gamma \vdash (\lambda y.M)[N/x] : \tau \to \tau'$.

(iii) It follows immediately by applying (ii) repeatedly.

□

Now we can isolate the well formed term of FPC.

Definition 4.1.11. *Let* $Exp_\tau(\Gamma)$ *be the set of terms of type* τ *in* Γ *defined as:*

$$Exp_\tau(\Gamma) \stackrel{def}{=} \{M \in Term \mid (\Gamma \vdash M : \tau) \in \Theta\}$$

Hence we say that a term M is **typable** if and only if there exist $\Gamma \in TEnv$ and $\tau \in Type$ such that $M \in Exp_\tau(\Gamma)$. We usually will write Exp_τ instead of $Exp_\tau(\emptyset)$ and we think it is clear what we mean. Finally we can isolate the set of programs, executable terms of the language.

Definition 4.1.12. *The set of* ***programs*** *of FPC is the set Exp defined as follows:*

$$Exp \stackrel{def}{=} \{M \in Term \mid \exists\, \tau \in Type,\ M \in Exp_\tau(\emptyset)\}$$

Hence FPC **programs**, are closed and typable FPC terms.

Example of typing As examples of typable terms we prove that the derived constructor $\mathbf{Y}$ and the term Ω are typable. We have defined $\mathbf{Y}$ as:

$$\lambda f.(\lambda x.f(\mathbf{rep}(x)x))(\mathbf{abs}_{\mu\alpha.\alpha\to\tau}(\lambda x.f(\mathbf{rep}(x)x)))$$

so we can first prove that $\lambda x.f(\mathbf{rep}(x)x)$ is typable:

$$\dfrac{x:\mu\alpha.\alpha\to\tau \qquad \dfrac{f:\tau\to\tau \qquad \dfrac{\dfrac{x:\mu\alpha.\alpha\to\tau}{\mathbf{rep}(x):(\mu\alpha.\alpha\to\tau)\to\tau} \quad x:\mu\alpha.\alpha\to\tau}{\mathbf{rep}(x)x:\tau}}{f(\mathbf{rep}(x)x):\tau}}{\lambda x.f(\mathbf{rep}(x)x):(\mu\alpha.\alpha\to\tau)\to\tau}$$

and so we can prove that $\mathbf{Y}$ is typable as follows:

$$\dfrac{f:\tau\to\tau \qquad \dfrac{\lambda x.f(\mathbf{rep}(x)x):(\alpha')\to\tau \quad \dfrac{\lambda x.f(\mathbf{rep}(x)x):(\alpha')\to\tau}{\mathbf{abs}_{\alpha'}(\lambda x.f(\mathbf{rep}(x)x)):\alpha'}}{(\lambda x.f(\mathbf{rep}(x)x))(\mathbf{abs}_{\alpha'}(\lambda x.f(\mathbf{rep}(x)x))):\tau}}{\lambda f.(\lambda x.f(\mathbf{rep}(x)x))(\mathbf{abs}_{\alpha'}(\lambda x.f(\mathbf{rep}(x)x))):(\tau\to\tau)\to\tau}$$

It is clear that we have used α' as abbreviation for the recursively defined type $\mu\alpha.\alpha\to\tau$. Therefore we have $\emptyset\vdash\mathbf{Y}:(\tau\to\tau)\to\tau$ where τ represents any type, hence we can define a family of derived costructors $\mathbf{Y}^\tau$, one for each $\tau\in \mathit{Type}$. From the above proof we have that the following rule usually given for **rec** as a basic constructor of FPC, for example in [Winskel, 1993]:

$$\dfrac{\Gamma,x:\tau\vdash F:\tau}{\Gamma\vdash\mathbf{rec}\ x.F:\tau}$$

is equivalent to the rule for **rec** when it is a derived constructor:

$$\dfrac{\Gamma\vdash\mathbf{Y}:(\tau\to\tau)\to\tau \quad \dfrac{\Gamma,x:\tau\vdash F:\tau}{\Gamma\vdash\lambda x.F:\tau\to\tau}}{\Gamma\vdash\mathbf{rec}\ x.F:\tau}$$

Barendregt in [Barendregt, 1992] gives an exaustive explanation of the proof for a version of lambda calculus without an explicit constructor for recursive types. Similarly we can prove that Ω defined as:

$$(\lambda x.\mathbf{rep}(x)x)(\mathbf{abs}_{\mu\alpha.\alpha\to\tau}(\lambda x.\mathbf{rep}(x)x))$$

is typable. We first prove that $(\lambda x.\mathbf{rep}(x)x)$ is typable as follows.

$$\dfrac{x:\mu T.T\to\tau \qquad \dfrac{\dfrac{x:\mu T.T\to\tau}{\mathbf{rep}(x):(\mu T.T\to\tau)\to\tau} \quad x:\mu T.T\to\tau}{\mathbf{rep}(x)x:\tau}}{(\lambda x.\mathbf{rep}(x)x):(\mu T.T\to\tau)\to\tau}$$

Now we can show how to type Ω.

$$\dfrac{(\lambda x.\mathbf{rep}(x)x):(\mu T.T\to\tau)\to\tau \quad \dfrac{(\lambda x.\mathbf{rep}(x)x):(\mu T.T\to\tau)\to\tau}{\mathbf{abs}_{\mu T.T\to\tau}(\lambda x.\mathbf{rep}(x)x):\mu T.T\to\tau}}{(\lambda x.\mathbf{rep}(x)x)\mathbf{abs}_{\mu T.T\to\tau}(\lambda x.\mathbf{rep}(x)x):\tau}$$

Hence we have $\emptyset\vdash\Omega:\tau$ and since τ is any type we have that we can define a family of divergent terms Ω^τ, one for each $\tau\in \mathit{Type}$.

4.2 Dynamic Semantics

Dynamic semantics, in an operational framework, describes how a program must be executed. In order to specify an operational semantics we need a kind of relation to relate terms in a computation. Depending on the abstraction level of specification, one can choose beetween different relations, but all of them must specify how the computations shall be performed.
During the last two decades emerged two approaches which became the standard ways to express operational semantics: *big-step* and *small-step*. They are both example of structural operational semantics.
The *big-step* approach was pionereed in 1979 by Martin-Löf see [Martin-Löf, 1982], was indipendently suggested by Plotkin in his notes [Plotkin, 1981] but became really popular only after the work of Kahn [Kahn, 1987], who called it *natural semantics*. Big-step semantics is intended to capture the meaning of an entire computation. We give the rules for two big-step relations, one for the semantics of convergent terms (*evaluation*), and one for semantics of divergent ones (*divergence*). Evaluation is also known as *natural semantics* [Kahn, 1987] or *big-step relation*.
The *small-step* approach was firstly formalized by Plotkin in [Plotkin, 1981], which is a good reference to different examples of its application. This approach uses a relation between terms which represents all the feasible single steps in the computational framework. We introduce one small-step relation *transition*, also known as *reduction* or *small-step relation*. We show that with transition relations we are able to capture the semantics of both convergent and divergent terms.
Remark. Since in what follows we will only deal with well typed closed terms (i.e. *Exp*) we usually omit type information; to make our treatment work we will prove that our relations respects typing.

4.2.1 Big-Step

In specifying operational semantics through the big-step approach we use relations which represent entire computations from terms. We firstly introduce rules to define inductively a (big-step) *evaluation* on FPC terms which captures entire convergent computations from terms to values. The rules we introduce are now accepted as standard for convergent computation.
Fiore in [Fiore, 1994] gives an operational semantics based on big-step approach for the call-by-value FPC. Winskel in [Winskel, 1993], Gunter in [Gunter, 1992] and McCusker in [McCusker, 1996b] give an operational semantics for a call-by-name version of FPC through an evaluation relation. Our presentation differs from the latters only in form rather then content, because we want to emphasize the role of inductive definition in the evaluation relation.
We then introduce the rules to define coinductively a big-step relation *divergence* on FPC terms, which captures entire divergent computations. Similar definition of divergence are already given for different languages. See for an example [Pitts, 1994] where a big-step coinductive definition of PCF language is given. Our presentation of divergence follows the line of the one for evaluation, in order to emphasize the role of coinductive definition.
We follow the presentation of operational semantics given by Moran in [Moran, 1994] for a non deterministic version of lambda calculus.

Evaluation

$$\frac{}{\lambda x.M \Downarrow \lambda x.M} \quad (+ \text{ lambda})$$

$$\frac{M_1 \Downarrow \lambda x.M \qquad M[M_2/x] \Downarrow C}{M_1 M_2 \Downarrow C} \quad (+ \text{ app})$$

Divergence

$$\frac{M_1 \Uparrow}{M_1 M_2 \Uparrow} \quad (- \text{ app})$$

$$\frac{M_1 \Downarrow \lambda x.M \qquad M[M_2/x] \Uparrow}{M_1 M_2 \Uparrow} \quad (- \text{ app lambda})$$

Reduction

Basic Reduction

$$(\lambda x.M)M_2 \longrightarrow M[M_2/x] \quad (\text{app tran})$$

Simplification Reduction

$$\frac{M \longrightarrow M'}{M M_1 \longrightarrow M' M_1} \quad (\text{app simpl})$$

Table 4.3: Function application - (call by name)

Evaluation We are interested in defining a binary relation in order to capture the relation of convergence of computations from a term to a value.
We firstly introduce the set $Can \subseteq Exp$ of terms of the language in **canonical form**. Terms in canonical form or *values* are defined by the following productions:

$$C ::= \lambda x.M \mid \langle M, M \rangle \mid \mathbf{inl}(M) \mid \mathbf{inr}(M) \mid \mathbf{abs}_{\mu T.\tau}(C)$$

we let C range over Can, note that the property to be a term in canonical form or not depends only from the outermost operators, this beacuse we have chosen a *lazy* evaluation. Can is the set of terms which represent the *values* of the language. We sometimes call the terms in canonical form, *canonical terms* or *values* but we think it is clear what we mean.
Restating our request, we are interested in a binary relation to capture only the computations from general terms which reach terms in canonical form.
Evaluation will be defined as a relation $\Downarrow \subseteq Exp \times Can$ through rules of tables 4.3-4.6. We note that rules (+ lambda), (+ pair), (+ inl), (+ inr) and (+ abs can) are all instances of the rule:

$$C \Downarrow C \qquad (+ \text{ can})$$

We read $M \Downarrow C$ as "the term M evaluates to term C". Using a logical view, evaluation relation can be considered as a kind of *convergence judgement*, when we write $M \Downarrow C$ this means that "the computation starting from expression M converges to canonical form C".
Since FPC is a deterministic sequential language we can consider evaluation

Evaluation

$$\frac{}{\langle M_1, M_2\rangle \Downarrow \langle M_1, M_2\rangle} \quad (+ \text{ pair})$$

$$\frac{M \Downarrow \langle M_1, M_2\rangle \qquad M_1 \Downarrow C}{\mathbf{fst}(M) \Downarrow C} \quad (+ \text{ fst})$$

$$\frac{M \Downarrow \langle M_1, M_2\rangle \qquad M_2 \Downarrow C}{\mathbf{snd}(M) \Downarrow C} \quad (+ \text{ snd})$$

Divergence

$$\frac{M \Uparrow}{\mathbf{fst}(M) \Uparrow} \quad (- \text{ fst})$$

$$\frac{M \Downarrow \langle M_1, M_2\rangle \qquad M_1 \Uparrow}{\mathbf{fst}(M) \Uparrow} \quad (- \text{ fst pair})$$

$$\frac{M \Uparrow}{\mathbf{snd}(M) \Uparrow} \quad (- \text{ snd})$$

$$\frac{M \Downarrow \langle M_1, M_2\rangle \qquad M_2 \Uparrow}{\mathbf{snd}(M) \Uparrow} \quad (- \text{ snd pair})$$

Reduction

Basic Reduction

$$\mathbf{fst}(\langle M_1, M_2\rangle) \longrightarrow M_1 \quad (\text{fst tran})$$
$$\mathbf{snd}(\langle M_1, M_2\rangle) \longrightarrow M_2 \quad (\text{snd tran})$$

Simplification Reduction

$$\frac{M \longrightarrow M'}{\mathbf{fst}(M) \longrightarrow \mathbf{fst}(M')} \quad (\text{fst simpl})$$

$$\frac{M \longrightarrow M'}{\mathbf{snd}(M) \longrightarrow \mathbf{snd}(M')} \quad (\text{snd simpl})$$

Table 4.4: Products

Evaluation

$$\frac{}{\mathbf{inl}_{\tau_1,\tau_2}(M) \Downarrow \mathbf{inl}_{\tau_1,\tau_2}(M)} \quad (+ \text{ inl}) \qquad \frac{}{\mathbf{inr}_{\tau_1,\tau_2}(M) \Downarrow \mathbf{inr}_{\tau_1,\tau_2}(M)} \quad (+ \text{ inr})$$

$$\frac{M \Downarrow \mathbf{inl}_{\tau_1,\tau_2}(M') \qquad M_1[M'/x_1] \Downarrow C}{\mathbf{case}\ M\ \mathbf{of}\ \{\mathbf{inl}_{\tau_1,\tau_2}(x_1).M_1 \mid \mathbf{inr}_{\tau_1,\tau_2}(x_2).M_2\} \Downarrow C} \quad (+ \text{ cond inl})$$

$$\frac{M \Downarrow \mathbf{inl}_{\tau_1,\tau_2}(M') \qquad M_1[M'/x_1] \Downarrow C}{\mathbf{case}\ M\ \mathbf{of}\ \{\mathbf{inl}_{\tau_1,\tau_2}(x_1).M_1 \mid \mathbf{inr}_{\tau_1,\tau_2}(x_2).M_2\} \Downarrow C} \quad (+ \text{ cond inl})$$

$$\frac{M \Downarrow \mathbf{inr}_{\tau_1,\tau_2}(M') \qquad M_2[M'/x_2] \Downarrow C}{\mathbf{case}\ M\ \mathbf{of}\ \{\mathbf{inl}_{\tau_1,\tau_2}(x_1).M_1 \mid \mathbf{inr}_{\tau_1,\tau_2}(x_2).M_2\} \Downarrow C} \quad (+ \text{ cond inr})$$

Divergence

$$\frac{M \Uparrow}{\mathbf{case}\ M\ \mathbf{of}\ \{\mathbf{inl}_{\tau_1,\tau_2}(x_1).M_1 \mid \mathbf{inr}_{\tau_1,\tau_2}(x_2).M_2\} \Uparrow} \quad (- \text{ cond})$$

$$\frac{M \Downarrow \mathbf{inl}_{\tau_1,\tau_2}(M') \qquad M_1[M'/x_1] \Uparrow}{\mathbf{case}\ M\ \mathbf{of}\ \{\mathbf{inl}_{\tau_1,\tau_2}(x_1).M_1 \mid \mathbf{inr}_{\tau_1,\tau_2}(x_2).M_2\} \Uparrow} \quad (- \text{ cond inl})$$

$$\frac{M \Downarrow \mathbf{inr}_{\tau_1,\tau_2}(M') \qquad M_2[M'/x_2] \Uparrow}{\mathbf{case}\ M\ \mathbf{of}\ \{\mathbf{inl}_{\tau_1,\tau_2}(x_1).M_1 \mid \mathbf{inr}_{\tau_1,\tau_2}(x_2).M_2\} \Uparrow} \quad (- \text{ cond inr})$$

Reduction

Basic Reduction

$\mathbf{case}\ \mathbf{inl}_{\tau_1,\tau_2}(M')\ \mathbf{of}\ \{\mathbf{inl}_{\tau_1,\tau_2}(x_1).M_1 \mid \mathbf{inr}_{\tau_1,\tau_2}(x_2).M_2\} \longrightarrow M_1[M'/x_1]$ (cond tran inl)

$\mathbf{case}\ \mathbf{inr}_{\tau_1,\tau_2}(M')\ \mathbf{of}\ \{\mathbf{inl}_{\tau_1,\tau_2}(x_1).M_1 \mid \mathbf{inr}_{\tau_1,\tau_2}(x_2).M_2\} \longrightarrow M_2[M'/x_2]$ (cond tran inr)

Simplification Reduction (cond simpl)

$$\frac{M \longrightarrow M'}{\mathbf{case}\ M\ \mathbf{of}\ \{\mathbf{inl}_{\tau_1,\tau_2}(x_1).M_1 \mid \mathbf{inr}_{\tau_1,\tau_2}(x_2).M_2\} \longrightarrow \mathbf{case}\ M'\ \mathbf{of}\ \{\mathbf{inl}_{\tau_1,\tau_2}(x_1).M_1 \mid \mathbf{inr}_{\tau_1,\tau_2}(x_2).M_2\}}$$

Table 4.5: Sums

Evaluation

$$\frac{}{\mathbf{abs}_{\mu T.\tau}(C) \Downarrow \mathbf{abs}_{\mu T.\tau}(C)} \quad (+ \text{ abs can})$$

$$\frac{M \Downarrow \mathbf{abs}_{\mu T.\tau}(C)}{\mathbf{rep}(M) \Downarrow C} \quad (+ \text{ rep})$$

$$\frac{M \Downarrow C}{\mathbf{abs}_{\mu T.\tau}(M) \Downarrow \mathbf{abs}_{\mu T.\tau}(C)} \quad (+ \text{ abs})$$

Divergence

$$\frac{M \Uparrow}{\mathbf{rep}(M) \Uparrow} \quad (- \text{ rep})$$

$$\frac{M \Uparrow}{\mathbf{abs}_{\mu T.\tau}(M) \Uparrow} \quad (- \text{ abs})$$

Reduction

Basic reduction

$$\mathbf{rep}(\mathbf{abs}_{\mu T.\tau}(C)) \longrightarrow C \quad (\text{rep tran})$$

Simplification Reduction

$$\frac{M \longrightarrow M'}{\mathbf{rep}(M) \longrightarrow \mathbf{rep}(M')} \quad (\text{rep simpl})$$

$$\frac{M \longrightarrow M'}{\mathbf{abs}_{\mu T.\tau}(M) \longrightarrow \mathbf{abs}_{\mu T.\tau}(M')} \quad (\text{abs simpl})$$

Table 4.6: Recursive types

rules in tables 4.3-4.6 as *L-rules*. Following [Ibraheem and Schmidt, 2000] we define **L-rules** as follow:

Definition 4.2.1. *A rule r, instance of the following rule schema:*

$$\frac{M_1 \diamond C_1 \ \cdots \ M_k \diamond C_k}{M \diamond C}$$

*is an **L-rule** if*

- *the value of each M_i with $0 \leq i \leq k$ is a function of M and of all C_j with $0 \leq j < i$*
- *the value of each C_i with $0 \leq i \leq k$ is a function of M_i*
- *the value of C is a function of C_i with $0 \leq i \leq k$*

where $\diamond$ is some binary relation symbol.

Particularly the dependencies of each M_i from M and all C_j in our case are *structural* dependencies, which reflects the structural property of the semantics. The intuition behind L-rules is that the computation is performed following left to right evaluation.
Using again logical view, we can call the evaluation rules in tables 4.3-4.6 *inference rules*, the judgements above the line are the *antecedents* and the judgement below the line *consequent*. So we can see a derivation as a *proof tree* justifying why a term M can be evaluated to a canonical form C. Finally the rules without antecedents are called *axioms*.
We note that in the evaluation rules defined in tables 4.3-4.6 the axioms can be referred by (+ can), they are sufficient to capture all the convergent FPC terms.
Now we are able to define inductively evaluation relation: taken the operator $\phi_{\Downarrow} : \mathcal{P}(Exp \times Can) \to \mathcal{P}(Exp \times Can)$ induced by the set of instances of evaluation rules in tables 4.3-4.6 defined as:

$$\phi_{\Downarrow}(X) \stackrel{\text{def}}{=} \{(M, C) \mid \exists \ \frac{M_1 \Downarrow C_1 \cdots M_n \Downarrow C_n}{M \Downarrow C}, (n \geq 0) \\ \& \ (M_1, C_1), \cdots, (M_n, C_n) \in X\}$$

we can easily see that it is monotonic. Hence it possesses a least fixed point $\mu X.\phi_{\Downarrow}(X)$ and since rules of tables 4.3-4.6 are all *finitary* we can define it constructively with respect to $\subseteq$ and with $\emptyset$ as basis.

Definition 4.2.2. ***Evaluation relation*** *is the relation $\Downarrow \subseteq \ Exp \times Can$ inductively defined as:*

$$\Downarrow = \ \mu X.\phi_{\Downarrow}(X)$$

It is clear that when we write $M \Downarrow C$ we mean $(M, C) \ \in \Downarrow$.
We note that given a set of convergent judgements, $\phi_{\Downarrow}$ gives the set of all convergent judgement which can be inferred from this set in one step through the rule.
We are sometimes interested in convergent terms instead of pairs consisting of terms with their values, so we define $\Downarrow_{Exp} \subseteq Exp$ the set of convergent terms as:

$$\Downarrow_{Exp} \stackrel{\text{def}}{=} \ \{M | \exists \ C \ (M \Downarrow C)\}$$

When is clear from the context what we mean, we write $\Downarrow$ instead of $\Downarrow_{Exp}$ omitting the subsctipt *Exp*.
From (+ can) we directly have that evaluation relation is reflexive on canonical terms. We note that the evaluation rules are labelled with a +, differently from the rules for divergence. This notation, due to the Cousots [Cousot and Cousot, 1992], means that they should be read inductively, in opposite to rules labelled with − which, as we will see, should be read coinductively. As for typing rules it is easy to verify the following *inversion* lemma.

Lemma 4.2.3 (Inversion).

1. *If* $M_1 M_2 \Downarrow C$ *then there exists* $x \in Var$, $M \in Exp$ *such that* $M_1 \Downarrow \lambda x.M$ *and* $M[M_2/x] \Downarrow C$
2. *If* $\mathbf{fst}(M) \Downarrow C$ *then there exists* $\langle M_1, M_2 \rangle \in Exp$ *such that* $M \Downarrow \langle M_1, M_2 \rangle$ *and* $M_1 \Downarrow C$
3. *If* $\mathbf{snd}(M) \Downarrow C$ *then there exists* $\langle M_1, M_2 \rangle \in Exp$ *such that* $M \Downarrow \langle M_1, M_2 \rangle$ *and* $M_2 \Downarrow C$
4. *If* $\mathbf{abs}_{\mu T.\tau}(M) \Downarrow \mathbf{abs}_{\mu T.\tau}(C)$ *then* $M \Downarrow C$
5. *If* $\mathbf{rep}(M) \Downarrow C$ *then* $M \Downarrow \mathbf{abs}_{\mu T.\tau}(C)$
6. *If* $\mathbf{case}\ M\ \mathbf{of}\ \{\mathbf{inl}_{\tau_1,\tau_2}(x_1).M_1 \mid \mathbf{inr}_{\tau_1,\tau_2}(x_2).M_2\} \Downarrow C$ *then there exists* $M' \in Exp$ *such that either* $M \Downarrow \mathbf{inl}_{\tau_1,\tau_2}(M')$ *and* $M_1[M'/x_1] \Downarrow C$ *or* $M \Downarrow \mathbf{inr}_{\tau_1,\tau_2}(M')$ *and* $M_2[M'/x_2] \Downarrow C$

Proof. All cases are obvious by simple case inspection. □

We note that in case (4) of lemma 4.2.3 we have that the height of the proof tree of $M \Downarrow C$ is smaller than the heigh of the proof tree of $\mathbf{abs}_{\mu T.\tau}(M) \Downarrow \mathbf{abs}_{\mu T.\tau}(C)$. Since we can apply the same argument to the other cases we have that lemma 4.2.3 permits us to reason inductively on the height of proof tree. For example we can easily prove the following.

Proposition 4.2.4 (Determinacy). *Evaluation of FPC terms is deterministic, that is:*

$$\textit{if } M \Downarrow C \textit{ and } M \Downarrow C' \textit{ then } C \equiv C'$$

Proof. We can easily prove it for different cases by induction on the height of derivation tree of $M \Downarrow C$. We show only the case $M \equiv M_1 M_2$, the others are similar.
If $M_1 M_2 \Downarrow C$ by lemma 4.2.3-(1) we have that there exists $\lambda x.M \in Exp$ such that $M_1 \Downarrow \lambda x.M$ and $M[M_2/x] \Downarrow C$ and analogously if $M_1 M_2 \Downarrow C'$ we have that there exists $\lambda x'.M' \in Exp$ such that $M_1 \Downarrow \lambda x'.M'$ and $M'[M_2/x'] \Downarrow C'$. By induction hypothesis we have $\lambda x.M \equiv \lambda x'.M'$ but then $M'[M_2/x'] \equiv M[M_2/x]$ and again by induction hypothesis $C \equiv C'$. □

Proposition 4.2.5 (Type preservation). *Evaluation of FPC preserves types, that is:*

$$\textit{if } \emptyset \vdash M : \tau \textit{ and } M \Downarrow C \textit{ then } \emptyset \vdash C : \tau$$

Proof. We can easily prove it for different cases by induction on the height of derivation tree of $M \Downarrow C$.
Base case is trivial. We show only the case $M \equiv M_1 M_2$, the others are similar. Suppose $\emptyset \vdash M_1 M_2 : \tau$, by inversion rule 4.1.8-(3) we have that there exists τ_1 such that $\emptyset \vdash M_1 : \tau_1 \to \tau$ and $\emptyset \vdash M_2 : \tau_1$. Since $M_1 M_2 \Downarrow C$ by lemma 4.2.3-(1) we have that there exists $\lambda x.M$ such that $M_1 \Downarrow \lambda x.M$ and $M[M_2/x] \Downarrow C$. By induction hypothesis we have $\emptyset \vdash \lambda x.M : \tau_1 \to \tau$ and by inversion rule 4.1.8-(2) $\emptyset, x : \tau_1 \vdash M : \tau$. By lemma 4.1.10-(ii) we have $\emptyset \vdash M[M_2/x] : \tau$ hence by induction hypothesis $\emptyset \vdash C : \tau$. □

The above proposition justifies the fact that in our treatment of evaluation relation we have not considered type informations.
Evaluation is an high-level specification; it can be used, for example, to specify which canonical form an expression may have, but it is more useful when you want to prove properties of the convergent derivations by induction on the height of the derivation tree.

Example of Convergence We show an example of convergence which will be useful in the sequel. We prove the following convergence judgement:

$$(\mathbf{fst}\langle \lambda x.\mathbf{snd}(x), \Omega^\tau\rangle)\langle \Omega^\tau, \lambda z.z\rangle \Downarrow \lambda z.z$$

For clarity we first prove the following.

$$\frac{\langle \lambda x.\mathbf{snd}(x), \Omega^\tau\rangle \Downarrow \langle \lambda x.\mathbf{snd}(x), \Omega^\tau\rangle \quad \lambda x.\mathbf{snd}(x) \Downarrow \lambda x.\mathbf{snd}(x)}{\mathbf{fst}\langle \lambda x.\mathbf{snd}(x), \Omega^\tau\rangle \Downarrow \lambda x.\mathbf{snd}(x)}$$

With the above derived convergence judgement the proof of the whole judgement is the following.

$$\frac{\mathbf{fst}\langle \lambda x.\mathbf{snd}(x), \Omega^\tau\rangle \Downarrow \lambda x.\mathbf{snd}(x) \qquad \dfrac{\langle \Omega^\tau, \lambda z.z\rangle \Downarrow \langle \Omega^\tau, \lambda z.z\rangle \quad \lambda z.z \Downarrow \lambda z.z}{\mathbf{snd}\langle \Omega^\tau, \lambda z.z\rangle \Downarrow \lambda z.z}}{(\mathbf{fst}\langle \lambda x.\mathbf{snd}(x), \Omega^\tau\rangle)\langle \Omega^\tau, \lambda z.z\rangle \Downarrow \lambda z.z}$$

Divergence A divergent computation is intuitively a computation which never ends, a typical example of divergent terms is Ω^τ. Clearly convergence is the negative of divergence, so by property 4.2.4 we have that a term $M \in \mathit{Exp}$ diverges if and only if $(\nexists\ C,\ M \Downarrow C)$. This involves a quantification over all canonical terms and it is not always easy to prove. So we need a more direct definition for divergence.
Following the ideas which have lead us to the definition of evaluation we can consider a *divergence judgement* $M \Uparrow$ which means "the computation starting from expression M never ends". We read $M \Uparrow$ as "the term M diverges".
To define divergence we use the divergence rules in tables 4.3-4.6. We have chosen to follow the notation used by Pitts in [Pitts, 1994] for divergence and to use as rules instances of the following schema:

$$\frac{M_1 \Downarrow C_1 \ \cdots \ M_k \Downarrow C_k \quad M_{k+1} \Uparrow}{M \Uparrow}$$

instead we could have choosen to use a rules schema such the following:

$$\frac{M_1 \Downarrow C_1 \ \cdots \ M_k \Downarrow C_k \quad M_{k+1} \Downarrow \Omega}{M \Downarrow \Omega}$$

where $\Downarrow \Omega$ is only a notation for $\Uparrow$. This choice gives some notational problems when we want to define the divergence relation but emphasize that our divergence rules are instances of L-rules. We prefer the former notation which simplifies the following definitions but we remark that divergence rules have left to right nature.
We now do not need axioms because we define divergence coinductively [Cousot and Cousot, 1992] instead of inductively.
Similarly to the evaluation case we can take the operator $\phi_{\Uparrow} : \mathcal{P}(Exp) \rightarrow \mathcal{P}(Exp)$ induced by the set of instances of divergence rules in tables 4.3-4.6:

$$\phi_{\Uparrow}(X) \stackrel{\text{def}}{=} \{M \mid \exists \ \frac{M_1 \Downarrow C_1 \cdots M_n \Downarrow C_n \ M_{n+1} \Uparrow}{M \Uparrow}, (n \geq 0) \ \& \ M_{n+1} \in X\}$$

we can easily verify that $\phi_{\Uparrow}$ is monotonic. Hence it possesses a greatest fixed point $\nu X.\phi_{\Uparrow}(X)$ and since rules of tables 4.3-4.6 are all *deterministic* we can define it constructively with respect to $\subseteq$ and with *Exp* as basis.

Definition 4.2.6. *The set $\Uparrow$ of **divergent terms** is the set coinductively defined by $\phi_{\Uparrow}$:*

$$\Uparrow \stackrel{def}{=} \nu X.\phi_{\Uparrow}(X)$$

Given a set of divergent terms, $\phi_{\Uparrow}$ yields all divergent terms which can be deduced in one step from this set by applying the rules. The set coinductively defined by $\phi_{\Uparrow}$ is obtained by iterating $\phi_{\Uparrow}$ starting from the set of all well formed terms *Exp*, each iteration remove the terms that cannot be deduced by applying the rules once from the conclusion of previous iteration.
Note that some rules have, in the antecedents, convergence judgments. This is possible because convergence is constructed without reference to divergence. This means that our definition of divergence depends in a certain sense from convergence.
It is easy to verify the following inversion lemma.

Lemma 4.2.7 (Inversion).

1. *If $M_1 M_2 \Uparrow$ then either $M \Uparrow$ or there exists $x \in Var$, $M \in Exp$ such that $M_1 \Downarrow \lambda x.M$ and $M[M_2/x] \Uparrow$*
2. *If $\mathbf{fst}(M) \Uparrow$ then either $M \Uparrow$ or there exists $\langle M_1, M_2 \rangle \in Exp$ such that $M \Downarrow \langle M_1, M_2 \rangle$ and $M_1 \Uparrow$*
3. *If $\mathbf{snd}(M) \Uparrow$ then either $M \Uparrow$ or there exists $\langle M_1, M_2 \rangle \in Exp$ such that $M \Downarrow \langle M_1, M_2 \rangle$ and $M_2 dv$*
4. *If $\mathbf{abs}_{\mu T.\tau}(M) \Uparrow$ then $M \Uparrow$*
5. *If $\mathbf{rep}(M) \Uparrow$ then $M \Uparrow$*
6. *If $\mathbf{case}\ M\ \mathbf{of}\ \{\mathbf{inl}_{\tau_1,\tau_2}(x_1).M_1 \mid \mathbf{inr}_{\tau_1,\tau_2}(x_2).M_2\} \Uparrow$ then either $M \Uparrow$ or there exists $M' \in Exp$ such that either $M \Downarrow \mathbf{inl}_{\tau_1,\tau_2}(M')$ and $M_1[M'/x_1] \Uparrow$ or $M \Downarrow \mathbf{inr}_{\tau_1,\tau_2}(M')$ and $M_2[M'/x_2] \Uparrow$*

Proof. All cases are obvious by a simple case analysis. □

We remark that the divergence rules are labelled with $-$ to stress that they form a coinductive definition and not because they are negative rules. This definition of divergence is a high-level specification and is useful when onr wants to prove properties of the divergent derivations by coinduction. We will see in the next section that is possible to characterize divergence coinductively also with a small step semantics.

Example of divergence As an example of divergence we show the divergence of Ω^{τ_1}. We have defined it as:

$$(\lambda x.\mathbf{rep}(x)x)\mathbf{abs}_{\mu T.\tau}(\lambda x.\mathbf{rep}(x)x)$$

We have that its proof of divergence is an infinite tree. For this reason we only give an itnitial part of proof tree, for clearity of notation we omitt the obvious convergence judgements for canonical terms.

$$\dfrac{\dfrac{\mathbf{rep}(\mathbf{abs}_{\mu T.\tau}(\lambda x.\mathbf{rep}(x)x)) \Downarrow \lambda x.\mathbf{rep}(x)x \quad \dfrac{\vdots}{(\lambda x.\mathbf{rep}(x)x)\mathbf{abs}_{\mu T.\tau}(\lambda x.\mathbf{rep}(x)x) \Uparrow}}{\mathbf{rep}(\mathbf{abs}_{\mu T.\tau}(\lambda x.\mathbf{rep}(x)x))\mathbf{abs}_{\mu T.\tau}(\lambda x.\mathbf{rep}(x)x) \Uparrow}}{(\lambda x.\mathbf{rep}(x)x)\mathbf{abs}_{\mu T.\tau}(\lambda x.\mathbf{rep}(x)x) \Uparrow}$$

4.2.2 Small-Step

The idea which underlies small-step approaches is to capture single computation steps. We first give the rules to define inductively a small-step *transition* relation on FPC terms which captures all the single steps of all computations. We know that a computation is not only a set of single steps but a sequence of single steps, hence we introduce for transition an intuitive kind of composition property. We finally show how characterize inductively and coinductively the notions of convergence and divergence, respectively.
Small-step relation was used to define operational semantics of different language, either functional and not. For many examples of the use of these relations see [Plotkin, 1981]. Moran in [Moran, 1994] gives operational semantics to a non deterministic version of lambda calculus through a small-step relation. The same happens in [Birkedal and Harper, 1999] for a call-by-value language similar to FPC. An operational semantics for a call-by-name version of FPC, through a small-step relation, was given by Gordon in [Gordon, 1995].
Our presentation is close to the those given by Gordon in [Gordon, 1995] and Moran in [Moran, 1994].

Transition To capture all valid computation single steps, we need a binary relation $\longrightarrow \subseteq \mathit{Exp} \times \mathit{Exp}$ as general as possible. Given the set of instances of reduction rules in tables 4.3-4.6 we can define, similarly to big-step case, an operator $\phi_{\rightarrow} : \mathcal{P}(\mathit{Exp} \times \mathit{Exp}) \rightarrow \mathcal{P}(\mathit{Exp} \times \mathit{Exp})$ as:

$$\phi_{\rightarrow}\ (X) \stackrel{\text{def}}{=} \{(M,N) \mid \exists\ \frac{M'_n \longrightarrow N'_n}{M \longrightarrow N}\ (0 \leq n \leq 1)\ \&\ (M'_n, N'_n) \in X\}$$

$\phi_{\rightarrow}$ is monotonic and so it possesses a least fixed point $\mu X.\phi_{\rightarrow}(X)$ and since rules of tables 4.3-4.6 are all *finitary* we can define it constructively with

respect to $\subseteq$ and with $\emptyset$ as basis.

Definition 4.2.8. ***Transition relation*** *is the relation* $\longrightarrow \subseteq \ Exp \times Exp$ *inductively defined as:*

$$\longrightarrow \stackrel{def}{=} \mu X.\phi_{\rightarrow}(X)$$

We write $(M \longrightarrow N)$ instead of $(M, N) \in \longrightarrow$ and it can be read as "term M reduces to term N".
$\longrightarrow$ is often called transition relation because represents the transition from a state to another, but it is also known as *reduction* or *small-step relation*. We use the term transition when we want to stress the fact that, to capture an entire computation, we need the transitive closure of the relation. Analogously we call *reduction*, the operation of rewriting a single term into another.
We say that M is a **redex** if it appears on the left of any basic reduction rules, a **contractum** if it appears on the right. The set *Rdx* of *redex* is the set of terms defined by the following production:

$$\begin{aligned} R ::= \ & (\lambda x.M)N \mid \mathbf{fst}(\langle M_1, M_2\rangle) \mid \mathbf{snd}(\langle M_1, M_2\rangle) \mid \mid \mathbf{rep}(\mathbf{abs}_{\mu T.\tau}(C)) \\ & \mid \mathbf{case}\ \mathbf{inl}_{\tau_1,\tau_2}(M)\ \mathbf{of}\ \{\mathbf{inl}_{\tau_1,\tau_2}(x_1).M_1 \mid \mathbf{inr}_{\tau_1,\tau_2}(x_2).M_2\} \\ & \mid \mathbf{case}\ \mathbf{inr}_{\tau_1,\tau_2}(M)\ \mathbf{of}\ \{\mathbf{inl}_{\tau_1,\tau_2}(x_1).M_1 \mid \mathbf{inr}_{\tau_1,\tau_2}(x_2).M_2\} \end{aligned}$$

We remark that there are no rules for term in canonical form, as shown by a simple analysis of rules, so these terms do not reduce further. Particularly the terms in canonical form are the only terms in *Exp* which do not reduce as will be proved (also stuck terms don't reduce further, but we have decided not to consider them). This justifies the fact that we can think to them as values.
A computation is not a general set of single steps, but a sequence of steps. According to this intuition, in order to express computation by a transition relation, we need to compose single steps.
We can define the set of computations of n steps, or transitions of n steps as:

$$\longrightarrow^n \stackrel{\text{def}}{=} \{(M, N) \mid \exists M_1, \ldots, M_{n-1}\ \&\ (M, M_1), (M_1, M_2), \ldots, (M_{n-1}, N) \in \longrightarrow\}$$

When we want to stress that in a computation $(M \longrightarrow^n N)$ after m steps of computation occur the term M_m we can write $(M \longrightarrow^m M_m \longrightarrow^{(n-m)} N)$.
We can now define the **transitive closure of** $\longrightarrow$ as:

$$\longrightarrow^+ \stackrel{\text{def}}{=} \bigcup_{n>0} \{(M \longrightarrow^n N)\}$$

Transitive closure captures all the computations of length $n > 0$. To add reflexivity we sometimes want to consider as computation also transitions of 0 steps. These computations have no steps, so we need a relation which do not modify terms. We know that the needed relation is the identity relation. So we have:

$$\longrightarrow^0 \stackrel{\text{def}}{=} \{(M, M) \mid M \in Exp\}$$

Now we can define the **reflexive-transitive closure of** $\longrightarrow$ as:

$$\longrightarrow^* \stackrel{\text{def}}{=} \bigcup_{n\geq 0} \{(M \longrightarrow^n N)\}$$

The reflexive-transitive closure of $\longrightarrow$ captures all the finite computations, whereas we write $M \longrightarrow^{\omega}$ to mean that starting from the term M there is an infinite sequence of transition. We define later the set of terms spawning infinitely long computations.
Note that we have two kind of reduction rules: *basic reductions* and *simplification reductions*. We can consider basic reductions as rules which define how we can reduce a term in the specified form into another term. Simplification reductions can be viewed as rules which define how to propagate basic reductions to contexts.
A simple analysis of the simplification reduction rules shows that all the rules can be given through a general schema. It can be done using the notion of **evaluation context** pioneered in [Felleisen and Friedman, 1986]. We introduce evaluation contexts, analogously to Gordon in [Gordon, 1995], as compositions of atomic experiments.

Definition 4.2.9. *Given an arbitrary variable $\bullet$ which can be viewed as a* hole, *the set Exper of* ***experiments*** *is defined as:*

$$\begin{array}{l}\mathcal{E} ::= [\bullet]\ M_1 \mid \boldsymbol{fst}([\bullet]) \mid \boldsymbol{snd}([\bullet]) \mid \boldsymbol{rep}([\bullet]) \mid \boldsymbol{abs}_{\mu T.\tau}([\bullet]) \\ \quad \mid \boldsymbol{case}\ [\bullet]\ \boldsymbol{of}\ \{\boldsymbol{inl}_{\tau_1,\tau_2}(x_1).M_1 \mid \boldsymbol{inr}_{\tau_1,\tau_2}(x_2).M_2\}\end{array}$$

Definition 4.2.10. *Let Ectx be the set of* ***evaluation context*** $\mathcal{E}$ *recursively defined as:*

$$\begin{array}{rcll}\vec{\mathcal{E}} & ::= & \mathcal{I}d & \textit{empty} \\ & \mid & \vec{\mathcal{E}} \circ \mathcal{E} & \textit{non-empty}\end{array}$$

where $\mathcal{E} \in Exper$ and $\mathcal{I}d \equiv [\bullet]$ and composition $\vec{\mathcal{E}} \circ \mathcal{E}$ means the syntactical substitution of $\mathcal{E}$ to $\bullet$ in $\vec{\mathcal{E}}$ which we write as $\vec{\mathcal{E}}[\bullet/\mathcal{E}[\bullet]]$.

Pitts in [Pitts, 2002] use two notions strictly related to experiments and evaluation contexts for a fragment of **ML**; they are called *frame* and *frame stack* respectively.
The definition of evaluation context gives strategies of reduction, the variable $\bullet$ in a evaluation context indicates, where in an expression, a reduction can take place without affecting the context. Note that there is no experiment $\mathcal{E}[\bullet]$ which can bind a variable of M in the substitution $\mathcal{E}[M/\bullet]$. We do not add type information in the definition of evaluation context but we remark that types are always used implicitly.
It is easy to verify some properties of evaluation contexts.

Lemma 4.2.11. *For each evaluation context $\vec{\mathcal{E}}$ there exist unique $\mathcal{E}_1, \cdots, \mathcal{E}_n \in Exper$ such that $\mathcal{E} \equiv \mathcal{I}d \circ \mathcal{E}_1 \circ \cdots \circ \mathcal{E}_n$.*

Lemma 4.2.12 (Unique redex decomposition). *For each $R \in Rdx$, R can be written uniquely as $\mathcal{E}[C]$ where C is a value and $\mathcal{E} \in Exper$ is an experiment.*

Proof. It follows immediately from definition of *Rdx* by case analysis. □

Furthermore we can prove the following.

Lemma 4.2.13 (Unique decomposition). *For all $M \in Exp$ either M is a value or M can be written uniquely as $\vec{\mathcal{E}}[R]$ where R is a redex.*

Proof. We can prove the lemma by structural induction. We suppose the lemma valid for all subterms and analyze the different cases:

$M \equiv C$. Trivial

$M \equiv M_1 M_2$ .We have two different possibilities:

- M_1 is a value, because M is well typed, it can be only a lambda abstraction $\lambda x.M''$, so we have the redex of rule (app tran)
- M_1 is not a value, by induction hypothesis it can be written uniquely as $\vec{\mathcal{E}}_{M_1}[M']$ where M' is a redex, so we can write uniquely $M_1 M_2$ as $\vec{\mathcal{E}}_{M_1}[M']M_2$. Thus M can be written uniquely as $\vec{\mathcal{E}}[M']$ where $\vec{\mathcal{E}}[\bullet] \equiv \vec{\mathcal{E}}_{M_1}[\bullet]M_2$

$M \equiv \mathbf{fst}(M')$ .We have two different possibilities:

- M' is a value, because M is well typed, it can be only a pair $\langle M_1, M_2 \rangle$, so we have the redex of the rule (fst tran)
- M' is not a value, by hypothesis it can be written uniquely as $\vec{\mathcal{E}}_{M'}[M'']$ where M'' is a redex, so we can write uniquely $\mathbf{fst}(M')$ as $\mathbf{fst}(\vec{\mathcal{E}}_{M'}[M''])$. Thus M can be written uniquely as $\vec{\mathcal{E}}[M'']$ where $\vec{\mathcal{E}}[\bullet] \equiv \mathbf{fst}(\vec{\mathcal{E}}_{M'}[\bullet])$.

$M \equiv \mathbf{snd}(M')$ .We have two different possibilities:

- M' is a value, as before it can be only a pair $\langle M_1, M_2 \rangle$, so we have the redex of the rule (snd tran)
- M' is not a value, by hypothesis it can be written uniquely as $\vec{\mathcal{E}}_{M'}[M'']$ where M'' is a redex, so we can write uniquely $\mathbf{snd}(M')$ as $\mathbf{snd}(\vec{\mathcal{E}}_{M'}[M''])$. Thus M can be written uniquely as $\vec{\mathcal{E}}[M'']$ where $\vec{\mathcal{E}}[\bullet] \equiv \mathbf{snd}(\vec{\mathcal{E}}_{M'}[\bullet])$.

$M \equiv \mathbf{case}\ M'\ \mathbf{of}\ \{\mathbf{inl}_{\tau_1,\tau_2}(x_1).M_1 \mid \mathbf{inr}_{\tau_1,\tau_2}(x_2).M_2\}$. We have only two different possibilities:

- M' is a value, because M is well typed, it can be either $\mathbf{inl}(M'')$ or $\mathbf{inr}(M'')$, the former is the redex of the rule (cond tran inl) the latter the redex of the rule (cond tran inr)
- M' is not a value, by hypothesis it can be written uniquely as $\vec{\mathcal{E}}_{M'}[M'']$ where M'' is a redex, so we can write uniquely $(\mathbf{case}\ M'\ \mathbf{of}\ \{\mathbf{inl}_{\tau_1,\tau_2}(x_1).M_1 \mid \mathbf{inr}_{\tau_1,\tau_2}(x_2).M_2\})$ as $(\mathbf{case}\ \vec{\mathcal{E}}_{M'}[M'']\ \mathbf{of}\ \{\mathbf{inl}_{\tau_1,\tau_2}(x_1).M_1 \mid \mathbf{inr}_{\tau_1,\tau_2}(x_2).M_2\})$. Thus M can be written uniquely as $\vec{\mathcal{E}}[M'']$ where $\vec{\mathcal{E}}[\bullet] \equiv (\mathbf{case}\ \vec{\mathcal{E}}_{M'}[\bullet]\ \mathbf{of}\ \{\mathbf{inl}_{\tau_1,\tau_2}(x_1).M_1 \mid \mathbf{inr}_{\tau_1,\tau_2}(x_2).M_2\})$

$M \equiv \mathbf{abs}_{\mu T.\tau}(M')$. We have two different possibilities:

- M' is a value, but hence by definition of canonical terms also $\mathbf{abs}_{\mu T.\tau}(M')$ is a value.

- M' is not a value, and by hypothesis it can be written uniquely as $\vec{\mathcal{E}}_{M'}[M'']$ where M'' is a redex, so we can write uniquely $\mathbf{abs}_{\mu T.\tau}(M')$ as $\mathbf{abs}_{\mu T.\tau}(\vec{\mathcal{E}}_{M'}[M''])$. Thus M can be written uniquely as $\vec{\mathcal{E}}[M'']$ where $\vec{\mathcal{E}}[\bullet] \equiv \mathbf{abs}_{\mu T.\tau}(\vec{\mathcal{E}}_{M'}[\bullet])$.

$M \equiv \mathbf{rep}(M')$. We have two different possibilities:

- M' is a value, because M is well typed, it can be only $\mathbf{abs}_{\mu T.\tau}(C)$, so we have the redex of the rule (rep tran)
- M' is not a value, by hypothesis it can be written uniquely as $\vec{\mathcal{E}}_{M'}[M'']$ where M'' is a redex, so we can write uniquely $\mathbf{rep}(M')$ as $\mathbf{rep}(\vec{\mathcal{E}}_{M'}[M''])$. Thus M can be written uniquely as $\vec{\mathcal{E}}[M'']$ where $\vec{\mathcal{E}}[\bullet] \equiv \mathbf{rep}(\vec{\mathcal{E}}_{M'}[\bullet])$.

□

By the last two lemmas we have the following.

Lemma 4.2.14 (Unique decomposition). *For all $M \in Exp$, either M is a value or M can be written uniquely as $\vec{\mathcal{E}}[\mathcal{E}[C]]$ where C is a value, $\vec{\mathcal{E}} \in Ectx$ and $\mathcal{E} \in Exper$.*

Now we are ready to add a generic rule to substitute all the simplification reduction rules.

$$\frac{M \longrightarrow M'}{\vec{\mathcal{E}}[M] \longrightarrow \vec{\mathcal{E}}[M']} \qquad \text{(red tran)}$$

So the transition relation becomes the binary relation inductively defined as the smallest relation closed under the basic rules in tables 4.3-4.6 and the general rule (red tran). Note that (red tran) rule tells that transition relation is a special kind of pre-congruence.
As an immediate consequence of this definition we can prove that transition is deterministic:

Lemma 4.2.15 (Determinacy). *If $(M \longrightarrow N)$ and $(M \longrightarrow N')$ then $N \equiv N'$.*

Proof. We can prove the lemma by induction on the structure of M. By lemma 4.2.13 we know that either M is a value or M has a unique decomposition $\vec{\mathcal{E}}[M']$ where M' is a redex. We have three distinct cases. If M is a value then we have no transition. If M is a redex and $\vec{\mathcal{E}}[\bullet] \equiv \bullet$ conclusion follows immediately by a simple analysis of the basic rules. Otherwise by induction hypothesis we have that if $M' \longrightarrow M_1$ and $M' \longrightarrow M_2$ then $M_1 \equiv M_2$, hence we have $M \equiv \vec{\mathcal{E}}[M'] \longrightarrow \vec{\mathcal{E}}[M_1] \equiv N$ and $M \equiv \vec{\mathcal{E}}[M'] \longrightarrow \vec{\mathcal{E}}[M_2] \equiv N'$ but $\vec{\mathcal{E}}[M_1] \equiv \vec{\mathcal{E}}[M_2]$ and so $N \equiv N'$. □

Lemma 4.2.16 (Type preservation). *If $(M \longrightarrow N)$ and $\emptyset \vdash M : \tau$ then $\emptyset \vdash N : \tau$.*

Proof. If M is a redex it follows immediately by an analysis of the different case and type inversion 4.1.8. The only special cases are when M is equivalent to $(\lambda x.M')N$,**case** $\mathbf{inl}_{\tau_1,\tau_2}(M)$ **of** $\{\mathbf{inl}_{\tau_1,\tau_2}(x_1).M_1 \mid \mathbf{inr}_{\tau_1,\tau_2}(x_2).M_2\}$ and **case** $\mathbf{inr}_{\tau_1,\tau_2}(M)$ **of** $\{\mathbf{inl}_{\tau_1,\tau_2}(x_1).M_1 \mid \mathbf{inr}_{\tau_1,\tau_2}(x_2).M_2\}$. We consider the first case, the others are similar.
By determinacy of transition we have $(\lambda x.M')N \longrightarrow M'[N/x]$. Suppose $\emptyset \vdash (\lambda x.M')N : \tau$ then by inversion rule 4.1.8-(3) we have that there exists τ_1 such that $\emptyset \vdash \lambda x.M' : \tau_1 \to \tau$ and $\emptyset \vdash N : \tau_1$. Furthermore by 4.1.8-(2) we have that $\emptyset, x : \tau \vdash M : \tau$ hence by lemma 4.1.10-(ii) it follows that $\emptyset \vdash M'[N/x] : \tau$.
Otherwise if M is not a redex we can apply to it a simplification reduction, hence by induction hypothesis and determinacy of typing the conclusion follows. □

The above proposition justifies the fact that in our treatment of reduction relation we have not considered type informations.
Giving an operational semantics via a transition relation is useful to stress the importance, not only of the values, but also of the reduction strategies.
As evaluation relation, the transition relation is syntax-directed, but in a different way. Looking at the transition rules we can see that, differently from evaluation relation, the analysis for the application of rules is not strictly on the syntactic structure but rather on where in an expression a reduction is feasible. It depends strictly from the language.
The idea behind transition relation is that the reduction rules are essentially rewriting rules from terms to terms and the evaluation contexts tell which term can be rewritten. Transition relation is particularly useful when you want to prove properties about convergent and divergent derivations by induction on the length of computations.
We define by a transition point of view some useful notion.

Definition 4.2.17. *For all* $M \in Exp$

M ***reduces***	$(M \longrightarrow)$	*iff*	$\{\exists\ N \mid (M \longrightarrow N)\}$
M ***value***	$\neg(M \longrightarrow)$	*iff*	$\{\nexists\ N \mid (M \longrightarrow N)\}$
M ***evaluates to*** N	$(M \downarrow N)$	*iff*	$\{(M \longrightarrow^* N)$ *for some value* $N\}$
M ***evaluates***	$(M \downarrow)$	*iff*	$\{\exists$ *a value* $N \mid (M \downarrow N))\}$
M ***diverges***	$(M \uparrow)$	*iff*	$\{\forall\ N\ (M \longrightarrow^* N) \Longrightarrow (N \longrightarrow)\}$

Since transition is deterministic we write $(M \longrightarrow^* C)$ instead of $(M \downarrow C)$ and $(M \longrightarrow^\omega)$ instead of $(M \uparrow)$ when we want to emphasize that these are defined from a transition relation. We will see in the next section that these definitions of evaluation and of divergence correspond exactly to the ones given by natural semantics rules. We first prove an easy lemma:

Lemma 4.2.18. *If* M *is a value then* $M \in Can$

Proof. We prove the contrapositive by induction on the structure of M. Assume $M \notin Can$, we show two cases, the other are similar.

$M \equiv M_1 M_2$ .We have two different possibilities:

- M_1 is a value, by induction hypothesis and because M is well typed, it can be only a lambda abstraction $\lambda x.M''$, and since we can apply rule (app tran) we have that M is not a value.

- M_1 is not a value hence there exists M_1' such that $M_1 \longrightarrow M_1'$. Thus by (red tran) rule we have $M_1 M_2 \longrightarrow M_1' M_2$ and hence it is not a value.

$M \equiv \mathbf{fst}(M_1)$ .We have two different possibilities:

- M_1 is a value, by induction hypothesis and because M is well typed, it can be only a pair $\langle M_1, M_2 \rangle$, and since we can apply rule (fst tran) we have that M is not a value.
- M_1 is not a value hence there exists M_1' such that $M_1 \longrightarrow M_1'$. Thus by (red tran) rule we have $\mathbf{fst}(M_1) \longrightarrow \mathbf{fst}(M_1')$ and hence it is not a value.

□

We now state an important result which follows immediately from lemma 4.2.13 and lemma 4.2.15:

Lemma 4.2.19. *For all $M \in Exp$ we have that either exists $C \in Can$ unique such that $M \Downarrow C$ or $M \Uparrow$.*

Proof. For each $M \in Exp$ by lemma 4.2.13 we have that either M is a value so $M \longrightarrow^0 M$ and hence $M \Downarrow$ or it can be written uniquely as $\vec{\mathcal{E}}[R]$ where $R \in Rdx$. So $R \longrightarrow R'$ where R' is not necessarily in Rdx and by (red tran) we have:

$$\frac{R \longrightarrow R'}{\vec{\mathcal{E}}[R] \longrightarrow \vec{\mathcal{E}}[R']}$$

and hence M reduce to a term $N \equiv \vec{\mathcal{E}}[R']$ and by lemma 4.2.15 this is unique. Now again by lemma 4.2.13 we have that N is either a value or it reduces and by applying repeatedly the above argument by lemma 4.2.15 we have that either exists unique $C \in Can$ such that $M \longrightarrow^* C$ or it do not exists and then $\forall\, N\ (M \longrightarrow^* N) \Longrightarrow (N \longrightarrow)$ hence $M \Uparrow$. □

Examples of transition We show two examples of transition.The two examples lead to the interesting considerations of the following sections.
As a first example we show the convergence of the term

$$(\mathbf{fst}\langle \lambda x.\mathbf{snd}(x), \Omega^\tau \rangle)\langle \Omega^\tau, \lambda z.z \rangle$$

already used in the example for evaluation relation. We have the following proof.

$$\frac{\mathbf{fst}\langle \lambda x.\mathbf{snd}(x), \Omega^\tau \rangle \longrightarrow \lambda x.\mathbf{snd}(x)}{\mathbf{fst}\langle \lambda x.\mathbf{snd}(x), \Omega^\tau \rangle\langle \Omega^\tau, \lambda z.z \rangle \longrightarrow (\lambda x.\mathbf{snd}(x))\langle \Omega^\tau, \lambda z.z \rangle} \longrightarrow \mathbf{snd}\langle \Omega^\tau, \lambda z.z \rangle \longrightarrow \lambda z.z$$

Usually when we write a proof by transition we do not write the premisses of the rules so it becomes:

$$\mathbf{fst}\langle \lambda x.\mathbf{snd}(x), \Omega^\tau \rangle\langle \Omega^\tau, \lambda z.z \rangle \longrightarrow (\lambda x.\mathbf{snd}(x))\langle \Omega^\tau, \lambda z.z \rangle \longrightarrow \mathbf{snd}\langle \Omega^\tau, \lambda z.z \rangle \longrightarrow \lambda z.z$$

which can finally be rewritten as:

$$(\mathbf{fst}\langle \lambda x.\mathbf{snd}(x), \Omega^\tau \rangle)\langle \Omega^\tau, \lambda z.z \rangle \longrightarrow^3 \lambda z.z$$

The second example show the divergence of the term Ω^{τ_1}.

$$\begin{aligned}\Omega^{\tau_1} \equiv (\lambda x.\mathbf{rep}(x)x)\mathbf{abs}_{\mu T.\tau}(\lambda x.\mathbf{rep}(x)x) &\longrightarrow \\ \mathbf{rep}(\mathbf{abs}_{\mu T.\tau}(\lambda x.\mathbf{rep}(x)x))\mathbf{abs}_{\mu T.\tau}(\lambda x.\mathbf{rep}(x)x) &\longrightarrow \\ (\lambda x.\mathbf{rep}(x)x)\mathbf{abs}_{\mu T.\tau}(\lambda x.\mathbf{rep}(x)x) &\longrightarrow \cdots\end{aligned}$$

It is easy to verify that the above is an infinite sequence of transitions.

Following Gordon in [Gordon, 1995], we now characterize inductively and coinductively respectively the notions of convergence and divergence by a transition relation.

Inductive Characterization of Convergence By definition 4.2.17 we have the set of term which can be evaluated to other $\downarrow$ defined as:

$$\downarrow \stackrel{\text{def}}{=} \{(M,N) \mid M \downarrow N\} = \{(M,N) \mid (M \longrightarrow^* N) \;\&\; \neg(N \longrightarrow)\}$$

Taken the operator $\psi_\downarrow : \mathcal{P}(Exp \times Exp) \to \mathcal{P}(Exp \times Exp)$ defined as:

$$\psi_\downarrow(X) \stackrel{\text{def}}{=} \{(N,N) \mid \neg(N \longrightarrow)\} \cup \{(M,N) \mid \exists N' \; (M \longrightarrow N') \;\&\; (N',N) \in X\}$$

we can easily see that it is monotonic. Hence it possesses a least fixpoint, $\mu X.\psi_\downarrow(X)$ which is the least $\psi_\downarrow$-closed set and since rules are all *finitary* we can define it constructively with respect to $\subseteq$ and with $\emptyset$ as basis. We can prove that $\downarrow$ can be inductively characterized.

Theorem 4.2.20. $\downarrow = \mu X.\psi_\downarrow(X)$

Proof.

$(\mu X.\psi_\downarrow(X) \subseteq \downarrow)$.

By induction we only need to show that $\downarrow$ is $\psi_\downarrow$-closed: $\psi_\downarrow(\downarrow) \subseteq \downarrow$.
Suppose that (M,N) is in $\psi_\downarrow(\downarrow)$, we have two cases:

- $(M,N) \in \{(N,N) \mid \neg(N \longrightarrow)\}$, so $(N \equiv M)$ and we have that $\neg(M \longrightarrow)$ but we know that $M \longrightarrow^0 M$ and so $M \longrightarrow^* M$ so we have that $(M,M) \in \downarrow$.
- $(M,N) \in \{(M,N) \mid \exists N' \; (M \longrightarrow N') \;\&\; (N',N) \in \downarrow\}$ we have $((M \longrightarrow N') \;\&\; (N',N) \in \downarrow)$ hence $(M \longrightarrow N' \longrightarrow^* N) \;\&\; \neg(N \longrightarrow)$, which can be rewritten as $(M \longrightarrow^* N) \;\&\; \neg(N \longrightarrow)$ and so $(M,N) \in \downarrow$.

$(\downarrow \subseteq \mu X.\psi_\downarrow(X))$.

We know that if $(M,N) \in \mu X.\psi_\downarrow(X)$ then $(M,N) \in \psi_\downarrow(\mu X.\psi_\downarrow(X))$ by the fixpoint property. Suppose $(M,N) \in \downarrow$ then $(M \longrightarrow^* N) \;\&\; \neg(N \longrightarrow)$, so $\exists\, n \; (M \longrightarrow^n N) \;\&\; \neg(N \longrightarrow)$. We prove the assertion by induction on n.

- if $(n = 0)$ then we have $(M \longrightarrow^0 N) \;\&\; \neg(N \longrightarrow)$ but we know that the only term N with the property $M \longrightarrow^0 N$ is M so we have $\neg(M \longrightarrow)$ and $(M,M) \in \mu X.\psi_\downarrow(X)$.

- suppose the assertion valid for n, hence for n we have that for all M' such that if $(M' \longrightarrow^n N)$ & $\neg(N \longrightarrow)$ then $(M', N) \in \mu X.\psi_{\downarrow}(X)$. We want to show that if $(M \longrightarrow^{n+1} N)$ & $\neg(N \longrightarrow)$ then also $(M, N) \in \mu X.\psi_{\downarrow}(X)$. If $(M \longrightarrow^{n+1} N)$ & $\neg(N \longrightarrow)$ we know that $\exists\ M'\ (M \longrightarrow M' \longrightarrow^n N)$ & $\neg(N \longrightarrow)$, by induction hypotesis we have $(M', N) \in \mu X.\psi_{\downarrow}(X)$ and so $(M \longrightarrow M')$ & $((M', N) \in \mu X.\psi_{\downarrow}(X))$ hence $(M, N) \in \psi_{\downarrow}(\mu X.\psi_{\downarrow}(X))$ but it is equal to $(M, N) \in \mu X.\psi_{\downarrow}(X)$.

□

Coinductive Characterization of Divergence By definition 4.2.17 we have the set of divergent terms $\uparrow$ defined as:

$$\uparrow \stackrel{\text{def}}{=} \{M \mid \forall\ N\ (M \longrightarrow^* N) \implies (N \longrightarrow)\}$$

Taken the operator $\psi_{\uparrow} : \mathcal{P}(Exp) \to \mathcal{P}(Exp)$ defined as:

$$\psi_{\uparrow}(X) \stackrel{\text{def}}{=} \{M \mid \exists N\ (M \longrightarrow N)\ \&\ N \in X\}$$

we can easily see that it is monotonic. Hence it possess a greatest fixpoint, $\nu X.\psi_{\uparrow}(X)$ which is the largest $\psi_{\uparrow}$-dense set and since rules are all *deterministic* we can define it constructively with respect to $\subseteq$ and with *Exp* as basis. We can prove that $\uparrow$ can be coinductively characterized by the following theorem.

Theorem 4.2.21. $\uparrow = \nu X.\psi_{\uparrow}(X)$

Proof.

$(\uparrow \subseteq \nu X.\psi_{\uparrow}(X))$.

By coinduction we only need to show that $\uparrow$ is $\psi_{\uparrow}$-dense: $\uparrow \subseteq \psi_{\uparrow}(\uparrow)$.
Suppose that M is in $\uparrow$, we know that whenever $(M \longrightarrow^* M')$ then $(M' \longrightarrow)$, in particular $\exists\ N\ (M \longrightarrow N)$ & $(N \longrightarrow)$. We only need to show that $N \in \uparrow$. Now if $(N \longrightarrow^* N')$ also $(M \longrightarrow^* N')$ and by hypothesis $(N' \longrightarrow)$ hence $N \in \uparrow$.

$(\nu X.\psi_{\uparrow}(X) \subseteq \uparrow)$.

By definition of $\phi_{\Uparrow}$, for each $X \subseteq \mathcal{P}(Exp)$ we have that $M \in \psi^n_{\uparrow}(X)$ if and only if there exist $N_{n-1} \in \psi^{n-1}_{\uparrow}(X), \ldots, N_1 \in \psi_{\uparrow}(X), N \in X$ such that $M \longrightarrow N_{n-1} \longrightarrow \cdots \longrightarrow N_1 \longrightarrow N$, since transition is deterministic they are unique.
Suppose that M is in $\nu X.\psi_{\uparrow}(X)$, by fixpoint property $\forall n \in \mathbb{N}$ we have $\nu X.\psi_{\uparrow}(X) = \psi^n_{\uparrow}(\nu X.\psi_{\uparrow})$. Hence $\forall n \in \mathbb{N}$ we have $M \in \psi^n_{\uparrow}(\nu X.\psi_{\uparrow})$ and by the above remarks there exist unique M'_n such that $(M \longrightarrow^n M'_n)$ and $M'_n \in \nu X.\psi_{\uparrow}(X)$ so $M' \longrightarrow$. Thus we have $M \in \uparrow$.

□

Furthermore we can coinductively characterize $\uparrow$ in a different manner which will be useful in the sequel. Taken the operator $\psi^+_{\uparrow} : \mathcal{P}(Exp) \to \mathcal{P}(Exp)$ defined as:

$$\psi^+_{\uparrow}(X) \stackrel{\text{def}}{=} \{M \mid \exists N\ (M \longrightarrow^+ N)\ \&\ N \in X\}$$

we can easily see that it is monotonic. Hence it possess a greatest fixpoint, $\nu X.\psi_{\uparrow}^{+}(X)$ which is the largest $\psi_{\uparrow}^{+}$-dense set and since rules are all *deterministic* we can define it constructively with respect to $\subseteq$ and with *Exp* as basis. The $+$ in $\psi_{\uparrow}^{+}$ means that it can be considered as the transitive closure of $\psi_{\uparrow}$. We can prove the following.

Theorem 4.2.22. $\uparrow = \nu X.\psi_{\uparrow}^{+}(X)$

Proof. By theorem 4.2.21 we have that we only need to prove that $\nu X.\psi_{\uparrow}^{+}(X)$ is $\psi_{\uparrow}$-dense and analogously that $\nu X.\psi_{\uparrow}(X)$ is $\psi_{\uparrow}^{+}$-dense. The latter is immediately by definition of $\psi_{\uparrow}^{+}$ and since $\nu X.\psi_{\uparrow}(X) = \psi_{\uparrow}(\nu X.\psi_{\uparrow}(X))$. For the former take $M \in \nu X.\psi_{\uparrow}^{+}(X)$. By fixpoint property we have that there exists N such that $M \longrightarrow^{n+1} N$ and $N \in \nu X.\psi_{\uparrow}^{+}(X)$. If $n = 0$ we have the conclusion. Otherwise we have that there exists M_1 such that $M \longrightarrow M_1$ and $M_1 \longrightarrow^{n} N$ but hence $M_1 \in \nu X.\psi_{\uparrow}^{+}(X)$ and so the conclusion. □

We sometimes write $M \longrightarrow^{\omega}$ instead of $M \in\uparrow$ when we want to stress the fact that a term $M \in\uparrow$ is the beginning of an infinitely long computation. An easy corollary to theorem 4.2.22 is the following.

Corollary 4.2.23. *If $M \longrightarrow^{*} N$ and $M \longrightarrow^{\omega}$ then $N \longrightarrow^{\omega}$.*

4.2.3 Relating Evaluation and Transition

In the definition of an operational semantics we are interested in giving a partial function from programs to values. This function can be defined as:

$$\{\ (M, C) \mid (M \Downarrow C)\ \}$$

it is partial because not all M evaluate to some C. We quote a phrase of Pitts in [Pitts, 1994] with reference to this partial function:

> The evaluation relation provides a convenient formulation for *reasoning* about properties of this partial function, but it is less convenient for *calculating* the value of the partial function (if any) at particular programs.

So, in order to calculate the value of the partial function for a particular program, we must use another relation instead of evaluation relation.
Transition relation captures the single steps of computations so can be useful for calculating values; furthermore, in the last section, we saw how we can use it to define the notion of convergence. We can think of using evaluation relation in reasoning about properties and transition relation calculating the values of the partial function, but this can be done if and only if the evaluation relation agrees with the transition relation.
We now prove that our definition of evaluation relation is *sound* with respect to transition relation. We only need to remember that in order to capture computation we need the reflexive-transitive closure of the transition relation and that canonical terms do not reduce.

Theorem 4.2.24 (Soundness).

$$\Downarrow \subseteq \downarrow$$

Proof. We want to show that for all $(M, C) \in \Downarrow$ then $(M, C) \in \downarrow$. This can be rewritten as:

$$(M \Downarrow C) \implies (M \longrightarrow^* C)$$

We prove this by induction on height of evaluation trees depending on the structure of M.

- $M \equiv C$
  $(C \Downarrow C) \implies (C \longrightarrow^0 C)$

- $M \equiv M_1 M_2$
  By lemma 4.2.3 $(M_1 M_2 \Downarrow C)$ if and only if $(M_1 \Downarrow \lambda x.M')$ and $(M'[M_2/x] \Downarrow C)$. By induction hypothesis $(M_1 \Downarrow \lambda x.M') \implies (M_1 \longrightarrow^* \lambda x.M')$ and $M'[M_2/x] \Downarrow C \implies (M'[M_2/x] \longrightarrow^* C)$, so by (app tran) and transitivity we have the conclusion.

- $M \equiv \mathbf{fst}(M')$
  By lemma 4.2.3 $(\mathbf{fst}(M') \Downarrow C)$ if and only if $(M' \Downarrow \langle M_1, M_2 \rangle)$ and $(M_1 \Downarrow C)$. By induction hypothesis $(M' \Downarrow \langle M_1, M_2 \rangle) \implies (M' \longrightarrow^* \langle M_1, M_2 \rangle)$ and $(M_1 \Downarrow C) \implies (M_1 \longrightarrow^* C)$, so by (fst tran) and transitivity we have the conclusion.

- $M \equiv \mathbf{snd}(M')$
  By lemma 4.2.3 $(\mathbf{snd}(M') \Downarrow C)$ if and only if $(M' \Downarrow \langle M_1, M_2 \rangle)$ and $(M_2 \Downarrow C)$. By induction hypothesis $(M' \Downarrow \langle M_1, M_2 \rangle) \implies (M' \longrightarrow^* \langle M_1, M_2 \rangle)$ and $(M_2 \Downarrow C) \implies (M_2 \longrightarrow^* C)$, so by (snd tran) and transitivity we have the conclusion.

- $M \equiv \mathbf{case}\ M_1\ \mathbf{of}\ \{\mathbf{inl}_{\tau_1,\tau_2}(x_1).M_2 \mid \mathbf{inr}_{\tau_1,\tau_2}(x_2).M_3\}$
  By lemma 4.2.3 we have two cases depending either $(M_1 \Downarrow \mathbf{inl}(M'))$ or $(M_1 \Downarrow \mathbf{inr}(M'))$.
  If $(M_1 \Downarrow \mathbf{inl}(M'))$ then $(\mathbf{case}\ M_1\ \mathbf{of}\ \{\mathbf{inl}_{\tau_1,\tau_2}(x_1).M_2 \mid \mathbf{inr}_{\tau_1,\tau_2}(x_2).M_3\} \Downarrow C) \implies (M_2[M'/x_1] \Downarrow C)$, by induction hypothesis we have $(M_1 \Downarrow \mathbf{inl}(M')) \implies (M_1 \longrightarrow^* \mathbf{inl}(M'))$ and $(M_2[M'/x_1] \Downarrow C) \implies (M_2[M'/x_1] \longrightarrow^* C)$, so by (cond tran inl) and transitivity we have the conclusion.
  Similarly if $(M_1 \Downarrow \mathbf{inr}(M'))$ then $(\mathbf{case}\ M_1\ \mathbf{of}\ \{\mathbf{inl}_{\tau_1,\tau_2}(x_1).M_2 \mid \mathbf{inr}_{\tau_1,\tau_2}(x_2).M_3\} \Downarrow C) \implies (M_3[M'/x_2] \Downarrow C)$, by induction hypothesis we have $(M_1 \Downarrow \mathbf{inr}(M')) \implies (M_1 \longrightarrow^* \mathbf{inr}(M'))$ and $(M_3[M'/x_2] \Downarrow C) \implies (M_3[M'/x_2] \longrightarrow^* C)$, so by (cond tran inr) and transitivity we have the conclusion.

- $M \equiv \mathbf{abs}(M_1)$
  By lemma 4.2.3 $(\mathbf{abs}(M_1) \Downarrow C)$ if and only if $(M_1 \Downarrow C')$ and $C \equiv \mathbf{abs}_{\mu T.\tau}(C'))$. By induction hypothesis $(M_1 \Downarrow C') \implies (M_1 \longrightarrow^* C')$ so by (abs simpl) we have the conclusion.

- $M \equiv \mathbf{rep}(M_1)$
  By lemma 4.2.3 $(\mathbf{rep}(M_1) \Downarrow C)$ if and only if $(M_1 \Downarrow \mathbf{abs}_{\mu T.\tau}(C'))$. By induction hypothesis $(M_1 \Downarrow \mathbf{abs}_{\mu T.\tau}(C')) \implies (M_1 \longrightarrow^* \mathbf{abs}_{\mu T.\tau}(C'))$ so by (rep tran) we have the conclusion.

□

We now want to prove that evaluation relation is *complete* with respect to transition relation but we prove first the following:

Lemma 4.2.25. *For all M, C:*

$$\textit{if } (M \longrightarrow M') \textit{ then } (M' \Downarrow C) \Longrightarrow (M \Downarrow C)$$

Proof. We prove the lemma by induction on the proof of the transition $(M \longrightarrow M')$. To prove the lemma for basic reduction rules we do not need assumption other then $(M' \Downarrow C)$. We proceed by cases:

- $M \equiv (\lambda x.M_1)M_2 \longrightarrow M_1[M_2/x] \equiv M'$
  assume $(M_1[M_2/x] \Downarrow C)$. Then, as $(\lambda x.M_1 \Downarrow \lambda x.M_1)$, an application of rule (+ app) shows that $(\lambda x.M_1)M_2 \Downarrow C)$.

- $M \equiv \mathbf{fst}(\langle M_1, M_2\rangle) \longrightarrow M_1 \equiv M'$
  assume $(M_1 \Downarrow C)$. Then, as $(\langle M_1, M_2\rangle \Downarrow \langle M_1, M_2\rangle)$, an application of rule (+ fst) shows that $(\mathbf{fst}(\langle M_1, M_2\rangle) \Downarrow C)$.

- $M \equiv \mathbf{snd}(\langle M_1, M_2\rangle) \longrightarrow M_2 \equiv M'$
  assume $(M_2 \Downarrow C)$. Then, as $(\langle M_1, M_2\rangle \Downarrow \langle M_1, M_2\rangle)$, an application of rule (+ snd) shows that $(\mathbf{snd}(\langle M_1, M_2\rangle) \Downarrow C)$.

- $M \equiv \mathbf{case}\ \mathbf{inl}(M')\ \mathbf{of}\ \{\mathbf{inl}_{\tau_1,\tau_2}(x_1).M_1 \mid \mathbf{inr}_{\tau_1,\tau_2}(x_2).M_2\} \longrightarrow M_1[M'/x_1] \equiv M'$
  assume $(M_1[M'/x_1] \Downarrow C)$. Then, as $(\mathbf{inl}(M') \Downarrow \mathbf{inl}(M'))$, an application of rule (+ cond inl) shows that $(\mathbf{case}\ \mathbf{inl}(M')\ \mathbf{of}\ \{\mathbf{inl}_{\tau_1,\tau_2}(x_1).M_1 \mid \mathbf{inr}_{\tau_1,\tau_2}(x_2).M_2\} \Downarrow C)$.

- $M \equiv \mathbf{case}\ \mathbf{inr}(M')\ \mathbf{of}\ \{\mathbf{inl}_{\tau_1,\tau_2}(x_1).M_1 \mid \mathbf{inr}_{\tau_1,\tau_2}(x_2).M_2\} \longrightarrow M_2[M'/x_2] \equiv M'$
  assume $(M_2[M'/x_2] \Downarrow C)$. Then, as $(\mathbf{inr}(M') \Downarrow \mathbf{inr}(M'))$, an application of rule (+ cond inr) shows that $(\mathbf{case}\ \mathbf{inr}(M')\ \mathbf{of}\ \{\mathbf{inl}_{\tau_1,\tau_2}(x_1).M_1 \mid \mathbf{inr}_{\tau_1,\tau_2}(x_2).M_2\} \Downarrow C)$,

- $M \equiv \mathbf{rep}(\mathbf{abs}_{\mu T.\tau}(C)) \longrightarrow C \equiv M'$
  $(\mathbf{abs}_{\mu T.\tau}(C) \Downarrow \mathbf{abs}_{\mu T.\tau}(C))$, then an application of rule (+ rep) shows that $(\mathbf{rep}(\mathbf{abs}_{\mu T.\tau}(C)) \Downarrow C))$.

To prove the reduction step rules we add an assumption. We assume the lemma is valid for $(N \longrightarrow N')$, the premisses of the rules.

- $M \equiv NM_1 \longrightarrow N'M_1 \equiv M'$
  assume $(N'M_1 \Downarrow C)$, then the only possibility is that $(N' \Downarrow \lambda x.M_2)$ and $(M_2[M_1/x] \Downarrow C)$. By induction hypothesis $(N \Downarrow \lambda x.M_2)$ and an application of rule (+ app) shows that $(N(M_1) \Downarrow C)$.

- $M \equiv \mathbf{fst}(N) \longrightarrow \mathbf{fst}(N') \equiv M'$
  assume $(\mathbf{fst}(N') \Downarrow C)$, then the only possibility is that $(N' \Downarrow \langle M_1, M_2\rangle)$ and $(M_1 \Downarrow C)$. By induction hypothesis $(N \Downarrow \langle M_1, M_2\rangle)$ and an application of rule (+ fst) shows that $\mathbf{fst}(N) \Downarrow C)$.

- $M \equiv \mathbf{snd}(N) \longrightarrow \mathbf{snd}(N') \equiv M'$
  assume $(\mathbf{snd}(N') \Downarrow C)$, then the only possibility is that $(N' \Downarrow \langle M_1, M_2\rangle)$ and $(M_2 \Downarrow C)$. By induction hypothesis $(N \Downarrow \langle M_1, M_2\rangle)$ and an application of rule (+ snd) shows that $\mathbf{snd}(N) \Downarrow C)$.

- $M \equiv \mathbf{case}\ N\ \mathbf{of}\ \{\mathbf{inl}_{\tau_1,\tau_2}(x_1).M_1 \mid \mathbf{inr}_{\tau_1,\tau_2}(x_2).M_2\} \longrightarrow \mathbf{case}\ N'\ \mathbf{of}\ \{\mathbf{inl}_{\tau_1,\tau_2}(x_1).M_1 \mid \mathbf{inr}_{\tau_1,\tau_2}(x_2).M_2\} \equiv M'$
  assume $(\mathbf{case}\ N'\ \mathbf{of}\ \{\mathbf{inl}_{\tau_1,\tau_2}(x_1).M_1 \mid \mathbf{inr}_{\tau_1,\tau_2}(x_2).M_2\} \Downarrow C)$, then we have only two possibilities: $(N' \Downarrow \mathbf{inl}(N_1)\ \&\ (M_1[N_1/x_1] \Downarrow C)$ or $(N' \Downarrow \mathbf{inr}(N_1)\ \&\ (M_2[N_1/x_2] \Downarrow C)$.
  If $(N' \Downarrow \mathbf{inl}(N_1)\ \&\ (M_1[N_1/x_1] \Downarrow C)$, by induction hypothesis $(N \Downarrow \mathbf{inl}(N_1))$ and an application of rule (+ cond inl) shows that $(\mathbf{case}\ N\ \mathbf{of}\ \{\mathbf{inl}_{\tau_1,\tau_2}(x_1).M_1 \mid \mathbf{inr}_{\tau_1,\tau_2}(x_2).M_2\} \Downarrow C)$.
  Otherwise if $(N' \Downarrow \mathbf{inr}(N_1)\ \&\ (M_2[N_1/x_2] \Downarrow C)$, by induction hypothesis $(N \Downarrow \mathbf{inr}(N_1)$ and an application of rule (+ cond inr) shows that $(\mathbf{case}\ N\ \mathbf{of}\ \{\mathbf{inl}_{\tau_1,\tau_2}(x_1).M_1 \mid \mathbf{inr}_{\tau_1,\tau_2}(x_2).M_2\} \Downarrow C)$.
- $M \equiv \mathbf{rep}(N) \longrightarrow \mathbf{rep}(N') \equiv M'$
  assume $(\mathbf{rep}(N') \Downarrow C)$, then the only possibility is that $(N' \Downarrow \mathbf{abs}_{\mu T.\tau}(N_1))$ and $(N_1 \Downarrow C)$. By induction hypothesis $(N \Downarrow \mathbf{abs}_{\mu T.\tau}(N_1))$ and an application of rule (+ rep) shows that $(\mathbf{rep}(N) \Downarrow C)$.
- $M \equiv \mathbf{abs}(N) \longrightarrow \mathbf{abs}(N') \equiv M'$
  assume $(\mathbf{abs}(N') \Downarrow C)$, then the only possibility is that $(N' \Downarrow C')$ and $(C \equiv \mathbf{abs}(C'))$. By induction hypothesis $(N \Downarrow C')$ and an application of rule (+ abs) shows that $(\mathbf{abs}(N) \Downarrow C)$.

□

We can now prove completeness theorem.

Theorem 4.2.26 (Completeness).

$$\downarrow \subseteq \Downarrow$$

Proof. We want to show that for all $(M, N) \in \downarrow$ we have $(M, N) \in \Downarrow$. We know that if $(M, N) \in \downarrow$ then $\neg(N \longrightarrow)$ and remembering that the class of terms which do not reduce are exactly terms in canonical form, we can rewrite the thesis as:

$$(M \longrightarrow^* C) \implies (M \Downarrow C)$$

We prove this by induction on the number of step:

Base The base case is trivial.
$(M \longrightarrow^0 C) \Longrightarrow (M \equiv C)$, so we have $C \Downarrow C$.

Inductive step We assume that for all M, C if $(M \longrightarrow^n C)$ then $(M \Downarrow C)$ and we want to show that if $(M \longrightarrow^{n+1} C)$ then $(M \Downarrow C)$.
If $(M \longrightarrow^{n+1} C)$ then by transitivity $(M \longrightarrow M' \longrightarrow^n C)$ so we have $(M \longrightarrow M')$ and by induction hypothesis $(M' \Downarrow C)$. By lemma 4.2.25 we obtain $(M \Downarrow C)$

□

Corollary 4.2.27.

$$\Downarrow = \downarrow$$

Now we have two relations which capture convergence, one useful in reasoning about programs, the other useful in calculating the value of the programs. Furthermore, we have an inductive characterization of evaluation relation in terms of transition relation. This justifies definition 4.2.17.
We are interested in finding out a direct correspondence beetween the proofs of convergence through the two relations.

4.2.4 Relating divergence and transition

We have defined divergence coinductively through big-step rules, we have also characterized divergence coinductively through transition relation. We now want to show that the two definitions agree. We first show that the big-step coinductive definition of divergence is sound with respect to divergence coinductively defined by transition steps.

Theorem 4.2.28 (Soundness).

$$\Uparrow \subseteq \uparrow$$

Proof. We know that $\uparrow$ is defined coinductively as $\nu X.\psi_{\uparrow}(X)$ so by coinduction we only need to show that $\Uparrow$ is $\psi_{\uparrow}$-dense. Remembering the definition of $\psi_{\uparrow}$, we want to show:

$$\Uparrow \subseteq \ \{M \mid \exists N \ (M \longrightarrow N) \ \& \ N \in \Uparrow\}$$

which can be rewritten:

$$M \Uparrow \implies \ (\exists N \ (M \longrightarrow N) \ \& \ N \in \Uparrow)$$

Assume $M \Uparrow$, we prove this by induction on the structure of M:

$M \equiv M_1 M_2$ we have two distinct cases:

- $(M_1 \Uparrow)$. By induction hypothesis $(M_1 \longrightarrow M_1')$ and $M_1' \Uparrow$, from (red tran) we have $(M_1 M_2 \longrightarrow M_1' M_2)$ and an application of $(-$ app) shows that $M_1' M_2 \Uparrow$.
- $(M_1 \Downarrow \lambda x.M)$ & $(M[M_2/x] \Uparrow)$. We have again two sub-cases: if $M_1 \equiv \lambda x.M'$ then by (app tran) $(\lambda x.M' M_2 \longrightarrow M[M_2/x])$ and so the conclusion; otherwise $(M_1 \longrightarrow M_1')$, by lemma 4.2.25 we have $M_1' \Downarrow \lambda x.M'$ and an application of rule$(-$ app lambda) shows that $M_1' M_2 \Uparrow$.

$M \equiv \mathbf{fst}(M')$ we have two distinct cases:

- $(M' \Uparrow)$. By induction hypothesis $(M' \longrightarrow M'')$ and $M'' \Uparrow$, from (red tran) we have $(\mathbf{fst}(M') \longrightarrow \mathbf{fst}(M''))$ and an application of $(-$ fst) shows that $\mathbf{fst}(M'') \Uparrow$.
- $(M' \Downarrow \langle M_1, M_2 \rangle)$ & $(M_1 \Uparrow)$. We have again two sub-cases: if $(M' \equiv \langle M_1, M_2 \rangle)$ then by (fst tran) $(\mathbf{fst}(\langle M_1, M_2 \rangle) \longrightarrow M_1)$ and so the conclusion; otherwise $(M' \longrightarrow M'')$, by lemma 4.2.25 we have $(M'' \Downarrow \langle M_1, M_2 \rangle)$ and an application of rule $(-$ fst pair) shows that $\mathbf{fst}(M'') \Uparrow$.

$M \equiv \mathbf{snd}(M')$ we have two distinct cases:

- $(M' \Uparrow)$. By induction hypothesis $(M' \longrightarrow M'')$ and $M'' \Uparrow$, from (red tran) we have $(\mathbf{snd}(M') \longrightarrow \mathbf{snd}(M''))$ and an application of rule $(-$ snd) shows that $\mathbf{snd}(M'') \Uparrow$.
- $(M' \Downarrow \langle M_1, M_2 \rangle)$ & $(M_2 \Uparrow)$. We have again two sub-cases: if $(M' \equiv \langle M_1, M_2 \rangle)$ then by (snd tran) $(\mathbf{snd}(\langle M_1, M_2 \rangle) \longrightarrow M_2)$ and so the conclusion; otherwise $(M' \longrightarrow M'')$, by lemma 4.2.25 we have $(M'' \Downarrow \langle M_1, M_2 \rangle)$ and an application $(-$ snd pair) shows that $\mathbf{snd}(M'') \Uparrow$.

$M \equiv \mathbf{case}\ M'\ \mathbf{of}\ \{\mathbf{inl}_{\tau_1,\tau_2}(x_1).M_1 \mid \mathbf{inr}_{\tau_1,\tau_2}(x_2).M_2\}$ we have three distinct cases:

- $(M' \Uparrow)$. By induction hypothesis $(M' \longrightarrow M'')$ and $M'' \Uparrow$, from (red tran) we have $(\mathbf{case}\ M'\ \mathbf{of}\ \{\mathbf{inl}_{\tau_1,\tau_2}(x_1).M_1 \mid \mathbf{inr}_{\tau_1,\tau_2}(x_2).M_2\} \longrightarrow \mathbf{case}\ M''\ \mathbf{of}\ \{\mathbf{inl}_{\tau_1,\tau_2}(x_1).M_1 \mid \mathbf{inr}_{\tau_1,\tau_2}(x_2).M_2\})$ and an application of rule $(-$ cond) shows that $\mathbf{case}\ M''\ \mathbf{of}\ \{\mathbf{inl}_{\tau_1,\tau_2}(x_1).M_1 \mid \mathbf{inr}_{\tau_1,\tau_2}(x_2).M_2\} \Uparrow$.
- $(M' \Downarrow \mathbf{inl}(M'')) \ \&\ (M_1[M''/x_1] \Uparrow)$. We have again two sub-cases: if $(M' \equiv \mathbf{inl}(M''))$ then by (cond tran inl) $(\mathbf{case}\ \mathbf{inl}(M'')\ \mathbf{of}\ \{\mathbf{inl}_{\tau_1,\tau_2}(x_1).M_1 \mid \mathbf{inr}_{\tau_1,\tau_2}(x_2).M_2\} \longrightarrow M_1[M''/x_1])$ and so the conclusion; otherwise $(M' \longrightarrow \overline{M})$, by lemma 4.2.25 we have $(\overline{M} \Downarrow \mathbf{inl}(M''))$ and an application of $(-$ cond inl) shows that $\mathbf{case}\ \overline{M}\ \mathbf{of}\ \{\mathbf{inl}_{\tau_1,\tau_2}(x_1).M_1 \mid \mathbf{inr}_{\tau_1,\tau_2}(x_2).M_2\} \Uparrow$.
- $(M' \Downarrow \mathbf{inr}(M'')) \ \&\ (M_2[M''/x_2] \Uparrow)$. We have again two sub-cases: if $(M' \equiv \mathbf{inr}(M''))$ then by (cond tran inr) $(\mathbf{case}\ \mathbf{inr}(M'')\ \mathbf{of}\ \{\mathbf{inl}_{\tau_1,\tau_2}(x_1).M_1 \mid \mathbf{inr}_{\tau_1,\tau_2}(x_2).M_2\} \longrightarrow M_2[M''/x_2])$ and so the conclusion; otherwise $(M' \longrightarrow \overline{M})$, by lemma 4.2.25 we have $(\overline{M} \Downarrow \mathbf{inr}(M''))$ and an application of $(-$ cond inr) shows that $\mathbf{case}\ \overline{M}\ \mathbf{of}\ \{\mathbf{inl}_{\tau_1,\tau_2}(x_1).M_1 \mid \mathbf{inr}_{\tau_1,\tau_2}(x_2).M_2\} \Uparrow$.

$M \equiv \mathbf{rep}(M')$ The only possibility is $M' \Uparrow$ hence by induction hypothesis $(M' \longrightarrow M'')\ \&\ M'' \Uparrow$, from (red tran) we have $(\mathbf{rep}(M') \longrightarrow \mathbf{rep}(M''))$ and an application of rule $(-$ rep) shows that $\mathbf{rep}(M'') \Uparrow$.

$M \equiv \mathbf{abs}_{\mu T.\tau}(M')$ The only possibility is $M' \Uparrow$ hence by induction hypothesis $(M' \longrightarrow M'')\ \&\ M'' \Uparrow$, from (red tran) we have $(\mathbf{abs}_{\mu T.\tau}(M') \longrightarrow \mathbf{abs}_{\mu T.\tau}(M''))$ and an application of rule $(-$ abs) shows that $\mathbf{abs}_{\mu T.\tau}(M'') \Uparrow$.

□

Theorem 4.2.29 (Completeness).

$$\uparrow \subseteq \Uparrow$$

Proof. We know that $\Uparrow$ is defined coinductively as $\nu X.\phi_{\Uparrow}(X)$ so by coinduction we only need to show that $\uparrow$ is $\phi_{\Uparrow}$-dense. Remembering the definition of $\phi_{\Uparrow}$, we want to show:

$$\uparrow \subseteq \ \{M \mid \exists\ \frac{M_1 \Downarrow C_1 \cdots M_n \Downarrow C_n\ M_{n+1} \Uparrow}{M \Uparrow}, (n \geq 0)\ \&\ M_{n+1} \in \uparrow\}$$

which can be rewritten:

$$M \longrightarrow^{\omega} \Longrightarrow\ (\exists\ \frac{M_1 \Downarrow C_1 \cdots M_n \Downarrow C_n\ M_{n+1} \Uparrow}{M \Uparrow}, (n \geq 0)\ \&\ M_{n+1} \longrightarrow^{\omega})$$

We prove this by induction on the structure of M. Assume $M \longrightarrow^{\omega}$.

$M \equiv M_1 M_2$ to prove that it is $\phi_{\Uparrow}$-dense we need to show that either $M_1 \longrightarrow^{\omega}$ or there exists $\lambda x.M'$ such that $M_1 \Downarrow \lambda x.M'$ and $M'[M_2/x] \longrightarrow^{\omega}$. Since $M_1 M_2 \longrightarrow^{\omega}$ we have two cases:

- $(M_1 M_2 \longrightarrow M_1' M_2)$. By lemma 4.2.19 either $M_1 \longrightarrow^\omega$ or there exists $\lambda x.M' \in \mathit{Can}$ such that $M_1 \longrightarrow^+ \lambda x.M'$. If the former, the conclusion follows immediately. Otherwise by (red tran) rule we have $M_1 M_2 \longrightarrow^+ \lambda x.M' M_2$ but by (app tran) rule $\lambda x.M' M_2 \longrightarrow M'[M_2/x]$ hence $M_1 M_2 \longrightarrow^+ M'[M_2/x]$ and by corollary 4.2.23 $M'[M_2/x] \longrightarrow^\omega$, the conclusion.
- $(M_1 M_2 \longrightarrow M'[M_2/x]$ where $M_1 \equiv \lambda x.M')$. By corollary 4.2.23 the conclusion follows immediately.

$M \equiv \mathbf{fst}(M')$ to prove that it is $\phi_{\Uparrow}$-dense we need to show that either $M' \longrightarrow^\omega$ or there exists $\langle M_1, M_2\rangle$ such that $M' \Downarrow \langle M_1, M_2\rangle$ and $M_1 \longrightarrow^\omega$. Since $\mathbf{fst}(M') \longrightarrow^\omega$ we have two cases:

- $(\mathbf{fst}(M') \longrightarrow \mathbf{fst}(M''))$. By lemma 4.2.19 either $M' \longrightarrow^\omega$ or there exists $\langle M_1, M_2\rangle \in \mathit{Can}$ such that $M' \longrightarrow^+ \langle M_1, M_2\rangle$. If the former, the conclusion follows immediately. Otherwise by (red tran) rule we have $\mathbf{fst}(M') \longrightarrow^+ \mathbf{fst}(\langle M_1, M_2\rangle)$ and by (fst tran) rule $\mathbf{fst}(\langle M_1, M_2\rangle) \longrightarrow M_1$ hence $\mathbf{fst}(M') \longrightarrow^+ M_1$ and by corollary 4.2.23 $M_1 \longrightarrow^\omega$, the conclusion.
- $(\mathbf{fst}(M') \longrightarrow M_1$ where $M' \equiv \langle M_1, M_2\rangle)$. By corollary 4.2.23 the conclusion follows immediately.

$M \equiv \mathbf{snd}(M')$ to prove that it is $\phi_{\Uparrow}$-dense we need to show that either $M' \longrightarrow^\omega$ or there exists $\langle M_1, M_2\rangle$ such that $M' \Downarrow \langle M_1, M_2\rangle$ and $M_2 \longrightarrow^\omega$. Since $\mathbf{snd}(M') \longrightarrow^\omega$ we have two cases:

- $(\mathbf{snd}(M') \longrightarrow \mathbf{snd}(M''))$. By lemma 4.2.19 either $M' \longrightarrow^\omega$ or there exists $\langle M_1, M_2\rangle \in \mathit{Can}$ such that $M' \longrightarrow^+ \langle M_1, M_2\rangle$. If the former, the conclusion follows immediately. Otherwise by (red tran) rule we have $\mathbf{snd}(M') \longrightarrow^+ \mathbf{snd}(\langle M_1, M_2\rangle)$ and by (snd tran) rule $\mathbf{snd}(\langle M_1, M_2\rangle) \longrightarrow M_2$ hence $\mathbf{snd}(M') \longrightarrow^+ M_2$ and by corollary 4.2.23 $M_2 \longrightarrow^\omega$, the conclusion.
- $(\mathbf{snd}(M') \longrightarrow M_2$ where $M' \equiv \langle M_1, M_2\rangle)$. By corollary 4.2.23 the conclusion follows immediately.

$M \equiv \mathbf{case}\ M'\ \mathbf{of}\ \{\mathbf{inl}_{\tau_1,\tau_2}(x_1).M_1 \mid \mathbf{inr}_{\tau_1,\tau_2}(x_2).M_2\}$ to prove that it is $\phi_{\Uparrow}$-dense we need to show that either $M' \longrightarrow^\omega$ or there exists $\mathbf{inl}(M'') \in \mathit{Can}$ such that $M' \Downarrow \mathbf{inl}(M'')$ and $M_1[M''/x_1] \longrightarrow^\omega$ or there exists $\mathbf{inr}(M'') \in \mathit{Can}$ such that $M' \Downarrow \mathbf{inr}(M'')$ and $M_2[M''/x_2] \longrightarrow^\omega$. Since $\mathbf{fst}(M') \longrightarrow^\omega$ we have three cases:

- $(\mathbf{case}\ M'\ \mathbf{of}\ \{\mathbf{inl}_{\tau_1,\tau_2}(x_1).M_1 \mid \mathbf{inr}_{\tau_1,\tau_2}(x_2).M_2\} \longrightarrow \mathbf{case}\ M''\ \mathbf{of}\ \{\mathbf{inl}_{\tau_1,\tau_2}(x_1).M_1 \mid \mathbf{inr}_{\tau_1,\tau_2}(x_2).M_2\})$. By lemma 4.2.19 either $M' \longrightarrow^\omega$ or there exists unique $C \in \mathit{Can}$ such that $M' \longrightarrow^+ C$. In the former case we have the conclusion. In the latter we have $(\mathbf{case}\ M'\ \mathbf{of}\ \{\mathbf{inl}_{\tau_1,\tau_2}(x_1).M_1 \mid \mathbf{inr}_{\tau_1,\tau_2}(x_2).M_2\} \longrightarrow^+ \mathbf{case}\ C\ \mathbf{of}\ \{\mathbf{inl}_{\tau_1,\tau_2}(x_1).M_1 \mid \mathbf{inr}_{\tau_1,\tau_2}(x_2).M_2\})$ where C can be either $\mathbf{inl}(M'')$ or $\mathbf{inl}(M'')$ for some M''. If $C \equiv \mathbf{inl}(M'')$ then $\mathbf{case}\ C\ \mathbf{of}\ \{\mathbf{inl}_{\tau_1,\tau_2}(x_1).M_1 \mid \mathbf{inr}_{\tau_1,\tau_2}(x_2).M_2\}) \longrightarrow M_1$ and so by transitivity and by corollary 4.2.23 we have the conclusion. Otherwise $C \equiv \mathbf{inr}(M'')$ then

case C **of** $\{\mathbf{inl}_{\tau_1,\tau_2}(x_1).M_1 \mid \mathbf{inr}_{\tau_1,\tau_2}(x_2).M_2\}) \longrightarrow M_2$ and again by transitivity and by corollary 4.2.23we have the conclusion.

- (**case** M' **of** $\{\mathbf{inl}_{\tau_1,\tau_2}(x_1).M_1 \mid \mathbf{inr}_{\tau_1,\tau_2}(x_2).M_2\} \longrightarrow M_1[M''/x_1]$ where $M' \equiv \mathbf{inl}(M'')$). By corollary 4.2.23 the conclusion follows immediately.
- (**case** M' **of** $\{\mathbf{inl}_{\tau_1,\tau_2}(x_1).M_1 \mid \mathbf{inr}_{\tau_1,\tau_2}(x_2).M_2\} \longrightarrow M_2[M''/x_2]$ where $M' \equiv \mathbf{inr}(M'')$. By corollary 4.2.23 the conclusion follows immediately.

$M \equiv \mathbf{rep}(M')$ to prove that it is $\phi_{\Uparrow}$-dense we need to show that $M' \longrightarrow^{\omega}$. By lemma 4.2.19 either $M' \longrightarrow^{\omega}$ or there exists $\mathbf{abs}_{\mu T.\tau}(C) \in \mathit{Can}$ such that $M' \longrightarrow^{+} \mathbf{abs}_{\mu T.\tau}(C)$. If the former, the conclusion follows immediately. Otherwise by theorem 4.2.26 and (+ rep) rule we have $\mathbf{rep}(M') \Downarrow C$ which contradict the assumption that $\mathbf{rep}(M') \longrightarrow^{\omega}$.

$M \equiv \mathbf{abs}_{\mu T.\tau}(M')$ to prove that it is $\phi_{\Uparrow}$-dense we need to show that $M' \longrightarrow^{\omega}$. By lemma 4.2.19 either $M' \longrightarrow^{\omega}$ or there exists $C \in \mathit{Can}$ such that $M' \longrightarrow^{+} C$. If the former, the conclusion follows immediately. Otherwise by theorem 4.2.26 and (+ abs) rule we have $\mathbf{abs}_{\mu T.\tau}(M') \Downarrow \mathbf{abs}_{\mu T.\tau}(C)$ which contradict the assumption that $\mathbf{rep}(M') \longrightarrow^{\omega}$.

□

By soundness and completeness we have the following corollary:

Corollary 4.2.30.

$$\Uparrow = \uparrow$$

So we have two ways to define coinductively divergence. We note that the above corollary justifies the fact that in our treatment of divergence we have not considered type informations.
As for the case of convergence, one relation is useful to reason about properties of divergent terms and the other describes better infinite computation paths. Again we are interested in finding out a direct correspondence beetween the proofs of divergence through the two relations.

Chapter 5

From Big-step to Small-step

> The thesis is then this: manuals for translating one language into another can be set up in divergent ways, all compatible with the totality of speech dispositions, yet incompatible with one another. In countless places they will diverge in giving, as their respective translations of a sentence of the one language, sentences of the other language which stand to each other in no plausible sort of equivalence however loose.
>
> W.V. Quine. "Word and Object"

Soundness theorem 4.2.24 shows that if $M \Downarrow C$ then $M \longrightarrow^* C$, soundness theorem 4.2.28 shows that if $M \Uparrow$ then $M \longrightarrow^\omega$. Furthermore lemmas 4.2.4 and 4.2.15 show that both evaluation and transition are deterministic, hence we can suppose that we can map the proof of $M \Downarrow C$ to the proof of $M \longrightarrow^* C$ and the proof of $M \Uparrow$ to the proof of $M \longrightarrow^\omega$ respectively.
We know that the proof by big-step semantics is a possibly infinite tree where the leaves are axioms, hence the only essential assumptions for our proof are the axioms, in particular since the only axiom scheme is (+ can) we are only interested in canonical terms.
Moreover we have seen that our big-step rules are instances of *L-rules*, this implies that the order of occurrence of canonical terms is important. So we can suppose that we can characterize a proof by big-step semantics by the sequence of the axioms which occur in its proof tree. Now we want to formalize this intuition.
The sequel is a refinement of the method proposed by Ibraheem and Schmidt in [Ibraheem and Schmidt, 2000].

5.1 Proof Assumptions

For convergence judgement we can give the following definition.

Definition 5.1.1. *The* ***convergence proof assumption*** *of a convergence judgement* $M \Downarrow C$

$$HP(M \Downarrow C) \in Can^+$$

is defined as follow:

- $HP(C \Downarrow C) = \langle C \rangle$
- $HP(M \Downarrow C) = \langle C_1^1, \cdots, C_1^{k_1}, \cdots, C_n^1, \cdots, C_n^{k_n} \rangle$
  if there exists a rule
  $$\frac{M_1 \Downarrow C_1 \cdots M_n \Downarrow C_n}{M \Downarrow C}$$
  such that:
  $$\begin{array}{c} HP(M_1 \Downarrow C_1) = \langle C_1^1, \cdots, C_1^{k_1} \rangle \\ \vdots \\ HP(M_n \Downarrow C_n) = \langle C_n^1, \cdots, C_n^{k_n} \rangle \end{array}$$

where Can^+ is the set of all finite not empty sequences of canonical terms.

Analogously for divergence judgement we have the following.

Definition 5.1.2. *The **divergence proof assumption** of a divergence judgement $M \Uparrow$*
$$HP(M \Uparrow) \in Can^\omega$$
is defined as follow:
$$HP(M \Uparrow) = \langle C_1^1, \cdots, C_1^{k_1}, \cdots, C_n^1, \cdots, C_n^{k_n}, C_{n+1}^1 \cdots C_{n+1}^{k_{n+1}} \cdots \rangle$$
if there exists a rule
$$\frac{M_1 \Downarrow C_1 \cdots M_n \Downarrow C_n \;\; M_{n+1} \Uparrow}{M \Uparrow}$$
such that:
$$\begin{array}{rcl} HP(M_1 \Downarrow C_1) & = & \langle C_1^1, \cdots, C_1^{k_1} \rangle \\ & \vdots & \\ HP(M_n \Downarrow C_n) & = & \langle C_n^1, \cdots, C_n^{k_n} \rangle \\ HP(M_{n+1} \Uparrow) & = & \langle C_{n+1}^1, \cdots, C_{n+1}^{k_{n+1}}, \cdots \rangle \end{array}$$

where Can^ω is the set of all infinite sequences of canonical terms.

Hence we can define a total function $HP : Exp \to Can^{\leq \omega}$, where $Can^{\leq \omega}$ is the set of all non empty finite and infinite sequences of canonical terms.

Definition 5.1.3. *The **proof assumption** of a term $M \in Exp$*
$$HP(M) \in Can^{\leq \omega}$$
is defined as:

- $HP(M) = HP(M \Downarrow C)$ *if* $M \Downarrow C$
- $HP(M) = HP(M \Uparrow)$ *if* $M \Uparrow$

The set of all proof assumptions is denoted by *Hyp*. It is easy to prove the following lemma.

Lemma 5.1.4. *For each $M \in Exp$ and $S_1, S_2 \in Can^{\leq\omega}$, if $HP(M) = S_1$ and $HP(M) = S_2$ then $S_1 = S_2$.*

Proof. We first consider the case where there exists $C \in Can$ such that $M \Downarrow C$ and we prove the statement by induction.
The case $HP(C \Downarrow C)$ is trivial. Suppose $HP(M \Downarrow C) = \langle C_1, \cdots, C_n\rangle$ and $HP(M \Downarrow C) = \langle C'_1, \cdots, C'_n\rangle$, by definition of convergence proof assumption we have that there exist:

$$\frac{M''_1 \Downarrow C''_1 \cdots M''_n \Downarrow C''_n}{M \Downarrow C} \qquad \frac{M'''_1 \Downarrow C'''_1 \cdots M'''_m \Downarrow C'''_m}{M \Downarrow C}$$

but by Inversion (lemma 4.2.3) we have that $n = m$ and:

$$M''_1 \equiv M'''_1, C''_1 \equiv C'''_1, \cdots, M''_n \equiv M'''_n, C''_n \equiv C'''_n$$

By induction hypothesis we have:

$$\begin{aligned} HP(M''_1 \Downarrow C''_1) = \langle C^1_1, \cdots, C^{k_1}_1\rangle = HP(M'''_1 \Downarrow C'''_1) \\ \vdots \\ HP(M''_n \Downarrow C''_n) = \langle C^1_n, \cdots, C^{k_n}_n\rangle = HP(M'''_n \Downarrow C'''_n) \end{aligned}$$

whence the conclusion.
We now treat the case where $M \Uparrow$. We prove it by an analysis of the case and by using the fact that $\Uparrow = \bigcap_{n\in\mathbb{N}} \phi^n_{\Uparrow}(Exp)$. Suppose $HP(M \Uparrow) = \langle C_1, \cdots, C_n, \cdots\rangle$ and $HP(M \Uparrow) = \langle C'_1, \cdots, C'_n, \cdots\rangle$. By definition of divergence proof assumption we have that there exist:

$$\frac{M''_1 \Downarrow C''_1 \cdots M''_n \Downarrow C''_n \quad \begin{array}{c}\vdots\\ M''_{n+1} \Uparrow\end{array}}{M \Uparrow} \qquad \frac{M'''_1 \Downarrow C'''_1 \cdots M'''_m \Downarrow C'''_m \quad \begin{array}{c}\vdots\\ M'''_{n+1} \Uparrow\end{array}}{M \Uparrow}$$

such that

$$\begin{aligned} HP(M''_1 \Downarrow C''_1) = \langle C^1_1, \cdots, C^{k_1}_1\rangle \qquad & HP(M'''_1 \Downarrow C'''_1) = \langle \hat{C}^1_1, \cdots, \hat{C}^{k_1}_1\rangle \\ \vdots & \\ HP(M''_n \Downarrow C''_n) = \langle C^1_n, \cdots, C^{k_n}_n\rangle \qquad & HP(M'''_n \Downarrow C'''_n) = \langle \hat{C}^1_n, \cdots, \hat{C}^{k_n}_n\rangle \\ HP(M''_{n+1} \Uparrow) = \langle C^1_{n+1}, \cdots, C^{k_{n+1}}_{n+1}, \cdots\rangle \qquad & HP(M'''_{n+1}) = \langle \hat{C}^1_{n+1}, \cdots, \hat{C}^{k_{n+1}}_{n+1}, \cdots\rangle \end{aligned}$$

but by Inversion lemma 4.2.7 we have that $n = m$ and:

$$M''_1 \equiv M'''_1, C''_1 \equiv C'''_1, \cdots, M''_n \equiv M'''_n, C''_n \equiv C'''_n, M''_{n+1} \equiv M'''_{n+1}$$

By rules analysis we have two cases: either $n = 0$ or $n = 1$. If $n = 0$ then by rule analysis we can apply inductive hypothesis to M''_{n+1} and hence the conclusion follows. Otherwise by the previous case we have:

$$HP(M''_1 \Downarrow C''_1) = \langle C^1_1, \cdots, C^{k_1}_1\rangle = HP(M'''_1 \Downarrow C'''_1)$$

and

$$HP(M''_{n+1} \Uparrow) = \langle C^1_{n+1}, \cdots, C^{k_{n+1}}_{n+1}, \cdots\rangle \qquad HP(M''_{n+1}) = \langle \hat{C}^1_{n+1}, \cdots, \hat{C}^{k_{n+1}}_{n+1}, \cdots\rangle$$

We can repeatedly apply the same argument, hence the conclusion follows. □

The definition of proof assumptions toghether with the above lemma assure that for each $M \in Exp$ there exists a unique proof assumption $HP(M)$. Furthermore one can try to prove a converse statement like the following:

> For each $(M, C), (M', C') \in \Downarrow,\ S \in Can^{\leq \omega}$ if $HP(M \Downarrow C) = S$ and $HP(M' \Downarrow C') = S$ then $(M, C) \equiv (M', C')$.

Unfortunately this fails as shown for example by

$$HP(\mathbf{abs}_\tau((\lambda y : \alpha.\lambda x : \tau.\mathbf{rep}(x)x)\Omega^\alpha)) = HP((\lambda y : \alpha.\lambda x : \tau.\mathbf{rep}(x)x)\Omega^\alpha)$$

in the case of convergence and by

$$HP(\Omega^{\tau_1 \times \tau_2}) = HP(\mathbf{fst}(\Omega^{\tau_1 \times \tau_2})) = HP(\mathbf{snd}(\Omega^{\tau_1 \times \tau_2}))$$

in the case of divergence. Hence we need to add more information to proof assumption in order to consider equivalently a term or its proof assumption. We will introduce in the next paragraph proof assumption with context information. Now to make our map explicit we need a notion of *length* of a proof assumption.

Definition 5.1.5. *The **length** of a proof assumption is a function* $len : Hyp \to \mathbb{N} \cup \{\omega\}$ *defined for all* $M \in Exp$ *as:*

- $len(HP(M)) = (n-1)$ *if* $\exists C \in Can\ (M \Downarrow C)\ \&\ HP(M) = \langle C_1, \cdots, C_n \rangle$
- $len(HP(M)) = \omega$ *if* $M \Uparrow$

Note that length differs from the height of proof trees. We want to prove the following propositions which state that the length of a proof assumption is the number of small steps in the proof by small-steps semantics.

Proposition 5.1.6.

Let $M \in Exp$; *if there exists* $n \in \mathbb{N}$ *such that* $len(HP(M)) = n$ *then* $M \longrightarrow^n C$

Proposition 5.1.7.

$$\textit{Let } M \in Exp \textit{ if } len(HP(M)) = \omega \textit{ then } M \longrightarrow^\omega$$

We are not only interested in proving the propositions, which can be easily done by means of a *cost semantics* as in [Harper, 2003], but also in the method to prove them. We will see that it gives us an algorithm to translate big-step into small-step proofs.
Before proving the above propositions we add context information to evaluation and divergence rules.

5.2 Judgements with context information

In our definition of transition relation we used the notion of evaluation context; rule (red tran) states that reductions are preserved in evaluation context. We now want to consider an analogous statement also for the evaluation relation. We add to convergence and divergence judgements some context information. Context informations are not interesting for the computation but only for the traslation from big-step to small-step. To do this we can use the notions of

experiment and of *evaluation context* already introduced in 4.2.9 and 4.2.10 respectively.
We can now define two new kinds of judgement. We have *evaluation with context information judgement*:

$$\vec{\mathcal{E}} \vdash M \Downarrow C$$

where $\vec{\mathcal{E}}$ is an evaluation context and $M \Downarrow C$ is a convergence judgement and *divergence with context information judgement*

$$\vec{\mathcal{E}} \vdash M \Uparrow$$

where $\vec{\mathcal{E}}$ is an evaluation context and $M \Uparrow$ is a divergence judgement.
Since $\vec{\mathcal{E}}[C]$ is not always a canonical term we have that a judgement $\vec{\mathcal{E}} \vdash M \Downarrow C$ does not mean that $\vec{\mathcal{E}}[M] \Downarrow \vec{\mathcal{E}}[C]$ but, by remembering corollary 4.2.27, only that $\vec{\mathcal{E}}[M] \longrightarrow^* \vec{\mathcal{E}}[C]$. Hence we can consider judgements with context information as "we are in the evaluation context $\vec{\mathcal{E}}$ and the term M evaluates to C" and "we are in the evaluation context $\vec{\mathcal{E}}$ and the term M diverges " for evaluation and divergence respectively.
By these observations we obtain rules in tables 5.1 and 5.2. We note that in the definition of rule (ctx rep) we added the assumption ($\vec{\mathcal{E}} \vdash C \Downarrow C$). The same assumption without context information has been used implicitly in the rule (+ rep), we decided to explicit it now because we need the next structural properties.

Lemma 5.2.1 (+ Structural). *For each instance of + rules with context information of table 5.1:*

$$\frac{\vec{\mathcal{E}} \circ \mathcal{E}_1 \vdash M_1 \Downarrow C_1 \cdots \vec{\mathcal{E}} \circ \mathcal{E}_n \vdash M_n \Downarrow C_n}{\vec{\mathcal{E}} \vdash M \Downarrow C}$$

we have:

(i) $\vec{\mathcal{E}}[M] \equiv \vec{\mathcal{E}}[\mathcal{E}_1[M_1]] \qquad \& \qquad \vec{\mathcal{E}}[C] \equiv \vec{\mathcal{E}}[\mathcal{E}_n[C_n]]$

(ii) $\forall i\ (0 < i < n)\ \vec{\mathcal{E}}[\mathcal{E}_i[C_i]] \longrightarrow \vec{\mathcal{E}}[\mathcal{E}_{i+1}[M_{i+1}]]$

Proof. (i) follows immediately by an ispection of the rules.
(ii) an ispection of the rules shows immediately that for i such that $(0 < i < n)$ each $\mathcal{E}_i[C_i]$ is a redex such that $\mathcal{E}_i[C_i] \longrightarrow \mathcal{E}_{i+1}[M_{i+1}]$ so (red tran) and show the conclusion. □

Lemma 5.2.2 (− Structural). *For each instance of − rules with context information of table 5.2:*

$$\frac{\vec{\mathcal{E}} \circ \mathcal{E}_1 \vdash M_1 \Downarrow C_1 \cdots \vec{\mathcal{E}} \circ \mathcal{E}_n \vdash M_n \quad \vec{\mathcal{E}} \circ \mathcal{E}_{n+1} \vdash M_{n+1} \Uparrow}{\vec{\mathcal{E}} \vdash M \Uparrow}$$

we have:

(i) $\vec{\mathcal{E}}[M] \equiv \vec{\mathcal{E}}[\mathcal{E}_1[M_1]]$

(ii) $\forall i\ (0 < i \leq n)\ \vec{\mathcal{E}}[\mathcal{E}_i[C_i]] \longrightarrow \vec{\mathcal{E}}[\mathcal{E}_{i+1}[M_{i+1}]]$

Proof. (i) follows immediately by an ispection of the rules.
(ii) an ispection of the rules shows immediately that for i such that $(0 < i \leq n)$ each $\mathcal{E}_i[C_i]$ is a redex such that $\mathcal{E}_i[C_i] \longrightarrow \mathcal{E}_{i+1}[M_{i+1}]$ so (red tran) show the conclusion. □

$\dfrac{}{\vec{\mathcal{E}} \vdash C \Downarrow C}$	(+ ctx can)
$\dfrac{\vec{\mathcal{E}} \circ [\bullet] M_2 \vdash M_1 \Downarrow \lambda x.M \qquad \vec{\mathcal{E}} \vdash M[M_2/x] \Downarrow C}{\vec{\mathcal{E}} \vdash M_1 M_2 \Downarrow C}$	(+ ctx app)
$\dfrac{\vec{\mathcal{E}} \circ \mathbf{fst}([\bullet]) \vdash M \Downarrow \langle M_1, M_2 \rangle \qquad \vec{\mathcal{E}} \vdash M_1 \Downarrow C}{\vec{\mathcal{E}} \vdash \mathbf{fst}(M) \Downarrow C}$	(+ ctx fst)
$\dfrac{\vec{\mathcal{E}} \circ \mathbf{snd}([\bullet]) \vdash M \Downarrow \langle M_1, M_2 \rangle \qquad \vec{\mathcal{E}} \vdash M_2 \Downarrow C}{\vec{\mathcal{E}} \vdash \mathbf{snd}(M) \Downarrow C}$	(+ ctx snd)
$\dfrac{\vec{\mathcal{E}} \circ \mathbf{rep}([\bullet]) \vdash M \Downarrow \mathbf{abs}_{\mu T.\tau}(C) \qquad \vec{\mathcal{E}} \vdash C \Downarrow C}{\vec{\mathcal{E}} \vdash \mathbf{rep}(M) \Downarrow C}$	(+ ctx rep)
$\dfrac{\vec{\mathcal{E}} \circ \mathbf{abs}_{\mu T.\tau}([\bullet]) \vdash M \Downarrow C}{\vec{\mathcal{E}} \vdash \mathbf{abs}_{\mu T.\tau}(M) \Downarrow \mathbf{abs}_{\mu T.\tau}(C)}$	(+ ctx abs)
$\dfrac{\vec{\mathcal{E}} \circ \mathbf{case}\ [\bullet]\ \mathbf{of}\ \{\mathbf{inl}_{\tau_1,\tau_2}(x_1).M_1 \mid \mathbf{inr}_{\tau_1,\tau_2}(x_2).M_2\} \vdash M \Downarrow \mathbf{inl}_{\tau_1,\tau_2}(M') \qquad \vec{\mathcal{E}} \vdash M_1[M'/x_1] \Downarrow C}{\vec{\mathcal{E}} \vdash \mathbf{case}\ M\ \mathbf{of}\ \{\mathbf{inl}_{\tau_1,\tau_2}(x_1).M_1 \mid \mathbf{inr}_{\tau_1,\tau_2}(x_2).M_2\} \Downarrow C}$	(+ ctx cond inl)
$\dfrac{\vec{\mathcal{E}} \circ \mathbf{case}\ [\bullet]\ \mathbf{of}\ \{\mathbf{inl}_{\tau_1,\tau_2}(x_1).M_1 \mid \mathbf{inr}_{\tau_1,\tau_2}(x_2).M_2\} \vdash M \Downarrow \mathbf{inr}_{\tau_1,\tau_2}(M') \qquad \vec{\mathcal{E}} \vdash M_2[M'/x_2] \Downarrow C}{\vec{\mathcal{E}} \vdash \mathbf{case}\ M\ \mathbf{of}\ \{\mathbf{inl}_{\tau_1,\tau_2}(x_1).M_1 \mid \mathbf{inr}_{\tau_1,\tau_2}(x_2).M_2\} \Downarrow C}$	(+ ctx cond inr)

Table 5.1: Evaluation Rules with Context Information

$\dfrac{\vec{\mathcal{E}} \circ [\bullet] M_2 \vdash M_1 \Uparrow}{\vec{\mathcal{E}} \vdash M_1 M_2 \Uparrow}$	(− ctx app)
$\dfrac{\vec{\mathcal{E}} \circ [\bullet] M_2 \vdash M_1 \Downarrow \lambda x.M \qquad \vec{\mathcal{E}} \vdash M[M_2/x] \Uparrow}{\vec{\mathcal{E}} \vdash M_1 M_2 \Uparrow}$	(− ctx app lambda)
$\dfrac{\vec{\mathcal{E}} \circ \mathbf{fst}([\bullet]) \vdash M \Uparrow}{\vec{\mathcal{E}} \vdash \mathbf{fst}(M) \Uparrow}$	(− ctx fst)
$\dfrac{\vec{\mathcal{E}} \circ \mathbf{fst}([\bullet]) \vdash M \Downarrow \langle M_1, M_2 \rangle \qquad \vec{\mathcal{E}} \vdash M_1 \Uparrow}{\vec{\mathcal{E}} \vdash \mathbf{fst}(M) \Uparrow}$	(− ctx fst pair)
$\dfrac{\vec{\mathcal{E}} \circ \mathbf{snd}([\bullet]) \vdash M \Uparrow}{\vec{\mathcal{E}} \vdash \mathbf{snd}(M) \Uparrow}$	(− ctx fst)
$\dfrac{\vec{\mathcal{E}} \circ \mathbf{snd}([\bullet]) \vdash M \Downarrow \langle M_1, M_2 \rangle \qquad \vec{\mathcal{E}} \vdash M_2 \Downarrow C}{\vec{\mathcal{E}} \vdash \mathbf{snd}(M) \Downarrow C}$	(− ctx snd pair)
$\dfrac{\vec{\mathcal{E}} \circ \mathbf{rep}([\bullet]) \vdash M \Uparrow}{\vec{\mathcal{E}} \vdash \mathbf{rep}(M) \Uparrow}$	(− ctx rep)
$\dfrac{\vec{\mathcal{E}} \circ \mathbf{abs}_{\mu T.\tau}([\bullet]) \vdash M \Uparrow}{\vec{\mathcal{E}} \vdash \mathbf{abs}_{\mu T.\tau}(M) \Uparrow}$	(− ctx abs)
$\dfrac{\vec{\mathcal{E}} \circ \mathbf{case}\ [\bullet]\ \mathbf{of}\ \{\mathbf{inl}_{\tau_1,\tau_2}(x_1).M_1 \mid \mathbf{inr}_{\tau_1,\tau_2}(x_2).M_2\} \vdash M \Uparrow}{\vec{\mathcal{E}} \vdash \mathbf{case}\ M\ \mathbf{of}\ \{\mathbf{inl}_{\tau_1,\tau_2}(x_1).M_1 \mid \mathbf{inr}_{\tau_1,\tau_2}(x_2).M_2\} \Uparrow}$	(− ctx cond)
$\dfrac{\vec{\mathcal{E}} \circ \mathbf{case}\ [\bullet]\ \mathbf{of}\ \{\mathbf{inl}_{\tau_1,\tau_2}(x_1).M_1 \mid \mathbf{inr}_{\tau_1,\tau_2}(x_2).M_2\} \vdash M \Downarrow \mathbf{inl}_{\tau_1,\tau_2}(M') \qquad \vec{\mathcal{E}} \vdash M_1[M'/x_1] \Uparrow}{\vec{\mathcal{E}} \vdash \mathbf{case}\ M\ \mathbf{of}\ \{\mathbf{inl}_{\tau_1,\tau_2}(x_1).M_1 \mid \mathbf{inr}_{\tau_1,\tau_2}(x_2).M_2\} \Uparrow}$	(− ctx cond inl)
$\dfrac{\vec{\mathcal{E}} \circ \mathbf{case}\ [\bullet]\ \mathbf{of}\ \{\mathbf{inl}_{\tau_1,\tau_2}(x_1).M_1 \mid \mathbf{inr}_{\tau_1,\tau_2}(x_2).M_2\} \vdash M \Downarrow \mathbf{inr}_{\tau_1,\tau_2}(M') \qquad \vec{\mathcal{E}} \vdash M_2[M'/x_2] \Uparrow}{\vec{\mathcal{E}} \vdash \mathbf{case}\ M\ \mathbf{of}\ \{\mathbf{inl}_{\tau_1,\tau_2}(x_1).M_1 \mid \mathbf{inr}_{\tau_1,\tau_2}(x_2).M_2\} \Uparrow}$	(− ctx cond inr)

Table 5.2: Divergence Rules with Context Information

5.3 Proof assumption with context information

In definition 4.2.2 and 4.2.6 we have defined relations without any reference to context. We want that our definition of evaluation relation with context information have exactly the same meaning of evaluation and only add some useful syntactical information.
Now taken the operator $\phi^{ctx}_{\Downarrow} : \mathcal{P}(Ectx \times Exp \times Can) \to \mathcal{P}(Ectx \times Exp \times Can)$ induced by the set of instances of + rules with context information in table 5.1 defined as:

$$\phi^{ctx}_{\Downarrow}(X) \stackrel{\text{def}}{=} \{(\vec{\mathcal{E}}, M, C) \mid \exists \ \frac{\vec{\mathcal{E}}_1 \vdash M_1 \Downarrow C_1 \cdots \vec{\mathcal{E}}_n \vdash M_n \Downarrow C_n}{\vec{\mathcal{E}} \vdash M \Downarrow C}, (n \geq 0)$$
$$\& \ (\vec{\mathcal{E}}_1, M_1, C_1), \cdots, (\vec{\mathcal{E}}_n, M_n, C_n) \in X\}$$

we can easily see that it is monotonic. Hence it possesses a least fixed point $\mu X.\phi^{ctx}_{\Downarrow}(X)$ and since rules are all *finitary* we can define it constructively with respect to $\subseteq$ and with $\emptyset$ as basis.

Definition 5.3.1. ***Evaluation relation with context information*** *is the relation $\Downarrow^{ctx} \subseteq Ectx \times Exp \times Can$ inductively defined as:*

$$\Downarrow^{ctx} = \mu X.\phi^{ctx}_{\Downarrow}(X)$$

we usually write $\vec{\mathcal{E}} \vdash M \Downarrow C$ instead of $(\vec{\mathcal{E}}, M, C) \in \Downarrow^{ctx}$.

Analogously taken the operator $\phi^{ctx}_{\Uparrow} : \mathcal{P}(Ectx \times Exp) \to \mathcal{P}(Ectx \times Exp)$ induced by the set of instances of − rules with context information in table 5.2 defined as:

$$\phi^{ctx}_{\Uparrow}(X) \stackrel{\text{def}}{=} \{(\vec{\mathcal{E}}, M) \mid \exists \ \frac{\vec{\mathcal{E}}_1 \vdash M_1 \Downarrow C_1 \cdots \vec{\mathcal{E}}_n \vdash M_n \Downarrow C_n \ \vec{\mathcal{E}}_{n+1} \vdash M_{n+1}}{\vec{\mathcal{E}} \vdash M \Downarrow C},$$
$$(n \geq 0) \ \& \ (\vec{\mathcal{E}}_1, M_1, C_1), \cdots, (\vec{\mathcal{E}}_n, M_n, C_n) \ \in \Downarrow^{ctx}, (\vec{\mathcal{E}}_{n+1}, M_{n+1}) \in X\}$$

we can easily see that it is monotonic. Hence it possesses a greatest fixed point $\nu X.\phi^{ctx}_{\Downarrow}(X)$ and since rules are all *deterministic* we can define it constructively with respect to $\subseteq$ and with Exp as basis.

Definition 5.3.2. ***Divergence relation with context information*** *is the relation $\Uparrow^{ctx} \subseteq Ectx \times Exp$ coinductively defined as:*

$$\Uparrow^{ctx} = \nu X.\phi^{ctx}_{\Uparrow}(X)$$

we usually write $\vec{\mathcal{E}} \vdash M \Uparrow$ instead of $(\vec{\mathcal{E}}, M) \in \Uparrow^{ctx}$.

It is now easy to verify the following lemmas:

Lemma 5.3.3. *For $M \in Exp$:*

(+i) $M \Downarrow C \Longrightarrow \forall \vec{\mathcal{E}} \ (\vec{\mathcal{E}} \vdash M \Downarrow C)$

(−i) $M \Uparrow \Longrightarrow \forall \vec{\mathcal{E}} : \ (\vec{\mathcal{E}} \vdash M \Uparrow)$

(+ii) *If for some $\vec{\mathcal{E}}_i$ we have $\vec{\mathcal{E}}_i \vdash M \Downarrow C$ then for each $\vec{\mathcal{E}}$: $\vec{\mathcal{E}} \vdash M \Downarrow C$*

(−ii) *If for some* $\vec{\mathcal{E}}_i$ *we have* $\vec{\mathcal{E}}_i \vdash M \Uparrow$ *then for each* $\vec{\mathcal{E}}$: $\vec{\mathcal{E}} \vdash M \Uparrow$

(+iii) *If for some* $\vec{\mathcal{E}}$ *we have* $(\vec{\mathcal{E}} \vdash M \Downarrow C)$ *then* $M \Downarrow C$

(−iii) *If for some* $\vec{\mathcal{E}}$ *we have* $(\vec{\mathcal{E}} \vdash M \Uparrow)$ *then* $M \Uparrow$

Now we can add context information to assumption.

Definition 5.3.4. *The* ***evaluation proof assumption with context information*** *of a convergence judgement with context information* $\vec{\mathcal{E}} \vdash M \Downarrow C$*:*

$$HP(\vec{\mathcal{E}} \vdash M \Downarrow C) \in (Ectx, Can)^+$$

is defined as follow:

- $HP(\vec{\mathcal{E}} \vdash C \Downarrow C) = \langle(\vec{\mathcal{E}}, C)\rangle$
- $HP(\vec{\mathcal{E}} \vdash M \Downarrow C) = \langle(\vec{\mathcal{E}}_1^1, C_1^1), \cdots, (\vec{\mathcal{E}}_1^{k_1}, C_1^{k_1}), \cdots, (\vec{\mathcal{E}}_n^1, C_n^1), \cdots, (\vec{\mathcal{E}}_n^{k_n}, C_n^{k_n})\rangle$ *if there exists a rule:*

$$\frac{\vec{\mathcal{E}}_1 \vdash M_1 \Downarrow C_1 \cdots \vec{\mathcal{E}}_n \vdash M_n \Downarrow C_n}{\vec{\mathcal{E}} \vdash M \Downarrow C}$$

such that:

$$\begin{array}{c} HP(\vec{\mathcal{E}}_1 \vdash M_1 \Downarrow C_1) = \langle(\vec{\mathcal{E}}_1^1, C_1^1), \cdots, (\vec{\mathcal{E}}_1^{k_1}, C_1^{k_1})\rangle \\ \vdots \\ HP(\vec{\mathcal{E}}_n \vdash M_n \Downarrow C_n) = \langle(\vec{\mathcal{E}}_n^1, C_n^1), \cdots, (\vec{\mathcal{E}}_n^{k_n}, C_n^{k_n})\rangle \end{array}$$

Definition 5.3.5. *The* ***divergence proof assumption with context information*** *of a divergence judgement with context information* $\vec{\mathcal{E}} \vdash M \Uparrow$*:*

$$HP(\vec{\mathcal{E}} \vdash M \Uparrow) \in (Ectx, Can)^\omega$$

is defined as follow:

$$\begin{array}{r} HP(\vec{\mathcal{E}} \vdash M \Uparrow) = \langle(\vec{\mathcal{E}}_1^1, C_1^1), \cdots, (\vec{\mathcal{E}}_1^{k_1}, C_1^{k_1}), \cdots, (\vec{\mathcal{E}}_n^1, C_n^1), \cdots \\ \cdots, (\vec{\mathcal{E}}_n^{k_n}, C_n^{k_n}), (\vec{\mathcal{E}}_{n+1}^1, C_{n+1}^1), \cdots, (\vec{\mathcal{E}}_{n+1}^{k_{n+1}}, C_{n+1}^{k_{n+1}}), \cdots\rangle \end{array}$$

if there exists a rule:

$$\frac{\vec{\mathcal{E}}_1 \vdash M_1 \Downarrow C_1 \cdots \vec{\mathcal{E}}_n \vdash M_n \Downarrow C_n \quad \vec{\mathcal{E}}_{n+1} \vdash M_{n+1} \Uparrow}{\vec{\mathcal{E}} \vdash M \Uparrow}$$

such that:

$$\begin{array}{rcl} HP(\vec{\mathcal{E}}_1 \vdash M_1 \Downarrow C_1) & = & \langle(\vec{\mathcal{E}}_1^1, C_1^1), \cdots, (\vec{\mathcal{E}}_1^{k_1}, C_1^{k_1})\rangle \\ & \vdots & \\ HP(\vec{\mathcal{E}}_n \vdash M_n \Downarrow C_n) & = & \langle(\vec{\mathcal{E}}_n^1, C_n^1), \cdots, (\vec{\mathcal{E}}_n^{k_n}, C_n^{k_n})\rangle \\ HP(\vec{\mathcal{E}}_{n+1} \vdash M_{n+1} \Uparrow) & = & \langle(\vec{\mathcal{E}}_{n+1}^1, C_{n+1}^1), \cdots, (\vec{\mathcal{E}}_{n+1}^{k_{n+1}}, C_{n+1}^{k_{n+1}})), \cdots\rangle \end{array}$$

Definition 5.3.6. *The **proof assumption with context information** of a term with context information $(\vec{\mathcal{E}}, M)$ where $M \in Exp$ and $\vec{\mathcal{E}} \in Ectx$*

$$HP(\vec{\mathcal{E}}, M) \in (Ectx, Can)^{\leq \omega}$$

is defined as:

- $HP(\vec{\mathcal{E}}, M) = HP(\vec{\mathcal{E}} \vdash M \Downarrow C)$ *if* $M \Downarrow C$
- $HP(\vec{\mathcal{E}}, M) = HP(\vec{\mathcal{E}} \vdash M \Uparrow)$ *if* $M \Uparrow$

The set of all proof assumptions with context information is denoted by Hyp^{ctx}. The notion of *length* is defined analogously to the one given for proof assumptions.

Definition 5.3.7. *The **length** of a proof assumption with context information is a function* $len : \ Hyp^{ctx} \to \mathbb{N} \cup \{\omega\}$ *defined for each* $M \in Exp$ *and* $\vec{\mathcal{E}} \in Ectx$ *as:*

- $len(HP(\vec{\mathcal{E}}, M)) = (n-1)$ *if* $\exists C \in Can \ (\vec{\mathcal{E}} \vdash M \Downarrow C) \ \& \ HP(\vec{\mathcal{E}}, M) = \langle (\vec{\mathcal{E}}_1, C_1), \cdots, (\vec{\mathcal{E}}_n, C_n) \rangle$
- $len(HP(\vec{\mathcal{E}}, M)) = \omega$ *if* $\vec{\mathcal{E}} \vdash M \Uparrow$

We have that the computation of a term can be together characterized by its proof assumption. In particular the structure of proof trees with the fact that the rules are L-rules, captures the direction of the computation. Hence we can consider the set of the assumptions of a proof tree: We now prove the following lemma:

Lemma 5.3.8 (+ Structural).
For each $M \in Exp, \vec{\mathcal{E}} \in Ectx$ such that $M \Downarrow C$ for some C and

$$HP(\vec{\mathcal{E}}, M) = \langle (\vec{\mathcal{E}}_1, C_1), \cdots, (\vec{\mathcal{E}}_n, C_n) \rangle$$

we have:

(i) $\vec{\mathcal{E}}[M] \equiv \vec{\mathcal{E}}_1[C_1] \qquad \& \qquad \vec{\mathcal{E}}[C] \equiv \vec{\mathcal{E}}_n[C_n]$

(ii) $\forall i \ (0 < i < n) \ \vec{\mathcal{E}}_i[C_i] \longrightarrow \vec{\mathcal{E}}_{i+1}[C_{i+1}]$

Proof. We prove it by induction on the height of proof tree of $M \Downarrow C$.
The base case is trivial. Now suppose $M \Downarrow C$ and:

$$HP(\vec{\mathcal{E}}, M) = \langle (\vec{\mathcal{E}}_1^1, C_1^1), \cdots, (\vec{\mathcal{E}}_1^{k_1}, C_1^{k_1}), \cdots, (\vec{\mathcal{E}}_{n-1}^{k_{n-1}}, C_{n-1}^{k_{n-1}}), (\vec{\mathcal{E}}_n^1, C_n^1), \cdots, (\vec{\mathcal{E}}_n^{k_n}, C_n^{k_n}) \rangle$$

by lemma 5.3.3 we have that $\vec{\mathcal{E}} \vdash M \Downarrow C$ and hence we have that there exists a + rule such that:

$$\frac{\vec{\mathcal{E}} \circ \mathcal{E}_1 \vdash M_1 \Downarrow C_1 \cdots \vec{\mathcal{E}} \circ \mathcal{E}_n \vdash M_n \Downarrow C_n}{\vec{\mathcal{E}} \vdash M \Downarrow C}$$

and by definition and determinacy of HP we have:

$$\begin{array}{rcl} HP(\vec{\mathcal{E}} \circ \mathcal{E}_1 \vdash M_1 \Downarrow C_1) & = & \langle (\vec{\mathcal{E}}_1^1, C_1^1), \cdots, (\vec{\mathcal{E}}_1^{k_1}, C_1^{k_1}) \rangle \\ & \vdots & \\ HP(\vec{\mathcal{E}} \circ \mathcal{E}_n \vdash M_n \Downarrow C_n) & = & \langle (\vec{\mathcal{E}}_n^1, C_n^1), \cdots, (\vec{\mathcal{E}}_n^{k_n}, C_n^{k_n}) \rangle \end{array}$$

By induction hypothesis we have:

$$\begin{array}{lcl} \vec{\mathcal{E}}[\mathcal{E}_1[M_1]] \equiv \vec{\mathcal{E}}_1^1[C_1^1] & \& & \vec{\mathcal{E}}[\mathcal{E}_1[C_1]] \equiv \vec{\mathcal{E}}_1^{k_1}[C_1^{k_1}] \\ & \vdots & \\ \vec{\mathcal{E}}[\mathcal{E}_n[M_n]] \equiv \vec{\mathcal{E}}_n^1[C_n^1] & \& & \vec{\mathcal{E}}[\mathcal{E}_n[C_n]] \equiv \vec{\mathcal{E}}_n^{k_n}[C_n^{k_n}] \end{array}$$

and

$$\begin{array}{c} \forall i \ (0 < i < k_1) \ \vec{\mathcal{E}}_1^i[C_1^i] \longrightarrow \vec{\mathcal{E}}_1^{i+1}[C_1^{i+1}] \\ \vdots \\ \forall j \ (0 < j < k_n) \ \vec{\mathcal{E}}_n^j[C_n^j] \longrightarrow \vec{\mathcal{E}}_n^{j+1}[C_n^{j+1}] \end{array}$$

By lemma 5.2.1 $\vec{\mathcal{E}}[M] \equiv \vec{\mathcal{E}}[\mathcal{E}_1[M_1]]$ and $\vec{\mathcal{E}}[C] \equiv \vec{\mathcal{E}}[\mathcal{E}_n[C_n]]$ hence $\vec{\mathcal{E}}[M] \equiv \vec{\mathcal{E}}_1^1[C_1^1]$ and $\vec{\mathcal{E}}[C] \equiv \vec{\mathcal{E}}_n^{k_n}[C_n^{k_n}]$ respectively and so (i) follows.
Now by lemma 5.2.1 we have that

$$\forall k \ (0 < k < n) \ \vec{\mathcal{E}}[\mathcal{E}_k[C_k]] \longrightarrow \vec{\mathcal{E}}[\mathcal{E}_{k+1}[M_{k+1}]]$$

and hence

$$\begin{array}{rcl} \vec{\mathcal{E}}_1^1[C_1^1] \longrightarrow \cdots \longrightarrow \vec{\mathcal{E}}_1^{k_1}[C_1^{k_1}] & \equiv & \vec{\mathcal{E}}[\mathcal{E}_1[C_1]] \longrightarrow \vec{\mathcal{E}}[\mathcal{E}_2[M_2]] \equiv \\ \vec{\mathcal{E}}_2^1[C_2^1] \longrightarrow \cdots & \equiv & \cdots \longrightarrow \vec{\mathcal{E}}_{n-1}^{k_{n-1}}[C_{n-1}^{k_{n-1}}] \equiv \\ \vec{\mathcal{E}}[\mathcal{E}_{n-1}[C_{n-1}]] \longrightarrow \vec{\mathcal{E}}[\mathcal{E}_n[M_n]] & \equiv & \vec{\mathcal{E}}_n^1[C_n^1] \longrightarrow \cdots \longrightarrow \vec{\mathcal{E}}_n^{k_n}[C_n^{k_n}] \end{array}$$

Whence the conclusion. □

Lemma 5.3.9 (− Structural).
For each $M \in Exp, \vec{\mathcal{E}} \in Ectx$ such that $M \Uparrow$ and

$$HP(\vec{\mathcal{E}}, M) = \langle (\vec{\mathcal{E}}_1, C_1), \cdots, (\vec{\mathcal{E}}_n, C_n), \cdots \rangle$$

we have:

(i) $\vec{\mathcal{E}}[M] \equiv \vec{\mathcal{E}}_1[C_1]$

(ii) $\forall i \ (0 < i) \ \vec{\mathcal{E}}_i[C_i] \longrightarrow \vec{\mathcal{E}}_{i+1}[C_{i+1}]$

Proof. We prove the lemma by induction on the abstract structure of M and the fact that $\Uparrow^{ctx}$ can be constructively defined as $\bigcap_{n \in \mathbb{N}} \phi_{\Uparrow}^{ctx}(Exp)$. Assume that $\vec{\mathcal{E}} \vdash M \Uparrow$ then we have there exists a $-$rule such that:

$$\frac{\vec{\mathcal{E}} \circ \mathcal{E}_1 \vdash M_1 \Downarrow C_1 \cdots \vec{\mathcal{E}} \circ \mathcal{E}_n \vdash M_n \Downarrow C_n \quad \vec{\mathcal{E}} \circ \mathcal{E}_{n+1} \vdash M_{n+1} \Uparrow}{\vec{\mathcal{E}} \vdash M \Uparrow}$$

By an inspection of the rules for each M we have at most two possibilities: either $n = 0$ or $n = 1$.
If $n = 0$ then we have

$$\frac{\vec{\mathcal{E}} \circ \mathcal{E}_1 \vdash M_1 \Uparrow}{\vec{\mathcal{E}} \vdash M \Uparrow}$$

where by lemma 5.2.2 $\vec{\mathcal{E}}[M] \equiv \vec{\mathcal{E}}[\mathcal{E}_1[M_1]]$ and by definition of proof assumption $HP(\vec{\mathcal{E}} \vdash M \Uparrow) = HP(\vec{\mathcal{E}} \circ \mathcal{E}_1 \vdash M_1 \Uparrow)$. By induction hypothesis we have $\vec{\mathcal{E}}[\mathcal{E}_1[M_1]] \equiv \vec{\mathcal{E}}_1[C_1]$ hence by lemma 5.2.2 (i) follows. Again by induction hypothesis we have $\forall i\ (0 < i)\ \vec{\mathcal{E}}_i[C_i] \longrightarrow \vec{\mathcal{E}}_{i+1}[C_{i+1}]$ and since transition is deterministic and $HP(\vec{\mathcal{E}} \vdash M \Uparrow) = HP(\vec{\mathcal{E}} \circ \mathcal{E}_1 \vdash M_1 \Uparrow)$ we have the conclusion.
Otherwise if $n = 1$ we have

$$\frac{\vec{\mathcal{E}} \circ \mathcal{E}' \vdash M' \Downarrow C' \qquad \dfrac{\vec{\mathcal{E}} \circ \mathcal{E}'' \circ \mathcal{E}_k \vdash M''_k \Downarrow C''_k \quad \vec{\mathcal{E}} \circ \mathcal{E}'' \circ \mathcal{E}_{k+1} \vdash M''_{k+1} \Uparrow}{\vec{\mathcal{E}} \circ \mathcal{E}'' \vdash M'' \Uparrow}}{\vec{\mathcal{E}} \vdash M \Uparrow}$$

By lemma 5.2.2 $\vec{\mathcal{E}}[M] \equiv \vec{\mathcal{E}}[\mathcal{E}'[M']]$ and by definition of proof assumption with context information we have:

$$HP(\vec{\mathcal{E}} \vdash M \Uparrow) = \langle (\vec{\mathcal{E}}_1, C_1), \cdots, (\vec{\mathcal{E}}_m, C_m), (\vec{\mathcal{E}}_{m+1}, C_{m+1}) \cdots \rangle$$

where:

$$\begin{aligned} HP(\vec{\mathcal{E}} \circ \mathcal{E}' \vdash M' \Downarrow C') &= \langle (\vec{\mathcal{E}}_1, C_1), \cdots, (\vec{\mathcal{E}}_m, C_m) \rangle \\ HP(\vec{\mathcal{E}} \circ \mathcal{E}'' \vdash M'' \Uparrow) &= \langle (\vec{\mathcal{E}}_{m+1}, C_{m+1}) \cdots \rangle \end{aligned}$$

Now by lemma 5.3.8 we have $\vec{\mathcal{E}}[\mathcal{E}'[M']] \equiv \vec{\mathcal{E}}_1[C_1]$ hence (i) follows. Furthermore again by lemma 5.3.8 we have $\forall i\ (0 < i < m)\ \vec{\mathcal{E}}_i[C_i] \longrightarrow \vec{\mathcal{E}}_{i+1}[C_{i+1}]$ and $\vec{\mathcal{E}}[\mathcal{E}'[C']] \equiv \vec{\mathcal{E}}_m[C_m]$. By lemma 5.2.2 $\vec{\mathcal{E}}[\mathcal{E}'[C_m]] \longrightarrow \vec{\mathcal{E}}[\mathcal{E}''[M'']]$. We can now apply the same argument for n to k. By repeatedly apply the above argument (ii) follows. □

Proposition 5.3.10. *Let $M \in Exp, \vec{\mathcal{E}} \in Ectx$, if there exists $n \in \mathbb{N}$ such that $(HP(\vec{\mathcal{E}}, M)) = n$ then:*

$$\vec{\mathcal{E}}[M] \longrightarrow^n \vec{\mathcal{E}}[C]$$

Proof. Lemma 5.3.8 and *len* definition give the conclusion. □

Proposition 5.3.11. *Let $M \in Exp, \vec{\mathcal{E}} \in Ectx$, if $len(HP(\vec{\mathcal{E}}, M)) = \omega$ then:*

$$\vec{\mathcal{E}}[M] \longrightarrow^\omega$$

Proof. Lemma 5.3.8 and *len* definition give the conclusion. □

Propositions 5.1.6 and 5.1.7 follows from proposition 5.3.10 and 5.3.11 respectively by taking $\vec{\mathcal{E}}$ equals to $\mathcal{I}d$.

$$\dfrac{\dfrac{\mathcal{E}_1[\mathcal{E}_2[\mathcal{E}_3]] \vdash \langle \lambda x.\mathbf{snd}(x), \Omega^\tau\rangle \Downarrow \langle \lambda x.\mathbf{snd}(x), \Omega^\tau\rangle \xrightarrow{\mathbf{1}} \mathcal{E}_1[\mathcal{E}_2] \vdash \lambda x.\mathbf{snd}(x) \Downarrow \lambda x.\mathbf{snd}(x)}{\mathcal{E}_1[\mathcal{E}_2] \vdash \mathbf{fst}\langle \lambda x.\mathbf{snd}(x), \Omega^\tau\rangle \Downarrow \lambda x.\mathbf{snd}(x)} \xrightarrow{\mathbf{2}} \dfrac{\mathcal{E}_1[\mathcal{E}_4] \vdash \langle \Omega^\tau, \lambda z.z\rangle \Downarrow \langle \Omega^\tau, \lambda z.z\rangle \xrightarrow{\mathbf{3}} \mathcal{E}_1 \vdash \lambda z.z \Downarrow \lambda z.z}{\mathcal{E}_1 \vdash \mathbf{snd}\langle \Omega^\tau, \lambda z.z\rangle \Downarrow \lambda z.z}}{\mathcal{E}_1 \vdash (\mathbf{fst}\langle \lambda x.\mathbf{snd}(x), \Omega^\tau\rangle)\langle \Omega^\tau, \lambda z.z\rangle \Downarrow \lambda z.z}$$

where Experiments are respectively:

$$\begin{aligned}
&\mathcal{E}_1 = [\bullet]\\
&\mathcal{E}_2 = [\bullet]\langle \Omega^\tau, \lambda z.z\rangle\\
&\mathcal{E}_3 = \mathbf{fst}([\bullet])\\
&\mathcal{E}_4 = \mathbf{snd}([\bullet])
\end{aligned}$$

can be rewritten as:

$$\begin{aligned}
&\mathcal{E}_1[\mathcal{E}_2[\mathcal{E}_3[\langle \lambda x.\mathbf{snd}(x), \Omega^\tau\rangle]]] \xrightarrow{\mathbf{1}}\\
&\mathcal{E}_1[\mathcal{E}_2[\lambda x.\mathbf{snd}(x)]] \xrightarrow{\mathbf{2}}\\
&\mathcal{E}_1[\mathcal{E}_4[\langle \Omega^\tau, \lambda z.z\rangle]] \xrightarrow{\mathbf{3}} \lambda z.z
\end{aligned}$$

and finally:

$$\begin{aligned}
&(\mathbf{fst}\langle \lambda x.\mathbf{snd}(x), \Omega^\tau\rangle)\langle \Omega^\tau, \lambda z.z\rangle \xrightarrow{\mathbf{1}}\\
&(\lambda x.\mathbf{snd}(x))\langle \Omega^\tau, \lambda z.z\rangle \xrightarrow{\mathbf{2}}\\
&\mathbf{snd}\langle \Omega^\tau, \lambda z.z\rangle \xrightarrow{\mathbf{3}} \lambda z.z
\end{aligned}$$

Table 5.3: Example of translation algorithm for convergent term

$$\dfrac{\dfrac{\mathcal{E}_1[\mathcal{E}_2] \vdash \lambda y.(\lambda x.xx)y \Downarrow \lambda y.(\lambda x.xx)y \quad \xrightarrow{\mathbf{1}} \quad \dfrac{\mathcal{E}_1[\mathcal{E}_3] \vdash (\lambda x.xx) \Downarrow (\lambda x.xx) \quad \xrightarrow{\mathbf{2}} \quad \dfrac{\mathcal{E}_1[\mathcal{E}_4] \vdash (\lambda x.xx) \Downarrow (\lambda x.xx) \quad \xrightarrow{\mathbf{3}} \quad \dfrac{\vdots}{\mathcal{E}_1 \vdash (\lambda x.xx)(\lambda x.xx) \Uparrow}}{\mathcal{E}_1 \vdash (\lambda x.xx)(\lambda x.xx) \Uparrow}}{\mathcal{E}_1 \vdash (\lambda x.xx)(\lambda x.xx) \Uparrow}}{\mathcal{E}_1 \vdash (\lambda y.(\lambda x.xx)y)(\lambda x.xx) \Uparrow}$$

where Experiments are respectively:

$$\begin{aligned}
\mathcal{E}_1 &= [\bullet] \\
\mathcal{E}_2 &= [\bullet](\lambda x.xx) \\
\mathcal{E}_3 &= [\bullet](\lambda x.xx) \\
\mathcal{E}_4 &= [\bullet](\lambda x.xx) \\
&\vdots
\end{aligned}$$

can be rewritten as:

$$\begin{aligned}
&\mathcal{E}_1[\mathcal{E}_2[\lambda y.(\lambda x.xx)y]] \xrightarrow{\mathbf{1}} \\
&\mathcal{E}_1[\mathcal{E}_3[\lambda x.xx]] \xrightarrow{\mathbf{2}} \\
&\mathcal{E}_1[\mathcal{E}_4[\lambda x.xx]] \xrightarrow{\mathbf{3}} \cdots
\end{aligned}$$

and finally:

$$\begin{aligned}
&(\lambda y.(\lambda x.xx)y)(\lambda x.xx) \xrightarrow{\mathbf{1}} \\
&(\lambda x.xx)(\lambda x.xx) \xrightarrow{\mathbf{2}} \\
&(\lambda x.xx)(\lambda x.xx) \xrightarrow{\mathbf{3}} \cdots
\end{aligned}$$

Table 5.4: Example of translation algorithm for divergent term

5.4 Conclusion

The proofs of structural lemmas 5.3.8 and 5.3.9 give us an algorithm to translate a proof of convergence by big-step semantics in a proof of convergence by small-step semantics.

Translation Algorithm	
STEP 1	Add to the proof tree of M context information to obtain the proof tree of $(\mathcal{I}d, M)$
STEP 2	take the sequence $HP(\mathcal{I}d, M) = \langle(\mathcal{E}_1, C_1), \cdots\rangle$
STEP 3	for all $0 \leq i \leq len(HP(\mathcal{I}d\, M))$ set $\mathcal{E}_i[C_i] \longrightarrow \mathcal{E}_{i+1}[C_{i+1}]$

Translation Algorithm can be applied both to convergent and divergent terms. In table 5.3 we show an example of the algorithm applied to the term:

$$(\mathbf{fst}\langle\lambda x.\mathbf{snd}(x), \Omega^\tau\rangle)\langle\Omega^\tau, \lambda z.z\rangle$$

In table 5.4 we show an example of the algorithm applied to the term:

$$(\lambda y.(\lambda x.xx)y)(\lambda x.xx)$$

We note that in order to justify the above method the validity of structural lemmas 5.3.8 and 5.3.9 is needed. This does not work for all languages. In particular it seems to work only for *sequential deterministic languages* and under the assumption that all the premisses are made explicit.
In non sequential languages the order of premisses is not always defined. So it is not possible to define the big-step semantics through instances of L-rules. The request of explicitate all the implicit premisses of each rule is needed in order to explicitate all the single steps of computation which are involved in a big-step rule.
We remark that our method is only an application and refinement of the one first outlined by Ibraheem and Schmidt in [Ibraheem and Schmidt, 2000]. The method is formulated and analyzed there in a more abstract way.

Chapter 6

Contextual Equivalence

> There is no assurance here that the extensional agreement of 'bachelor' and 'unmarried man' rests on meaning rather than merely on accidental matters of fact, as does extensional agreement of 'creature with a heart' and 'creature with a kidney'.
>
> W.V. Quine. "Two Dogmas of Empiricism"

We have so far defined operational semantics for FPC language through two different relation which we have proved to be equivalent. In particular, a semantics induces a notion of program equivalence. Intuitively, by an operational point of view, two programs are equivalent if they behave in the same way when they are executed on the same machine. This was more clearly expressed by Ong in [Ong, 1995]:

> Two program fragments are equivalent if they can always be *interchanged* without affecting the visible or observable outcome of the computation. This criterion of sameness which is called *observational equivalence* is formally expressed in terms of invariance of observable outcome under all program contexts.

We shall refer to program equivalence by a different name and when we refer to the name *observational equivalence* we do not mean a concrete equivalence relation on our language but the abstract process of deciding equivalence of things, in general, by observing their properties. In this sense we say that the equivalence induced by operational semantics is the *observational equivalence* for FPC terms.
An equivalence notion with the above properties was first introduced for lambda calculus by Morris in [Morris, 1968] with the name of *Extensional Equivalence*. We call it **Contextual Equivalence** and this is widely accepted as the natural notion of operational semantics for FPC.
In the sequel we formalize the above informal definition. In order to have a formal definition we need to define exactly what does it means *interchanging two program fragments* and what are the *observable outcomes* of a program. We can formalize the former via the general notion of contexts, the latter via the familiar notion of evaluation.

Once we have formalized contextual equivalence for FPC we prove some interesting properties. In particular we show the validity of a kind of *syntactic continuity* and *rational completeness*.
Our presentation of context and contextual equivalence follows exactly the one given by Pitts in [Pitts, 1995]. Differently from Pitts, as an exercise, we prove all the properties of contextual equivalence directly by induction. Finally the proofs of syntactic continuity and rational completeness follows in spirit those by Sands in [Sands, 1997] and by Smith in [Smith, 1991].

6.1 Context

We have seen that to give a formal definition of *contextual equivalence* we are interested in a notion of *interchanging program fragments*. Since program fragments are FPC terms we need a notion of interchanging terms in programs. To give a definition as general as possible, we use the notion of **context**. Informally a *context* is a syntax tree containing some sorts of parameters which represent *holes*. When all the holes are replaced by terms we have a program. In this sense we can think the *holes* as meta-variables. So we can define contexts by adding parameters to the definition of our language as follows.

Definition 6.1.1 (Context). *Let Ctx be the set of* ***FPC contexts*** *defined by the following grammar specified via BNF:*

$$\begin{aligned}\mathcal{C} \;::=\;& x \mid p \mid \lambda x.\mathcal{C} \mid (\mathcal{C}\mathcal{C}) \mid \langle \mathcal{C}, \mathcal{C}\rangle \mid \boldsymbol{fst}(\mathcal{C}) \mid \boldsymbol{snd}(\mathcal{C}) \\ \mid\;& \boldsymbol{case}\ \mathcal{C}\ \boldsymbol{of}\ \{\boldsymbol{inl}_{\tau_1,\tau_2}(x_1).\mathcal{C} \mid \boldsymbol{inr}_{\tau_1,\tau_2}(x_2).\mathcal{C}\} \\ \mid\;& \boldsymbol{inl}_{\tau_1,\tau_2}(\mathcal{C}) \mid \boldsymbol{inr}_{\tau_1,\tau_2}(\mathcal{C}) \mid \boldsymbol{abs}_{\mu T.\tau}(\mathcal{C}) \mid \boldsymbol{rep}(\mathcal{C})\end{aligned}$$

where p ranges over some fixed set of parameters

We note that by the above definition we have that either evaluation contexts and expressions are particular cases of the more general notion of context. In particular evaluation contexts form a subset of the set of contexts which contains exactly one occurence of a unique parameter $\bullet$ and *Exp* is the set of α-equivalence classes of contexts which do not contain parameter at all.
By the above remark it is natural extends to contexts the definitions which we have so far stated for FPC terms. It is easy to extends the notions of free variable, bound variable, α-equivalence.
In the sequel we usually work with only one parameters which we write $\circ$. We write $\mathcal{C}[\circ]$ to stress that $\mathcal{C}$ is a context and $\circ$ is the only parameter contained in $\mathcal{C}$.

Context substitution

The name "context" underlines the importance of filling the hole $\circ$ with a term M to obtain a new term. In general, filling the occurrences of a hole $\circ$ with a term M is different from the natural extension to contexts of substitution operation defined in 4.1.3. Since a term M can be viewed as a context, to generalize we can define the operation of substitution of a context for a parameter in a context.

Definition 6.1.2. *The* ***substitution in a context*** *of a context $\mathcal{C}'$ for a parameter $\circ$ in a context $\mathcal{C}$, written $\mathcal{C}[\mathcal{C}'/\circ]$, is inductively defined as follows:*

$$
\begin{array}{lcll}
\mathcal{C}[\mathcal{C}'/\circ] & = & \mathcal{C} & \text{if } \mathcal{C} \text{ is a variable} \\
\circ[\mathcal{C}'/\circ] & = & \mathcal{C}' & \\
(\lambda y.\mathcal{C})[\mathcal{C}'/\circ] & = & \lambda y.(\mathcal{C}[\mathcal{C}'/\circ]) & \\
(\mathcal{C}_1\mathcal{C}_2)[\mathcal{C}'/\circ] & = & (\mathcal{C}_1[\mathcal{C}'/\circ]\mathcal{C}_2[\mathcal{C}'/\circ]) & \\
\langle \mathcal{C}_1, \mathcal{C}_2\rangle[\mathcal{C}'/\circ] & = & \langle \mathcal{C}_1[\mathcal{C}'/\circ], \mathcal{C}_2[\mathcal{C}'/\circ]\rangle & \\
\boldsymbol{fst}(\mathcal{C})[\mathcal{C}'/\circ] & = & \boldsymbol{fst}(\mathcal{C}[\mathcal{C}'/\circ]) & \\
\boldsymbol{snd}(\mathcal{C})[\mathcal{C}'/\circ] & = & \boldsymbol{snd}(\mathcal{C}[\mathcal{C}'/\circ]) & \\
\boldsymbol{inl}_{\tau_1,\tau_2}(\mathcal{C})[\mathcal{C}'/\circ] & = & \boldsymbol{inl}_{\tau_1,\tau_2}(\mathcal{C}[\mathcal{C}'/\circ]) & \\
\boldsymbol{inr}_{\tau_1,\tau_2}(\mathcal{C})[\mathcal{C}'/\circ] & = & \boldsymbol{inr}_{\tau_1,\tau_2}(\mathcal{C}[\mathcal{C}'/\circ]) & \\
\boldsymbol{abs}_{\mu T.\tau}(\mathcal{C})[\mathcal{C}'/\circ] & = & \boldsymbol{abs}_{\mu T.\tau}(\mathcal{C}[\mathcal{C}'/\circ])) & \\
\boldsymbol{rep}(\mathcal{C})[\mathcal{C}'/\circ] & = & \boldsymbol{rep}(\mathcal{C}[\mathcal{C}'/\circ])) & \\
\multicolumn{4}{l}{\boldsymbol{case}\ \mathcal{C}\ \boldsymbol{of}\ \{\boldsymbol{inl}_{\tau_1,\tau_2}(x_1).\mathcal{C}_1 \mid \boldsymbol{inr}_{\tau_1,\tau_2}(x_2).\mathcal{C}_2\}[\mathcal{C}'/\circ]} \\
\multicolumn{4}{l}{= \boldsymbol{case}\ \mathcal{C}[\mathcal{C}'/\circ]\ \boldsymbol{of}\ \{\boldsymbol{inl}_{\tau_1,\tau_2}(x_1).\mathcal{C}_1[\mathcal{C}'/\circ] \mid \boldsymbol{inr}_{\tau_1,\tau_2}(x_2).\mathcal{C}_2[\mathcal{C}'/\circ]\}}
\end{array}
$$

By the above definition it is clear that in the substitution $\mathcal{C}[\mathcal{C}'/\circ]$, the free variables of $\mathcal{C}'$ may become bound. We write $traps(\mathcal{C}[\circ])$ for the set of variables that occur in $\mathcal{C}[\circ]$ associated to binders containing the hole $\circ$ within their scope. Thus any $x \in FV(M)$ such that $x \in traps(\mathcal{C}[\circ])$ becomes bound in $\mathcal{C}[M]$.
The above remarks implies that substitution in α-equivalent contexts does not preserve α-equivalence, as shown by the following.

Lemma 6.1.3.

$$\mathcal{C}_1 \equiv^{\alpha} \mathcal{C}_2 \not\Longrightarrow \mathcal{C}_1[\mathcal{C}/\circ] \equiv^{\alpha} \mathcal{C}_2[\mathcal{C}/\circ]$$

Proof. We only need to show a counter example: taken $\mathcal{C}_1 \equiv \lambda x.\circ \equiv^{\alpha} \lambda y.\circ \equiv \mathcal{C}_2$ and $\mathcal{C} \equiv x$ we have $\mathcal{C}_1[\mathcal{C}/\circ] \equiv \lambda x.x$ which is not α-equivalent to $\mathcal{C}_2[\mathcal{C}/\circ] \equiv \lambda y.x$. □

Nevertheless substituting α-equivalent contexts results in α-equivalent contexts, as shown by the following.

Lemma 6.1.4.

$$\mathcal{C}_1 \equiv_{\alpha} \mathcal{C}_2 \Longrightarrow \mathcal{C}[\mathcal{C}_1/\circ] \equiv_{\alpha} \mathcal{C}[\mathcal{C}_2/\circ]$$

Proof. We prove it by induction on the structure of $\mathcal{C}$. We show the case $\mathcal{C} \equiv \lambda x.\mathcal{C}'$ the other cases are similar.
We assume $\mathcal{C}_1 \equiv_{\alpha} \mathcal{C}_2$. We want to show that $(\lambda x.\mathcal{C}')[\mathcal{C}_1/\circ] \equiv_{\alpha} (\lambda x.\mathcal{C}')[\mathcal{C}_2/\circ]$ By definition of substitution we have $(\lambda x.\mathcal{C}')[\mathcal{C}_1/\circ] = \lambda x.(\mathcal{C}'[\mathcal{C}_1/\circ])$ and $(\lambda x.\mathcal{C}')[\mathcal{C}_2/\circ] = \lambda x.(\mathcal{C}'[\mathcal{C}_2/\circ])$ but by induction hypothesis we have $\mathcal{C}'[\mathcal{C}_1/\circ] \equiv_{\alpha} \mathcal{C}'[\mathcal{C}_2/\circ]$. Hence if $x \in Bound(\mathcal{C}'[\mathcal{C}_1/\circ])$ we have immediately the conclusion, otherwise $x \in FV(\mathcal{C}'[\mathcal{C}_1/\circ])$ and by (α-cong lambda) the conclusion. □

By the above lemma, the operation of substituting for a parameter in a context induces a well-defined operation on α-equivalence classes of contexts. In particular it is well-defined on the set of α-equivalence classes of contexts with no occurrences of parameters at all, *Term*.
By lemma 6.1.4 we have the following remark which is quoted by Pitts in [Pitts, 1995].

> The term $\mathcal{C}[M]$ will denote the term resulting from choosing a representative syntax tree for M, substituting it for the parameter in $\mathcal{C}$, and forming the α-equivalence class of the resulting FPC syntax tree which is indipendent of the choice of representative for M.

Typed Contexts

Since FPC is a typed language we need that contexts be typed terms with typed *holes* as subterms which can be filled with type-compatible terms. Hence we have that contexts and parameters must assume types. We usually write p_τ to indicate that parameter p has type τ.
In particular we are interested in contexts which are well-typed. Hence we need a kind of **context type judgement** which can take the form

$$\Gamma \vdash \mathcal{C} : \tau$$

where $\Gamma \in \mathit{TEnv}$ and $\tau \in \mathit{Type}$.
We have that context type judgements are of the same kind of type judgements for FPC terms hence we can think that **context type assignment relation** is the natural extension of type assignment relation for FPC terms. So we have that typing rules for contexts are rules in table 4.2, considering meta-variables $M, N, M_1, \ldots$ as contexts $\mathcal{C}, \mathcal{C}_1, \mathcal{C}_2, \ldots$ instead of terms, with the following axiom.

$$\Gamma \vdash \circ_\tau : \tau \qquad (\vdash \text{param})$$

Given typing rules for contexts it is easy to define *context type assignment relation* analogously to definition 4.1.7 as the set of relation inductively defined by typing rules.
By quoting Pitts in [Pitts, 1995] we have the following important remark.

> When typing rules are applied to syntax tree rather than α-equivalence class of syntax trees then they enforce a separation beetween free and bound variables and hence are not closed under α-equivalence.

In [Pitts, 1994] Pitts gives an alternative treatment which avoid this problem by using the notion of "function variables".
Context type assignment relation permits to isolate the set of well-typed contexts:

Definition 6.1.5. *Let $Ctx_\tau(\Gamma)$ be the set of **contexts** that can be assigned type τ, given Γ:*

$$Ctx_\tau(\Gamma) \stackrel{\text{def}}{=} \{\mathcal{C} \mid \Gamma \vdash \mathcal{C} : \tau\}$$

in particular we define:

$$Ctx_\tau \stackrel{\text{def}}{=} Ctx_\tau(\emptyset)$$

It is clear by definition of context type assignment relation that many properties on type assignment on term can be naturally extended to type assignment on contexts.

We need that context substitution preserves typing, we have that it is preserved in the following sense.

Lemma 6.1.6. *For each* $M \in Exp_\tau(\Gamma, \Gamma')$, $\mathcal{C}[\circ_\tau] \in Ctx_{\tau'}(\Gamma')$ *such that* $dom(\Gamma) \subseteq traps(\mathcal{C}[\circ_\tau])$ *we have* $\mathcal{C}[M] \in Exp_{\tau'}(\Gamma')$

Proof. The proof is by induction on the derivation of $\Gamma \vdash C[\circ_\tau] : \tau'$. We prove it for $\mathcal{C}[\circ_\tau] \equiv \lambda x.\mathcal{C}'[\circ_\tau]$ with $\tau' = \tau_1 \to \tau_2$, the other cases are similar.
We assume that $\Gamma, \Gamma' \vdash M : \tau$ and $\Gamma' \vdash \lambda x.\mathcal{C}'[\circ_\tau] : \tau_1 \to \tau_2$ with $dom(\Gamma) \subseteq traps(\mathcal{C}[\circ_\tau])$. We want to prove that $\Gamma' \vdash \lambda x.\mathcal{C}'[M] : \tau_1 \to \tau_2$.
If $\Gamma' \vdash \lambda x.\mathcal{C}'[\circ_\tau] : \tau_1 \to \tau_2$ then by inversion property $\Gamma', x : \tau_1 \vdash \mathcal{C}'[\circ_\tau] : \tau_2$. It is easy to verify that $traps(\mathcal{C}'[\circ_\tau]) = traps(\mathcal{C}[\circ_\tau]) - \{x\}$ and so $dom(\Gamma) - \{x\} \subseteq traps(\mathcal{C}'[\circ_\tau])$. Hence by induction hypothesis we have $\Gamma', x : \tau_1 \vdash \mathcal{C}[M] : \tau_2$ and by ($\vdash$ lam) $\Gamma' \vdash \lambda x.\mathcal{C}[M] : \tau_1 \to \tau_2$. □

A closed and typable context $\mathcal{C} \in Ctx_\tau$, analogously to closed and typable terms can be considered as *program contexts*. Intuitively program contexts are different in nature with respect to programs, as clearly noted by Ong in [Ong, 1995]:

> Program contexts are an auxiliary syntactic construction; they are not objects that computer manipulate, but a conceptual device.

Lemma 6.1.6 show the importance of program contexts and how we can use them. Taken a program context $\mathcal{C}[\circ_\tau] \in Ctx_{\tau_1}$, by the above remark a conceptual device, the substitution of all occurrences of $\circ_\tau$ with a term M such that $\Gamma \vdash M : \tau$ where $dom(\Gamma) \subseteq traps(\mathcal{C}[\circ_\tau])$ give us a program term, an executable object of the language.

6.2 Evaluation Context with holes

In 4.2.10 we have defined the notion of evaluation context. An evaluation context is a context which contains a single occurrence of a single hole. In particular by lemma 4.2.13 we have that for each $M \in Exp$ either it is a value or it can be written uniquely as $\vec{\mathcal{E}}[M']$ where $M' \in Rdx$. Hence the hole specify the position in a term where the next reduction step can be performed.
By lemma 6.1.6 we have that for each $M \in Exp_\tau(\Gamma)$, $\mathcal{C}[\circ_\tau] \in Ctx_{\tau'}$ such that $dom(\Gamma) \subseteq traps(\mathcal{C}[\circ_\tau])$ we have $\mathcal{C}[M] \in Exp_{\tau'}$. Hence either each $\mathcal{C}[M]$ is a value or it can be written uniquely as $\vec{\mathcal{E}}[M']$ where $M' \in Rdx$. Unfortunately this does not give us information about the relation beetween M' and M. For this reason we introduce the following notion.

Definition 6.2.1. ***Evaluation context with holes*** *are defined by the following grammar specified via BNF:*

$$\begin{aligned}\hat{\mathcal{E}} ::= \ & \bullet \mid \hat{\mathcal{E}}\ (\mathcal{C}) \mid \mathbf{fst}(\hat{\mathcal{E}}) \mid \mathbf{snd}(\hat{\mathcal{E}}) \mid \mathbf{rep}(\hat{\mathcal{E}}) \mid \mathbf{abs}_{\mu T.\tau}(\hat{\mathcal{E}}) \\ & \mid \mathbf{case}\ \hat{\mathcal{E}}\ \mathbf{of}\ \{\mathbf{inl}_{\tau_1,\tau_2}(x_1).\mathcal{C}_1 \mid \mathbf{inr}_{\tau_1,\tau_2}(x_2).\mathcal{C}_2\}\end{aligned}$$

Similarly to context it is easy to assign type to evaluation context with holes. Since for a context we usually use only one parameter then we usually write an evaluation context with hole as $\hat{\mathcal{E}}[\circ_\tau]\langle\bullet_{\tau'}\rangle$ where $\circ_\tau$ is the hole of the context and $\bullet_{\tau'}$ indicates where the next reduction can occur. We used two different type of brackets only for clearity of notation.
The notion of evaluation context with holes is extensively used by Smith in

[Smith, 1991] and by Mason, Talcott and Smith in [Mason et al., 1996]. Sands uses it in [Sands, 1997] with the name *reduction context with holes*.
The following lemma is the analogous of lemma 4.2.13 for evaluation context with holes.

Lemma 6.2.2 (Unique Context Factorization). *For each* $M \in Exp_\tau(\Gamma)$, $C[\circ_\tau] \in Ctx_{\tau_1}$, *where* $dom(\Gamma) \subseteq traps(C)$, *either* $C[M]$ *is a value or there exists a unique* $\hat{\mathcal{E}}[\circ_\tau]\langle \bullet_{\tau_2} \rangle : \tau_1$ *and* $C_1[\circ_\tau] : \tau_2$ *such that* $C[\circ_\tau] \equiv \hat{\mathcal{E}}[\circ_\tau]\langle C_1[\circ_\tau]\rangle$ *and* $C_1[M]$ *is a redex.*

Proof. By lemma 6.1.6 we have that $C[M]$ is a closed expression and there exists $M' \in Exp_{\tau_1}$ such that $C[M] \equiv M'$. By lemma 4.2.13 either M' is a value or we can write $C[M]$ uniquely as $\mathcal{E}\langle M'' \rangle : \tau$ where $M'' : \tau_2$ is a redex. M can occur in $\mathcal{E}\langle \bullet_{\tau_2} \rangle$ hence we write $\hat{\mathcal{E}}_1[\circ_\tau]\langle \bullet_{\tau_2} \rangle$ for $\hat{\mathcal{E}}[\circ_\tau/M]\langle \bullet_{\tau_2} \rangle$. Analogously we write $C_1[\circ_\tau]$ for $M''[\circ/M]$ where $C_1[\circ_\tau] \in Ctx_{\tau_2}$. Thus, we have $C[\circ_\tau] \equiv \hat{\mathcal{E}}_1[\circ_\tau]\langle C_1[\circ_\tau]\rangle$ and $C_1[M] \equiv M''$ is a redex. □

We show a simple example which explains how the above lemma works. Let $M, N \in Exp_{\tau\to(\tau_1+\tau_2)}$ and $Q \in Exp_\tau$ where $M \equiv \lambda x.M'$:

$$\mathcal{C}[\circ_{\tau\to(\tau_1+\tau_2)}] \equiv \mathbf{case}\ (\circ_{\tau\to(\tau_1+\tau_2)})Q\ \mathbf{of}\ \{\mathbf{inl}_{\tau_1,\tau_2}(x_1).\circ_{\tau\to(\tau_1+\tau_2)} \mid \mathbf{inr}_{\tau_1,\tau_2}(x_2).N\}$$

we have that:

$$\mathcal{C}[M] \equiv \mathbf{case}\ MQ\ \mathbf{of}\ \{\mathbf{inl}_{\tau_1,\tau_2}(x_1).M \mid \mathbf{inr}_{\tau_1,\tau_2}(x_2).N\} \equiv \hat{\mathcal{E}}[M]\langle \mathcal{C}_1[M]\rangle$$

where

$$\hat{\mathcal{E}}[\circ_{\tau\to(\tau_1+\tau_2)}]\langle \bullet_{\tau_1+\tau_2}\rangle \equiv \mathbf{case}\ \bullet_{\tau_1+\tau_2}\ \mathbf{of}\ \{\mathbf{inl}_{\tau_1,\tau_2}(x_1).\circ_{\tau\to(\tau_1+\tau_2)} \mid \mathbf{inr}_{\tau_1,\tau_2}(x_2).N\}$$

and

$$\mathcal{C}_1[\circ_{\tau\to(\tau_1+\tau_2)}] \equiv (\circ_{\tau\to(\tau_1+\tau_2)})Q$$

and since $M \equiv \lambda x.M'$ we have that $\mathcal{C}_1[M]$ is a redex.
The following lemma formalizes a common case analysis usually used informally in proofs of properties for contextual equivalence. This is an adaptation of lemma A.6 in [Sands, 1997] to FPC.

Lemma 6.2.3 (Activity). *For each* $M \in Exp_\tau, C[\circ_\tau] \in Ctx_{\tau_1}$ *if* $C[M]$ *reduces then* $C[M] \equiv \hat{\mathcal{E}}[M]\langle C_1[M]\rangle$ *and we have that one of the following applies:*

(i) the reduction is "uniform" in M *which means that there exists* $C_1'[\circ_\tau] \in Ctx_{\tau_1}$ *such that* $\forall\ N \in Exp_\tau,\ C_1[N] \longrightarrow C_1'[N]$

(ii) M *is a redex and* $C_1[\circ_\tau] \equiv \circ_\tau$.

(iii) M *is a value and there exists* $\mathcal{E} \in Exper$ *such that* $C_1[M] \equiv \mathcal{E}[M]$

Proof. By lemma 6.1.6 we have that $C[M] \in Exp_{\tau_1}$ and by lemma 6.2.2 since $C[M]$ reduces we can write it uniquely as $\equiv \hat{\mathcal{E}}[M]\langle C_1[M]\rangle$ where $\mathcal{C}_1[M]$ is a redex. By a simple analysis of redex cases we have three distinct possibilities:

$\mathcal{C}_1[\circ_\tau] \equiv \circ_\tau$ hence M is a redex and (ii) follows

$\mathcal{C}_1[\circ_\tau] \equiv \mathcal{E}[\circ_\tau]$ hence by lemma 4.2.12 we have that M is a value and (iii) follows

$\mathcal{C}_1[\circ_\tau] \not\equiv \mathcal{E}[-_\tau]$ & $\mathcal{C}_1[\circ_\tau] \not\equiv \circ_\tau$ we show that by a simple analysis of redex cases and of transition rules (i) follows.

- $\mathcal{C}_1[M] \equiv (\lambda x.\mathcal{C}_2[M])\mathcal{C}_3[M]$. For such a redex we can only apply (app tran) rule, in the reduction, the occurrences of M in $\mathcal{C}_3[M]$ remain untouched and hence the reduction is uniform in $\mathcal{C}_3[M]$. Now consider the occurrences of M in $\mathcal{C}_2[M]$, in the reduction we have that the free occurrences of x in $\mathcal{C}_2[M]$ are replaced by $\mathcal{C}_3[M]$, but since $M \in Exp_\tau$ we have that $x \notin FV(M) = \emptyset$. Hence the reduction is uniform in $\mathcal{C}_2[M]$ and so (i) follows.
- $\mathcal{C}_1[M] \equiv \mathbf{fst}(\langle \mathcal{C}_2[M], \mathcal{C}_3[M] \rangle)$. For such a redex we can only apply (fst tran) rule, in the reduction, the occurrences of M in $\mathcal{C}_2[M]$ remain untouched and hence the reduction is uniform in $\mathcal{C}_2[M]$. Now consider $\mathcal{C}_3[M]$, in the reduction we have that it is eliminated but we have that $\mathcal{C}_3[M]$ is eliminated indipendently from its structure and hence the reduction is uniform in $\mathcal{C}_3[M]$ and so (i) follows.
- $\mathcal{C}_1[M] \equiv \mathbf{snd}(\langle \mathcal{C}_2[M], \mathcal{C}_3[M] \rangle)$. For such a redex we can only apply (snd tran) rule, in the reduction, the occurrences of M in $\mathcal{C}_3[M]$ remain untouched and hence the reduction is uniform in $\mathcal{C}_3[M]$. Now consider $\mathcal{C}_2[M]$, in the reduction we have that it is eliminated but we have that $\mathcal{C}_2[M]$ is eliminated indipendently from its structure and hence the reduction is uniform in $\mathcal{C}_2[M]$ and so (i) follows.
- $\mathcal{C}_1[M] \equiv \mathbf{case}\ \mathbf{inl}(\mathcal{C}'[M])\ \mathbf{of}\ \{\mathbf{inl}_{\tau_1,\tau_2}(x_1).\mathcal{C}_2[M] \mid \mathbf{inr}_{\tau_1,\tau_2}(x_2).\mathcal{C}_3[M]\}$. For such a redex we can only apply (cond tran inl) rule, in the reduction, the occurrences of M in $\mathcal{C}'[M]$ remain untouched and hence the reduction is uniform in $\mathcal{C}'[M]$.
  Now consider $\mathcal{C}_3[M]$, in the reduction we have that it is eliminated but we have that it is eliminated indipendently from its structure and hence the reduction is uniform in $\mathcal{C}_3[M]$.
  Finally consider the occurrences of M in $\mathcal{C}_2[M]$, in the reduction we have that the free occurrences of x_1 in $\mathcal{C}_2[M]$ are replaced by $\mathcal{C}'[M]$, but since $M \in Exp_\tau$ we have that $x_1 \notin FV(M) = \emptyset$. Hence the reduction is uniform in $\mathcal{C}_2[M]$ and (i) follows.
- $\mathcal{C}_1[M] \equiv \mathbf{case}\ \mathbf{inr}(\mathcal{C}'[M])\ \mathbf{of}\ \{\mathbf{inl}_{\tau_1,\tau_2}(x_1).\mathcal{C}_2[M] \mid \mathbf{inr}_{\tau_1,\tau_2}(x_2).\mathcal{C}_3[M]\}$. For such a redex we can only apply (cond tran inl) rule, in the reduction, the occurrences of M in $\mathcal{C}'[M]$ remain untouched and hence the reduction is uniform in $\mathcal{C}'[M]$.
  Now consider $\mathcal{C}_2[M]$, in the reduction we have that it is eliminated but we have that it is eliminated indipendently from its structure and hence the reduction is uniform in $\mathcal{C}_2[M]$.
  Finally consider the occurrences of M in $\mathcal{C}_3[M]$, in the reduction we have that the free occurrences of x_2 in $\mathcal{C}_3[M]$ are replaced by $\mathcal{C}'[M]$, but since $M \in Exp_\tau$ we have that $x_2 \notin FV(M) = \emptyset$. Hence the reduction is uniform in $\mathcal{C}_3[M]$ and (i) follows.

□

6.3 Observable types

We have so far seen that two program fragments, e.g. two terms, are equivalent if they can always be interchanged in all program contexts, e.g. substituted in closed and typable context, without affecting the observable outcome.
In order to give a formal definition of contextual equivalence we now need to formalize what can be considered the *observable outcome* of a program.
It is natural for a programmer to consider as the observable outcome of a program the final output of the program's computation. In section 4 we have specified the operational behavior of FPC programs through relations which capture both the convergence and divergence of programs computation. In particular an evaluation judgement $M \Downarrow C$ captures the convergence of the computation originating from a term M to a value C, and lemma 4.2.4 shows that evaluation is deterministic. Hence we can consider as *observation* for a term M the result of its evaluation, if it exists.
Thus two program fragments M and N are equivalent if for each $\mathcal{C}[\circ]$ such that $\mathcal{C}[M]$ and $\mathcal{C}[N]$ are programs of the same types we have:

$$\mathcal{C}[M] \Downarrow C \iff \mathcal{C}[N] \Downarrow C$$

However the above condition is sometimes too strong, in particular for our language, since we have chosen a lazy evaluation, the observation stops on the outermost operator, hence for example we have that $\langle A, B\rangle$ and $\langle \mathbf{fst}(\langle A, B\rangle), \mathbf{snd}(\langle A, B\rangle)\rangle$ are two different canonical terms. It seems reasonable by lemma 4.2.19 and the quantification over all contexts to observe for a term M only if its computation converges or diverges.
Thus two program fragments M and N are equivalent if for each $\mathcal{C}[\circ]$ such that $\mathcal{C}[M]$ and $\mathcal{C}[N]$ are programs of the same type we have:

$$\mathcal{C}[M] \Downarrow \iff \mathcal{C}[N] \Downarrow$$

Since FPC is a typed language we now must define at what type should convergence be observable. We call these the *observable types* of FPC. McCusker in [McCusker, 1996a] observes that the choice of observable types in a call-by-value language does not make any difference. This is because the convergence of a term $(\lambda x.M)N$, where M is a convergent closed expression of observable type and N is an expression of any type, depends only on the convergence of N. In a call-by-name language as our version of FPC the situation is different. For example the choice to observe convergence at product types permits to distinguish beetween the terms $\Omega^{\tau_1 \times \tau_2}$ and $\langle \Omega^{\tau_1}, \Omega^{\tau_2}\rangle$ which otherwise are considered equal; the choice to observe convergence at function types permits to distinguish beetween the terms $\Omega^{\tau_1 \to \tau_2}$ and $\lambda x.\Omega^{\tau_2}$ where $x : \tau_1$ which otherwise are considered equal.
Furthermore McCusker in [McCusker, 1996a] suggested to restrict observable type only to sum types. This because we have the **case** constructor which forces convergence to be observable at sum types, if it is observable anywhere. Consider the term

$$\mathbf{case}\ M\ \mathbf{of}\ \{\mathbf{inl}_{\tau_1,\tau_2}(x_1).N \mid \mathbf{inr}_{\tau_1,\tau_2}(x_2).N\}$$

where M is a closed expression of sum type and N is a convergent closed expression of any type. The convergence of this term depends only on the convergence

of M. This means that for each context of sum type we can build a context of any type which has the same behaviour. Unfortunately it seems that the converse do not generally holds. This can be reasonable for our version of FPC, depending on our type constructor, nevertheless we prefer a strictly finer notion of contextual equivalence. Hence we observe the convergence at each type τ, instead of restricting our attention to sum types only. Thus finally we have that two program fragments M and N are equivalent if for any $\tau_1 \in \mathit{Type}$ and for each $\mathcal{C}[\circ] \in \mathit{Exp}_{\tau_1}$ such that $\mathcal{C}[M]$ and $\mathcal{C}[N]$ are programs we have:

$$\mathcal{C}[M] \Downarrow \iff \mathcal{C}[N] \Downarrow$$

6.4 Contextual Equivalence

We can now state the formal definition of *contextual approximability.*

Definition 6.4.1 (Contextual preorder). *Let $M, M' \in \mathit{Exp}_\tau(\Gamma)$ for some $\Gamma \in \mathit{TEnv}, \tau \in \mathit{Type}$. We say that M **contextually approximates** M', written $\Gamma \vdash M \lesssim_\tau M'$ if and only if for each $\tau_1 \in \mathit{Type}$ and for each $\mathcal{C}[\circ_\tau] \in \mathit{Ctx}_{\tau_1}$ such that $dom(\Gamma) \subseteq traps$ we have:*

$$\mathcal{C}[M] \Downarrow \Longrightarrow \mathcal{C}[M'] \Downarrow$$

We read $\Gamma \vdash M \lesssim_\tau M'$ as "in the environment Γ, M contextually approximate M'". We note that by our definition, contextual preorder is a family of relations, one for each $\Gamma \in \mathit{TEnv}, \tau \in \mathit{Type}$. We now must verify that we have really defined a preorder on expressions.

Lemma 6.4.2 (Preorder). *For each $\Gamma \in \mathit{TEnv}, \tau \in \mathit{Type}$, $\lesssim_\tau$ is a preorder on $\mathit{Exp}_\tau(\Gamma)$. Hence:*

(i) ***Reflexivity:*** $\Gamma \vdash M : \tau \Longrightarrow (\Gamma \vdash M \lesssim_\tau M)$

(ii) ***Transitivity:*** $\Gamma \vdash M \lesssim_\tau M' \;\&\; \Gamma \vdash M' \lesssim_\tau M'' \Longrightarrow \Gamma \vdash M \lesssim_\tau M''$

Proof.

(i) Reflexivity is trivial.

(ii) Assume $\Gamma \vdash M \lesssim_\tau M'$ and $\Gamma \vdash M' \lesssim_\tau M''$ and let $\mathcal{C}[\circ_\tau] \in \mathit{Ctx}_{\tau_1}$ be an arbitrarily context with $dom(\Gamma) \subseteq traps(C)$, then $\mathcal{C}[M], \mathcal{C}[M'], \mathcal{C}[M''] \in \mathit{Exp}_{\tau_1}$. By assumption $\mathcal{C}[M] \Downarrow\Longrightarrow \mathcal{C}[M'] \Downarrow$ and $\mathcal{C}[M'] \Downarrow\Longrightarrow \mathcal{C}[M''] \Downarrow$, so $\mathcal{C}[M] \Downarrow\Longrightarrow \mathcal{C}[M''] \Downarrow$. Since $\mathcal{C}[\circ_\tau]$ was choosen arbitrarily we have $\Gamma \vdash M \lesssim_\tau M''$.

□

We can finally define formally **contextual equivalence** as the equivalence relation induced by $\lesssim_\tau$.

Definition 6.4.3 (Contextual equivalence). *Given $\Gamma \in \mathit{TEnv}, \tau \in \mathit{Type}$, let $M, M' \in \mathit{Exp}_\tau(\Gamma)$:*

$$\Gamma \vdash M \cong_\tau M' \stackrel{\text{def}}{=} (\Gamma \vdash M \lesssim_\tau M') \;\&\; (\Gamma \vdash M' \lesssim_\tau M)$$

We read $\Gamma \vdash M \cong_\tau M'$ as "in the environment Γ, M is contextually equivalent to M'". Since we are usually interested in closed terms we write $M \cong_\tau M'$ and $M \lesssim_\tau M'$ for $\emptyset \vdash M \cong_\tau M'$ and $\emptyset \vdash M \lesssim_\tau M'$ respectively. As for contextual preorder it is clear that contextual equivalence is a family of relations, one for each $\Gamma \in \mathit{TEnv}, \tau \in \mathit{Type}$.

Lemma 6.4.4 (Equivalence). *For each Γ, $\cong_\tau$ is an equivalence relation on $\mathit{Exp}_\tau(\Gamma)$. Hence:*

Symmetry $\Gamma \vdash M \cong_\tau M' \iff \Gamma \vdash M' \cong_\tau M$

An immediate consequence of the definition of contextual equivalence is the following property.

Lemma 6.4.5.

$$\Gamma \vdash M \lesssim_\tau M' \ \&\ \Gamma \subseteq \Gamma_1 \Longrightarrow \Gamma_1 \vdash M \lesssim_\tau M'$$

Proof. Take any $\mathcal{C}[\circ_\tau] \in \mathit{Ctx}_{\tau_1}$, $\mathit{dom}(\Gamma_1) \subseteq \mathit{traps}(\mathcal{C})$ such that $\mathcal{C}[M] \Downarrow$. $\Gamma \subseteq \Gamma_1$ and hence $\mathit{dom}(\Gamma) \subseteq \mathit{traps}(\mathcal{C})$, so by assumption $\mathcal{C}[M'] \Downarrow$. □

6.4.1 Contextual Equivalence and congruence

A **congruence** is an equivalence relation that is preserved by all contexts. The following lemma shows that contextual preorder is a *precongruence* on FPC with the consequence that contextual equivalence is a *congruence* on FPC. This was originally proved by Morris in [Morris, 1968]. Our proof is a typed version of the one used by Morris.

Lemma 6.4.6 (Precongruence). *Given $\Gamma, \Gamma_1 \in \mathit{TEnv}, \tau \in \mathit{Type}$, for each $M, M' \in \mathit{Exp}_\tau(\Gamma, \Gamma_1)$, if $\Gamma, \Gamma_1 \vdash M \lesssim_\tau M'$ then $\forall\ \tau_1 \in \mathit{Type}, \forall\ \mathcal{C}[\circ_\tau] \in \mathit{Ctx}_{\tau_1}(\Gamma)$, $\mathit{dom}(\Gamma_1) \subseteq \mathit{traps}(\mathcal{C}[\circ_\tau])$ we have:*

$$\Gamma \vdash \mathcal{C}[M] \lesssim_{\tau_1} \mathcal{C}[M']$$

Proof. Assume $\Gamma, \Gamma_1 \vdash M \lesssim_\tau M'$ and let $\mathcal{C}[\circ_\tau]$ be any context in $\mathit{Ctx}_{\tau_1}(\Gamma)$ with $\mathit{dom}(\Gamma_1) \subseteq \mathit{traps}(\mathcal{C})$. For all τ_2 let $\mathcal{C}_1[\circ_{\tau_1}]$ be an arbitrary context in Ctx_{τ_2} with $\mathit{dom}(\Gamma) \subseteq \mathit{traps}(\mathcal{C})$. Then $\mathcal{C}_1[\mathcal{C}[\circ_\tau]] \equiv \mathcal{C}_2[\circ_\tau]$ is a context with $\mathit{dom}(\Gamma) \subseteq \mathit{traps}(\mathcal{C}_2[\circ_\tau])$, $\mathit{dom}(\Gamma_1) \subseteq \mathit{traps}(\mathcal{C}_2[\circ_\tau])$, and by hypothesis $\mathcal{C}_2[M] \Downarrow \Longrightarrow \mathcal{C}_2[M'] \Downarrow$. Thus, since $\mathcal{C}_1[\circ_{\tau_1}]$ was choosen arbitrarily and by lemma 6.1.6, we have $\Gamma \vdash \mathcal{C}[M] \cong_\tau \mathcal{C}[M']$. □

Corollary 6.4.7 (Congruence). *Given $\Gamma, \Gamma_1 \in \mathit{TEnv}, \tau \in \mathit{Type}$, for each $M, M' \in \mathit{Exp}_\tau(\Gamma, \Gamma_1)$, if $\Gamma, \Gamma_1 \vdash M \cong_\tau M'$ then $\forall\ \tau_1, \forall\ \mathcal{C}[\circ_\tau] \in \mathit{Ctx}_{\tau_1}(\Gamma)$, $\mathit{dom}(\Gamma_1) \subseteq \mathit{traps}(\mathcal{C}[\circ_\tau])$ we have:*

$$\Gamma \vdash \mathcal{C}[M] \cong_{\tau_1} \mathcal{C}[M']$$

The precongruence property of contextual preorder has several interesting consequences. For example the following.

Corollary 6.4.8 (Lambda abstraction).

$$\Gamma, x : \tau_1 \vdash M \lesssim_\tau M' \implies (\Gamma \vdash \lambda x.M \lesssim_{\tau_1 \to \tau} \lambda x.M')$$

Proof. It follows immediately from precongruence properties with $\mathcal{C}[\circ_\tau] \equiv \lambda x.\circ_\tau$ where $\mathcal{C}[\circ_\tau] \in Ctx_{\tau_1 \to \tau}(\Gamma)$ and $x \in traps(\mathcal{C})$. □

It is now possible to prove the following lemma, which states a kind of *monotonicity*.

Lemma 6.4.9 (Monotonic). *For each* $M_1, M_1' \in Exp_{\tau_1}(\Gamma, \Gamma_1), \ldots, M_n, M_n' \in Exp_{\tau_n}(\Gamma, \Gamma_1)$, *if* $\Gamma, \Gamma_1 \vdash M_1 \lesssim_{\tau_1} M_1', \ldots, \Gamma, \Gamma_1 \vdash M_n \lesssim_{\tau_n} M_n'$ *then:*

$$\forall\ \tau, \forall\ \mathcal{C}[\circ_{\tau_1}, \ldots, \circ_{\tau_n}] \in Ctx_\tau(\Gamma), dom(\Gamma_1) \subseteq traps(\mathcal{C}[\circ_\tau])$$
$$\Gamma \vdash \mathcal{C}[M_1, \ldots, M_n] \lesssim_\tau \mathcal{C}[M_1', \ldots, M_n']$$

Proof. Assume

$$\Gamma, \Gamma_1 \vdash M_1 \lesssim_{\tau_1} M_1',\ \Gamma, \Gamma_1 \vdash M_2 \lesssim_{\tau_2} M_2', \ldots,\ \Gamma, \Gamma_1 \vdash M_n \lesssim_{\tau_n} M_n'$$

Then by precongruence property for some $\mathcal{C}_1[\circ_{\tau_1}], \ldots, \mathcal{C}_n[\circ_{\tau_n}] \in Ctx_\tau(\Gamma)$ such that $\forall\ i,\ 1 \leq i \leq n,\ dom(\Gamma_1) \subseteq traps(\mathcal{C}_i)$:

$$\begin{array}{lcl}
\Gamma \vdash \mathcal{C}[M_1, M_2, \ldots, M_n] : \tau \equiv \mathcal{C}_1[M_1] & \lesssim_\tau & \mathcal{C}_1[M_1'] \equiv \mathcal{C}[M_1', M_2, \ldots, M_n] \\
\Gamma \vdash \mathcal{C}[M_1', M_2, \ldots, M_n] : \tau \equiv \mathcal{C}_2[M_2] & \lesssim_\tau & \mathcal{C}_2[M_2'] \equiv \mathcal{C}[M_1', M_2', \ldots, M_n] \\
\vdots & & \\
\Gamma \vdash \mathcal{C}[M_1, M_2', \ldots, M_n] : \tau \equiv \mathcal{C}_n[M_n] & \lesssim_\tau & \mathcal{C}_n[M_n'] \equiv \mathcal{C}[M_1', M_2', \ldots, M_n']
\end{array}$$

so by transitivity we have the conclusion. □

6.4.2 Contextual Equivalence and Computation

Intuitively in order to have a theory of program equivalence we need to show that contextual equivalence respects computations. We now prove different properties of both evaluation and transition. We prove the properties by induction on the length of computations, following the schema extensively used by Smith in [Smith, 1991], it is easy to prove the same results by induction on the height of the big-step proof trees.

Lemma 6.4.10. *For each* $M, M' \in Exp_\tau$*:*

$$M \longrightarrow M' \Longrightarrow \emptyset \vdash M \lesssim_\tau M'$$

Proof. Assume $(M \longrightarrow M')$ and for some $\mathcal{C}[\circ_\tau] \in Ctx_{\tau_1}$, $\mathcal{C}[M] \Downarrow$. We proceed by induction on the length of computation.
For the base case $\mathcal{C}[M]$ is a value, hence $\mathcal{C}[M] \Downarrow \mathcal{C}[M]$, and since $(M \longrightarrow M')$ we have that the canonical form depends on $\mathcal{C}[\circ_\tau]$ so $\mathcal{C}[M'] \Downarrow \mathcal{C}[M']$.
Now assume the lemma valid for shorter computation. By lemma 6.2.2, $\mathcal{C}[M]$ can be written uniquely as $\hat{\mathcal{E}}[M]\langle \mathcal{C}_1[M] \rangle$ where $\mathcal{C}_1[M]$ is a redex. Consider the next computation step:
$\hat{\mathcal{E}}[M]\langle \mathcal{C}_1[M] \rangle \longrightarrow \hat{\mathcal{E}}[M]\langle M'' \rangle$
Observe that the occurences of M outside the reduction context scope remain untouched, so we are interested in M''. By lemma 6.2.3, since $M \in Exp_\tau$ and $M \notin Can$ we have two cases depending on the structure of M'':

$\hat{\mathcal{E}}[M]\langle\mathcal{C}_1[M]\rangle \longrightarrow \hat{\mathcal{E}}[M]\langle\mathcal{C}_2[M]\rangle$
for some $\mathcal{C}_2[\circ_\tau] \not\equiv \circ_\tau$. By assumption $\hat{\mathcal{E}}[M]\langle\mathcal{C}_2[M]\rangle \Downarrow$ but it is a shorter computation hence we can apply induction hypothesis and obtain $\hat{\mathcal{E}}[M']\langle\mathcal{C}_2[M']\rangle \Downarrow$. Since $\hat{\mathcal{E}}[M']\langle\mathcal{C}_1[M']\rangle \longrightarrow \hat{\mathcal{E}}[M']\langle\mathcal{C}_2[M']\rangle$ we have the conclusion.
Otherwise:
$\hat{\mathcal{E}}[M]\langle\mathcal{C}_1[M]\rangle \longrightarrow \hat{\mathcal{E}}[M]\langle\mathcal{C}_1[M']\rangle$
where $\mathcal{C}_1[\circ_\tau] \equiv \circ_\tau$. Hence $\hat{\mathcal{E}}[M]\langle\mathcal{C}_1[M']\rangle \Downarrow$ and fixed M' by induction hypothesis $\hat{\mathcal{E}}[M']\langle\mathcal{C}_1[M']\rangle \Downarrow$.

□

Lemma 6.4.11. *For each $M, M' \in Exp_\tau$:*

$$M \longrightarrow M' \Longrightarrow \emptyset \vdash M' \lesssim_\tau M$$

Proof. Assume $(M \longrightarrow M')$ and for $\mathcal{C}[\circ_\tau] \in Ctx_{\tau_1}$, $\mathcal{C}[M'] \Downarrow$. We proceed by induction on the length of computation.
For the base case $\mathcal{C}[M']$ is a value, hence either $\mathcal{C}[M']$ is a value depending on $\mathcal{C}[\circ_\tau]$ or M' is a value with $\tau = \tau_1$. In the former case, also $\mathcal{C}[M]$ is a value, in the latter we know that $M \longrightarrow M'$ so $M \Downarrow M'$ by lemma 4.2.26. Then both give the conclusion.
Now assume the lemma valid for shorter computation. By lemma 6.2.2, $\mathcal{C}[M']$ can be written uniquely as $\hat{\mathcal{E}}[M']\langle\mathcal{C}_1[M']\rangle$ where $\mathcal{C}_1[M']$ is a redex. Consider the next computation step:
$\hat{\mathcal{E}}[M']\langle\mathcal{C}_1[M']\rangle \longrightarrow \hat{\mathcal{E}}[M']\langle M''\rangle$
Observe that the occurences of M' outside the reduction context scope remain untouched, so we are interested in M''. By lemma 6.2.3, since $M \in Exp_\tau$ and $M \notin Can$ we have two cases depending on the structure of M'':
$\hat{\mathcal{E}}[M']\langle\mathcal{C}_1[M']\rangle \longrightarrow \hat{\mathcal{E}}[M']\langle\mathcal{C}_2[M']\rangle$
for some $\mathcal{C}_2[\circ_\tau] \not\equiv \circ_\tau$. By assumption $\hat{\mathcal{E}}[M']\langle\mathcal{C}_2[M']\rangle \Downarrow$ but it is a shorter computation hence we can apply induction hypothesis and obtain $\hat{\mathcal{E}}[M]\langle\mathcal{C}_2[M]\rangle \Downarrow$. Since $\hat{\mathcal{E}}[M]\langle\mathcal{C}_1[M]\rangle \longrightarrow \hat{\mathcal{E}}[M]\langle\mathcal{C}_2[M]\rangle$ we have the conclusion.
Otherwise:
$\hat{\mathcal{E}}[M']\langle\mathcal{C}_1[M']\rangle \longrightarrow \hat{\mathcal{E}}[M']\langle\mathcal{C}_1[M'']\rangle$
for some M'' where $\mathcal{C}_1[\circ_\tau] \equiv \circ_\tau$. Hence $\hat{\mathcal{E}}[M']\langle M''\rangle \Downarrow$ and fixed M'' by induction hypothesis $\hat{\mathcal{E}}[M]\langle M''\rangle \Downarrow$. But by assumption $M \longrightarrow M'$ and $M' \longrightarrow M''$ then we have:
$\hat{\mathcal{E}}[M]\langle M\rangle \longrightarrow \hat{\mathcal{E}}[M]\langle M'\rangle \longrightarrow \hat{\mathcal{E}}[M]\langle M''\rangle$
the conclusion.

□

Hence for transition relation we have the following.

Corollary 6.4.12 ($\longrightarrow\subseteq\cong$). *For all $M, M' \in Exp_\tau$:*

$$M \longrightarrow M' \Longrightarrow M \cong_\tau M'$$

We can now easily prove the followings.

Lemma 6.4.13. *For each $M \in Exp_\tau, C \in Can_\tau$:*

$$M \Downarrow C \Longrightarrow M \lesssim_\tau C$$

Proof. By corollary 4.2.27 we have that $M \Downarrow C \iff M \longrightarrow M_1 \longrightarrow \cdots \longrightarrow C$. By lemma 6.4.10 we have $M \lesssim_\tau M_1 \lesssim_\tau \cdots \lesssim_\tau C$ and transitivity gives the conclusion. □

Lemma 6.4.14. *For each $M \in Exp_\tau, C \in Can_\tau$:*

$$M \Downarrow C \Longrightarrow C \lesssim_\tau M$$

Proof. By corollary 4.2.27 we have that $M \Downarrow C \iff M \longrightarrow M_1 \longrightarrow \cdots \longrightarrow C$. By lemma 6.4.11 we have $C \lesssim_\tau \cdots \lesssim_\tau M_1 \lesssim_\tau M$ and by transitivity the conclusion. □

Hence for evaluation relation we have the following.

Corollary 6.4.15 ($\Downarrow \subseteq \cong$). *For each $M \in Exp_\tau, C \in Can_\tau$:*

$$M \Downarrow C \Rightarrow M \cong_\tau C$$

We have proved important properties of computation with respect to contextual equivalence, in particular for convergent computation. Contextual equivalence is useful also in the case where computation diverges. This is stated by the two following lemmas.

Lemma 6.4.16 (Syntactic Bottom).

$$\Gamma \vdash M : \tau \Longrightarrow \Gamma \vdash \Omega^\tau \lesssim_\tau M$$

Proof. We only need to show $\forall \mathcal{C}[\circ_\tau] \in Ctx_{\tau_1}$ if $\mathcal{C}[\Omega^\tau] \Downarrow$ also $\mathcal{C}[M] \Downarrow$. Assume $\mathcal{C}[\Omega^\tau] \Downarrow$ for some $\mathcal{C}[\circ_\tau] \in Ctx_{\tau_1}$. We prove the lemma by induction on the length of computation.
For the base case $\mathcal{C}[\Omega^\tau] \Downarrow \mathcal{C}[\Omega^\tau]$ is surely due to $\mathcal{C}[\circ_\tau]$ because Ω^τ is divergent, so we have also $\mathcal{C}[M] \Downarrow \mathcal{C}[M]$.
Now assume the lemma valid for shorter computation. By lemma 6.2.2, $\mathcal{C}[\Omega^\tau]$ can be written uniquely as $\hat{\mathcal{E}}[\Omega^\tau]\langle \mathcal{C}_1[\Omega^\tau] \rangle$ where $\mathcal{C}_1[\Omega^\tau]$ is a redex. Consider the next step of computation:
$\hat{\mathcal{E}}[\Omega^\tau]\langle \mathcal{C}_1[\Omega^\tau] \rangle \longrightarrow \hat{\mathcal{E}}[\Omega^\tau]\langle M'' \rangle$
Observe that the occurences of Ω^τ outside the reduction context scope remain untouched, so we are interested in M''. By lemma 6.2.3, since $\Omega^\tau \in Exp_\tau$ and $\Omega^\tau \notin Can$ we have only one possibility, otherwise we contradict the assumption:
$\hat{\mathcal{E}}[\Omega^\tau]\langle \mathcal{C}_1[\Omega^\tau] \rangle \longrightarrow \hat{\mathcal{E}}[\Omega^\tau]\langle \mathcal{C}_2[\Omega^\tau] \rangle$
for some $\mathcal{C}_2[\circ_\tau] \not\equiv \circ_\tau$. By assumption $\hat{\mathcal{E}}[\Omega^\tau]\langle \mathcal{C}_2[\Omega^\tau] \rangle \Downarrow$ but it is a shorter computation hence we can apply induction hypothesis and obtain $\hat{\mathcal{E}}[M]\langle \mathcal{C}_2[M] \rangle \Downarrow$. Since $\hat{\mathcal{E}}[M]\langle \mathcal{C}_1[M] \rangle \longrightarrow \hat{\mathcal{E}}[M]\langle \mathcal{C}_2[M] \rangle$ we have the conclusion.

□

Lemma 6.4.17 (Divergence). *For each $M \in Exp_\tau$*

$$M \Uparrow \iff \Omega^\tau \cong_\tau M$$

Proof. ($\Longrightarrow$) From lemma 6.4.16 we only need to show that $M \Uparrow \Longrightarrow M \lesssim_\tau \Omega^\tau$. We show that $M \Uparrow \Longrightarrow \forall \mathcal{C}[\circ_\tau] \in Ctx_{\tau_1}$ ($\mathcal{C}[M] \Downarrow \Longrightarrow \mathcal{C}[\Omega^\tau] \Downarrow$). Assume $\mathcal{C}[M] \Downarrow$ for some $\mathcal{C}[\circ_\tau] \in Ctx_{\tau_1}$. We proceed by induction on the length of computation. For the base case $\mathcal{C}[M] \Downarrow \mathcal{C}[M]$ is surely due to $\mathcal{C}[\circ_\tau]$ because $M \Uparrow$, so we have

also $\mathcal{C}[\Omega^\tau] \Downarrow \mathcal{C}[\Omega^\tau]$.
Now assume the conclusion valid for shorter computation. By lemma 6.2.2, $\mathcal{C}[M]$ can be written uniquely as $\hat{\mathcal{E}}[M]\langle\mathcal{C}_1[M]\rangle$ where $\mathcal{C}_1[M]$ is a redex. Consider the next step of computation:
$\hat{\mathcal{E}}[M]\langle\mathcal{C}_1[M]\rangle \longrightarrow \hat{\mathcal{E}}[M]\langle M''\rangle$
Observe that the occurences of M outside the reduction context scope remain untouched, so we are interested in M''. By lemma 6.2.3, since $M \in Exp_\tau$ and $M \notin Can$ we have only one possibility, otherwise we contradict the assumption:
$\hat{\mathcal{E}}[M]\langle\mathcal{C}_1[M]\rangle \longrightarrow \hat{\mathcal{E}}[M]\langle\mathcal{C}_2[M]\rangle$
for some $\mathcal{C}_2[\circ_\tau] \not\equiv \circ_\tau$. By assumption $\hat{\mathcal{E}}[M]\langle\mathcal{C}_2[M]\rangle \Downarrow$ but it is a shorter computation hence we can apply induction hypothesis and obtain $\hat{\mathcal{E}}[\Omega^\tau]\langle\mathcal{C}_2[\Omega^\tau]\rangle \Downarrow$. Since $\hat{\mathcal{E}}[\Omega^\tau]\langle\mathcal{C}_1[\Omega^\tau]\rangle \longrightarrow \hat{\mathcal{E}}[M]\langle\mathcal{C}_2[\Omega^\tau]\rangle$ we have the conclusion.
($\Longleftarrow$) If $\Omega^\tau \cong_\tau M$ then for all $\mathcal{C}[\circ_\tau] \in Ctx_{\tau_1}$, $\mathcal{C}[\Omega^\tau] \Downarrow \Longleftrightarrow \mathcal{C}[M] \Downarrow$ but we know that for each $\mathcal{C}[\circ_\tau] \equiv \hat{\mathcal{E}}[\circ_\tau]\langle\bullet_\tau\rangle$ we have $\hat{\mathcal{E}}[\Omega^\tau]\langle\Omega^\tau\rangle \Uparrow$ which implies $\hat{\mathcal{E}}[M]\langle M\rangle \Uparrow$ and hence $M \Uparrow$. □

We can now prove the following interesting properties.

Corollary 6.4.18 (Beta reduction). *For each $x : \tau_1 \vdash M : \tau,\ A \in Exp_{\tau_1}$ we have:*

$$(\lambda x.M)A \cong_\tau M[A/x]$$

Proof. By reflexivity $x : \tau_1 \vdash M \cong_\tau M$ and so by lemma 6.4.8 we have $\emptyset \vdash \lambda x.M \cong_{\tau_1 \to \tau} \lambda x.M$. By congruence property of contextual equivalence we have $(\lambda x.M)A \cong_\tau (\lambda x.M)A$ and hence since $(\lambda x.M)A \longrightarrow M[A/x]$ by corollary 6.4.12 and transitivity $(\lambda x.M)A \cong_\tau M[A/x]$ □

Corollary 6.4.19 (Beta reduction). *For each $\langle M_1, M_2\rangle : \tau_1 \times \tau_2$ we have:*

$$\boldsymbol{fst}(\langle M_1, M_2\rangle) \cong_{\tau_1} M_1 \qquad \boldsymbol{snd}(\langle M_1, M_2\rangle) \cong_{\tau_2} M_2$$

Proof. Since $\mathbf{fst}\langle M_1, M_2\rangle \longrightarrow M_1$ and $\mathbf{snd}\langle M_1, M_2\rangle \longrightarrow M_2$ by corollary 6.4.12 the conclusion follows. □

Corollary 6.4.20 (Beta reduction). *For each $\boldsymbol{inl}(M) : \tau_1 + \tau_2$, $\boldsymbol{inr}(N) : \tau_1 + \tau_2$ we have:*

$$\boldsymbol{case}\ \boldsymbol{inl}_{\tau_1,\tau_2}(M)\ \boldsymbol{of}\ \{\boldsymbol{inl}_{\tau_1,\tau_2}(x_1).x_1 \mid \boldsymbol{inr}_{\tau_1,\tau_2}(x_2).x_2\} \cong_{\tau_1} M$$
$$\boldsymbol{case}\ \boldsymbol{inr}_{\tau_1,\tau_2}(N)\ \boldsymbol{of}\ \{\boldsymbol{inl}_{\tau_1,\tau_2}(x_1).x_1 \mid \boldsymbol{inr}_{\tau_1,\tau_2}(x_2).x_2\} \cong_{\tau_2} N$$

Proof. Since $\mathbf{case}\ \mathbf{inl}_{\tau_1,\tau_2}(M)\ \mathbf{of}\ \{\mathbf{inl}_{\tau_1,\tau_2}(x_1).x_1 \mid \mathbf{inr}_{\tau_1,\tau_2}(x_2).x_2\} \longrightarrow M$ and $\mathbf{case}\ \mathbf{inr}_{\tau_1,\tau_2}(N)\ \mathbf{of}\ \{\mathbf{inl}_{\tau_1,\tau_2}(x_1).x_1 \mid \mathbf{inr}_{\tau_1,\tau_2}(x_2).x_2\} \longrightarrow N$ by corollary 6.4.12 the conclusion follows. □

Corollary 6.4.21 (Beta reduction). *For each $\boldsymbol{abs}_{\mu T.\tau}(C) : muT.\tau$ we have:*

$$\boldsymbol{rep}(\boldsymbol{abs}_{\mu T.\tau}(C)) \cong_{\tau[\mu T.\tau/T]} C$$

Proof. Since $\mathbf{rep}(\mathbf{abs}_{\mu T.\tau}(C)) \longrightarrow C$ by corollary 6.4.12 the conclusion follows. □

We have proved the above lemmas by using transition relation, Pitts in [Pitts, 1995] uses a big-step semantics for PCFL hence he needs to introduce a new program equivalence called *kleene equivalence* in order to prove them.

6.4.3 Contextual Equivalence and Extensionality

An interesting property which will be useful in the sequel is **Extensionality** We first prove the following lemma.

Lemma 6.4.22. *For each* $x : \tau_1 \vdash N \lesssim_\tau N'$ *and* $M \in Exp_{\tau_1}$ *we have:*

$$N[M/x] \lesssim_\tau N'[M/x]$$

Proof. If $x : \tau_1 \vdash N \lesssim_\tau N'$ by lemma 6.4.8 we have $\emptyset \vdash \lambda x.N \lesssim_{\tau_1 \to \tau} \lambda x.N'$. By the congruence property of contextual equivalence for each $M \in Exp_{\tau_1}$ we have $(\lambda x.N)M \cong_\tau (\lambda x.N')M$ but by lemma 6.4.18 we have $(\lambda x.N)M \cong_\tau N[M/x]$ and $(\lambda x.N')M \cong_\tau N'[M/x]$ and hence by transitivity $N[M/x] \lesssim_\tau N'[M/x]$. □

Now we can prove the following generalization of the above lemma which will be useful in the sequel.

Lemma 6.4.23. *For all* $x_1 : \tau_1, \ldots, x_n : \tau_n \vdash N \lesssim_\tau N'$ *we have:*

$$\begin{gathered}\forall M_1 \in Exp_{\tau_1}, \ldots, M_n \in Exp_{\tau_n} \\ N[M_1/x_1, \cdots, M_n/x_n] \lesssim_\tau N'[M_1/x_1, \cdots, M_n/x_n]\end{gathered}$$

Proof. Take $x_1 : \tau_1, \ldots, x_n : \tau_n \vdash N \lesssim_\tau N'$ and general $M_1 \in Exp_{\tau_1}, \ldots, M_n \in Exp_{\tau_n}$. By lemma 6.4.8 we have:

$$x_2 : \tau_2, \ldots, x_n : \tau_n \vdash \lambda x_1.N \lesssim_\tau \lambda x_1.N'$$

and by congruence property of contextual equivalence:

$$x_2 : \tau_2, \ldots, x_n : \tau_n \vdash (\lambda x_1.N)M_1 \lesssim_\tau (\lambda x_1.N')M_1$$

Now we can repeat this argument to find:

$$\vdash (\lambda x_n. \cdots ((\lambda x_1.N)M_1) \cdots)M_n \lesssim_\tau (\lambda x_n. \cdots ((\lambda x_1.N')M_1) \cdots)M_n$$

Now by lemma 6.4.18 we have:

$$\begin{aligned}\vdash &((\lambda x_{n-1}. \cdots ((\lambda x_1.N)M_1) \cdots)M_{n-1})[M_n/x_n] \\ &\lesssim_\tau ((\lambda x_{n-1}. \cdots ((\lambda x_1.N')M_1) \cdots)M_{n-1})[M_n/x_n]\end{aligned}$$

and by repeating we have:

$$N[M_1/x_1, \cdots, M_n/x_n] \lesssim_\tau N'[M_1/x_1, \cdots, M_n/x_n]$$

□

It is interesting to note that we use extensively contexts as part of our metalanguage in the proofs of proposition as the above.

Lemma 6.4.24. *For each* $N, N' \in Exp_\tau(x_1 : \tau_1, \ldots, x_n : \tau_n)$*:*

$$\begin{gathered}\forall M_1 \in Exp_{\tau_1}, \ldots, M_n \in Exp_{\tau_n}(N[M_1/x_1, \cdots, M_n/x_n] \lesssim_\tau N'[M_1/x_1, \cdots, M_n/x_n]) \\ \Longrightarrow x_1 : \tau_1, \ldots, x_n : \tau_n \vdash N \lesssim_\tau N'\end{gathered}$$

Proof. If $N \in Exp_\tau(\emptyset)$ the conclusion follows immediately. Otherwise take $\mathcal{C}[\circ_\tau] \in Ctx_{\tau_2+\tau_3}$ such that $x \in traps(C)$ and assume that $\mathcal{C}[N] \Downarrow$. We prove the lemma by induction on the length of computation.
Since $N \notin Exp_\tau(\emptyset)$ we have $N \notin Can$ hence for the base case the only possibility is $\mathcal{C}[N] \Downarrow \mathcal{C}[N]$ depending on $\mathcal{C}[\circ]$ and hence we have $\mathcal{C}[N'] \Downarrow \mathcal{C}[N']$. Now assume the conclusion valid for shorter computation.
By lemma 6.2.2, $\mathcal{C}[M]$ can be written uniquely as $\hat{\mathcal{E}}[N]\langle \mathcal{C}_1[N]\rangle$ where $\mathcal{C}_1[N]$ is a redex. Consider the next step of computation:

$\hat{\mathcal{E}}[N]\langle \mathcal{C}_1[N]\rangle \longrightarrow \hat{\mathcal{E}}[N]\langle N''\rangle$

Observe that the occurrences of M outside the reduction context scope remain untouched, so we are interested in N''. By observing that N is neither a redex nor a value by lemma 4.2.12 we have that $\mathcal{C}_1[N] = \mathcal{E}[\mathcal{C}_2[N]]$ where $\mathcal{C}_2[N] \in Can$. Now by an analysis of transition rules, we have two distinct cases. If $\mathcal{E}$ equals $\mathbf{fst}([\bullet])$, $\mathbf{snd}([\bullet])$ $\mathbf{rep}([\bullet])$ or $\mathbf{abs}_{\mu T.\tau}([\bullet])$ then we have:

$\hat{\mathcal{E}}[N]\langle \mathcal{C}_1[N]\rangle \longrightarrow \hat{\mathcal{E}}[N]\langle \mathcal{C}_3[N]\rangle$

for some $\mathcal{C}_3[\circ_\tau]$ where the computation is uniform in N because it is not a value. Hence $\hat{\mathcal{E}}[N]\langle \mathcal{C}_3[N]\rangle \Downarrow$ but it is a shorter computation, hence we can apply induction hypothesis and obtain $\hat{\mathcal{E}}[N']\langle \mathcal{C}_3[N']\rangle \Downarrow$. Since $\hat{\mathcal{E}}[N']\langle \mathcal{C}_1[N']\rangle \longrightarrow \hat{\mathcal{E}}[N']\langle \mathcal{C}_3[N']\rangle$ we have the conclusion.
Otherwise $\mathcal{E}$ equals $\mathbf{case}\ [\bullet]\ \mathbf{of}\ \{\mathbf{inl}_{\tau_1,\tau_2}(x_1).M_1 \mid \mathbf{inr}_{\tau_1,\tau_2}(x_2).M_2\}$ or $[\bullet]M_1$, and one of the rules (app tran), (cond tran inl) or (cond tran inl) apply. We show the case $\mathcal{E} \equiv [\bullet]M_1$, the others are similar. To apply (app tran) rule $\mathcal{C}_2[N]$ should be of the form $\lambda x.\mathcal{C}'[N]$, M_1 can contain N hence we can write it as $\mathcal{C}''[N]$ so we have:

$\hat{\mathcal{E}}[N]\langle(\lambda x.\mathcal{C}'[N])\mathcal{C}''[N]\rangle \longrightarrow \hat{\mathcal{E}}[N]\langle(\mathcal{C}'[N])[\mathcal{C}''[N]/x]\rangle$

Hence $\hat{\mathcal{E}}[N]\langle(\mathcal{C}'[N])[\mathcal{C}''[N]/x]\rangle \Downarrow$ but it is a shorter computation, hence we can apply the induction hypothesis to occurrences outside the reduction context scope and obtain $\hat{\mathcal{E}}[N']\langle(\mathcal{C}'[N])[\mathcal{C}''[N]/x]\rangle \Downarrow$. The occurrences of N in $\mathcal{C}''[N]$ are untouched and hence computation is uniform in $\mathcal{C}''[N]$, so again we can apply induction hypothesis to occurrences of N in $\mathcal{C}''[N]$ and obtain $\hat{\mathcal{E}}[N']\langle(\mathcal{C}'[N])[\mathcal{C}''[N']/x]\rangle \Downarrow$. Finally consider the occurrences of N in $\mathcal{C}'[N]$. We have two cases: either $x \notin FV(N)$ or $x \in FV(N)$ and hence $x \equiv x_i$ for any $1 \leq i \leq n$. If the former holds then computation is uniform in $\mathcal{C}'[N]$ and hence we obtain $\hat{\mathcal{E}}[N']\langle(\mathcal{C}'[N'])[\mathcal{C}''[N']]\rangle \Downarrow$, and since

$\hat{\mathcal{E}}[N']\langle(\lambda x.\mathcal{C}'[N'])\mathcal{C}''[N']\rangle \longrightarrow \hat{\mathcal{E}}[N']\langle(\mathcal{C}'[N'])[\mathcal{C}''[N']/x]\rangle$

we get the conclusion.
In the latter case $\hat{\mathcal{E}}[N']\langle(\mathcal{C}'[N])[\mathcal{C}''[N']/x]\rangle \Downarrow$ can written as $\mathcal{C}_4[N[\mathcal{C}''[N']/x]]$ and since $\mathcal{C}_4[N[\mathcal{C}''[N']/x]] \Downarrow$ by assumption we have for each M' that $N[M'/x] \lesssim_\tau N[M'/x]$ hence $\mathcal{C}_4[N'[\mathcal{C}''[N']/x]] \Downarrow$ and since

$\hat{\mathcal{E}}[N']\langle(\lambda x.\mathcal{C}'[N'])\mathcal{C}''[N']\rangle \longrightarrow \hat{\mathcal{E}}[N']\langle(\mathcal{C}'[N'])[\mathcal{C}''[N']/x]\rangle$

the conclusion. □

By the above lemmas we have that contextual equivalence is **extensional** in the sense of the following corollary.

Corollary 6.4.25 (Extensionality).
For each $N, N' \in Exp_\tau(x_1 : \tau_1, \ldots, x_n : \tau_n)$ we have:

$$\forall M_1 \in Exp_{\tau_1}, \ldots, M_n \in Exp_{\tau_n}(N[M_1/x_1, \cdots, M_n/x_n] \lesssim_\tau N'[M_1/x_1, \cdots, M_n/x_n]) \iff x_1 : \tau_1, \ldots, x_n : \tau_n \vdash N \lesssim_\tau N'$$

6.4.4 Recursively defined term

In chapter 4 we have shown that the derived constructor **rec** $x.F$ is typable with respect to FPC typing rules. We now show that the usual dynamic semantics rules for **rec** $x.F$ can be derived by our relations.

Small-step

In [Gordon, 1995] Gordon gives the following small-step rule for **rec** $x.F$.

$$\mathbf{rec}\ x.F \longrightarrow F[\mathbf{rec}\ x.F/x] \qquad \text{(rec tran)}$$

We want to prove that this rule is equivalent to our semantics. In the sequel we do not consider type information only for clarity of notation. By our operational semantics we have for each **rec** $x.F \in Exp_\tau$ the following transitions.

$$\begin{array}{l}\mathbf{rec}\ x.F \equiv \\ \lambda f.(\lambda y.f(\mathbf{rep}(y)y))(\mathbf{abs}(\lambda y.f(\mathbf{rep}(y)y)))(\lambda x.F) \longrightarrow \\ (\lambda y.(\lambda x.F)(\mathbf{rep}(y)y))(\mathbf{abs}(\lambda y.(\lambda x.F)(\mathbf{rep}(y)y))) \longrightarrow \\ \lambda x.F(\mathbf{rep}(\mathbf{abs}(\lambda y.(\lambda x.F)(\mathbf{rep}(y)y)))(\mathbf{abs}(\lambda y.(\lambda x.F)(\mathbf{rep}(y)y)))) \longrightarrow \\ F[\mathbf{rep}(\mathbf{abs}(\lambda y.(\lambda x.F)(\mathbf{rep}(y)y)))(\mathbf{abs}(\lambda y.(\lambda x.F)(\mathbf{rep}(y)y)))/x]\end{array}$$

Hence we want to prove that

$$\begin{array}{c}F[\mathbf{rep}(\mathbf{abs}(\lambda y.(\lambda x.F)(\mathbf{rep}(y)y)))(\mathbf{abs}(\lambda y.(\lambda x.F)(\mathbf{rep}(y)y)))/x] \\ \cong_\tau \\ F[\mathbf{rec}\ x.F/x]\end{array}$$

By corollary 6.4.12 and transition rules we have

$$\begin{array}{c}\mathbf{rep}(\mathbf{abs}(\lambda y.(\lambda x.F)(\mathbf{rep}(y)y)))(\mathbf{abs}(\lambda y.(\lambda x.F)(\mathbf{rep}(y)y))) \\ \cong_\tau \\ \lambda y.(\lambda x.F)(\mathbf{rep}(y)y)(\mathbf{abs}(\lambda y.(\lambda x.F)(\mathbf{rep}(y)y)))\end{array}$$

and by lemma 6.4.18 the following

$$\begin{array}{c}\lambda f.(\lambda y.f(\mathbf{rep}(y)y))(\mathbf{abs}(\lambda y.f(\mathbf{rep}(y)y)))\lambda x.F \\ \cong_\tau \\ \lambda y.(\lambda x.F)(\mathbf{rep}(y)y)(\mathbf{abs}(\lambda y.(\lambda x.F)(\mathbf{rep}(y)y)))\end{array}$$

Hence by transitivity we have

$$\mathbf{rep}(\mathbf{abs}(\lambda y.(\lambda x.F)(\mathbf{rep}(y)y)))(\mathbf{abs}(\lambda y.(\lambda x.F)(\mathbf{rep}(y)y))) \cong_\tau \mathbf{rec}\ x.F$$

and by congruence property the conclusion follows. Thus it is clear that (rec tran) rule involve not a single step but different steps but we can consider it correct with respect to our small-step operational semantics, and when will be useful we will use it as an abbreviation for the reduction of recursively defined terms.

Big-step

In [Winskel, 1993] Winskel gives the following big-step rule for **rec** $x.F$.

$$\frac{F[\mathbf{rec}\ x.F/x] \Downarrow C}{\mathbf{rec}\ x.F \Downarrow C} \qquad (+\ \text{rec})$$

We want to prove that it is equivalent to our semantics. In the sequel we do not consider type information and we do not state convergence judgement for canonical terms only for clarity of notation. By our operational semantics we have for each **rec** $x.F \in Exp_\tau$ the following transitions.

$$\frac{\dfrac{\dfrac{F[\mathbf{rep}(\mathbf{abs}(\lambda y.(\lambda x.F)(\mathbf{rep}(y)y)))(\mathbf{abs}(\lambda y.(\lambda x.F)(\mathbf{rep}(y)y)))/x] \Downarrow C}{\lambda x.F(\mathbf{rep}(\mathbf{abs}(\lambda y.(\lambda x.F)(\mathbf{rep}(y)y)))(\mathbf{abs}(\lambda y.(\lambda x.F)(\mathbf{rep}(y)y)))) \Downarrow C}}{(\lambda y.(\lambda x.F)(\mathbf{rep}(y)y))(\mathbf{abs}(\lambda y.(\lambda x.F)(\mathbf{rep}(y)y))) \Downarrow C}}{\mathbf{rec}\ x.F \equiv \lambda f.(\lambda y.f(\mathbf{rep}(y)y))(\mathbf{abs}(\lambda y.f(\mathbf{rep}(y)y)))\lambda x.F \Downarrow C}$$

In the last paragraph we have show that

$$\begin{array}{c} F[\mathbf{rep}(\mathbf{abs}(\lambda y.(\lambda x.F)(\mathbf{rep}(y)y)))(\mathbf{abs}(\lambda y.(\lambda x.F)(\mathbf{rep}(y)y)))/x] \\ \cong_\tau \\ F[\mathbf{rec}\ x.F/x] \end{array}$$

and hence we can consider (+ rec) rule correct with respect to our big-step operational semantics. Thus in the sequel we will use it when we need an abbreviation for evaluation of recursively defined terms.

6.4.5 Rational completeness and syntactic continuity

We have shown how we can recursively define terms through a derived constructor **rec** $x.F$. Furthermore we have proved that it is possible to define an operational semantics for **rec** $x.F$ which corresponds with the usual operational semantics of languages which provide it as basic constructor.
Now we want to prove that the term **rec** $x.F \in Exp_\tau$ is contextual approximated by its finite unrollings. We define recursively the *n-th approximation* of **rec** $x.F$ as:

$$\begin{array}{rcl} \mathbf{rec}^0\ x.F & \stackrel{\text{def}}{=} & \Omega^\tau \\ \mathbf{rec}^{n+1}\ x.F & \stackrel{\text{def}}{=} & F[\mathbf{rec}^n\ x.F/x] \end{array}$$

We can now prove the following lemmas.

Lemma 6.4.26.

$$\forall\ n\ (\boldsymbol{rec}^n\ x.F \lesssim_\tau \boldsymbol{rec}^{n+1}\ x.F)$$

Proof. We prove it by induction on n. The base case is trivial. Assume $\mathcal{C}[\mathbf{rec}^n\ x.F] \Downarrow$ for $\mathcal{C}[\circ_\tau] \in Ctx_{\tau_1}$. $\mathcal{C}[\mathbf{rec}^{n+1}\ x.F]$ can be written as $\mathcal{C}[F[\mathbf{rec}^n\ x.F/x]] \equiv \mathcal{C}_1[\mathbf{rec}^n\ x.F]$, but we have that also $\mathcal{C}[F[\mathbf{rec}^{n-1}\ x.F/x]] \equiv \mathcal{C}_1[\mathbf{rec}^{n-1}\ x.F]$. By induction hypothesis $\mathcal{C}_1[\mathbf{rec}^{n-1}\ x.F] \Downarrow$ implies that $\mathcal{C}_1[\mathbf{rec}^n\ x.F] \Downarrow$ and hence the conclusion. □

Lemma 6.4.27.

$$\forall\ n\ (\boldsymbol{rec}^n\ x.F \lesssim_\tau \boldsymbol{rec}\ x.F)$$

Proof. For each n we want to show that for each $\mathcal{C}[\circ_\tau] \in Ctx_{\tau_1}$ if $\mathcal{C}[\mathbf{rec}^n\ x.F] \Downarrow$ then $\mathcal{C}[\mathbf{rec}\ x.F] \Downarrow$. We prove this by induction on n.

Base $\mathbf{rec}^0\ x.F \stackrel{\text{def}}{=} \Omega^\tau$ hence by syntactic bottom property if $\mathcal{C}[\Omega^\tau] \Downarrow$ we have $\mathcal{C}[\mathbf{rec}\ x.F] \Downarrow$

Induction Step Assuming $\forall\ \mathcal{C}[\circ_\tau] \in Ctx_{\tau_1}\ \mathcal{C}[\mathbf{rec}^n\ x.F] \Downarrow \Longrightarrow \mathcal{C}[\mathbf{rec}\ x.F] \Downarrow$ we want to show $\forall\ \mathcal{C}[\circ_\tau] \in Ctx_{\tau_1}\ \mathcal{C}[\mathbf{rec}^{n+1}\ x.F] \Downarrow \Longrightarrow \mathcal{C}[\mathbf{rec}\ x.F] \Downarrow$.
$\mathcal{C}[\mathbf{rec}^{n+1}\ x.F] \equiv \mathcal{C}[F[\mathbf{rec}^n\ x.F/x]] \equiv \mathcal{C}_1[\mathbf{rec}^n\ x.F]$, if $\mathcal{C}[\mathbf{rec}^{n+1}\ x.F] \Downarrow$ then $\mathcal{C}_1[\mathbf{rec}^n\ x.F] \Downarrow$ hence by induction hypothesis $\mathcal{C}_1[\mathbf{rec}\ x.F] \Downarrow$ but this implies that $\mathcal{C}[F[\mathbf{rec}\ x.F/x]] \Downarrow$ and so by (+ fix) $\mathcal{C}[\mathbf{rec}\ x.F] \Downarrow$.

□

Lemma 6.4.28. *Let* $\mathcal{C}[\circ_\tau] \in Ctx_{\tau_1}$ *then*

$$\forall\ n\ (\mathcal{C}[\boldsymbol{rec}^n\ x.F] \lesssim_{\tau_1} \mathcal{C}[\boldsymbol{rec}\ x.F])$$

Proof. It follows immediately by lemma 6.4.27 and precongruence property of $\lesssim$. □

Lemma 6.4.29.

$$(\mathcal{C}[\boldsymbol{rec}\ x.F] \Downarrow \Longrightarrow\ \exists\ n\ \mathcal{C}[\boldsymbol{rec}^n x.F] \Downarrow)$$

Proof. We need to show that $\forall \mathcal{C}[\circ_\tau] : \tau_1$ if $\mathcal{C}[\mathbf{rec}\ x.F] \Downarrow$ then exists n such that $\mathcal{C}[\mathbf{rec}^n\ x.F] \Downarrow$. We prove this on the length of the computation.

Base $\mathcal{C}[\mathbf{rec}\ x.F] \Downarrow \mathcal{C}[\mathbf{rec}\ x.F]$ this implies that for all n $\mathcal{C}[\mathbf{rec}^n\ x.F] \Downarrow \mathcal{C}[\mathbf{rec}^n\ x.F]$.

Induction Step By lemma 6.2.2, we can write $\mathcal{C}[\mathbf{rec}\ x.F]$ uniquely as $\hat{\mathcal{E}}[\mathbf{rec}\ x.F]\langle\mathcal{C}_1[\mathbf{rec}\ x.F]\rangle$. Consider the next step of computation:

$\hat{\mathcal{E}}[\mathbf{rec}\ x.F]\langle\mathcal{C}_1[\mathbf{rec}\ x.F]\rangle \longrightarrow \hat{\mathcal{E}}[\mathbf{rec}\ x.F]\langle M\rangle$

Observe that the occurences of $\mathbf{rec}\ x.F$ outside the reduction context scope remain untouched, so we are interested in M. We have only two possibilities:

$\hat{\mathcal{E}}[\mathbf{rec}\ x.F]\langle\mathcal{C}_1[\mathbf{rec}\ x.F]\rangle \longrightarrow \hat{\mathcal{E}}[\mathbf{rec}\ x.F]\langle\mathcal{C}_2[\mathbf{rec}\ x.F]\rangle$

where $\mathcal{C}_1 \not\equiv \circ_\tau$ but by induction hypothesis exists n such that $\hat{\mathcal{E}}[\mathbf{rec}^n\ x.F]\langle\mathcal{C}_2[\mathbf{rec}^n\ x.F]\rangle \Downarrow$ and hence from

$\hat{\mathcal{E}}[\mathbf{rec}^n\ x.F]\langle\mathcal{C}_1[\mathbf{rec}^n\ x.F]\rangle \longrightarrow \hat{\mathcal{E}}[\mathbf{rec}^n\ x.F]\langle\mathcal{C}_2[\mathbf{rec}^n\ x.F]\rangle$

we have the conclusion.

Otherwise we have

$\hat{\mathcal{E}}[\mathbf{rec}\ x.F]\langle\mathbf{rec}\ x.F\rangle \longrightarrow \hat{\mathcal{E}}[\mathbf{rec}\ x.F]\langle F[\mathbf{rec}\ x.F/x]\rangle$

hence where $\mathcal{C}_1 \equiv \circ_\tau$ but by fixing the occurrences of $\mathbf{rec}\ x.F$ outside the reduction context scope we can write $\hat{\mathcal{E}}[\mathbf{rec}\ x.F]\langle F[\mathbf{rec}\ x.F/x]\rangle$ as $\hat{\mathcal{E}}[\mathbf{rec}\ x.F]\langle\mathcal{C}_3[\mathbf{rec}\ x.F]\rangle$ and by induction hypothesis exists n such that $\hat{\mathcal{E}}[\mathbf{rec}^n\ x.F]\langle\mathcal{C}_3[\mathbf{rec}^n\ x.F]\rangle \Downarrow$ and hence $\hat{\mathcal{E}}[\mathbf{rec}^n\ x.F]\langle F[\mathbf{rec}^n\ x.F/x]\rangle \Downarrow$ but this can be written as $\hat{\mathcal{E}}[\mathbf{rec}^n\ x.F]\langle\mathbf{rec}^{n+1}\ x.F]\rangle \Downarrow$ and since $\mathbf{rec}^n\ x.F \lesssim_\tau \mathbf{rec}^{n+1}\ x.F$ we have the conclusion.

□

Lemma 6.4.30 (Rational Completeness).

$$\mathbf{rec}\ x.F \lesssim_\tau M \iff \forall n \in \mathbb{N}\ (\mathbf{rec}\ ^n x.F \lesssim_\tau M)$$

Proof.

($\Longrightarrow$) By lemma 6.4.27 and transitivity.

($\Longleftarrow$) Assume $\forall n \in \mathbb{N}\ (rec\ ^n x.F \lesssim_\tau M)$. Taken $\mathcal{C}[\circ_\tau] \in Ctx_{\tau_1}$ such that $\mathcal{C}[\mathbf{rec}\ x.F] \Downarrow$ by lemma 6.4.29 we have that there exists n such that $\mathcal{C}[\mathbf{rec}\ x.F] \Downarrow \Longrightarrow \mathcal{C}[\mathbf{rec}^n\ x.F] \Downarrow$. By assumption $\mathcal{C}[\mathbf{rec}^n\ x.F] \Downarrow \Longrightarrow \mathcal{C}[M] \Downarrow$ hence since we have choosen arbitrarily $\mathcal{C}[\circ_\tau]$ we have the conclusion.

□

Lemma 6.4.31 (Syntactic Continuity).

$$\mathcal{C}[\mathbf{rec}\ x.F] \lesssim_\tau M \iff \forall\ n \in \mathbb{N}(\mathcal{C}[\mathbf{rec}\ ^n x.F] \lesssim_\tau M)$$

Proof.

($\Longrightarrow$) By lemma 6.4.28 and transitivity.

($\Longleftarrow$) Assume $\forall n \in \mathbb{N}\ (\mathcal{C}[\mathbf{rec}\ ^n x.F] \lesssim_\tau M)$. Take $\mathcal{C}_1[\circ_\tau] \in Ctx_{\tau_1}$ such that $\mathcal{C}_1[\mathcal{C}[\mathbf{rec}\ x.F]] \Downarrow$. Hence there exists $\mathcal{C}_2[\circ] \equiv \mathcal{C}_1[\mathcal{C}[\circ]]$ such that $\mathcal{C}_2[\mathbf{rec}\ x.F] \Downarrow$. By lemma 6.4.29 we have that there exists n such that $\mathcal{C}_2[\mathbf{rec}\ x.F] \Downarrow \Longrightarrow \mathcal{C}_2[\mathbf{rec}^n\ x.F] \Downarrow$. By assumption $\mathcal{C}_2[\mathbf{rec}^n\ x.F] \equiv \mathcal{C}_1[\mathcal{C}[\mathbf{rec}^n\ x.F]] \Downarrow \Longrightarrow \mathcal{C}_1[M] \Downarrow$ hence since we have choosen arbitrarily $\mathcal{C}_1[\circ_\tau]$ we have the conclusion.

□

6.5 Guide to references

We have so far generalized the notion of contexts. Contexts are well known and studied device in different fields of computer science. We have introduced only context for FPC, but it is clear that it is possible to define a notion of context in each formal framework. We have seen that our definition of context does not preserve α-equivalence. Pitts in [Pitts, 1994] gives an alternative treatment of contexts which do not descend below the level of abstraction of α-equivalence classes of expressions by introducing a notion of "function variables".
The possible generalization do not only regard contexts but also contextual equivalence. We have already remarked that contextual equivalence is an instance of observational equivalence. The genericity of the notion has caused the proliferation of names in literature for the same equivalence, this can sometimes be misleading. Contextual equivalence was first introduced by Morris in [Morris, 1968] with the name of *extensional equivalence*. Howe in [Howe, 1989] use the name *contextual congruence*, Smith in [Smith, 1991] *observational congruence*, Mason, Talcott and Smith in [Mason et al., 1996] *operational equivalence* and Ong in [Ong, 1995], Pitts in [Pitts, 1995] *observational equivalence*. It must be clear that they are different names for the same notion.
The definitions of contextual equivalence depend also from the choices of observable types, for an interesting discussion on language constructors and observable

types see [Pitts, 1995] and [McCusker, 1996a].
We have proved different properties of contextual equivalence with respect to FPC programs and our operational semantics. In particular we have proved *rational completeness* and *syntactic continuity* which seem to be the more interesting properties. Our proofs are not too elegant but they are extremely direct and "operational". Our proofs follow in spirit the ones by Smith in [Smith, 1991], Mason, Talcott and Smith in [Mason et al., 1996] and by Sands in [Sands, 1997]. Pitts in [Pitts, 1995], Birkedal and Harper in [Birkedal and Harper, 1999] prove the properties by a more general and elegant method.

Chapter 7

Other Equivalence notions

> Entities are not to be multiplied beyond necessity.
> Entities are not to be reduced or eliminated beyond necessity.
> Properties of entities are not to be multiplied beyond necessity.
> Properties of entities are not to be reduced or eliminated beyond necessity
>
> "Occam's Sword", an extension of the famous "Occam's Razor".

Contextual equivalence is the natural operational equivalence relation. Our formulation is useful when we need to prove that two programs are different, in fact we need only to find a context where the two programs differs. Conversely, in order to prove the equality of two programs we need to show that the two program behaviours are equivalent in each context. This quantification in general is not easy to prove. For this reason, in the last years emerged different formulations of contextual equivalence. Since these formulations are simpler they can be used when we need to prove properties of contextual equivalence which are difficult to prove otherwise.
In the sequel, for simplicity, we usually prefer to begin to work with contextual equivalence restricted to closed terms, then in some cases we will show how to extends the results to open terms. Sometimes we use different but intuitively equivalent formulations of contextual equivalence.

7.1 Kleene Equivalence

We first introduce a simple preorder relation and its associated equivalence. Following Pitts in [Pitts, 1995], Birkedal and Harper in [Birkedal and Harper, 1999] we call this relation **Kleene equivalence**. As emphasized by Pitts this terminology is adopted from the logic of partially defined expressions.

Definition 7.1.1 (Kleene preorder). *For each* $M, M' \in Exp_\tau$*:*

$$M \lesssim_\tau^{kl} M' \stackrel{\text{def}}{=} \forall C \in Can_\tau \ (M \Downarrow C \Longrightarrow M' \Downarrow C)$$

Definition 7.1.2 (Kleene equivalence). *For each* $M, M' \in Exp_\tau$*:*

$$M \cong_\tau^{kl} M' \stackrel{\text{def}}{=} M \lesssim_\tau^{kl} M' \ \& \ M' \lesssim_\tau^{kl} M$$

It is easy to verify the following lemma.

Lemma 7.1.3. *For each τ:*

$\lesssim_\tau^{kl}$ *is a preorder and* $\cong_\tau^{kl}$ *is the induced equivalence relation on* Exp_τ

The two following lemmas show that Kleene equivalence is contained in contextual equivalence.

Lemma 7.1.4 ($\lesssim^{kl} \subseteq \lesssim$). *For all $M, M' \in Exp_\tau$:*

$$M \lesssim_\tau^{kl} M' \Longrightarrow M \lesssim_\tau M'$$

Proof. For all $M, M' \in Exp_\tau$ such that $M \lesssim_\tau^{kl} M'$ we have two cases: If $M \Downarrow C$ for some $C \in Can_\tau$ then $M' \Downarrow C$. By lemma 6.4.13 we have $M \lesssim_\tau C$ and by lemma 6.4.13 $C \lesssim_\tau M'$. By transitivity the conclusion. Otherwise $M \Uparrow$, by lemma 6.4.17 $M \lesssim_\tau \Omega^\tau$ and by lemma 6.4.16 $\Omega^\tau \lesssim_\tau M'$. By transitivity the conclusion. □

Lemma 7.1.5 ($\cong^{kl} \subseteq \cong$). *For all $M, M' \in Exp_\tau$:*

$$M \cong_\tau^{kl} M' \Longrightarrow M \cong_\tau M'$$

Clearly the converse of the above lemma is not generally true. This means that Kleene equivalence is a notion of program equivalence which does not capture the entire observational meaning of a term. In section 6.3 we have claimed that in the definition of contextual equivalence if we choose as observation for our language the convergence to the same value, we obtain too strong an equivalence relation. We now define a relation $\cong_\tau^{Can} \subseteq \mathcal{P}(Exp \times Exp)$ by observing convergence to the same value for each $M, N \in Exp$ as:

$$M \cong_\tau^{Can} N \iff \forall \tau_1, \forall \mathcal{C}[\circ_\tau] \in Exp_{\tau_1}\ (\mathcal{C}[M] \Downarrow C \iff \mathcal{C}[N] \Downarrow C)$$

It is easy to prove the following proposition.

Proposition 7.1.6. *For each $M, M' \in Exp_\tau$:*

$$M \cong_\tau^{Can} M' \Longrightarrow M \cong_\tau^{kl} M'$$

Proof. By lemma 4.2.19 we have that for each $M \in Exp_\tau$ either M evaluates to some $C \in Can$ or M diverges. Since evaluation is deterministic we have that C is unique. For each $M, M' \in Exp_\tau$ such that $M \cong_\tau^{Can} M'$ we have that if $M \Downarrow C$ where $C \in Can_\tau$ then $M' \Downarrow C$, otherwise we have that context $\mathcal{C}[\circ_\tau] \equiv \circ_\tau$ distinguishes beetween the two terms. If $M \Uparrow$ then also $M' \Uparrow$, and hence $M \cong_\tau^{kl} M'$. □

Hence the notion of equivalences where observations are on convergence to the same value are too strong in the sense that they distinguish beetween canonical form terms which can be considered operationally equivalent. We remark that the above propositions depend on the fact that we have chosen a lazy evaluation and to observe convergence at every type.
Nevertheless, Kleene equivalence is useful in the formulation of program equivalence notions where the language has ground types and convergence is observed at these ground types (for example see [Birkedal and Harper, 1999]).

7.2 Experimental Equivalence and Context Lemma

In [Milner, 1977] Milner show that contextual equivalence, defined by observing convergence of terms in all program contexts, for a combinatory-logic form of PCF is equivalent to the relation obtained by observing the convergences to values of all possible applications of terms to arguments. This result is well known as **context lemma**. This can be stated as:

7.2.1 (Milner Context Lemma). *For any closed PCF terms M and M'*

$$M \cong M' \iff \forall N_1, \ldots, N_m \ (MN_1, \ldots, N_m \Downarrow \Longleftrightarrow M'N_1, \ldots, N_m \Downarrow)$$

Milner's context lemma is based on the fact that for the simply typed language PCF the only type constructor is $\rightarrow$, this means that each term has a functional behaviour. By the well known *extensionality* principle for functions we know that two functions have the same behaviour if and only if for each arguments they compute the same value.
Hence we can restrict our attention only to contexts of the form $\mathcal{C}[\circ] \equiv \circ\, \mathcal{C}_1[\circ]$. This means to restrict our attention only to the set of contexts which are really useful to test the behavior of a term, for PCF they are called *applicative contexts*. Since we are interested in a language where there are different type constructors, we need to find a set of contexts which are useful to test FPC terms. It is easy to verify that they are exactly the evaluation contexts, defined before in 4.2.10.
We can define **experimental preorder** on closed terms as follows.

Corollary 7.2.2 (Experimental preorder). *For each $M, M' \in Exp_\tau$:*

$$M \lesssim_\tau^{Exp} M' \stackrel{\text{def}}{=} \forall \vec{\mathcal{E}}[\bullet_\tau] \in Ectx_{\tau_1} \ \vec{\mathcal{E}}[M] \Downarrow \Longleftrightarrow \vec{\mathcal{E}}[M'] \Downarrow$$

As usual we can now define the induced equivalence relation.

Corollary 7.2.3 (Experimental Equivalence). *For each $M, M' \in Exp_\tau$:*

$$M \cong_\tau^{Exp} M' \stackrel{\text{def}}{=} \forall \vec{\mathcal{E}}[\bullet_\tau] \in Ectx_{\tau_1} \ \vec{\mathcal{E}}[M] \Downarrow \iff \vec{\mathcal{E}}[M'] \Downarrow$$

We read $M \cong_\tau^{Exp} M'$ as "M is experimentally equivalent to M'". As for contextual equivalence it is clear that applicative equivalence is a family of relations, one for each $\tau \in Type$. When we write $\cong^{Exp}$ we clearly mean the entire family.
In the language as PCF, where the only type constructor is $\rightarrow$, the equivalent notion of experimental equivalence is called *applicative equivalence*. We have chosen, following [Gordon, 1995] and [Birkedal and Harper, 1999], the name *experimental equivalence* to stress that we need to test (or experiment) terms in different contexts.
Definition 7.2.3 is intuitively coinductive. Following Gordon in [Gordon, 1995] we now want to emphasize this fact by characterizing experimental equivalence as the greatest fixed point of a monotone operator on the complete lattice of relations on programs. We can now define an operator

$$\phi_{Exp} : \mathcal{P}(Exp \times Exp) \rightarrow \mathcal{P}(Exp \times Exp)$$

as

$$\phi_{Exp}(X) \stackrel{\text{def}}{=} \{(M,N)|M \Downarrow \iff N \Downarrow \;\&$$
$$\forall \mathcal{E} \in \mathit{Exper} \text{ such that } \mathcal{E}[M], \mathcal{E}[N] \in \mathit{Exp} : (\mathcal{E}[M], \mathcal{E}[N]) \in X\}$$

It is easy to verify that ϕ_{Exp} is monotonic and hence it possesses a greatest fixed point $\nu X.\phi_{Exp}(X)$. In particular ϕ_{Exp} is **extensional**, then by lemma 2.5.7 we have that $\nu X.\phi_{Exp}(X)$ is an equivalence relation. Furthermore we can prove the following lemma.

Lemma 7.2.4.

$$\cong^{Exp} = \nu X.\phi_{Exp}(X)$$

Proof. By lemmas 4.2.11 we have that each $\vec{\mathcal{E}}[\bullet]$ can be decomposed uniquely as $\mathcal{E}_1[\cdots[\mathcal{E}_n[\circ]]\cdots]$. Hence clearly $\cong_{Exp}$ is ϕ_{Exp}-dense and so by coinduction we have $\cong_{Exp} \subseteq \nu X.\phi_{Exp}(X)$. Now for each $M, N \in \mathit{Exp}_\tau$ such that $(M,N) \in \nu X.\phi(X)$, we have $M \Downarrow \iff N \Downarrow$ because, by fixpoint property, we have $\phi(\nu X.\phi(X)) = \nu X.\phi(X)$. Take any $\vec{\mathcal{E}}[\circ_\tau]$ such that $\vec{\mathcal{E}}[M], \vec{\mathcal{E}}[N] \in \mathit{Exp}_{\tau_1}$ and $\vec{\mathcal{E}}[M] \Downarrow$. Again by fixpoint property we have $(\vec{\mathcal{E}}[M], \vec{\mathcal{E}}[N]) \in \nu X.\phi(X)$ hence by induction on the structure of $\vec{\mathcal{E}}[\circ]$ and lemma 4.2.11 we have $\vec{\mathcal{E}}[M] \Downarrow \iff \vec{\mathcal{E}}[M'] \Downarrow$. □

It is easy to verify that the rules are deterministic hence we can characterize experimental equivalence as:

$$\bigcap_{n \in \mathbb{N}} \phi^n_{Exp}(Exp) \;=\; \nu X.\phi_{Exp}(X)$$

Hence we have characterized coinductively experimental equivalence. Furthermore we have that it can be costructively defined with respect to $\subseteq$ and with bases *Exp*. This permits to reason coinductively to prove that two programs are experimental equivalence.

Lemma 7.2.5. *For each $X \subseteq \mathit{Exp} \times \mathit{Exp}$ we have*

$$X \subseteq \phi_{Exp}(X) \Longrightarrow X \subseteq \cong^{Exp}$$

This can be view as stating that for each $M, N \in \mathit{Exp}$, if there exists a relation $\mathcal{S} \subseteq \mathit{Exp} \times \mathit{Exp}$ such that $\{(M,N)|M \Downarrow \iff N \Downarrow\} \subseteq \mathcal{S}$ and $M \;\mathcal{S}\; N$ and for each $\mathcal{E} \in \mathit{Exper}$ such that $\mathcal{E}[M], \mathcal{E}[N] \in \mathit{Exp}$ we have $\mathcal{E}[M] \;\mathcal{S}\; \mathcal{E}[N]$ then $M \cong^{Exp} N$.
To be an attractive equivalence relation, experimental equivalence must satisfies the following lemma which can be seen as a version for FPC of Milner context lemma. We now consider a definition of contextual equivalence where the observation are at any type. It is clear that it is equivalent to the one defined in 6.4.3. The proof is an evolution of the proof due to Berry in [Berry, 1981].

Lemma 7.2.6 (Context Lemma for Closed Terms).

$$M \cong^{Exp}_\tau N \iff M \cong_\tau N$$

Proof. Clearly since $Ectx \subseteq Ctx$ if $M \cong_\tau N$ then $M \cong_\tau^{Exp} N$. Now to prove the reverse implication assume for some $\mathcal{C}[\circ_\tau] \in Ctx_{\tau_1}$, $\mathcal{C}[M] \Downarrow$. We proceed by induction on the length of the computation.
For the base case $\mathcal{C}[M]$ either is a value and it depends from $\mathcal{C}[\circ_\tau]$ and so also $\mathcal{C}[N]$ is a value or $\mathcal{C}[\circ_\tau] \equiv \circ_\tau$ and hence $M \Downarrow M$ but hence since $M \cong_\tau^{app} N$ we have $N \Downarrow N$.
Now assume the lemma valid for shorter computation. By lemma 6.2.2, $\mathcal{C}[M]$ can be written uniquely as $\hat{\mathcal{E}}[M]\langle \mathcal{C}_1[M]\rangle$ where $\mathcal{C}_1[M]$ is a redex. Consider the next step of computation:
$\hat{\mathcal{E}}[M]\langle \mathcal{C}_1[M]\rangle \longrightarrow \hat{\mathcal{E}}[M]\langle M'\rangle$
Observe that the occurences of M outside the reduction context scope remain untouched, so we are interested in M'. By lemma 6.2.3, since $M \in Exp_\tau$ we have three distinct cases depending on the structure of M':
$\hat{\mathcal{E}}[M]\langle \mathcal{C}_1[M]\rangle \longrightarrow \hat{\mathcal{E}}[M]\langle \mathcal{C}_2[M]\rangle$
for some $\mathcal{C}_1[\circ_\tau] \not\equiv \circ_\tau$ and $\mathcal{C}_1[\circ_\tau] \not\equiv \mathcal{E}[\circ_\tau]$. By assumption $\hat{\mathcal{E}}[M]\langle \mathcal{C}_2[M]\rangle \Downarrow$ but it is a shorter computation hence we can apply induction hypothesis and obtain $\hat{\mathcal{E}}[N]\langle \mathcal{C}_2[N]\rangle \Downarrow$. Since $\hat{\mathcal{E}}[N]\langle \mathcal{C}_1[N]\rangle \longrightarrow \hat{\mathcal{E}}[N]\langle \mathcal{C}_2[N]\rangle$ we have the conclusion.
Now consider
$\hat{\mathcal{E}}[M]\langle \mathcal{C}_1[M]\rangle \longrightarrow \hat{\mathcal{E}}[M]\langle \mathcal{C}_1[M']\rangle$
for some M' where $\mathcal{C}_1[\circ_\tau] \equiv \circ_\tau$. Hence $\hat{\mathcal{E}}[M]\langle M'\rangle \Downarrow$ and fixed M' by induction hypothesis $\hat{\mathcal{E}}[N]\langle M'\rangle \Downarrow$. Then since reduction is deterministic we have:
$\hat{\mathcal{E}}[N]\langle M\rangle \longrightarrow \hat{\mathcal{E}}[N]\langle M'\rangle$
But now we have $\hat{\mathcal{E}}[N]\langle M\rangle \equiv \mathcal{E}\langle M\rangle$ and $\vec{\mathcal{E}}[M] \Downarrow$ hence since $M \cong_\tau^{Exp} N$ we have $\vec{\mathcal{E}}[N] \Downarrow$ and so $\mathcal{C}[N] \Downarrow$
Finally consider
$\hat{\mathcal{E}}[M]\langle \mathcal{C}_1[M]\rangle \longrightarrow \hat{\mathcal{E}}[M]\langle \mathcal{C}_3[M]\rangle$
for some $\mathcal{C}_3$ where $\mathcal{C}_1[\circ_\tau] \equiv \mathcal{E}[\circ_\tau]$. Hence $\hat{\mathcal{E}}[M]\langle \mathcal{C}_3[M]\rangle \Downarrow$ and fixed $\mathcal{C}_3[M]$ by induction hypothesis $\hat{\mathcal{E}}[N]\langle \mathcal{C}_3[M]\rangle \Downarrow$. Then since reduction is deterministic we have:
$\hat{\mathcal{E}}[N]\langle \mathcal{E}[M]\rangle \longrightarrow \hat{\mathcal{E}}[N]\langle \mathcal{C}_3[M]\rangle$
But now we have $\hat{\mathcal{E}}[N]\langle \mathcal{E}[M]\rangle \equiv \mathcal{E}'\langle M\rangle$ and $\vec{\mathcal{E}'}[M] \Downarrow$ hence since $M \cong_\tau^{Exp} N$ we have $\vec{\mathcal{E}'}[N] \Downarrow$ and so $\mathcal{C}[N] \Downarrow$

□

Hence in order to prove that two programs are contextually equivalent we have a proof principle which we call **Experimental coinduction** defined as follows.

Lemma 7.2.7 (Experimental Coinduction). *For each $M, N \in Exp_\tau$ and $\mathcal{S} \subseteq Exp \times Exp$ such that $\{(M,N) | M \Downarrow \iff N \Downarrow\} \subseteq \mathcal{S}$ and for each $\mathcal{E} \in Exper$ such that $\mathcal{E}[M], \mathcal{E}[N] \in Exp$*

$$M \;\mathcal{S}\; N \;\&\; \mathcal{E}[M] \;\mathcal{S}\; \mathcal{E}[N] \Longrightarrow M \cong_\tau N$$

In [Gordon, 1995] Gordon shows how it is possible to improve this coinductive principle with *strong coinduction*. It is easy to extend experimental equivalence to open terms.

Definition 7.2.8. *An **expression substitution** γ for a type environment Γ is a finite map $\gamma : dom(\Gamma) \to Exp$ with the following properties:*

(i) $dom(\gamma) = dom(\Gamma)$

(ii) $\forall x \in dom(\Gamma) : \emptyset \vdash \gamma(x) : \Gamma(x)$

It is clear that an expression substitution γ for a type environment $x_1 : \tau_1, \ldots, x_n : \tau_n$ applied to a term $M \in Exp_\tau(x_1 : \tau_1, \ldots, x_n : \tau_n)$ such that $\gamma(x_1) = M_1, \ldots, \gamma(x_n) = M_n$ can be written as $M[M_1/x_1, \ldots, M_n/x_n]$.

Definition 7.2.9 (Open Experimental Preorder). *Let $M, N \in Exp_\tau(\Gamma)$ for some $\Gamma \in TEnv, \tau \in Type$. We say that M **experimentally approximates** N, written $\Gamma \vdash M \lesssim_\tau^{Exp} N$ if and only if for each expression substitution γ for Γ we have:*

$$\gamma(M) \lesssim_\tau^{Exp} \gamma(N)$$

Analougusly we can define define experimental equivalence.

Definition 7.2.10 (Open Experimental Equivalence). *Let $M, N \in Exp_\tau(\Gamma)$ for some $\Gamma \in TEnv, \tau \in Type$. We say that M is **experimentally equivalent** to N, written $\Gamma \vdash M \cong_\tau^{Exp} N$ if and only if for each expression substitution γ for Γ we have:*

$$\gamma(M) \cong_\tau^{Exp} \gamma(N)$$

It is easy to verify the following:

Lemma 7.2.11 (Context Lemma for Open Terms).

$$\Gamma \vdash M \cong_\tau^{Exp} N \iff \Gamma \vdash M \cong_\tau N$$

Proof. Clearly by definition $\Gamma \vdash M \cong_\tau N$ implies $\Gamma \vdash M \cong_\tau^{Exp} N$. For the reverse implication consider $\Gamma \vdash M \cong_\tau^{Exp} N$ this means that for each expression substitution γ for Γ we have $\gamma(M) \cong_\tau^{Exp} \gamma(N)$ but if $\Gamma = \{x_1 : \tau_1, \ldots, x_n : \tau_n\}$ this means that for each $M_1 \in Exp_{\tau_1}, \ldots, M_n \in Exp_{\tau_n}$ we have:

$$N[M_1/x_1, \cdots, M_n/x_n] \cong_\tau^{Exp} N'[M_1/x_1, \cdots, M_n/x_n]$$

but by context lemma 7.2.6 we have

$$N[M_1/x_1, \cdots, M_n/x_n] \cong_\tau N'[M_1/x_1, \cdots, M_n/x_n]$$

and by extensionality property

$$x_1 : \tau_1, \ldots, x_n : \tau_n \vdash N \cong_\tau N'$$

□

Another interesting notion closed related to Experimental equivalence is *CIU equivalence* introduced by Mason and Talcott in [Mason and Talcott, 1991], where CIU means Closed Instances of all Uses. Birkedal and Harper in [Birkedal and Harper, 1999] define experimental equivalence using a notion of approximation on value substitution. They have chosen that definition for a technical reason, but they prove that their formulation is equivalent to our.

7.3 Similarity and Bisimilarity

Another interesting notion of programs equivalence derives by works of Milner [Milner, 1989] and Park [Park, 1981] in the field of concurrency. This is the notion of **bisimilarity**. By quoting Gordon in the introduction of [Gordon, 1995]:

> Bisimilarity is based on the intuition that a derivation tree represents the behaviour of a program. We say two programs are **bisimilar** if their derivation trees are the same when one ignores the syntactic structure at the nodes. Hence bisimilarity is a way to compare behaviour, represented by actions, whilst discarding syntactic structure.

In the last years bisimilarity has been applied to different deterministic functional languages. Abramsky and Ong in [Abramsky, 1990], [Ong and Abramsky, 1993] used a version of bisimilarity, based on evaluation relation, for a *lazy* version of lambda calculus. They called it **applicative bisimulation**. Howe in [Howe, 1989] used a version of bisimilarity generalized to a lazy computation system. Gordon in [Gordon, 1995] defined first a labelled transition system for PCF and then bisimilarity exactly as in CCS. Pitts in [Pitts, 1995] used a version of bisimilarity, based on evaluation relation, for his language PCFL.
Bisimilarity can be easily defined coinductively hence it makes possible to reason coinductively about program relations. Furthermore we must prove that bisimilarity coincides with contextual equivalence in order to reason coinductively on program equivalence.
We first should define the notion of **simulation** and **bisimulation** for FPC. For this reason we define two operators on the complete lattice of program relation $\langle\ \rangle : \mathcal{P}(Exp \times Exp) \to \mathcal{P}(Exp \times Exp)$ and $[\ ] : \mathcal{P}(Exp \times Exp) \to \mathcal{P}(Exp \times Exp)$. They are defined in table 7.1 and 7.2 respectively. It is easy to verify that for each $\mathcal{R} \in \mathcal{P}(Exp \times Exp)$ we can define $[\mathcal{R}]$ exactly as $\langle \mathcal{R} \rangle \cap \langle \mathcal{R}^{op} \rangle^{op}$. Furthermore both $[\ ]$ and $\langle\ \rangle$ are monotonic hence they possess greatest fixed points $\nu X.\langle X \rangle$ and $\nu X.[X]$ respectively.

Definition 7.3.1. *Given a family of relation $\mathcal{S} \in \mathcal{P}(Exp \times Exp)$ we say that it is an **FPC simulation** if $\mathcal{S} \subseteq \langle \mathcal{S} \rangle$; furthermore we say that it is an **FPC bisimulation** if $\mathcal{S} \subseteq [\mathcal{S}]$.*

Definition 7.3.2 (Similarity). *For each $M, N \in Exp_\tau$ we say that they are **similar**, written $M \lesssim_\tau^{bis} N$ if and only if (M, N) are in the greatest fixed point of $\langle\ \rangle_\tau$. Hence we have:*

$$\lesssim^{bis} \stackrel{\text{def}}{=} \nu X.\langle X \rangle$$

Definition 7.3.3 (Bisimilarity). *For each $M, N \in Exp_\tau$ we say that they are **bisimilar**, written $M \cong_\tau^{bis} N$ if and only if (M, N) are in the greatest fixed point of $[\ ]_\tau$. Hence we have:*

$$\cong^{bis} \stackrel{\text{def}}{=} \nu X.[X]$$

It is easy to verify that $\langle\ \rangle$ is **pre-extensional** hence by lemma 2.5.5 we have that $\nu X.\langle X \rangle$ is a preorder on Exp. Furthermore $[\ ]$ is **extensional** hence

$M\langle\mathcal{R}\rangle_{\tau\to\tau_1} N$	$\stackrel{def}{\iff}$ $(M \Downarrow \lambda x.M' \Longrightarrow \exists\lambda x.N'(N \Downarrow \lambda x.N' \;\&\; \forall A \in Exp_\tau(M[A/x]\mathcal{R}_{\tau_1} N[A/x])))$	(sim fun)
$M\langle\mathcal{R}\rangle_{\tau_1\times\tau_2} N$	$\stackrel{def}{\iff}$ $(M \Downarrow \langle M_1, M_2\rangle \Longrightarrow \exists\langle N_1, N_2\rangle(N \Downarrow \langle N_1, N_2\rangle \;\&\; (M_1\mathcal{R}_{\tau_1} N_1 \;\&\; M_2\mathcal{R}_{\tau_2} N_2)))$	(sim pair)
$M\langle\mathcal{R}\rangle_{\tau_1+\tau_2} N$	$\stackrel{def}{\iff}$ $(M \Downarrow \mathbf{inl}(M_1) \Longrightarrow \exists N_1(N \Downarrow \mathbf{inl}(N_1) \;\&\; (M_1\mathcal{R}_{\tau_1} N_1)))$	(sim sum inl)
$M\langle\mathcal{R}\rangle_{\tau_1+\tau_2} N$	$\stackrel{def}{\iff}$ $(M \Downarrow \mathbf{inr}(M_1) \Longrightarrow \exists N_1(N \Downarrow \mathbf{inr}(N_1) \;\&\; (M_1\mathcal{R}_{\tau_2} N_1)))$	(sim sum inr)
$M\langle\mathcal{R}\rangle_{\mu T.\tau} N$	$\stackrel{def}{\iff}$ $(M \Downarrow \mathbf{abs}_{\mu T.\tau}(C) \Longrightarrow \exists C_1(N \Downarrow \mathbf{abs}(C_1) \;\&\; C\mathcal{R}_{\tau[\mu T.\tau/T]} C_1))$	(sim rep)

Table 7.1: definition of $\langle\ \rangle$

$M[\mathcal{R}]_{\tau\to\tau_1} N$	$\stackrel{def}{\iff}$ $(M \Downarrow \lambda x.M' \Longrightarrow \exists\lambda x.N'(N \Downarrow \lambda x.N' \;\&\; \forall A \in Exp_\tau(M[A/x]\mathcal{R}_{\tau_1} N[A/x]))) \;\&$ $(N \Downarrow \lambda x.N' \Longrightarrow \exists\lambda x.M'(M \Downarrow \lambda x.M' \;\&\; \forall A \in Exp_\tau(M[A/x]\mathcal{R}_{\tau_1} N[A/x])))$	(bis fun)
$M[\mathcal{R}]_{\tau_1\times\tau_2} N$	$\stackrel{def}{\iff}$ $(M \Downarrow \langle M_1, M_2\rangle \Longrightarrow \exists\langle N_1, N_2\rangle(N \Downarrow \langle N_1, N_2\rangle \;\&\; (M_1\mathcal{R}_{\tau_1} N_1 \;\&\; M_2\mathcal{R}_{\tau_2} N_2))) \;\&$ $(N \Downarrow \langle N_1, N_2\rangle \Longrightarrow \exists\langle M_1, M_2\rangle(M \Downarrow \langle M_1, M_2\rangle \;\&\; (M_1\mathcal{R}_{\tau_1} N_1 \;\&\; M_2\mathcal{R}_{\tau_2} N_2)))$	(bis pair)
$M[\mathcal{R}]_{\tau_1+\tau_2} N$	$\stackrel{def}{\iff}$ $(M \Downarrow \mathbf{inl}(M_1) \Longrightarrow \exists N_1(N \Downarrow \mathbf{inl}(N_1) \;\&\; (M_1\mathcal{R}_{\tau_1} N_1))) \;\&$ $(N \Downarrow \mathbf{inl}(N_1) \Longrightarrow \exists M_1(M \Downarrow \mathbf{inl}(M_1) \;\&\; (M_1\mathcal{R}_{\tau_1} N_1)))$	(bis sum inl)
$M[\mathcal{R}]_{\tau_1+\tau_2} N$	$\stackrel{def}{\iff}$ $(M \Downarrow \mathbf{inr}(M_1) \Longrightarrow \exists N_1(N \Downarrow \mathbf{inr}(N_1) \;\&\; (M_1\mathcal{R}_{\tau_2} N_1))) \;\&$ $(N \Downarrow \mathbf{inr}(N_1) \Longrightarrow \exists M_1(N \Downarrow \mathbf{inr}(M_1) \;\&\; (M_1\mathcal{R}_{\tau_2} N_1)))$	(bis sum inr)
$M[\mathcal{R}]_{\mu T.\tau} N$	$\stackrel{def}{\iff}$ $(M \Downarrow \mathbf{abs}_{\mu T.\tau}(C) \Longrightarrow \exists C_1(N \Downarrow \mathbf{abs}(C_1) \;\&\; C\mathcal{R}_{\tau[\mu T.\tau/T]} C_1)) \;\&$ $(N \Downarrow \mathbf{abs}_{\mu T.\tau}(C_1) \Longrightarrow \exists C(M \Downarrow \mathbf{abs}(C) \;\&\; C\mathcal{R}_{\tau[\mu T.\tau/T]} C_1))$	(bis rep)

Table 7.2: definition of $[\ ]$

by lemma 2.5.7 we have that $\nu X.[X]$ is an equivalence relation on *Exp*.
Since we have coinductively defined similarity and bisimilarity, associated with them we have two different coinductive proof principles.

Definition 7.3.4 (Similarity Coinduction).
For each $M, N \in Exp_\tau$ *and* $\mathcal{S} \subseteq \mathcal{P}(Exp \times Exp)$

$$\mathcal{S} \subseteq \langle \mathcal{S} \rangle \ \& \ M \ \mathcal{S} \ N \Longrightarrow M \lesssim_\tau^{bis} N$$

Definition 7.3.5 (Bisimilarity Coinduction).
For each $M, N \in Exp_\tau$ *and* $\mathcal{B} \subseteq \mathcal{P}(Exp \times Exp)$

$$\mathcal{B} \subseteq [\mathcal{B}] \ \& \ M \ \mathcal{B} \ N \Longrightarrow M \cong_\tau^{bis} N$$

We have claimed that bisimilarity is an equivalence relations on *Exp* we now want to prove the following theorem which gives us an alternative coinductive characterization of contextual equivalence.

Theorem 7.3.6 (Bisimilarity equals contextual equivalence).

$$M \cong_\tau^{bis} N \iff M \cong_\tau N$$

Proof.

($\Longleftarrow$) We only need to show that $\cong^{Exp} \subseteq [\cong]$. Assume $M \cong_\tau N$; by lemma 4.2.19 we have either that $M \Uparrow$ or $M \Downarrow$ and if $M \Uparrow$ by property 6.4.17 we have $N \Uparrow$ and hence $M \cong_\tau^{bis} N$. Now consider the case $M \Downarrow$ and consider the different cases for τ.

- $M \cong_{\tau_1 \to \tau_2} N$ Suppose $M \Downarrow \lambda x.M'$ then $N \Downarrow$ and since it is well typed we have that there exists $\lambda y.N'$ such that $N \Downarrow \lambda y.N'$. By congruence property 6.4.7 we have $(\lambda x.M)A \cong_{\tau_2} (\lambda y.N)A$ and by beta reduction property 6.4.18 $M[A/x] \cong_{\tau_2} N[A/x]$.
- $M \cong_{\tau_1 \times \tau_2} N$ Suppose $M \Downarrow \langle M_1, M_2 \rangle$ then also $N \Downarrow$ and since N is well typed we have that there exists $\langle N_1, N_2 \rangle$ such that $N \Downarrow \langle N_1, N_2 \rangle$. By congruence property 6.4.7 we have $\mathbf{fst}(\langle M_1, M_2 \rangle) \cong_{\tau_2} \mathbf{fst}(\langle N_1, N_2 \rangle)$ and $\mathbf{snd}(\langle M_1, M_2 \rangle) \cong_{\tau_2} \mathbf{snd}(\langle N_1, N_2 \rangle)$. By beta reduction property 6.4.19 $M_1 \cong_{\tau_1} N_1$ and $M_2 \cong_{\tau_2} N_2$ follow.
- $M \cong_{\tau_1 + \tau_2} N$ We have two different cases.
  - Suppose $M \Downarrow \mathbf{inl}(M_1)$ then also $N \Downarrow$ and since N is well typed we have that there exists N_1 such that $N \Downarrow \mathbf{inl}(N_1)$ otherwise $N \Downarrow \mathbf{inr}(N_1)$ contraddicts the hypothesis that $M \cong_\tau N$. By congruence property 6.4.7 we have $\mathbf{case}\ \mathbf{inl}(M_1)\ \mathbf{of}\ \{\mathbf{inl}_{\tau_1,\tau_2}(x_1).x_1 \mid \mathbf{inr}_{\tau_1,\tau_2}(x_2).x_2\} \cong_{\tau_1} \mathbf{case}\ \mathbf{inl}(N_1)\ \mathbf{of}\ \{\mathbf{inl}_{\tau_1,\tau_2}(x_1).x_1 \mid \mathbf{inr}_{\tau_1,\tau_2}(x_2).x_2\}$ and by beta reduction property 6.4.20 $M_1 \cong_{\tau_1} N_1$.
  - Suppose $M \Downarrow \mathbf{inr}(M_1)$ then also $N \Downarrow$ and since N is well typed we have that there exists N_1 such that $N \Downarrow \mathbf{inr}(N_1)$ otherwise $N \Downarrow \mathbf{inl}(N_1)$ contraddicts the hypothesis that $M \cong_\tau N$. By congruence property 6.4.7 we have $\mathbf{case}\ \mathbf{inr}(M_1)\ \mathbf{of}\ \{\mathbf{inl}_{\tau_1,\tau_2}(x_1).x_1 \mid \mathbf{inr}_{\tau_1,\tau_2}(x_2).x_2\} \cong_{\tau_2} \mathbf{case}\ \mathbf{inr}(N_1)\ \mathbf{of}\ \{\mathbf{inl}_{\tau_1,\tau_2}(x_1).x_1 \mid \mathbf{inr}_{\tau_1,\tau_2}(x_2).x_2\}$ and by beta reduction property 6.4.20 $M_1 \cong_{\tau_2} N_1$.

$M \cong_{\mu T.\tau_1} N$ Suppose $M \Downarrow \mathbf{abs}_{\mu T.\tau_1}(C)$ then also $N \Downarrow$ and since N is well typed we have that there exists C_1 such that $N \Downarrow \mathbf{abs}_{\mu T.\tau_1}(C_1)$. By congruence property 6.4.7 we have $\mathbf{rep}(\mathbf{abs}_{\mu T.\tau_1}(C)) \cong_{\tau_1[\mu T.\tau_1/x]} \mathbf{rep}(\mathbf{abs}_{\mu T.\tau_1}(C_1))$ and by beta reduction property 6.4.21 $C \cong_{\tau_1[\mu T.\tau_1/T]} C_1$.

($\Longrightarrow$) It follows immediately by definition of bisimilarity and experimental coinduction 7.2.7.

□

By the above result we can reformulate our coinduction proof principle.

Definition 7.3.7 (Bisimilarity Coinduction).
For each $M, N \in Exp_\tau$ and $\mathcal{B} \subseteq \mathcal{P}(Exp \times Exp)$

$$\mathcal{B} \subseteq [\mathcal{B}] \;\&\; M \;\mathcal{B}\; N \Longrightarrow M \cong_\tau N$$

Finally we can extends our results to open terms. We can extends as usual the preorder on closed terms to open terms by using expression substitution.

Definition 7.3.8 (Open Similarity). *Let $M, N \in Exp_\tau(\Gamma)$ for some $\Gamma \in TEnv, \tau \in Type$. We say that M is **similar** to N, written $\Gamma \vdash M \lesssim_\tau^{Bis} N$ if and only if for each expression substitution γ for Γ we have:*

$$\gamma(M) \lesssim_\tau^{Bis} \gamma(N)$$

Analougsly we can define define open bisimilarity equivalence.

Definition 7.3.9 (Open Bisimilarity). *Let $M, N \in Exp_\tau(\Gamma)$ for some $\Gamma \in TEnv, \tau \in Type$. We say that M and N are **bisimilar**, written $\Gamma \vdash M \cong_\tau^{Bis} N$ if and only if for each expression substitution γ for Γ we have:*

$$\gamma(M) \cong_\tau^{Bis} \gamma(N)$$

It is easy to verify the following:

Lemma 7.3.10 (Open Bisimilarity equals Contextual Equivalence).

$$\Gamma \vdash M \cong_\tau^{Bis} N \iff \Gamma \vdash M \cong_\tau N$$

Proof. Clearly by definition $\Gamma \vdash M \cong_\tau N$ implies $\Gamma \vdash M \cong_\tau^{Bis} N$. For the reverse implication consider $\Gamma \vdash M \cong_\tau^{Bis} N$ this means that for each expression substitution γ for Γ we have $\gamma(M) \cong_\tau^{Bis} \gamma(N)$ but if $\Gamma = \{x_1 : \tau_1, \ldots, x_n : \tau_n\}$ this means that for each $M_1 \in Exp_{\tau_1}, \ldots, M_n \in Exp_{\tau_n}$ we have:

$$N[M_1/x_1, \cdots, M_n/x_n] \cong_\tau^{Bis} N'[M_1/x_1, \cdots, M_n/x_n]$$

but by lemma 7.3.6 we have

$$N[M_1/x_1, \cdots, M_n/x_n] \cong_\tau N'[M_1/x_1, \cdots, M_n/x_n]$$

and by extensionality property

$$x_1 : \tau_1, \ldots, x_n : \tau_n \vdash N \cong_\tau N'$$

□

We have proved lemmas 7.3.6 and 7.3.10 by using results proved in chapter 6. Howe in [Howe, 1989] invented a powerful method to prove the same results. Moreover Howe in [Howe, 1996] generalized the method to languages whose semantics follows a *certain natural form of structured operational semantics*. We prefer to use a more concrete approach but we stress that our results includes all the intermediate results of Howe method.

Chapter 8

FPC Constructions

> Our language may be seen as an ancient city: a maze of little streets and squares, of old and new houses, and of houses with additions from various periods; and this surrounded by a moltitude of new boroughs with straight regular streets and uniform houses.
>
> Ludwig Wittgenstein. "Philosophical Investigations"

We now give some examples of how FPC can be effectively used as a functional language. Gordon in [Gordon, 1995] use a version of FPC with two ground types: the *Boolean* type and the *Integer* type. Usually, functional programming languages have Boolean and Integer as built-in types. If a language does not provide them as built-in types it must allow to define them as derived types. As one expects in FPC it is possible to derive Boolean and Integer only using constructors defined in chapter 3 and 4. We now show this. Furthermore in the sequel we will show how to derive types which allow more interesting structure. We first define a *divergent* type:

$$\mathbf{0} \stackrel{\text{def}}{=} \mu t.t$$

A simple example of object of type $\mathbf{0}$ which will be useful in the sequel is $\Omega^{\mathbf{0}}$. Naturally $\Omega^{\mathbf{0}}$ is not the only term of type $\mathbf{0}$, for example $\mathbf{rec}\ x.x$ where $x : \mathbf{0}$ is a different term but it is easy to verify that they have the same behaviour. Indeed for each $M, N \in Exp_{\mathbf{0}}$ we have $M \cong_{\mathbf{0}} N$. Hence we can consider the equivalence class $Exp_{\mathbf{0}}/\cong_{\mathbf{0}}$ as comprising a unique element $\Omega^{\mathbf{0}}$.

8.1 Booleans

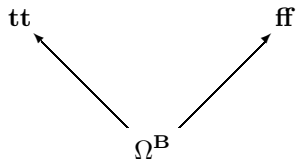
We now want to define the simple *Boolean* type. We can easily do it as:

$$\mathbf{B} \stackrel{\text{def}}{=} \mathbf{0} + \mathbf{0}$$

the above definition depends on our choice of sum constructor, in fact we have a separated sum. We can use the above remark to define the boolean term *true* and *false* in a natural way as:

$$\mathbf{tt} \stackrel{\text{def}}{=} \mathbf{inl}_{\mathbf{0},\mathbf{0}}(\Omega^{\mathbf{0}})$$
$$\mathbf{ff} \stackrel{\text{def}}{=} \mathbf{inr}_{\mathbf{0},\mathbf{0}}(\Omega^{\mathbf{0}})$$

Clearly **tt** and **ff** are both canonical terms. It is easy to verify that for each $M \in Exp_{\mathbf{B}}$ we have three distincts possibilities: either $M \cong_{\mathbf{B}} \mathbf{tt}$ or $M \cong_{\mathbf{B}} \mathbf{f}$ or $M \cong_{\mathbf{B}} \Omega^{\mathbf{B}}$. This means that the quotient $Exp_{\mathbf{B}}/\cong_{\mathbf{B}}$ has three elements. We can graphically represents $Exp_{\mathbf{B}}/\cong_{\mathbf{B}}$ as:



where at each node we have an equivalence class and each line through nodes represents the relation of contextual approximation.
Now we can define some functions with the help of boolean type. A useful function is for example the conditional function. Given $x : \mathbf{B}$, $y : \tau$, $z : \tau$, we define a family of conditional functions, $Cond_\tau : (\mathbf{B} \to (\tau \to (\tau \to \tau))$ one for each type τ.

$$Cond_\tau = \lambda x.\lambda y.\lambda z.\mathbf{case}\ x\ \mathbf{of}\ \{\mathbf{inl_{0,0}}(x_1).y \mid \mathbf{inr_{0,0}}(x_2).z\}$$

Now we can use the conditional function to define other functions. For example we can define $Not : \mathbf{B} \to \mathbf{B}$ as:

$$Not = \lambda x.(((Cond\ x)\mathbf{ff})\mathbf{tt})$$

as one expects $Not\ \mathbf{tt} \Downarrow \mathbf{ff}$ and $Not\ \mathbf{ff} \Downarrow \mathbf{tt}$. We show an example:

$$\begin{array}{l} Not\ \mathbf{tt} \equiv \lambda x.(((Cond\ x)\mathbf{ff})\mathbf{tt}) \longrightarrow (((Cond\ \mathbf{tt})\mathbf{ff})\mathbf{tt}) \equiv \\ (((\lambda x.\lambda y.\lambda z.\mathbf{case}\ x\ \mathbf{of}\ \{\mathbf{inl_{0,0}}(x_1).y \mid \mathbf{inr_{0,0}}(x_2).z\}\ \mathbf{tt})\mathbf{ff})\mathbf{tt}) \longrightarrow \\ ((\lambda y.\lambda z.\mathbf{case}\ \mathbf{tt}\ \mathbf{of}\ \{\mathbf{inl_{0,0}}(x_1).y \mid \mathbf{inr_{0,0}}(x_2).z\}\ \mathbf{ff})\mathbf{tt}) \longrightarrow \\ (\lambda z.\mathbf{case}\ \mathbf{tt}\ \mathbf{of}\ \{\mathbf{inl_{0,0}}(x_1).\mathbf{ff} \mid \mathbf{inr_{0,0}}(x_2).z\}\ \mathbf{tt}) \longrightarrow \\ \mathbf{case}\ \mathbf{tt}\ \mathbf{of}\ \{\mathbf{inl_{0,0}}(x_1).\mathbf{ff} \mid \mathbf{inr_{0,0}}(x_2).\mathbf{tt}\} \longrightarrow \mathbf{ff} \end{array}$$

It is easy to extend the above method to other boolean functions. By our definition of *Cond* we can define the usual **if** constructor for closed terms $B \in \mathbf{B}$ and $M_1, M_2 \in Exp_\tau$ as follows.

$$\mathbf{if}\ B\ \mathbf{then}\ M_1\ \mathbf{else}\ M_2 = Cond\ BM_1M_2$$

8.2 Natural numbers

We can now define a type **N** which represents the *natural numbers*, as follows.

$$\mathbf{N} \stackrel{\text{def}}{=} \mu T.(\mathbf{0} + T)$$

It is well known that, in order to define natural numbers, we need a constant function which represents the number 0 and a unary function which represents the successor function. We can define in FPC the constant 0 as:

$$0 \stackrel{\text{def}}{=} \mathbf{abs_N}(\mathbf{inl_{0,N}}(\Omega^{\mathbf{0}}))$$

Analogously we can define the successor function $Succ : \mathbf{N} \to \mathbf{N}$ as follows.

$$Succ \stackrel{\text{def}}{=} \lambda x.\mathbf{abs_N}(\mathbf{inr_{0,N}}(x))$$

We can define the $n-$th iterate of *Succ* applied to 0 by the following schema, for $n \geq 0$:

$$Succ^n 0 \stackrel{\text{def}}{=} \underbrace{Succ\ Succ \cdots Succ}_{n \text{ times}}\ 0$$

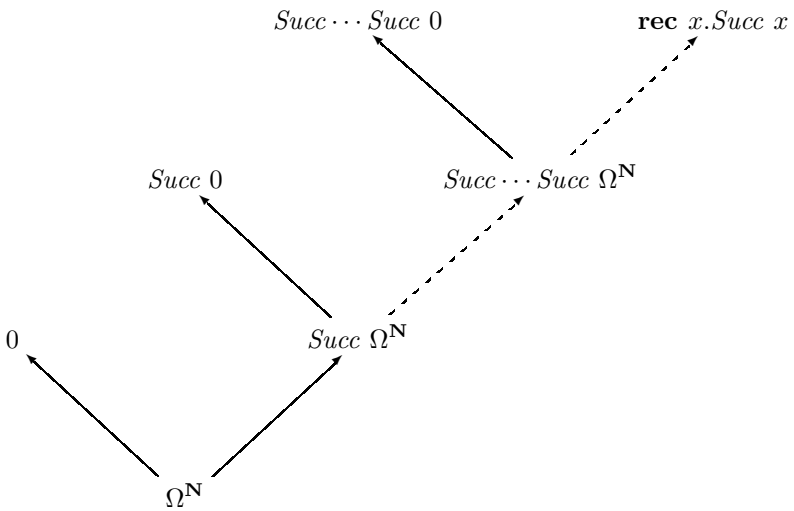
Hence we can define *natural numbers* as the canonical form obtained by applying the successor function to 0.

$$\begin{array}{ll}
0 & \stackrel{\text{def}}{=} \mathbf{abs_N}(\mathbf{inl_{0,N}}(\Omega^{\mathbf{0}})) \\
1 & \stackrel{\text{def}}{=} \mathbf{abs_N}(\mathbf{inr_{0,N}}(0)) = \mathbf{abs_N}(\mathbf{inr_{0,N}}(\mathbf{abs_N}(\mathbf{inl_{0,N}}(\Omega^{\mathbf{0}})))) \\
2 & \stackrel{\text{def}}{=} \mathbf{abs_N}(\mathbf{inr_{0,N}}(1)) = \mathbf{abs_N}(\mathbf{inr_{0,N}}(\mathbf{abs_N}(\mathbf{inr_{0,N}}(0)))) \\
\vdots & \\
n & \stackrel{\text{def}}{=} \mathbf{abs_N}(\mathbf{inr_{0,N}}(n-1)) = \underbrace{\mathbf{abs_N}(\mathbf{inr_{0,N}}(\cdots(\mathbf{abs_N}(\mathbf{inr_{0,N}}}(0)))\cdots))}_{n \text{ times}}
\end{array}$$

In particular we have that:

$$Succ\ n \cong_{\mathbf{N}} \mathbf{abs_N}(\mathbf{inr_{0,N}}(n)) \cong_{\mathbf{N}} Succ^{n+1} 0$$

Since we have $Succ\ n \longrightarrow n+1$ then $Succ\ n \cong_{\mathbf{N}} n+1$ hence we sometimes write $n+1$ for $Succ\ n$. It must be clear that when we write $+1$ we mean a metaoperation instead of one belonging to our language FPC.
To define natural numbers we have applied *Succ* to 0, unfortunately *Succ* can be applied also to $\Omega^{\mathbf{N}}$ obtaining canonical forms which have different behaviour with respect to *natural numbers*. Following Winskel in [Winskel, 1993] we call these numbers *partial numbers*. This means that the equivalence class $Exp_{\mathbf{N}}/\cong_{\mathbf{N}}$ is too large, in the sense that it contains elements which have different behaviour with respects to *natural numbers* and $\Omega^{\mathbf{N}}$. We can graphically represent $Exp_{\mathbf{N}}/\cong_{\mathbf{N}}$ as:



In the sequel it will be useful to consider only a subset of $Exp_{\mathbf{N}}$. Hence we can define inductively the set of *natural complete numbers* $Nat \subseteq Exp_{\mathbf{N}}$ as:

$$Nat \stackrel{\text{def}}{=} \{M \in Exp_{\mathbf{N}} | M \cong_{\mathbf{N}} 0 \text{ or } \exists N \in Nat(M \cong_{\mathbf{N}} Succ\ N)\}$$

When is clear what do we mean we identify natural numbers with natural complete numbers. We can represents graphically $Nat/\cong_{\mathbf{N}}$ as:

$$0 \qquad 1 \qquad 2 \qquad \cdots \qquad n \qquad \cdots$$

We now want to define some useful functions on $\mathbf{N}$. As a simple example we can define a test for zero $isZero : \mathbf{N} \to \mathbf{B}$ as:

$$isZero \stackrel{\text{def}}{=} \lambda x.\mathbf{case}\ \mathbf{rep}(x)\ \mathbf{of}\ \{\mathbf{inl}_{\mathbf{0},\mathbf{N}}(x_1).\mathbf{tt} \mid \mathbf{inr}_{\mathbf{0},\mathbf{N}}(x_2).\mathbf{ff}\}$$

As one expects the computation of the function *isZero* appllied to 0 converges to **tt**

$$\begin{array}{l} isZero\ 0 \equiv \lambda x.\mathbf{case}\ \mathbf{rep}(x)\ \mathbf{of}\ \{\mathbf{inl}_{\mathbf{0},\mathbf{N}}(x_1).\mathbf{tt} \mid \mathbf{inr}_{\mathbf{0},\mathbf{N}}(x_2).\mathbf{ff}\}\ 0 \longrightarrow \\ \mathbf{case}\ \mathbf{rep}(0)\ \mathbf{of}\ \{\mathbf{inl}_{\mathbf{0},\mathbf{N}}(x_1).\mathbf{tt} \mid \mathbf{inr}_{\mathbf{0},\mathbf{N}}(x_2).\mathbf{ff}\} \equiv \\ \mathbf{case}\ \mathbf{rep}(\mathbf{abs}_{\mathbf{N}}(\mathbf{inl}_{\mathbf{0},\mathbf{N}}(\Omega^{\mathbf{0}})))\ \mathbf{of}\ \{\mathbf{inl}_{\mathbf{0},\mathbf{N}}(x_1).\mathbf{tt} \mid \mathbf{inr}_{\mathbf{0},\mathbf{N}}(x_2).\mathbf{ff}\} \longrightarrow \\ \mathbf{case}\ \mathbf{inl}_{\mathbf{0},\mathbf{N}}(\Omega^{\mathbf{0}})\ \mathbf{of}\ \{\mathbf{inl}_{\mathbf{0},\mathbf{N}}(x_1).\mathbf{tt} \mid \mathbf{inr}_{\mathbf{0},\mathbf{N}}(x_2).\mathbf{ff}\} \longrightarrow \mathbf{tt} \end{array}$$

and the computation of the function *isZero* apllied to any $n \in \mathbf{N}$ converge to **ff**

$$\begin{array}{l} isZero\ n \equiv \lambda x.\mathbf{case}\ \mathbf{rep}(x)\ \mathbf{of}\ \{\mathbf{inl}_{\mathbf{0},\mathbf{N}}(x_1).\mathbf{tt} \mid \mathbf{inr}_{\mathbf{0},\mathbf{N}}(x_2).\mathbf{ff}\}\ n \longrightarrow \\ \mathbf{case}\ \mathbf{rep}(n)\ \mathbf{of}\ \{\mathbf{inl}_{\mathbf{0},\mathbf{N}}(x_1).\mathbf{tt} \mid \mathbf{inr}_{\mathbf{0},\mathbf{N}}(x_2).\mathbf{ff}\} \equiv \\ \mathbf{case}\ \mathbf{rep}(\mathbf{abs}_{\mathbf{N}}(\mathbf{inr}_{\mathbf{0},\mathbf{N}}(n-1)))\ \mathbf{of}\ \{\mathbf{inl}_{\mathbf{0},\mathbf{N}}(x_1).\mathbf{tt} \mid \mathbf{inr}_{\mathbf{0},\mathbf{N}}(x_2).\mathbf{ff}\} \longrightarrow \\ \mathbf{case}\ \mathbf{inr}_{\mathbf{0},\mathbf{N}}(n-1)\ \mathbf{of}\ \{\mathbf{inl}_{\mathbf{0},\mathbf{N}}(x_1).\mathbf{tt} \mid \mathbf{inr}_{\mathbf{0},\mathbf{N}}(x_2).\mathbf{ff}\} \longrightarrow \mathbf{ff} \end{array}$$

As one expects $isZero\ \Omega^{\mathbf{N}} \Uparrow$.
In the sequel will be useful a *predecessor* function $Pred : \mathbf{N} \to \mathbf{N}$ defined as

$$Pred \stackrel{\text{def}}{=} \lambda x.\mathbf{case}\ \mathbf{rep}(x)\ \mathbf{of}\ \{\mathbf{inl}_{\mathbf{0},\mathbf{N}}(x_1).\Omega^{\mathbf{N}} \mid \mathbf{inr}_{\mathbf{0},\mathbf{N}}(x_2).x_2\}$$

Since $Pred\ n \longrightarrow (n-1)$ and $Pred\ 0 \longrightarrow \Omega^{\mathbf{N}}$ as one expects we have $Pred\ n \cong_{\mathbf{N}} n-1$ and $Pred\ 0 \cong_{\mathbf{N}} \Omega^{\mathbf{N}}$ hence we sometimes write $n-1$ instead of $Pred\ n$ where $n \ncong_{\mathbf{N}} 0$. We prove the following:

Proposition 8.2.1. *For each $N \in Exp_{\mathbf{N}}$ we have:*

$$Pred(Succ(N)) \cong_{\mathbf{N}} N$$

Proof. $Pred(Succ(N)) \equiv$
$\lambda x.\mathbf{case}\ \mathbf{rep}(x)\ \mathbf{of}\ \{\mathbf{inl}_{\mathbf{0},\mathbf{N}}(x_1).\Omega^{\mathbf{N}} \mid \mathbf{inr}_{\mathbf{0},\mathbf{N}}(x_2).x_2\}Succ(N) \longrightarrow$
$\mathbf{case}\ \mathbf{rep}(Succ(N))\ \mathbf{of}\ \{\mathbf{inl}_{\mathbf{0},\mathbf{N}}(x_1).\Omega^{\mathbf{N}} \mid \mathbf{inr}_{\mathbf{0},\mathbf{N}}(x_2).x_2\} \equiv$
$\mathbf{case}\ \mathbf{rep}(\lambda x.\mathbf{abs}_{\mathbf{N}}(\mathbf{inr}_{\mathbf{0},\mathbf{N}}(x))N)\ \mathbf{of}\ \{\mathbf{inl}_{\mathbf{0},\mathbf{N}}(x_1).\Omega^{\mathbf{N}} \mid \mathbf{inr}_{\mathbf{0},\mathbf{N}}(x_2).x_2\} \longrightarrow$
$\mathbf{case}\ \mathbf{rep}(\mathbf{abs}_{\mathbf{N}}(\mathbf{inr}_{\mathbf{0},\mathbf{N}}(N)))\ \mathbf{of}\ \{\mathbf{inl}_{\mathbf{0},\mathbf{N}}(x_1).\Omega^{\mathbf{N}} \mid \mathbf{inr}_{\mathbf{0},\mathbf{N}}(x_2).x_2\} \longrightarrow$
$\mathbf{case}\ \mathbf{inr}_{\mathbf{0},\mathbf{N}}(N)\ \mathbf{of}\ \{\mathbf{inl}_{\mathbf{0},\mathbf{N}}(x_1).\Omega^{\mathbf{N}} \mid \mathbf{inr}_{\mathbf{0},\mathbf{N}}(x_2).x_2\} \longrightarrow N$
By lemma 6.4.12 the conclusion follows. □

8.2.1 Coinduction at N

In chapter 6 we have defined $\cong^{bis}$ as the greatest fixed point of $[\ ]$, hence we have that $\cong^{bis} = [\cong^{bis}]$. In particular lemma 7.3.6 shows that $\cong = [\cong]$ and hence that $\cong$ is the largest bisimulation. Furthermore 7.3.7 shows that associated with $\cong^{bis}$ we have a coinductive proof principle at each type. We now want to reformulate the coinductive proof principle at $\mathbf{N}$. We have defined $\mathbf{N}$ as

$\mu T.(\mathbf{0}+T)$. Since $\cong$ is a bisimulation and by definition of bisimulation we have that:

$$N \cong_{\mathbf{N}} N_1 \iff$$
$$((N \Downarrow \mathbf{abs_N}(C) \iff N_1 \Downarrow \mathbf{abs_N}(C_1)) \;\&\; C \cong_{\mathbf{0+N}} C_1)$$

And again since $\cong$ is a bisimulation and by definition of bisimulation we have that:

$$C \cong_{\mathbf{0+N}} C_1 \iff$$
$$((C \Downarrow \mathbf{inl_{0,N}}(M) \iff C_1 \Downarrow \mathbf{inl_{0,N}}(M_1) \;\&\; M \cong_{\mathbf{0}} M_1) \text{ or}$$
$$(C \Downarrow \mathbf{inr_{0,N}}(M) \iff C_1 \Downarrow \mathbf{inr_{0,N}}(M_1) \;\&\; M \cong_{\mathbf{N}} M_1))$$

We know that for each $M, M_1 \in Exp_{\mathbf{0}}$ we have $M \cong_{\mathbf{0}} M_1 \cong_{\mathbf{0}} \Omega^{\mathbf{0}}$ hence we can rewrite the above as

$$C \cong_{\mathbf{0+N}} C_1 \iff$$
$$((C \Downarrow \mathbf{inl_{0,N}}(M) \iff C_1 \Downarrow \mathbf{inl_{0,N}}(M_1)) \text{ or}$$
$$(C \Downarrow \mathbf{inr_{0,N}}(M) \iff C_1 \Downarrow \mathbf{inr_{0,N}}(M_1) \;\&\; M \cong_{\mathbf{N}} M_1))$$

Furthermore since evaluation is deterministic and for each $C \in Can$ we have $C \Downarrow C$ we have:

$$N \cong_{\mathbf{N}} N_1 \iff$$
$$((N \Downarrow \mathbf{abs_N}(\mathbf{inl_{0,N}}(M)) \iff N_1 \Downarrow \mathbf{abs_N}(\mathbf{inl_{0,N}}(M_1))) \text{ or}$$
$$((N \Downarrow \mathbf{abs_N}(\mathbf{inr_{0,N}}(M)) \iff N_1 \Downarrow \mathbf{abs_N}(\mathbf{inr_{0,N}}(M_1))) \;\&\; M \cong_{\mathbf{N}} M_1))$$

Definition 8.2.2 (Bisimulation at N). *$\mathcal{B} \subseteq Exp_{\mathbf{N}} \times Exp_{\mathbf{N}}$ is a* **N-*bisimulation*** *if whenever $N \;\mathcal{B}\; N_1$ we have:*

(i) $(N \Downarrow \boldsymbol{abs_N}(\boldsymbol{inl_{0,N}}(M)) \iff N_1 \Downarrow \boldsymbol{abs_N}(\boldsymbol{inl_{0,N}}(M_1)))$

(ii) $((N \Downarrow \boldsymbol{abs_N}(\boldsymbol{inr_{0,N}}(M)) \iff N_1 \Downarrow \boldsymbol{abs_N}(\boldsymbol{inr_{0,N}}(M_1))) \;\&\; M \;\mathcal{B}\; M_1)$

Finally we have the following proof principle.

Definition 8.2.3 (Coinduction at N). *For each $N, N_1 \in Exp_{\mathbf{N}}$ if there exists a* **N**-*bisimulation $\mathcal{B}$ such that $N \;\mathcal{B}\; N_1$ then $N \cong_{\mathbf{N}} N_1$.*

8.3 List type

As a last example we define a type which allows interesting possibly infinite structures. For each $\tau \in Type$ we can define $\mathbf{L}_\tau$ as follows.

$$\mathbf{L}_\tau \stackrel{\text{def}}{=} \mu T.(\mathbf{0} + (\tau \times T))$$

$\mathbf{L}_\tau$ represents the type of possibly infinite *lists* of object of type τ. For example we can define the type of the lists of natural numbers as:

$$\mathbf{L_N} \stackrel{\text{def}}{=} \mu T.(\mathbf{0} + (\mathbf{N} \times T))$$

Intuitively we can build lists through a constant function of type list and a *concatenation* operation. First we define the *empty* list as follows.

$$\varepsilon \stackrel{\text{def}}{=} \mathbf{abs}_{\mathbf{L}_\tau}(\mathbf{inl}_{\mathbf{0},\tau\times\mathbf{L}_\tau}(\Omega^{\mathbf{0}}))$$

Since for each $M, N \in Exp_{\mathbf{0}}$ we have $M \cong_{\mathbf{0}} N$, it is easy to verify that:

$$\forall M \in Exp_{\mathbf{0}}(\mathbf{abs}_{\mathbf{L}_\tau}(\mathbf{inl}_{\mathbf{0},\tau\times\mathbf{L}_\tau}(M)) \cong_{\mathbf{L}_\tau} \varepsilon$$

The canonical terms of the right part of the sum have the form:

$$\mathbf{abs}_{\mathbf{L}_\tau}(\mathbf{inr}_{\mathbf{0},\tau\times\mathbf{L}_\tau}(P))$$

where $P \in Exp_{\tau\times\mathbf{L}_\tau}$. As natural more interesting objects of type list have the form:

$$\mathbf{abs}_{\mathbf{L}_\tau}(\mathbf{inr}_{\mathbf{0},\tau\times\mathbf{L}_\tau}(\langle M, N\rangle))$$

where $M : \tau$ can be considered the *head* and $N : \mathbf{L}_\tau$ the *tail* of the lists. We sometimes write $M :: N$ instead of $\mathbf{abs}_{\mathbf{L}_\tau}(\mathbf{inr}_{\mathbf{0},\tau\times\mathbf{L}_\tau}(\langle M, N\rangle))$. By this observation we can define the function $Cons : \tau \to \mathbf{L}_\tau \to \mathbf{L}_\tau$ as:

$$Cons \stackrel{\text{def}}{=} \lambda x.\lambda y.\mathbf{abs}_{\mathbf{L}_\tau}(\mathbf{inr}_{\mathbf{0},\tau\times\mathbf{L}_\tau}(\langle x, y\rangle))$$

where $x : \tau$ and $y : \mathbf{L}_\tau$. As obvious we have:

$$Cons\ H\ T \cong_{\mathbf{L}_\tau} \mathbf{abs}_{\mathbf{L}_\tau}(\mathbf{inr}_{\mathbf{0},\tau\times\mathbf{L}_\tau}(\langle H, T\rangle)) \equiv H :: T$$

We have claimed that $\mathbf{L}_\tau$ represents all (possibly infinite) lists of objects of type τ. An example of infinite lists of type $\mathbf{L}_\tau$ is:

$$\mathbf{rec}\ l.Cons\ A\ l$$

which is the infinite list of object $A : \tau$. Unfortunately $Exp_{\mathbf{L}_\tau}$ is sometimes too large because contains objects which, analogously to partial number, can be considered *partial lists*, e.g. $\mathbf{abs}_{\mathbf{L}_\tau}(\mathbf{inr}_{\mathbf{0},\tau\times\mathbf{L}_\tau}(\langle H, \Omega^{\mathbf{L}_\tau}\rangle))$. Hence we want to isolate a set of *complete lists*. We define an operator $\phi_{\mathbf{L}} : \mathcal{P}(Exp_{\mathbf{L}_\tau}) \to \mathcal{P}(Exp_{\mathbf{L}_\tau})$ as:

$$\phi_{\mathbf{L}}(X) \stackrel{\text{def}}{=} \{L \mid L \cong_{\mathbf{L}_\tau} \varepsilon\} \cup \{L \mid L \cong_{\mathbf{L}_\tau} Cons\ N\ L'\ \&N \in Exp_\tau\ \&\ L' \in X\}$$

Since $\phi_{\mathbf{L}}$ is monotone it possesses a greatest fixed point, $\nu X.\phi_{\mathbf{L}}(X)$ which is the greatest $\phi_{\mathbf{L}}$-dense set. Hence we can define $List_\tau \subseteq \mathbf{L}$ as:

$$List_\tau \stackrel{\text{def}}{=} \nu X.\phi_{\mathbf{L}}(X)$$

In particular in the sequel will be useful to consider an operator $\phi_{Nat} : \mathcal{P}(Exp_{\mathbf{L_N}}) \to \mathcal{P}(Exp_{\mathbf{L_N}})$ defined as:

$$\phi_{Nat}(X) \stackrel{\text{def}}{=} \{L \mid L \cong_{\mathbf{L_N}} \varepsilon\} \cup \{L \mid L \cong_{\mathbf{L_N}} Cons\ N\ L'\ \&N \in Exp_{\mathbf{N}}\ \&\ L' \in X\}$$

And since ϕ_{Nat} is monotone it possesses a greatest fixed point, $\nu X.\phi_{Nat}(X)$ which is the greatest ϕ_{Nat}-dense set. Hence we can define $List_{Nat} \subseteq \mathbf{L}$ as:

$$List_{Nat} \stackrel{\text{def}}{=} \nu X.\phi_{Nat}(X)$$

8.3.1 Coinduction at List type

It is useful as for $\mathbf{N}$ to reformulate the coinduction proof principle to prove properties about objects of type $\mathbf{L}_\tau$. Since we have $\cong = [\cong]$ we have that $\cong$ is a bisimulation, in particular it is the largest. Hence since $\mathbf{L}_\tau = \mu T.(\mathbf{0} + (\tau \times T))$ we have that:

$$L \cong_{\mathbf{L}_\tau} L' \iff$$
$$((L \Downarrow \mathbf{abs}_{\mathbf{L}_\tau}(C) \iff L' \Downarrow \mathbf{abs}_{\mathbf{L}_\tau}(C_1)) \;\&\; C \cong_{\mathbf{0}+(\tau\times\mathbf{L}_\tau)} C_1)$$

and again by the fact that $\cong$ is a bisimulation and the fact that we have only one object of type $\mathbf{0}$ we have that:

$$C \cong_{\mathbf{0}+(\tau\times\mathbf{L}_\tau)} C_1 \iff$$
$$((C \Downarrow \mathbf{inl}_{\mathbf{0},(\tau\times\mathbf{L}_\tau)}(M) \iff C_1 \Downarrow \mathbf{inl}_{\mathbf{0},(\tau\times\mathbf{L}_\tau)}(M_1)) \text{ or}$$
$$(C \Downarrow \mathbf{inr}_{\mathbf{0},(\tau\times\mathbf{L}_\tau)}(M) \iff C_1 \Downarrow \mathbf{inr}_{\mathbf{0},(\tau\times\mathbf{L}_\tau)}(M_1) \;\&\; M \cong_{(\tau\times\mathbf{L}_\tau)} M_1))$$

Furthermore we have:

$$M \cong_{(\tau\times\mathbf{L}_\tau)} N \iff$$
$$((M \Downarrow \langle M_1, M_2\rangle \iff N \Downarrow \langle N_1, N_2\rangle) \;\&\; M_1 \cong_\tau N_1 \;\&\; M_2 \cong_{\mathbf{L}_\tau} N_2)$$

Hence since evaluation is deterministic and C and C_1 are in canonical form we have:

$$L \cong_{\mathbf{L}_\tau} L' \iff$$
$$((L \Downarrow \mathbf{abs}_{\mathbf{L}_\tau}(\mathbf{inl}_{\mathbf{0},(\tau\times\mathbf{L}_\tau)}(M)) \iff L' \Downarrow \mathbf{abs}_{\mathbf{L}_\tau}(\mathbf{inl}_{\mathbf{0},(\tau\times\mathbf{L}_\tau)}(N))) \text{ or}$$
$$((L \Downarrow \mathbf{abs}_{\mathbf{L}_\tau}(\mathbf{inr}_{\mathbf{0},(\tau\times\mathbf{L}_\tau)}(M)) \iff L' \Downarrow \mathbf{abs}_{\mathbf{L}_\tau}(\mathbf{inr}_{\mathbf{0},(\tau\times\mathbf{L}_\tau)}(N))) \;\&$$
$$((M \Downarrow \langle M_1, M_2\rangle \iff N \Downarrow \langle N_1, N_2\rangle) \;\&\; M_1 \cong_\tau N_1 \;\&\; M_2 \cong_{\mathbf{L}_\tau} N_2))$$

Definition 8.3.1 (Bisimulation at $\mathbf{L}_\tau$). *$\mathcal{B} \subseteq Exp_{\mathbf{L}_\tau} \times Exp_{\mathbf{L}_\tau}$ is a $\mathbf{L}_\tau$-**bisimulation** if whenever $L\ \mathcal{B}\ L_1$ we have:*

(i) $L \Downarrow \boldsymbol{abs}_{\mathbf{L}_\tau}(\boldsymbol{inl}_{\mathbf{0},(\tau\times\mathbf{L}_\tau)}(M)) \iff L' \Downarrow \boldsymbol{abs}_{\mathbf{L}_\tau}(\boldsymbol{inl}_{\mathbf{0},(\tau\times\mathbf{L}_\tau)}(N))$

(ii) $(L \Downarrow \boldsymbol{abs}_{\mathbf{L}_\tau}(\boldsymbol{inr}_{\mathbf{0},(\tau\times\mathbf{L}_\tau)}(M)) \iff L' \Downarrow \boldsymbol{abs}_{\mathbf{L}_\tau}(\boldsymbol{inr}_{\mathbf{0},(\tau\times\mathbf{L}_\tau)}(N)))$
$\&\; ((M \Downarrow \langle M_1, M_2\rangle \iff N \Downarrow \langle N_1, N_2\rangle) \;\&\; M_1 \cong_\tau N_1 \;\&\; M_2\ \mathcal{B}\ N_2)$

Finally we have the following proof principle.

Definition 8.3.2 (Coinduction at $\mathbf{L}_\tau$). *For each $L, L_1 \in Exp_{\mathbf{L}_\tau}$ if there exists a $\mathbf{L}_\tau$-bisimulation $\mathcal{B}$ such that $L\ \mathcal{B}\ L_1$ then $L \cong_{\mathbf{L}_\tau} L_1$.*

8.3.2 Take Lemma

We now show a classical example where is useful to reason coinductively. For each τ we define a function $Take : \tau \to (\mathbf{L}_\tau \to \mathbf{L}_\tau)$ as:

$$Take \stackrel{\text{def}}{=} \mathbf{rec}\ f.\lambda x.\lambda l.\mathbf{if}\ isZero(x)\ \mathbf{then}\ \varepsilon\ \mathbf{else}$$
$$\mathbf{case}\ \mathbf{rep}(l)\ \mathbf{of}\ \{\mathbf{inl}_{\mathbf{0},\tau\times\mathbf{L}_\tau}(x_1).\mathbf{abs}_{\mathbf{L}_\tau}(\mathbf{inl}_{\mathbf{0},(\tau\times\mathbf{L}_\tau)}(x_1)) \mid$$
$$\mathbf{inr}_{\mathbf{0},\tau\times\mathbf{L}_\tau}(x_2).Cons\ \mathbf{fst}(x_2)(f\ (x-1)\ \mathbf{snd}(x_2))\}$$

An important result for Lists which was first informally discussed by Bird and Wadler in [Bird and Wadler, 1988] is the following proposition.

Proposition 8.3.3 (Take lemma). *For each $L, L' \in Exp_{\mathbf{L}_\tau}$ we have:*

$$\forall N \in Exp_{\mathbf{N}}(Take\ N\ L \cong_{\mathbf{L}_\tau} Take\ N\ L') \Longrightarrow (L \cong_{\mathbf{L}_\tau} L')$$

Unfortunately in general the take lemma for FPC and our definition of Exp_τ fails. For example for each $N \in Exp_{\mathbf{N}}$ we have:

$$Take\ N\ \Omega^{\mathbf{L}_\tau} \cong_{\mathbf{L}_\tau} Take\ N\ \mathbf{abs}_{\mathbf{L}_\tau}(\mathbf{inr}_{\mathbf{0},(\tau\times\mathbf{L}_\tau)}(\Omega^{\tau\times\mathbf{L}_\tau}))$$

but

$$\Omega^{\mathbf{L}_\tau} \ncong_{\mathbf{L}_\tau} \mathbf{abs}_{\mathbf{L}_\tau}(\mathbf{inr}_{\mathbf{0},(\tau\times\mathbf{L}_\tau)}(\Omega^{\tau\times\mathbf{L}_\tau}))$$

Nevertheless we can prove the following restriction of Take lemma. Our proof is an adaptation to the one of Pitts in [Pitts, 1995].

Proposition 8.3.4 (Restricted Take lemma). *For each $L, L' \in List_{Nat}$ we have:*

$$\forall N \in Nat(Take\ N\ L \cong_{\mathbf{L_N}} Take\ N\ L') \Longrightarrow (L \cong_{\mathbf{L_N}} L')$$

Proof. Define $\mathcal{R} \subseteq List_{Nat} \times List_{Nat}$ as

$$\mathcal{R} \stackrel{\text{def}}{=} \{(L, L') | \forall N \in Nat(Take\ N\ L \cong_{\mathbf{L_N}} Take\ N\ L')\}$$

we note that:

$$Take\ N+1\ L \Downarrow \mathbf{abs}_{\mathbf{L_N}}(\mathbf{inl}_{\mathbf{0},(\mathbf{N}\times\mathbf{L_N})}(M)) \iff L \Downarrow \mathbf{abs}_{\mathbf{L_N}}(\mathbf{inl}_{\mathbf{0},(\mathbf{N}\times\mathbf{L_N})}(M)) \tag{8.1}$$

and

$$\begin{array}{c} Take\ N+1\ L \Downarrow \mathbf{abs}_{\mathbf{L_N}}(\mathbf{inr}_{\mathbf{0},(\mathbf{N}\times\mathbf{L_N})}(P))\ \&\ P \Downarrow \langle H, T\rangle \iff \exists P' \\ (L \Downarrow \mathbf{abs}_{\mathbf{L_N}}(\mathbf{inr}_{\mathbf{0},(\mathbf{N}\times\mathbf{L_N})}(P'))\ \&\ P' \Downarrow \langle H, T'\rangle\ \&\ T \equiv Take\ (N+1)-1\ T') \end{array} \tag{8.2}$$

Now assume $L\ \mathcal{R}\ L'$ and suppose $L \Downarrow$. We have two possibilities: either $L \Downarrow \mathbf{abs}_{\mathbf{L_N}}(\mathbf{inl}_{\mathbf{0},(\mathbf{N}\times\mathbf{L_N})}(M))$ where $M : \mathbf{0}$ or $L \Downarrow \mathbf{abs}_{\mathbf{L_N}}(\mathbf{inr}_{\mathbf{0},(\mathbf{N}\times\mathbf{L_N})}(P))$ where $P : (\mathbf{N} \times \mathbf{L_N})$.
If $L \Downarrow \mathbf{abs}_{\mathbf{L_N}}(\mathbf{inl}_{\mathbf{0},(\mathbf{N}\times\mathbf{L_N})}(M))$ then by 8.1 for each $N \in \mathbf{N}$ we have that $Take\ N+1\ L \Downarrow \mathbf{abs}_{\mathbf{L_N}}(\mathbf{inl}_{\mathbf{0},(\mathbf{N}\times\mathbf{L_N})}(M))$. Since $L\ \mathcal{R}\ L'$ we have $Take\ N+1\ L \cong_{\mathbf{L_N}} Take\ N+1\ L'$, hence since $\cong_{\mathbf{L_N}}$ is a $\mathbf{L_N}$-bisimulation there exists M_1 such that $Take\ N+1\ L' \Downarrow \mathbf{abs}_{\mathbf{L_N}}(\mathbf{inl}_{\mathbf{0},(\mathbf{N}\times\mathbf{L_N})}(M_1))$ thus by 8.1 $L' \Downarrow \mathbf{abs}_{\mathbf{L_N}}(\mathbf{inl}_{\mathbf{0},(\mathbf{N}\times\mathbf{L_N})}(M_1))$.
Now suppose $L \Downarrow \mathbf{abs}_{\mathbf{L_N}}(\mathbf{inr}_{\mathbf{0},(\mathbf{N}\times\mathbf{L_N})}(P))$. Since $L \in List_{Nat}$ we have that there exists $H \in Nat, L_1 \in List_{Nat}$ such that $L \cong_{\mathbf{L_N}} Cons\ H\ L_1$. In particular from this follows that $P \Downarrow$.
Hence suppose $L \Downarrow \mathbf{abs}_{\mathbf{L_N}}(\mathbf{inr}_{\mathbf{0},(\mathbf{N}\times\mathbf{L_N})}(P))$ and $P \Downarrow \langle H, T\rangle$, then by 8.2 for each $N \in \mathbf{N}$ we have that $Take\ N+1\ L \Downarrow \mathbf{abs}_{\mathbf{L_N}}(\mathbf{inr}_{\mathbf{0},(\mathbf{N}\times\mathbf{L_N})}(P'))$ and $P' \Downarrow \langle H, Take\ (N+1)-1\ T\rangle$. Since $L\ \mathcal{R}\ L'$ we have $Take\ N+1\ L \cong_{\mathbf{L_N}} Take\ N+1\ L'$, hence since $\cong_{\mathbf{L_N}}$ is a $\mathbf{L_N}$-bisimulation there exists P'' such that $Take\ N+1\ L' \Downarrow \mathbf{abs}_{\mathbf{L_N}}(\mathbf{inr}_{\mathbf{0},(\mathbf{N}\times\mathbf{L_N})}(P''))$ and $P'' \Downarrow \langle H', T'\rangle$ where $H' \cong_{\mathbf{N}} H$ and $T' \cong_{\mathbf{L_N}} Take\ (N+1)-1\ T$. Furthermore by 8.2 we have that there exists P''' such that $L' \Downarrow \mathbf{abs}_{\mathbf{L_N}}(\mathbf{inr}_{\mathbf{0},(\mathbf{N}\times\mathbf{L_N})}(P'''))$ and $P''' \Downarrow \langle H', T''\rangle$ where $T' \equiv Take\ (N+1)-1\ T''$. Hence we have:

$$Take\ (N+1)-1\ T'' \cong_{\mathbf{L_N}} Take\ (N+1)-1\ T$$

and by proposition 8.2.1 we have:

$$Take\ N\ T'' \cong_{\mathbf{L_N}} Take\ N\ T$$

so $T''\ \mathcal{R}\ T$. Hence $\mathcal{R}$ is a $\mathbf{L_N}$-bisimulation and by coinduction 8.3.2, the conclusion follows. □

We have shown some interesting examples of construction in FPC language. Clearly it is possible to extends these constructions: for example one can define other *numerals*, *trees* and many other construction. The last interesting example which we only mention is the following:

$$\Lambda \stackrel{\text{def}}{=} \mu X.(X \to X)$$

which can be considered the type of the *lazy lambda calculus*. See [Winskel, 1993] for an extensive study of this type.

Chapter 9

Conclusions and Future Work

> "And at last we've got to the end of this ideal race-course! Now that you accept A and B and C and D, OF COURSE you accept Z."
> "Do I?" said the Tortoise innocently. "Lets make that quite clear. I accept A and B and C and D. Suppose I STILL refused to accept Z?"
> "Then Logic would take you by the throat, and FORCE you to do it!" Achilles triumphantly replied. "Logic would tell you, 'You can't help yourself. Now that you've accepted A and B and C and D, you MUST accept Z.' So you've no choice, you see."
> "Whatever LOGIC is good enough to tell me is worth WRITING DOWN," said the Tortoise. "So enter it in your book, please"
>
> Lewis Carroll -"What the Tortoise said to Achilles" - Mind, 1895

We have so far introduced (co)inductive definitions and (co)induction as a proof principle and we have shown how they can be studied from a set-theoretic point of view. These notions are useful to define our language FPC and relations which capture the computational meaning of FPC constructs. We have shown that the different relations we have introduced capture the same meaning and how they can be strictly related. Then we have defined *contextual equivalence*, the natural notion of equivalence for FPC programs; furthermore we have proved many interesting properties of the language with respect to it, in particular *rational completeness* and *syntactic continuity*. We have introduced different notions of equivalence for FPC programs, having the advantage that they come equipped with coinductive proof principles. We have proved that they are equivalent to contextual equivalence. Finally we have defined some standard constructions within FPC.
We think that our development of an operational theory of functional programs can be seen as a first attempt to provide a basic formal treatment of FPC in the spirit of the following remark, due to [Pitts, 1998]:

> operational semantics can (sometimes—we are still learning how) be presented at a sufficient 'level of abstraction' to support quite palatable methods of reasoning, calculation and design.

We have introduced induction and coinduction as natural notions in operational theories and we have used them to define properties that capture the computational meaning of our language FPC; these relations can be used in *calculating*. Furthermore we have shown that in our theory we can derive notions analogous to the order-theoretic notions of domain theory and that we can derive proof principle to *reason* either inductively and coinductively about the properties of functional language. Finally we have *designed* some constructions to show how to use FPC as a real programming language.
A critique that can be made to our approach is that it seems too syntactic and at a 'low level of abstraction'. We remark that we have chosen to follow this approach in order to have a concrete base for future developments. We think that the results in the present work will make easy the future job of analyzing operational theory at a *higher level of abstraction*.
A necessary remark is that our work is not intended to show that operational semantics can take the place of the other approaches in the analysis of functional programming languages, but only that it can be widely used to study *some* fields that are usually investigated with other approach. In particular we think that the limits of operational approach must be investigated in relation to those of the denotational approach; or again to describe exactly our positions by the words of Pitts in [Pitts, 1995]:

> I believe that any serious attempt to develop a useful theory for verification of program properties has to involve *both* operational and denotational techniques.

We finally try to point out some interesting directions which seem to need further investigations.
We have shown how induction and coinduction can be effectively used as techniques to reason about program's properties. In particular induction is useful when one wants to prove that a set inductively defined has certain properties; analogously, coinduction is useful when one wants to prove that certain objects belong to a coinductively defined set. Furthermore we have seen that under certain assumptions we can "constructively" characterize inductive and coinductive definitions. This permits to reason "in an inductive manner" for coinductively defined set. This shows that the relations beetween the two concepts are not completely all clear. We think that it will be interesting to investigate in more depth the relations beetween the two notions. Coquand's work [Coquand, 1994] and especially his "guarded induction principle" can be considered a starting point for future works in this direction.
We have developed an operational theory of programs without any reference to denotational models. Nevertheless we think that in order to make a serious analysis of functional programs one needs to show how the operational model is related to the denotational one. In particular it can be useful, following the approach of [Bloom, 1990], [Bloom and Riecke, 1989], [Cosmadakis, 1989] and [Jim and Meyer, 1991], to show the consequences on the denotational model of the particular properties of the operational semantics. We think that an analysis in this direction can clarify the *limits* of the two models.
We have already admitted that our analysis is at a quite-low-level of abstraction. This can be considered a basic approach. To make an analysis at an higher level it is necessary to study more abstract proof methods Many steps in this direction have already been taken by Howe, Pitts, Smith etc.

We have studied operational semantics of a functional programming language. Operational semantics is also widely used to study *process algebras*, it can be interesting to relate the two area of study through an analysis of the difference and similarities based on operational semantics, and also to experiment the application of operational methods to other programming paradigms.
The recent works [Turi, 1996], [Turi and Plotkin, 1997] and [Power, 2003] have shown that it is possible to analyze the concepts of operational semantics in an abstract categorical framework. Because of the generality and the power of category theory this direction needs to be studied in depth.
Finally we think that it is challenging to study the operational approach in an abstract way like what happens in the theory of term rewriting system with various notions of *abstract rewriting*. This is a direction which to our knowledge hasn't been pursued systematically (but see, for example, the many studies of the various formats for labelled transition systems surveyed in [Aceto et al., 1999]).

Appendix A

A brief historical account

The aim of this brief historical discussion is to show how, with mindsight, the operational approach (even, to some extent, the structural one) was already present in the background of the early discussions on the foundations of computer science and how its concepts are "intellectually" related to the denotational approach.
The idea of "computability", introduced by Alan M. Turing in [Turing, 1936] as the common property of functions which can be calculated by a machine, is clearly operational in nature. Turing in [Turing, 1936] wrote:

> The possible behaviour of the machine at any moment is determined by the m-configuration q_n and the scanned symbol $\mathfrak{S}(r)$. This pair $q_n, \mathfrak{S}(r)$ will be called the "configuration": thus the configuration determines the possible behaviour of the machine.

The machine introduced by Turing which is nowadays well known as "Turing Machine" was the first *abstract* machine which was designed to capture all the computable functions.
In the development of the first programming languages the attention of computer scientists was centered on the efficency and expressive power of languages with little or no interest for the definitions of the meanings of commands. For example in the FORTRAN manuals [Backus et al., 1956] the command DO was explained as follows:

> The DO statement is a command to "Do the statements which follow, to and including the statement with statement number n, repeatedly, the first time with $i = m_1$ and with i increased by m_3 for each succeding time; after they have been done with i equal to the highest of this sequence of values which does not exceed m_2 let control reach the statement following the statement with statement number n".
> The *range* of a DO is the set of statements which will be executed repeatedly; it is the sequence of consecutive statements immediately following the DO, to and including the statement numbered n.
> The *index* of a DO is the fixed point variable i, which is controlled by the DO in such a way that its value begins at m_1 and is increased each time by m_8 until it is about to exceed m_2. Throughout

> the range it is available for computation, either as an ordinary fixed point variable or as the variable of a subscript. During the last execution of the range, the Do is said to be *satisfied*.

The number of programming languages grew quickly in the '50 and '60 and early emerged the difficulties of dealing with notions that were not stated formally. At this period we can date back the first attempts to give formal foundations to computer science.
A first step in this direction was due to John McCarthy. In his early works [McCarthy, 1962] ,[McCarthy, 1963] he tried to establish a *mathematical science of computation*, this also to give answer to questions like "What does a program mean?". In [McCarthy, 1962] we read:

> Programs are symbolic expressions representing procedures. The same procedure may be represented by different programs in different programming languages. We shall discuss the problem of defining a programming language semantically by stating what procedures the programs represent.

The contributes of McCarthy to the theory of computer science are not limited to the above, for more information see [Cardone and Hindley, 2004], in particular it is more interesting for our discussion the fact that McCarthy in [McCarthy, 1960] defined the meaning of his LISP through the explaination of how a function *apply*, 'evaluates' the expressions of the language, or by using McCarthy words:

> we describe the universal S-function *apply* which plays the theoretical role of a universal Turing machine and the practical role of an interpreter.

Furthermore McCarthy in [McCarthy, 1963] clarified the limits of Turing Machines as formal model of computation for real programming, and the necessity of new abstract models that capture computational meaning:

> Turing machines are not conceptually different from the automatic computers in general use, but they are very poor in their control structure. Any programmer who has also had to write down Turing machines to compute functions will observe that one has to invent a few artifies and that constructing Turing machines is like programming. Of course, most of the theory of computability deals with questions which are not concerned with the particular ways computations are represented. It is sufficient that computable functions be represented somehow by symbolic expressions.

Finally in [McCarthy, 1963] he goes a step forward and defines what is the meaning of a program:

> We can go farther and describe the meaning of a program in a programming language as follows: *The meaning of a program is defined by its effect on the state vector.*

The ideas of McCarthy were the basis for all the following developments of theories of computation.
At about the same time, Christopher Strachey was indipendently investigating how to formalize the fundamental concept of programming language. In

the introduction to [Strachey, 1967] he clarified which were the objects of his investigations:

> Any discussion on the foundations of computing runs into severe problems right at the start. The difficulty is that although we all use words such as 'name', 'value', 'program', 'expression' or 'command' which we think we understand, it often turns out on closer investigation that in point of fact we all mean different things by these words, so that communication is at best precarious. [...].
> An investigation of the meanings of these basic terms is undoubtedly an exercise in mathematical logic and neither to the taste nor within the field of competence of many people who work on programming languages.

In particular a collaborator of Strachey, Peter Landin in [Landin, 1965], inspired by the works of McCarthy and those of Alonzo Church on the lambda calculus, realized that λ-terms could be used to code constructs of the programming language Algol 60. The main advantage of a formal treatment of Algol-like language was argued by Landin as follows:

> It seems possible that the correspondence might form the basis of a formal description of the semantics of ALGOL 60. As presented here it reduces the problem of specifying ALGOL 60 semantics to that of specifying the semantics of a structurally simpler language.

Furthermore in his earlier work [Landin, 1964], Landin had already introduced an abstract machine for evaluating λ-expressions considered as programs, the SECD-Machine. This consisted of a transition system whose states were made of four components: Stack, Environment, Control and Dump. In [Landin, 1964] it is described how λ-expressions can be evaluated in a simple manner:

> We now describe a "mechanization" of evaluation in the following sense. We define a class of constructed objects, called "states", constructed of applicative expressions and their values; and we define a "transition" rule whose successive application starting at a "state" that contains an environment E and an applicative expression X (in a certain arrangement), leads eventually to a "state" that contains (in a certain position) either $valEX$ or a closure representing $valEX$.

and furthermore, in the spirit of "abstraction":

> It was earlier observed that the mechanization in terms of SECD-states is only one of many ways of mechanizing evaluation. Likewise, given a particular mechanization, there may be many ways of representing it with a digital computer.

It is clear that the above description of how "evaluation" works is closely related to relations that are normally used to define operational semantics, or more precisely, as noted in [Cardone and Hindley, 2004]:

> While the design of the SECD-machine was influenced by similar earlier devices for the interpretation of programming languages, notably the LISP interpreter by Paul Gilmore, it was however the first

> to be presented in a formal way, and may fairly be said to have opened the way to the study of the *formal* operational sematics of λ-calculus considered as a programming language, that would find a systematic exposition a decade later starting with Gordon Plotkin's [Plotkin, 1981].

In the same period the IBM Vienna group was engaged in the big project of trying to describe formally the language PL/I. As noted by Jones in [Jones, 2003]:

> The IBM Vienna group list McCarthy (along with Cal Elgot and Peter Landin) as a major influence on their ambitious attempt to define PL/I using an abstract interpreter.

The work of the Vienna group converged in what is now called *Vienna Definition Language*(VDL). This descriptive style was considered an abstract interpreter for different language. For this reason it was a further step in the direction of operational semantics.
VDL was used by the IBM Vienna group to clarify some aspects of PL/I and other languages and, as argued by Jones, the descriptions of those languages were used "as a basis from which compiler designs could be justified." Unfortunately these good results came with some disadvantages. Jones in [Jones, 2003] clearly states these difficulties.

> VDL definitions tended to use huge states which included control trees that coped with both abnormal sequencing and concurrency. These "grand states" presented gratuitous difficulties in proofs about VDL descriptions. One of the measures considered here of the usefulness of semantic descriptions is the extent to which they make reasoning straightforward.

Nevertheless these developments, with those studied by Landin and several other computer scientists, were the basis for the fundamental work [Plotkin, 1981] that Gordon Plotkin in 1981 wrote for a course at Aarhus University. We do not describe neither it's historical developments (on this, see [Plotkin, 2003] where he has described his "personal intellectual context in which the notes arose") nor the consequences of this systematic exposition because they belong to recent history.
We now focus our attention on the relations beetween the operational and the denotational approach. It is worth noting what Plotkin wrote in [Plotkin, 1981] about the origin of the term 'operational semantics':

> It would be interesting to know the origins of the term 'operational semantics'; an early use is in a paper of Dana's [64, 65] written in the context of discussions with Christopher Strachey where they came up with the denotational/operational distinction.

It is well known that Strachey, that we have already seen to be one of the main promoters of formal semantics, was the pioneer with Dana Scott of the approach to program semantics nowadays well known as *denotational semantics*. Furthermore Plotkin in his discussion clearly stated that there exists a strict relation between his development and the Strachey-Scott denotational approach:

> In fact ideas from denotational semantics pervade SOS.

Furthermore it seems that there exists also a converse relation. Strachey in [Strachey, 1967] wrote:

> all concept of sequencing appears to have vanished. It is, in fact, replaced by the partially ordered sequence of function applications which is specified by λ-expressions.

and again

> the ultimative machine required (and all methods of describing semantics come to a machine ultimately) is in no way specialised. Its only requirement is that it should be able to evaluate pure λ-expressions.

Mosses in [Mosses, 2000] about the above wrote:

> Finally, the reader should not be disconcerted by Stracheys operational interpretation of λ-expressions: the paper was written a full two years before Scott provided a model for the λ-calculus and established the domain theory that forms the mathematical foundations of denotational semantics

In opposition to Mosses' remark we think that Strachey's sentence above are not to be considered as Strachey's "defense" of an operational interpretation of a language but only as the sign that was already clear to Strachey himself that the development of an abstract mathematical semantics must be strictly related with a more concrete operational one. The two approaches are both necessaries and complementary. This was confirmed by a similar argument by Scott in [Scott, 1970]:

> 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.

Bibliography

[Abramsky, 1990] Abramsky, S. (1990). The lazy lambda calculus. In Turner, D. A., editor, *Research Topics in Functional Programming*, pages 65–116. Addison Wesley, Reading, Mass.

[Aceto et al., 1999] Aceto, L., Fokkink, W., and Verhoef, C. (1999). Structural operational semantics. In Bergstra, J., Ponse, A., and Smolka, S., editors, *Handbook of Process Algebra*. Elsevier.

[Aczel, 1977] Aczel, P. (1977). An introduction to inductive definitions. In Barwise, J., editor, *Handbook of Mathematical Logic*, volume 90 of *Studies in Logic and the Foundations of Mathematics*, chapter C.7, pages 739–782. North-Holland, Amsterdam.

[Aczel, 1995] Aczel, P. (1995). Lectures on Semantics : The initial algebra and final coalgebra perspectives. Four lectures given at the 1995 "Logic of Computation" Advanced Study Institute International Summer School at Marktoberdorf.

[Backus et al., 1956] Backus, J. W., S.Best, R. J. B., Goldberg, R., Herrick, H. L., Hughes, R. A., Mitchell, L. B., Nelson, R. A., Nutt, R., D. Sayre, P. B. S., Stern, H., and Ziller, I. (1956). Programmer's reference manual: FORTRAN automatic coding system for the IBM 704. Technical report C28-6000, IBM.

[Barendregt, 1992] Barendregt, H. P. (1992). Lambda calculi with types. In D. M. Gabbay, S. A. and Maiboum, T. S. E., editors, *Handbook of Logic in Computer Science*. Oxford University Press, Oxford.

[Berry, 1981] Berry, G. (1981). Some syntactic and categorical constructions of lambda calculus models. Rapport de Recherche 80, Institute National de Recherche en Informatique et en Automatique (INRIA).

[Bird and Wadler, 1988] Bird, R. and Wadler, P. (1988). *Introduction to Functional Programming*. International Series in Computer Science. Prentice-Hall.

[Birkedal and Harper, 1999] Birkedal, L. and Harper, R. (1999). Relational interpretations of recursive types in an operational setting. *Information and Computation*, 155:3–63.

[Bloom, 1990] Bloom, B. (1990). Can LCF be topped? Flat lattice models of typed λ-calculus. *Information and Computation*, 87(1/2):263–300.

[Bloom and Riecke, 1989] Bloom, B. and Riecke, J. G. (1989). LCF should be lifted. In Rus, T., editor, *Proc. Conf. Algebraic Methodology and Software Technology*, pages 133–136. Department of Computer Science, University of Iowa.

[Cardone and Hindley, 2004] Cardone, F. and Hindley, J. R. (2004). History of lambda-calculus and combinatory logic. Draft.

[Coquand, 1994] Coquand, T. (1994). Infinite objects in type theory. In Barendregt, H. and Nipkow, T., editors, *Selected Papers 1st Int. Workshop on Types for Proofs and Programs, TYPES'93, Nijmegen, The Netherlands, 24–28 May 1993*, volume 806, pages 62–78. Springer-Verlag, Berlin.

[Cosmadakis, 1989] Cosmadakis, S. (1989). Computing with recursive types. In *Symposium on Logic in Computer Science (LICS '89)*, pages 24–38, Washington, D.C., USA. IEEE Computer Society Press.

[Cousot and Cousot, 1979] Cousot, P. and Cousot, R. (1979). Constructive versions of Tarski's fixed point theorems. *Pacific Journal of Mathematics*, 81(1):43–57.

[Cousot and Cousot, 1992] Cousot, P. and Cousot, R. (1992). Inductive definitions, semantics and abstract interpretation. In *Conference Record of the Nineteenth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 83–94, Albequerque, New Mexico.

[Crole, 1998] Crole, R. L. (1998). Lectures on [Co]Induction and [Co]Algebras. Technical Report 1998/12, Department of Mathematics and Computer Science, University of Leicester.

[Crole, 2003] Crole, R. L. (2003). Coinduction and Bisimulation. Lectures for the International Summer School on the Foundations of Security, University of Oregon, Eugene, USA.

[Davey and Priestley, 2002] Davey, B. A. and Priestley, H. A. (2002). *Introduction to Lattices and Order: Second Edition*. Cambridge University Press.

[Felleisen and Friedman, 1986] Felleisen, M. and Friedman, D. P. (1986). Control operators, the SECD-machine, and the λ-calculus. In *Formal Description of Programming Concepts III*, pages 193–217. North-Holland.

[Fiore, 1994] Fiore, M. P. (1994). *Axiomatic domain theory in categories of partial maps*. PhD thesis, University of Edinburgh.

[Gordon, 1995] Gordon, A. D. (1995). Bisimilarity as a theory of functional programming: mini-course. Notes Series BRICS-NS-95-3, BRICS, Department of Computer Science, University of Aarhus.

[Gordon and Pitts, 1998] Gordon, A. D. and Pitts, A. M., editors (1998). *Higher Order Operational Techniques in Semantics*. Publications of the Newton Institute. Cambridge University Press.

[Gordon and Pitts, 1999] Gordon, A. D. and Pitts, A. M., editors (1999). *Higher Order Operational Techniques in Semantics, Third International Workshop, Paris, 1999*, volume 26 of *Electronic Notes in Theoretical Computer Science*. Elsevier.

[Gordon et al., 1998] Gordon, A. D., Pitts, A. M., and Talcott, C., editors (1998). *Second Workshop on Higher-Order Operational Techniques in Semantics (HOOTS II), Stanford University, December 8-12, 1997*, volume 10 of *Electronic Notes in Theoretical Computer Science*. Elsevier.

[Gunter, 1992] Gunter, C. A. (1992). *Semantics of Programming Languages: Structures and Techniques*. Foundations of Computing. MIT Press.

[Harper, 2003] Harper, R. (2003). Programming languages: Theory and practice. Draft.

[Howe, 1989] Howe, D. J. (1989). Equality in lazy computation systems. In *Proceedings, Fourth Annual Symposium on Logic in Computer Science*, pages 198–203, Asilomar Conference Center, Pacific Grove, California. IEEE Computer Society Press.

[Howe, 1996] Howe, D. J. (1996). Proving congruence of bisimulation in functional programming languages. *Information and Computation*, 124:103–112.

[Ibraheem and Schmidt, 2000] Ibraheem, H. and Schmidt, D. A. (2000). Adapting big-step semantics to small-step style: Coinductive interpretations and "higher-order" derivations. In Gordon, A., Pitts, A., and Talcott, C., editors, *Electronic Notes in Theoretical Computer Science*, volume 10. Elsevier.

[Jacobs, 1997] Jacobs, B. (1997). Invariants, bisimulations and the correctness of coalgebraic refinements. *Lecture Notes in Computer Science*, 1349:276–292.

[Jacobs and Rutten, 1996] Jacobs, B. and Rutten, J. (1996). A tutorial on (co)algebras and (co)induction. *Bulletin of the EATCS*, 62:222–259.

[Jeffrey, 2000] Jeffrey, A., editor (2000). *Higher Order Operational Techniques in Semantics, Fourth International Workshop, Montreal, 2000*, volume 41 of *Electronic Notes in Theoretical Computer Science*. Elsevier.

[Jim and Meyer, 1991] Jim, T. and Meyer, A. R. (1991). Full abstraction and the context lemma. In Ito, T. and Meyer, A. R., editors, *Proceedings of Theoretical Aspects of Computer Software.(TACS '91)*, volume 526 of *LNCS*, pages 131–151, Berlin,. Springer.

[Jones, 2003] Jones, C. B. (2003). Operational semantics: concepts and their expression. *Inf. Process. Lett.*, 88(1-2):27–32.

[Kahn, 1987] Kahn, G. (1987). Natural semantics. In *Proceedings of the Symposium on Theoretical Aspects of Computer Science*, volume 247 of *Lecture Notes in Computer Science*, pages 22–39. Springer-Verlag.

[Kleene, 1952] Kleene, S. C. (1952). *Introduction to Metamathematics*. North-Holland, Amsterdam, 7 edition.

[Landin, 1964] Landin, P. J. (1964). The mechanical evaluation of expressions. *Computer Journal*, 6(5):308–320.

[Landin, 1965] Landin, P. J. (1965). Correspondence between Algol 60 and Church's lambda-notation: part I. *Commun. ACM*, 8(2):89–101.

[Martin-Löf, 1982] Martin-Löf, P. (1982). Constructive mathematics and computer programming. In *Sixth International Congress for Logic, Methodology, and Philosophy of Science*, pages 153–175, Amsterdam. North-Holland.

[Mason et al., 1996] Mason, I. A., Smith, S., and Talcott, C. L. (1996). From operational semantics to domain theory. *INFCTRL: Information and Computation (formerly Information and Control)*, 128.

[Mason and Talcott, 1991] Mason, I. A. and Talcott, C. L. (1991). Equivalence in functional languages with effects. *Journal of Functional Programming*, 1(3):287–327.

[McCarthy, 1960] McCarthy, J. (1960). Recursive functions of symbolic expressions, and their computation by machine, part I. *Communications of the ACM*, 3(3):184–195. siehe McCarthy85.

[McCarthy, 1962] McCarthy, J. (1962). Towards a mathematical science of computation. In *Information Processing '62*, pages 21–28. North-Holland. Proceedings of 1962 IFIP Congress.

[McCarthy, 1963] McCarthy, J. (1963). A basis for a mathematical theory of computation. In Braffort, P. and Hirschberg, D., editors, *Computer Programming and Formal Systems*, pages 33–70. North-Holland, Amsterdam, NL.

[McCusker, 1996a] McCusker, G. (1996a). *Games and Full Abstraction for a Functional Metalanguage with Recursive Types*. PhD thesis, Department of Computing, Imperial College, University of London.

[McCusker, 1996b] McCusker, G. (1996b). Games and full abstraction for FPC. In *Proceedings of the Eleventh Annual IEEE Symposium on Logic In Computer Science (LICS'96)*, pages 174–183, New York, USA. IEEE Computer Society Press.

[Milner, 1977] Milner, R. (1977). Fully abstract models of typed lambda-calculus. *Theoretical Computer Science*, 4:1–22.

[Milner, 1989] Milner, R. (1989). *Communication and concurrency*. International series in computer science. Prentice Hall, New York.

[Moran, 1994] Moran, A. K. (1994). Natural Semantics for Non-Determinism. Licentiate Thesis, Chalmers University of Technology and University of Göteborg, Sweden.

[Morris, 1968] Morris, J. H. (1968). *Lambda-Calculus Models of Programming Languages*. Ph.D. thesis, MIT, Cambridge, MA.

[Mosses, 2000] Mosses, P. D. (2000). A foreword to 'fundamental concepts in programming languages'. *Higher-Order and Symbolic Computation*, 13(1–2):7–9.

[Ong, 1995] Ong, C.-H. L. (1995). Correspondence between operational and denotational semantics. In Abramsky, S., Gabbay, D., and Maibaum, T. S. E., editors, *Handbook of Logic in Computer Science, Vol 4*, pages 269–356. Oxford University Press.

[Ong and Abramsky, 1993] Ong, C.-H. L. and Abramsky, S. (1993). Full abstraction in the lazy lambda calculus. *Information and Computation*, 105:159–267.

[Park, 1981] Park, D. (1981). Concurrency and automata on infinite sequences. In *Theoretical Computer Science, 5th GI-Conf.*, LNCS 104, pages 167–183. Springer-Verlag, Karlsruhe.

[Pitts, 1994] Pitts, A. M. (1994). Some notes on inductive and co-inductive techniques in the semantics of functional programs. Notes Series BRICS-NS-94-5, BRICS, Department of Computer Science, University of Aarhus. vi+135 pp, draft version.

[Pitts, 1995] Pitts, A. M. (1995). Operationally-based theories of program equivalence. In Dybjer, P. and Pitts, A. M., editors, *Semantics and Logics of Computation*. Cambridge University Press. Based on lectures given at the CLICS-II Summer School on Semantics and Logics of Computation, Isaac Newton Institute for Mathematical Sciences, Cambridge UK, September 1995.

[Pitts, 1998] Pitts, A. M. (1998). Operational versus denotational methods in the semantics of higher order languages (tutorial). In Palamidessi, C., Glaser, H., and Meinke, K., editors, *Principles of Declarative Programming, 10th Int. Symp., PLILP'98*, volume 1490 of *Lecture Notes in Computer Science*, pages 282–283. Springer-Verlag, Berlin.

[Pitts, 2002] Pitts, A. M. (2002). Operational semantics and program equivalence. *Lecture Notes in Computer Science*, 2395:378–411.

[Pitts and Stark, 1998] Pitts, A. M. and Stark, I. D. B. (1998). Operational reasoning for functions with local state. In Gordon, A. D. and Pitts, A. M., editors, *Higher Order Operational Techniques in Semantics*, Publications of the Newton Institute, pages 227–273. Cambridge University Press.

[Plotkin, 1981] Plotkin, G. D. (1981). A structural approach to operational semantics. Technical Report DAIMI FN-19, Aarhus University.

[Plotkin, 1985] Plotkin, G. D. (1985). Denotational semantics with partial functions. Lecture notes, C.S.L.I. Summer School, Stanford.

[Plotkin, 2003] Plotkin, G. D. (2003). The Origins of Structural Operational Semantics. Unpublished.

[Power, 2003] Power, J. (2003). Towards a theory of mathematical operational semantics. In Gumm, H. P., editor, *Electronic Notes in Theoretical Computer Science*, volume 82. Elsevier.

[Rutten, 2001] Rutten, J. J. M. M. (2001). Elements of stream calculus (an extensive exercise in coinduction). In Brookes, S. and Mislove, M., editors, *Proc. of 17th Conf. on Mathematical Foundations of Programming Semantics, Aarhus, Denmark, 23–26 May 2001*, volume 45 of *Electronic Notes in Theoretical Computer Science*. Elsevier, Amsterdam.

[Sands, 1997] Sands, D. (1997). From SOS rules to proof principles: An operational metatheory for functional languages. In *Conference Record of POPL'97: The 24th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 428–441, Paris.

[Scott, 1970] Scott, D. (1970). Outline of a mathematical theory of computation. In *Proc. Fourth Annual Princeton Conference on Information Sciences and Systems*, pages 169–176. Princeton University.

[Scott and Strachey, 1971] Scott, D. and Strachey, C. (1971). Towards a mathematical semantics for computer languages. Technical Report PRG-6, Oxford University Computer Laboratory.

[Smith, 1991] Smith, S. F. (1991). From Operational to Denotational Semantics. In Main, M. et al., editors, *Mathematical Foundation of Programming Semantics'91*, volume 598 of *LNCS*, pages 54–76. Springer-Verlag.

[Strachey, 1967] Strachey, C. (1967). Fundamental concepts in programming languages. Lecture Notes, International Summer School in Computer Programming, Copenhagen. Reprinted in *Higher-Order and Symbolic Computation*, 13(1/2), pp. 1–49, 2000.

[Tarski, 1955] Tarski, A. (1955). A lattice-theoretic fixpoint theorem and its applications. *Pacific Journal of Mathematics*, 5(2):285–309.

[Turi, 1996] Turi, D. (1996). *Functorial Operational Semantics and Its Denotational Dual*. PhD thesis, Free University, Amsterdam.

[Turi and Plotkin, 1997] Turi, D. and Plotkin, G. D. (1997). Towards a mathematical operational semantics. In *Proc. of 12th Ann. IEEE Symp. on Logic in Computer Science, LICS'97, Warsaw, Poland, 29 June – 2 July 1997*, pages 280–291. IEEE Computer Society Press, Los Alamitos, CA.

[Turing, 1936] Turing, A. M. (1936). On computable numbers with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society Series 2*, 42:230–265.

[Winskel, 1993] Winskel, G. (1993). *The Formal Semantics of Programming Languages: An Introduction*. Foundation of Computing Series. The MIT Press, Cambridge, MA.