# BRICS

**Basic Research in Computer Science** 

# **Relational Reasoning about Functions and Nondeterminism**

Søren Bøgh Lassen

**BRICS Dissertation Series** 

DS-98-2

ISSN 1396-7002

December 1998

Copyright © 1998,BRICS, Department of Computer Science<br/>University of Aarhus. All rights reserved.

Reproduction of all or part of this work is permitted for educational or research use on condition that this copyright notice is included in any copy.

See back inner page for a list of recent BRICS Dissertation Series publications. Copies may be obtained by contacting:

> BRICS Department of Computer Science University of Aarhus Ny Munkegade, building 540 DK–8000 Aarhus C Denmark Telephone: +45 8942 3360 Telefax: +45 8942 3255 Internet: BRICS@brics.dk

**BRICS** publications are in general accessible through the World Wide Web and anonymous FTP through these URLs:

http://www.brics.dk
ftp://ftp.brics.dk
This document in subdirectory DS/98/2/

# Relational Reasoning about Functions and Nondeterminism

Søren Bøgh Lassen

# Ph.D. Dissertation



Department of Computer Science University of Aarhus Denmark

# Relational Reasoning about Functions and Nondeterminism

Dissertation presented to the Faculty of Science of the University of Aarhus in partial fulfillment of the requirements for the Ph.D. degree

> by Søren Bøgh Lassen December 7, 1998

## Abstract

This dissertation explores a uniform, relational proof style for operational arguments about program equivalences. It improves and facilitates many previously given proofs, and it is used to establish new proof rules for reasoning about term contexts, recursion, and nondeterminism in higher-order programming languages.

Part I develops an algebra of relations on terms and exploits these relations in operational arguments about contextual equivalence for a typed deterministic functional language. Novel proofs of the basic laws, sequentiality and continuity properties, induction rules, and the CIU Theorem are presented together with new relational proof rules akin to Sangiorgi's "bisimulation up to context" for process calculi.

Part II extends the results from the first part to nondeterministic functional programs. May and must operational semantics and contextual equivalences are defined and their properties are explored by means of relational techniques. For must contextual equivalence, the failure of ordinary syntactic  $\omega$ -continuity in the presence of countable nondeterminism is addressed by a novel transfinite syntactic continuity principle. The relational techniques are also applied to the study of lower and upper applicative simulation relations, yielding new results about their properties in the presence of countable and fair nondeterminism, and about their relationship with the contextual equivalences.

# Acknowledgments

I thank Peter Mosses for advice and support. I am also grateful to many others for discussions and valuable inputs to my work. In Aarhus, these included Olivier Danvy, Glynn Winskel, Jaap van Oosten, Torben Braüner, Peter Ørbæk, and Ian Stark. Special thanks go to Andrew Gordon and Andrew Pitts for inviting me to Cambridge and for their inputs and guidance throughout the work that led to this dissertation. I also wish to thank Dave Sands, Luke Ong, Paritosh Pandya, Carolyn Talcott, and Dave Schmidt, whom I communicated with and met on various occasions around the world. Dave Sands and Carolyn Talcott also provided valuable feedback as my Ph.D. examiners. During the last phase of the work on Part II of the dissertation, I benefitted immensely from extended discussions with Corin Pitcher and Andrew Moran. I wish them luck with their dissertations.

Last, but not least, I wish to thank my wife and children, Sharmila, Paul and Philip, for their incredible patience and support. We are grateful to Laju and Nitin for their help.

I was supported by grants from Aarhus University Research Foundation and the Danish Natural Science Research Council.

# Preface

The work reported in this dissertation took place at BRICS, Department of Computer Science, University of Aarhus, and during two visits to the Computer Laboratory, University of Cambridge. It was carried out under the supervision of Dr. Peter Mosses.

Parts of the dissertation are based on results that appear in other forms in two publications (Lassen 1997; Lassen 1998) and in joint work with Corin Pitcher from Oxford University (Lassen and Pitcher 1998).

My interest in the problems of semantic equivalence for higher-order languages with unbounded nondeterminism arose while investigating the theory of action semantics (Mosses 1992). It resulted in an operational treatment of a functional and declarative, nondeterministic fragment of action notation, the specification language of action semantics, in Lassen (1997) (which supersedes the earlier work in Lassen 1994, 1995). The work included a generalisation of a precongruence proof for applicative similarity in Ong (1992) to deal with unbounded nondeterminism. This generalisation was found independently by Corin Pitcher and appears together with some new results in a joint paper (Lassen and Pitcher 1998). The results about applicative simulation for bounded and unbounded nondeterminism in Lassen (1997) are expanded and reworked for a typed functional language in Part II of the dissertation.

The relational approach in the dissertation was first developed in Lassen (1998) but is here transfered to typed and nondeterministic settings and is applied to the study of contextual equivalence rather than applicative bisimulation. Several original results from Lassen (1998) about applicative bisimulation up to context and about improvement are not included here.

In joint work with Dr. Andrew Gordon, now at Microsoft Research in Cambridge, and Paul Hankin from University of Cambridge, I have been applying the relational techniques from this dissertation in investigations of operational equivalence for Abadi and Cardelli's imperative object calculi (Abadi and Cardelli 1996). Results for the simple untyped version of the calculus are reported in Gordon, Hankin and Lassen (1997a, 1997b).

After the dissertation was originally submitted in February 1998, I have further developed the material about improvement theory for ambiguous choice from Chapter 8 in cooperation with Andrew Moran. Preliminary results are included in his dissertation (Moran 1998). Moran's dissertation on ambiguous choice as well as the forthcoming dissertation work on nondeterminism by Corin Pitcher, Oxford University, and Russ Harmer, Imperial College, London, did not appear in time to be discussed in the present dissertation.

## Prerequisites

The reader is assumed to be familiar with functional programming languages, operational semantics, and the basic principles of inductive definitions and proofs by induction. These subjects are covered in textbooks on programming language semantics, e.g., Winskel (1993). In §6.7, a passing familiarity with transfinite induction and ordinals is assumed; see, e.g., Phillips (1992) or Rogers (1967).

# Contents

| 1 | $\mathbf{Intr}$ | oduction                                 | 1         |
|---|-----------------|--|-----------|
|   | 1.1             | Background                               | 1         |
|   | 1.2             | Overview                                 | 4         |
| Ι | Det             | terminism                                | 7         |
| 2 | A D             | eterministic Functional Language         | 9         |
|   | 2.1             | Syntax                                   | 9         |
|   | 2.2             | Operational semantics                    | 11        |
|   | 2.3             | Types                                    | 15        |
|   | 2.4             | Contexts                                 | 18        |
| 3 | Rela            | ations                                   | <b>21</b> |
|   | 3.1             | Binary relations                         | 22        |
|   | 3.2             | Open and closed relations                | 22        |
|   | 3.3             | Matching values                          | 23        |
|   | 3.4             | Substitution                             | 24        |
|   | 3.5             | Compatible refinement                    | 26        |
|   | 3.6             | Compatibility                            | 27        |
|   | 3.7             | Contextual equivalence                   | 29        |
|   | 3.8             | Compatibility and substitutivity         | 31        |
|   | 3.9             | Relations of higher arity                | 33        |
|   | 3.10            | Structural reduction                     | 34        |
|   | 3.11            | Future work                              | 35        |
| 4 | Rela            | ational Reasoning                        | 37        |
|   | 4.1             | Contextual approximation and equivalence | 38        |
|   | 4.2             | Simulation                               | 38        |
|   | 4.3             | Kleene approximation and equivalence     | 39        |
|   | 4.4             | Sequentiality                            | 43        |
|   | 4.5             | Unwinding and syntactic continuity       | 46        |
|   | 4.6             | Compatible simulations                   | 50        |
|   | 4.7             | The CIU Theorem                          | 52        |
|   | 4.8             | Similarity                               | 54        |

| II Nondeterminism                        |  |       |     |     |   | <b>57</b> |     |     |           |   |          |
|--|--|-------|-----|-----|---|-----------|-----|-----|-----------|---|----------|
| 5 A Nondeterministic Functional Language |  |       |     |     |   |           |     |     | <b>59</b> |   |          |
|  | 5.1 Syntax   | •••   | • • | • • | • | •••       | • • | ·   | •••       | • | 09<br>60 |
|  | 5.2 Operational semantics  | •••   | • • | • • | • | •••       | • • | ·   | •••       | • | 00<br>64 |
|  | 5.4 Polotiona  |       | ••• | • • | • | •••       | ••• | ·   | •••       | • | 04<br>64 |
|  | 5.4 Relations  |       | ••• | • • | • | •••       | ••• | •   | •••       | • | 04       |
| 6  | 6 Contexts   |       |     |     |   |           |     |     |           |   | 65       |
|  | 6.1 Contextual approximation and equivalence                       |       |     |     | • |           |     |     |           | • | 65       |
|  | 6.2 Lower and upper simulation $\ldots$ $\ldots$ $\ldots$ $\ldots$ |       |     |     | • |           |     |     |           | • | 68       |
|  | 6.3 Lower relational reasoning                                     |       |     |     | • |           |     |     |           | • | 68       |
|  | 6.4 May equational theory  |       |     |     |   |           |     |     |           | • | 71       |
|  | 6.5 Upper relational reasoning                                     |       |     |     |   |           |     |     |           |   | 73       |
|  | 6.6 Must equational theory   |       |     |     |   |           |     |     |           |   | 77       |
|  | 6.7 Transfinite unwinding and syntactic $\omega_1^{CK}$ -continuit |       |     |     |   |           |     |     |           |   | 79       |
|  | 6.8 Sequentiality  |       |     |     |   |           |     |     |           |   | 85       |
|  | 6.9 Related work   |       |     |     | • |           |     |     |           | • | 87       |
| 7  | 7 Simulation   |       |     |     |   |           |     |     |           |   | 89       |
|  | 7.1 Lower similarity   |       |     |     |   |           |     |     |           |   | 89       |
|  | 7.2 Upper similarity $\ldots$                                      |       |     |     |   |           |     |     |           |   | 90       |
|  | 7.3 Other simulation and bisimulation relations                    |       |     |     |   |           |     |     |           |   | 91       |
|  | 7.4 Syntactic continuity   |       |     |     |   |           |     |     |           |   | 92       |
|  | 7.5 Open problems  |       | • • | • • | • | •••       | • • | ·   | • •       | • | 94       |
|  | 7.6 Related work   | · · · | ••• | ••• | • | · ·       | ••• | :   |           |   | 94       |
| _  |  |       |     |     |   |           |     |     |           |   |          |
| 8  | 8 Fairness   |       |     |     |   |           |     | 95  |           |   |          |
|  | 8.1 Syntax and operational semantics                               |       |     | • • | • |           | • • | ·   |           | • | 95       |
|  | 8.2 Generalising the sequential theory                             |       | • • | • • | • |           | • • | •   | • •       | • | 96       |
|  | 8.3 Refinement   |       |     |     | • |           |     | •   |           | • | 99       |
|  | 8.4 Improvement  |       |     |     | • |           |     |     |           | • | 103      |
|  | 8.5 Future work  |       |     |     | • | • •       |     |     |           | • | 112      |
| Α  | A Encoding ordinal-bounded fixed point operators                   |       |     |     |   |           |     |     |           |   | 113      |
| Bibliography 117                         |  |       |     |     |   |           |     |     | 117       |   |          |
|  |  |       |     |     |   |           |     |     |           |   |          |
| Symbol Index 12                          |  |       |     |     |   |           |     | 125 |           |   |          |

# Chapter 1

# Introduction

This dissertation develops techniques for formal reasoning about higher-order programming languages and applies these techniques to reason about program equivalences and nondeterminism. The results extend and generalise existing syntactic techniques for deriving properties of higher-order programs from their operational semantics; and new results about program equivalences are obtained both for deterministic programs and for various forms of nondeterminism, including bounded and countable nondeterminism as well as fairness.

This chapter provides an introduction to the research on operational method for higherorder languages, and an outline of the remainder of this dissertation.

## 1.1 Background

Syntactic and operational methods are today widely used in work on semantics of programming languages. The attraction of this approach is the flexibility and the low mathematical overhead that it imposes. Syntax and operational semantics consist of inductively defined entities and the principal mathematical tool needed for their analysis is that of induction. In many cases, the main task is to master the syntactic complexity that one encounters. Since syntax is application specific, it is difficult to obtain general results and techniques that can be transferred from one application to the next. One recurrent concept, however, is that of relations between terms. They link syntax and operational semantics, and many semantic problems can be expressed in the language of term relations, in particular, notions of semantic equivalence and approximation between terms. Moreover, relations enjoy some general theory that is useful across many different applications. One of the results in this work is the formulation of a useful theory of term relations suited for syntactic reasoning about higherorder programming languages. The underlying relational theory is based on simple induction on syntax and inductively defined relations, yet it provides a non-trivial toolkit of notation and theory for mastering the syntactic complexity in expressing and reasoning about program properties based on operational semantics.

## **Operational equivalence**

The program properties that are considered here are mainly in the form of semantic equivalences and orderings between programs. In contextual equivalence we have, at least for deterministic programming languages, a standard and generally accepted syntactic definition of operational equivalence based on suitable observations of the operational behaviour of programs. Denotational semantics has provided operationally adequate models for reasoning about operational equivalence, but the underlying domain theory incurs a certain mathematical overhead, in particular, if we need recursively defined domains, models of (unbounded) nondeterminism or fairness, or fully abstract ("good-fit") models for features such as sequentiality and local state (see Fiore et al. 1996).

Progress on these aspects of domain theory has been complemented by a large body of work on operationally-based reasoning about operational equivalence. For higher-order languages, two approaches have been particularly influential. One is that of Ian Mason and Carolyn Talcott, surveyed in Talcott (1998). It is based on a so-called *CIU Theorem*, or Generalised Context Lemma, which gives a tractable characterisation of contextual equivalence. The CIU characterisation is valid and useful for reasoning about higher-order functions, even in the presence of mutable state (Mason and Talcott 1991), objects (Gordon, Hankin, and Lassen 1997a), control operators (Talcott 1998; Bierman 1998), and nondeterminism (Lassen 1997). Another approach is the operational theory of applicative bisimulation, first developed to model lazy and call-by-value calculi (Abramsky 1990; Howe 1989; Egidi et al. 1992; Perez 1991; Ong 1992) and later adapted to call-by-name theories (Gordon 1995a) and to state-less object calculi (Gordon and Rees 1996; Gordon 1998). For deterministic languages without side-effects, applicative bisimulation generally matches contextual equivalence, a property called operational extensionality (Bloom 1990), yielding a slightly stronger characterisation than the CIU Theorem. Forms of applicative bisimulation have also been used to model state (Ritter and Pitts 1995) and nondeterminism (Ong 1993; Lassen 1997), but in these cases it is more fine-grained than contextual equivalence—except if one adopts a continuation-passing style as in Wand and Sullivan (1997).

The characterisations of contextual equivalence provided by the CIU Theorem and operational extensionality offer the following benefits when reasoning about equivalences. Firstly, they permit simple soundness proofs of the reductions performed by the operational semantics. Secondly, they assert certain extensionality properties which are very useful, e.g., when reasoning about terms with free variables. Finally, applicative bisimulation provides a coinduction principle for reasoning about the infinite data structures found in higher-order languages. But neither the CIU Theorem nor operational extensionality are particularly helpful for proving general results about recursion, e.g., the validity of the fundamental induction rules for recursion such as recursion induction, syntactic continuity ( $\omega$  induction), and syntactic minimal invariance (syntactic projections).

## **Contexts and relations**

In general, when reasoning syntactically about recursion, we need to consider term contexts. They also arise in arguments about contextual equivalence (as the name suggests). For instance, the CIU Theorem and operational extensionality are proved by showing that appropriate equivalence relations are congruences, i.e., closed under contexts. Intuitively, a context is a term containing holes that may be filled by other terms. This is an evocative idea, but for formal arguments contexts are difficult to work with, both technically and notationally.

In his seminal operationally-based congruence proof for applicative bisimilarity, Howe (1989, 1996) uses a *relational* method to conduct a complex syntactic argument. Instead of considering contexts, the proof involves the construction of a "congruence candidate" relation, closed under contexts, which is shown to coincide with applicative bisimilarity by induction

on computations and co-induction. The relational approach results in a rigorous and precise proof, both technically and notationally. Moreover, the proof applies to many different typed and untyped higher-order languages and operational orderings; see, e.g., (Sands 1991; Ong 1992; Ferreira et al. 1996; Lassen 1997; Gordon 1998; Lassen 1996). Pitts (1995) extended Howe's congruence proof for applicative bisimilarity to also establish an "up to context" proof rule (Sangiorgi 1994) for applicative bisimulation. This work was generalised in Lassen (1998) by the introduction of an algebra of (untyped) term relations for constructing suitable context closed relations for a wide range of applications, including proofs of general induction principles for reasoning about recursion and bisimulation up to context proof rules, both for applicative bisimulation and for a variant of Sands' improvement theory (Sands 1998b). Parts of the results in Lassen (1998) are reworked below for typed terms and contextual equivalence in Chapters 3 and 4.

## Nondeterminism

The second part of the dissertation focuses on reasoning principles for nondeterministic programs. There are several motivations for incorporating nondeterminism in semantic theories of programming languages. One is that there are aspects of program behaviour, e.g., in the communication pattern of concurrent systems, that are best modelled as nondeterministic. Nondeterminism is also relevant for reasoning abstractly about under-specified systems where some freedom of choice is left to the implementation. The research that led to the results presented here began in the context of action semantics (Mosses 1992) which features both unbounded nondeterminism and fairness.

A number of challenges for semantic modelling arise when nondeterminism is introduced into a programming language. Some of the fundamental reasoning principles from deterministic settings are invalid for nondeterminism. In programming languages with unbounded nondeterminism the  $\omega$ -continuity properties required by conventional domain theory fail. Apt and Plotkin (1986) showed that this is inevitable, and it complicates the partial-order models that are used for reasoning about recursion as least fixed points. In the presence of fairness, things are even worse: recursion cannot be modelled by least fixed points because there are operators that are not monotone with respect to the semantic orderings. Only a few results have been obtained about fairness in higher-order languages; see Moran (1994) and Agha, Mason, Smith, and Talcott (1997).

Moreover, nondeterminism makes it more difficult to fix a notion of semantic equivalence. Different applications lead to conflicting requirements, suggesting that the semantic theory should accommodate a family of semantic equivalences. A multitude of behavioural equivalences have been proposed in the literature. There are many co-inductively defined bisimulation equivalences (Park 1981; Milner 1989) and they are attractive because they permit proofs by co-induction, but in the presence of nondeterminism it can be argued that they are finer-grained than is warranted by reasonable notions of observation. In contrast, testing equivalences (DeNicola and Hennessy 1984) are defined in terms of observations, more in the style of contextual equivalence.

In operationally-based theories for nondeterministic higher-order languages, the choice between bisimulation and testing equivalences is reflected in the choice between whether to generalise applicative bisimulation or contextual equivalence to the nondeterministic case.

Howe (1989, 1996), Ong (1992), and Moran (1994) have used forms of applicative bisimulation to model nondeterministic and concurrent extensions of higher-order languages. In Chapter 7 the continuity properties of two forms of applicative simulation preorders are investigated and Ong's results are extended to countable nondeterminism. This work is also reported in Lassen (1997) and Lassen and Pitcher (1998).

Contextual equivalences have been considered in (Hennessy and Ashcroft 1980; Astesiano and Costa 1980; Astesiano and Costa 1984; Sieber 1993; Dezani-Ciancaglini, de'Liguoro, and Piperno 1996; de'Liguoro and Piperno 1995) but mainly as a reference for judging full abstraction properties of domain-theoretic models. Sieber (1993) has demonstrated some fundamental full abstraction problems in such models. None of these works consider unbounded nondeterminism or fairness. These features are addressed in the few operationally-based studies of nondeterministic contextual equivalence for higher-order languages in the literature: Moran (1994), Agha, Mason, Smith, and Talcott (1997), and Lassen (1997); but they are all of a preliminary nature. In Chapters 6 and 8 the theory of contextual equivalences for bounded and countable nondeterminism and a form of fairness is developed in greater depth.

## Applications

The relational approach, developed in this dissertation, is applied here to reason about sequentiality, termination, equivalence, and continuity properties of higher-order, nondeterministic programs. But the underlying relational techniques should apply to many other formalisms and problems than those that are explored here.

A particularly interesting and challenging problem—for which the operational theory in this dissertation is intended—is to develop the theory of action semantics for reasoning about programs. Action semantics (Mosses 1992, 1996) is a structured framework for semantic description of realistic programming languages. A useful semantic theory for the specification language action notation would provide the tools for reasoning about the many programming languages with action semantic descriptions. Some results for a functional, declarative, and nondeterministic fragment of action notation are reported in Lassen (1997), but much remains to be done. The theory should at least be extended to the imperative facet of action notation to be of practical use for realistic action semantic descriptions. Research on the semantic foundations of action notation suggests some changes in the design of the notation that will facilitate the theoretical development and strengthen the resulting semantic theory. The work by Wansbrough and Hamer (1997) on combining action semantics and modular monadic semantics is a recent a valuable contribution to our understanding of the semantic structure of action notation. It should be important for further research in this area.

## 1.2 Overview

**Part I** introduces an algebra of relations on terms and uses it in operational reasoning about deterministic functional programs.

First, for the sake of concreteness, Chapter 2 fixes the syntax, types and operational semantics of a deterministic functional language for which the relational theory will be developed. The operational semantics is call-by-value and is given both as a big-step evaluation relation and a small-step transition relation. The presentation of the two is combined via a notion of primitive reductions—this factorisation simplifies the relational proofs about the operational semantics in later chapters. Chapter 2 also defines notions of term contexts.

In Chapter 3 the algebra of term relations is developed. Several useful relational constructions to be used in the sequel are introduced, including Howe's congruence candidate relation. The relational theory is based on simple inductive definitions and proofs by induction on derivations, avoiding unwieldy notions of syntactic occurrences and explicit term contexts.

Relations are used in Chapter 4 for developing the theory of contextual equivalence for the deterministic language. Novel proofs of the basic laws, induction rules, continuity properties, and the CIU Theorem are presented together with proof rules for simulation up to context. Finally, operational extensionality is proved by Howe's method.

**Part II** extends the results from Chapter 4 to different nondeterministic extensions of the functional language in Chapter 2.

Chapter 5 defines the syntax and operational semantics of erratic choice and countable choice combinators. The evaluation semantics comes in two flavours: a may evaluation relation which specifies possible outcomes and a must relation which relates a program with its set of outcomes if and only if all the program's computation paths terminate.

In Chapter 6 the may and must modalities give rise to two different nondeterministic generalisations of contextual equivalence. Most of the deterministic theory is shown to carry over in appropriately modified form. The failure of  $\omega$ -continuity in the presence of countable nondeterminism turns up for the must modality only and is addressed by a novel transfinite syntactic continuity principle.

Chapter 7 investigates lower and upper similarity preorders, the natural nondeterministic generalisations of applicative similarity. They turn out to differ subtly from the corresponding contextual approximation preorders associated with the contextual equivalences for nondeterminism. However, they are shown to be pre-congruences, by suitable adaptations of Howe's method, and are thus sound approximations to the contextual approximation preorders.

In the last chapter of the dissertation, Chapter 8, parts of the theory for nondeterminism is extended to ambiguous choice and thus a form of fairness. Much of the theory in the preceding chapters breaks down in the presence of ambiguous choice, including the CIU Theorem, operational extensionality, and induction and continuity principles. Nonetheless, Chapter 8 successfully establishes several useful results, in particular, a soundness property for the reductions of the operational semantics. Moreover, an improvement simulation preorder is introduced, shown to be a pre-congruence, and, hence, included in contextual approximation. This result provides us with co-induction proof rules.

A symbol index is provided on page 125.

# Part I

# Determinism

# Chapter 2

# A Deterministic Functional Language

For the sake of concreteness, the relational theory of the dissertation will be developed for a specific typed functional language. This chapter introduces the sequential language, essentially Plotkin's FPC (Plotkin 1985) or a monomorphically typed fragment of Standard ML (Milner, Tofte, and Harper 1990), which is the subject of study in the first part of the dissertation. In the second part, various nondeterministic extensions to the language are studied.

The syntax and the type system of the language are fairly standard. For convenience, a "reduced" syntax is chosen where expressions are restricted to values in certain syntactic positions, but the reduced syntax is easily inter-translatable with the usual unrestricted syntax of FPC. The types include function spaces, products, and recursive sum types.

The operational semantics is call-by-value and is given both as a big-step evaluation relation and a small-step transition relation. The presentation of the two is combined via a primitive reduction relation which forms the core of both semantic relations.

The chapter ends with definitions of term contexts, that is, extensions of the syntax with "holes", which are useful for comparison with the relational approach in the subsequent chapters.

## 2.1 Syntax

The syntax of expressions and values is defined by the grammar in Table 2.1. We operate with a *reduced syntax* where occurrences of expressions in many syntactic positions are restricted to be values. Values are a syntactic subcategory of expressions and include functions, tuples

Table 2.1: Syntax

of values and injections of values into sums. We let x, y, z, f, g, h range over an infinite set of variables. As the language is call-by-value, variables range over values. Therefore variables are themselves syntactically categorised as values. There is a primitive let construct which is the only means of sequencing expressions. The reduced syntax is important for the simplicity and uniformity of our operational arguments later on, but otherwise the restriction is immaterial: it is easy to map the richer syntax of FPC or Standard ML into reduced form by explicitly sequencing subexpressions in let bindings. Our presentation of the syntax is inspired by Moggi's computational  $\lambda$ -calculus (Moggi 1989) and the distinction between values and computations in his monadic meta-language (Moggi 1991); see Sabry and Wadler (1996).

The scope of  $\lambda$ , let, and case extends as far to the right as possible, e.g., the term  $\lambda x. x v$  parses as  $\lambda x. (x v)$ .

In let x = b in a and  $\lambda x. a, x$  is bound in a. In case u of  $\operatorname{inj}_1 x_1. a_1 [] \dots [] \operatorname{inj}_n x_n. a_n, x_i$  is bound in  $a_i$ , for  $i \in 1..n$ . In general, in phrases of the form x. a, x is bound in a. Expressions are identified up to  $\alpha$ -renaming of bound variables.

We use the notation  $\vec{a}, \vec{u}, \vec{x}, \ldots$  for finite lists of expressions, values, variables, etc. In the case of variables, the variables  $x_1 \ldots x_n$  in  $\vec{x}$  are always assumed to be pairwise distinct.

 $Exp_{\vec{x}}$  and  $Val_{\vec{x}}$  are the sets of expressions and values, respectively, with free variables contained in  $\vec{x}$ .  $Exp_0$  and  $Val_0$  are the sets of closed expressions and values, i.e., those with no free variables. Notice that  $Val_{\vec{x}} \subseteq Exp_{\vec{x}}$  and  $Val_0 \subseteq Exp_0$ .

Let a[u/x] denote the result of capture-free substitution of the value u for all free occurrences of the variable x in a, and let  $a[\vec{u}/\vec{x}]$  be the result of simultaneous, capture-free substitution of values  $\vec{u} = u_1 \dots u_n$  for variables  $\vec{x} = x_1 \dots x_n$  in a. If  $\vec{x}$  are not free in  $\vec{u}$ then  $a[\vec{u}/\vec{x}] = a[u_1/x_1] \dots [u_n/x_n]$ . (See Stoughton (1988) for a precise definition of simultaneous substitution.)

#### 2.1.1 Examples

When no confusion arises, we shall sometimes write expressions in value positions as abbreviation for the appropriate let-construction, e.g., for any non-value a,

$$\begin{array}{rcl} \mathbf{case} \ a \ \mathbf{of} \ \langle \vec{x} \rangle. \ b & \stackrel{\mathrm{def}}{=} & \mathbf{let} \ x = a \ \mathbf{in} \ \mathbf{case} \ x \ \mathbf{of} \ \langle \vec{x} \rangle. \ b \\ & a \ v & \stackrel{\mathrm{def}}{=} & \mathbf{let} \ x = a \ \mathbf{in} \ x \ v \end{array}$$

where x does not occur free in b and v. In accordance with the latter abbreviation we let function application associate to the left,  $u v_1 v_2 = (u v_1) v_2$ .

Sequential composition can be encoded using let,

$$(a;b) \stackrel{\text{def}}{=} \mathbf{let} \ x = a \ \mathbf{in} \ b$$

where x is not free in b.

Here is a list of some other useful abbreviations that we shall use.

$$\begin{array}{rcl} \mathbf{true} & \stackrel{\mathrm{def}}{=} & \mathbf{inj}_1 \langle \rangle \\ \mathbf{false} & \stackrel{\mathrm{def}}{=} & \mathbf{inj}_2 \langle \rangle \\ \mathbf{0} & \stackrel{\mathrm{def}}{=} & \mathbf{inj}_1 \langle \rangle \\ \mathbf{succ} \, u & \stackrel{\mathrm{def}}{=} & \mathbf{inj}_2 \, u \\ \mathbf{nil} & \stackrel{\mathrm{def}}{=} & \mathbf{inj}_1 \langle \rangle \\ \mathbf{ons} \langle u, v \rangle & \stackrel{\mathrm{def}}{=} & \mathbf{inj}_2 \langle u, v \rangle \end{array}$$

(true, 0, and nil are identical but they will be given different types in §2.3.)

1 0

C

Function bodies and the branches of case expressions for sums are of the form x.a. We shall use the abbreviation  $\langle x_1, \ldots, x_n \rangle.a$  for  $x.(\text{case } x \text{ of } \langle x_1, \ldots, x_n \rangle.a)$  when x is not free in a. Then we can construct a conditional expression as

if a then 
$$b_1$$
 else  $b_2 \stackrel{\text{def}}{=} \operatorname{case} a$  of  $\operatorname{inj}_1\langle \rangle . b_1 \| \operatorname{inj}_2\langle \rangle . b_2$ 

and we can express uncurried functions concisely, e.g.,  $\lambda \langle x, y \rangle$ .  $\langle y, x \rangle$  swaps the components of a pair. We can write case-splits on natural numbers and lists as follows,

```
case u of 0.a \parallel \operatorname{succ}(x).b
```

case u of nil. a  $\| \cos\langle x, y \rangle$ . b

For unary sums we write inj in place of  $inj_1$  and we use the abbreviation

$$\lambda \operatorname{inj} x. a \stackrel{\text{def}}{=} \lambda x'. \operatorname{case} x' \operatorname{of} \operatorname{inj} x. a$$

Unary sums become important in §2.3 when we want to type recursive functions because sum types are recursive. For example, if we write a diverging expression  $\Omega$  as

 $\boldsymbol{\Omega} \stackrel{\text{def}}{=} (\lambda \mathbf{inj} f. f(\mathbf{inj} f)) (\mathbf{inj} \lambda \mathbf{inj} f. f(\mathbf{inj} f))$ 

it will be well typed. Similarly, the following call-by-value version of Curry's fixed point combinator will be well typed in §2.3.

 $\mathbf{Y} \stackrel{\text{def}}{=} \lambda g. \operatorname{fix}[g]$ 

where

 $\mathsf{fix}[u] \stackrel{\text{def}}{=} (\lambda \mathbf{inj} \ y. \ u \ (\lambda x. \ y \ (\mathbf{inj} \ y) \ x)) \ (\mathbf{inj} \ \lambda \mathbf{inj} \ y. \ u \ (\lambda x. \ y \ (\mathbf{inj} \ y) \ x))$ 

# 2.2 Operational semantics

In this section we give a joint presentation of two conventional forms of operational semantics, a big-step evaluation semantics and a small-step transition semantics. First, a so-called primitive reduction relation formalises the beta-reductions for all non-value expression constructors other than let. This primitive reduction relation forms the core of both the evaluation relation and the transition relation. Each of these can be defined by just three rules, given the primitive reduction relation. This will be an advantage throughout the dissertation as it simplifies case analyses of derivations of evaluations and transitions.

(Redex apply) 
$$(\lambda x. a) v \to a[v/x]$$

(Redex case 
$$\times$$
) case  $\langle \vec{u} \rangle$  of  $\langle \vec{x} \rangle . a \to a[\vec{u}/\vec{x}]$ 

 $(\text{Redex case } +) \quad \textbf{case inj}_i \, u \, \, \textbf{of inj}_1 \, x_1. \, a_1 \, [\hspace{-1.5pt}] \, \dots \, [\hspace{-1.5pt}] \, \textbf{inj}_n \, x_n. \, a_n \rightarrow a_i [\hspace{-1.5pt}u/\hspace{-1.5pt}/ x_i], \hspace{1.5pt} \text{if} \hspace{1.5pt} i \in 1..n$ 

Table 2.2: Primitive reduction relation

(Eval value) 
$$v \rightsquigarrow v$$
  
(Eval redex)  $\frac{b \rightsquigarrow v}{a \rightsquigarrow v}$  if  $a \rightarrow b$   
(Eval let)  $\frac{a \rightsquigarrow u \quad b[u/x] \rightsquigarrow v}{\mathbf{let} \ x = a \ \mathbf{in} \ b \rightsquigarrow v}$ 

## Table 2.3: Evaluation relation

(Trans redex) 
$$a \rightarrow b$$
, if  $a \rightarrow b$   
(Trans let beta) **let**  $x = u$  **in**  $b \rightarrow b[u/x]$ 

(Trans let left) 
$$\frac{a \rightarrowtail a'}{\text{let } x = a \text{ in } b \rightarrowtail \text{let } x = a' \text{ in } b}$$

## Table 2.4: Transition relation

#### 2.2.1 Primitive reduction

The primitive reduction relation,  $\rightarrow$ , relates closed expressions as defined by the rules in Table 2.2. It is deterministic:

**Lemma 2.2.2** If  $a \rightarrow b$  and  $a \rightarrow b'$  then b = b'.

**Proof** By inspection of the primitive reduction rules.

Actually, the primitive reduction relation is a fully fledged transition relation for expressions without occurrences of let. (This sublanguage is not entirely uninteresting as it is the image of a CPS transformation of the full language; however, we shall not pursue this topic.) For instance,  $\Omega$  is defined without use of let, so all its transitions are primitive reductions. Indeed, there is an infinite sequence of reductions starting from  $\Omega$  as one would expect:

 $\Omega \rightarrow \mathbf{case} \ (\mathbf{inj} \ \lambda \mathbf{inj} \ f. \ f \ (\mathbf{inj} \ f)) \ \mathbf{of} \ \mathbf{inj} \ f. \ f \ (\mathbf{inj} \ f) \ \rightarrow \ \Omega \ \rightarrow \ \cdots$ 

The motivation for introducing the primitive reduction relation is twofold. Firstly, it saves some duplication in the definitions of the evaluation and transition relations below. Secondly, the defining rules are *structural*: reduction is determined by the outermost syntactic structure of expressions in a regular way; this regularity is captured in the next chapter,  $\S3.10$ , in a relational formulation which simplifies many relational proofs in the remainder of the dissertation. We discuss these points in more depth in  $\S2.2.8$  after the definitions of the evaluation and transition relations.

#### 2.2.3 Evaluation

The first form of operational semantics is an evaluation relation  $\rightsquigarrow$  between closed expressions and values,  $a \rightsquigarrow v$ ; we say that a evaluates to v and that v is the outcome of a. The evaluation relation is defined inductively as the least relation closed under the three rules in Table 2.3. The definition is quite concise because all other expression constructs than **let** are processed by the primitive reduction relation.

Evaluation is deterministic:

**Proposition 2.2.4** If  $a \rightsquigarrow v$  and  $a \rightsquigarrow v'$  then v = v'.

**Proof** By induction on the derivation of  $a \rightsquigarrow v$ , using Lemma 2.2.2.

An expression a terminates if  $a \rightsquigarrow v$  for some v. Otherwise a diverges. For instance, any derivation of an evaluation  $\Omega \rightsquigarrow v$  must be of the following form derived from (Eval redex) and the sequence of primitive reductions from  $\Omega$  mentioned at the end of §2.2.1.

 $\begin{array}{c} \vdots \\ \overline{\boldsymbol{\Omega} \rightsquigarrow \boldsymbol{v}} \\ \hline \mathbf{case} \; (\mathbf{inj} \; \lambda \mathbf{inj} \; f. \; f \; (\mathbf{inj} \; f)) \; \mathbf{of} \; \mathbf{inj} \; f. \; f \; (\mathbf{inj} \; f) \rightsquigarrow \boldsymbol{v} \\ \mathbf{\Omega} \rightsquigarrow \boldsymbol{v} \end{array}$ 

Because of the recurrence of  $\Omega \rightsquigarrow v$  as a premise in the derivation, there cannot be any finite derivation tree for  $\Omega \rightsquigarrow v$ . Therefore  $\Omega$  diverges.

#### 2.2.5 Transitions

The rules in Table 2.4 define a transition relation,  $\rightarrow$ , between closed expressions. The rule (Trans redex) includes the primitive reduction relation,  $\rightarrow$ , in the transition relation; (Trans let beta) is a beta reduction rule for let; and (Trans let left) specifies the reduction strategy.

A transition sequence is a sequence of closed expressions,  $a_0a_1...$ , such that  $a_0 \rightarrow a_1 \rightarrow ...$  A terminating transition sequence is a finite transition sequence that ends with a value.

There is the following relationship between evaluation and terminating transition sequences ( $\rightarrow^*$  denotes the reflexive transitive closure of  $\rightarrow$ ).

**Proposition 2.2.6**  $a \rightsquigarrow u$  if and only if  $a \rightarrowtail^* u$ .

Consequently, Proposition 2.2.4 asserts that terminal values of terminating transition sequences are unique.

The transition relation is deterministic:

**Proposition 2.2.7** If  $a \rightarrow b$  and  $a \rightarrow b'$  then b = b'.

**Proof** By induction on the derivation of  $a \rightarrow b$ , using Lemma 2.2.2.

As an example of transitions let us look at the operational behaviour of  $\mathbf{Y}$  and fix.

$$\mathbf{Y} u \rightarrow \mathsf{fix}[u] \rightarrow^2 u (\lambda x. \mathsf{fix}[u] x)$$

 $\lambda x$ . fix [u] x is a fixed point of u to the extent that

$$(\lambda x.\operatorname{fix}[u] x) v \rightarrow^{3} u (\lambda x.\operatorname{fix}[u] x) v$$

so that

$$(\lambda x.\operatorname{fix}[u] x) v \rightarrowtail^* v' \quad \operatorname{iff} \quad u(\lambda x.\operatorname{fix}[u] x) v \rightarrowtail^* v'$$

for all values v and v'.

#### 2.2.8 Discussion

We mentioned above that the definition of primitive reductions is *structural* in the sense that reduction is determined by the outermost syntactic structure of expressions. The definitions of the evaluation relation and the transition relation are not structural: the evaluation relation may reduce redexes everywhere in a term and the transition relation may rewrite a redex which occurs arbitrarily deep within an evaluation context. It is possible to define a structural transition relation by adopting the  $\beta$  and associativity reduction rules for **let** from Moggi's computational  $\lambda$ -calculus (Moggi 1989),

(Redex let beta) 
$$\operatorname{let} x = u \text{ in } b \to b[u/x]$$

(Redex let assoc) let 
$$x = (\text{let } x' = a' \text{ in } a)$$
 in  $b \to \text{let } x' = a'$  in  $(\text{let } x = a \text{ in } b)$ 

However, there is a subtle way in which these two reduction rules are "less structural" than primitive reductions: the primitive reductions are all determined by the outermost syntactic

non-value constructor and, in some cases, by the value constructors of immediate value subterms; they are all "uniform" with respect to all proper subterms which do not occur in value positions on the left hand side. It is a bit tricky to make this notion of uniformity precise in an explicit way but we shall formalise it in terms of relations in the next chapter, §3.10. For now, we just remark how the (Redex let beta) and (Redex let assoc) reduction rules are not uniform in the left expression subterm: the form of the left subexpression—value or **let**—determines the outcome and, moreover, (Redex let assoc) takes apart the left subexpression.

The bottom line is that the strong structural property of primitive reductions allows us to establish a precise and general result about primitive reductions in §3.10 which is very useful throughout in the relational reasoning in later chapters.

# 2.3 Types

The type system for the language is monomorphic and includes function spaces, product types and recursive sum types. It differs from FPC only in that sums and recursive types are fused into one type construct for recursive sums as in Standard ML—the difference is insignificant but our choice is convenient because it allows us to use the same expression syntax for introduction and elimination of sums and recursive types.

The syntax for types is given by the grammar (where  $n \ge 0$ ):

$$(Type)$$
  $t ::= t_1 \rightarrow t_2 \mid t_1 \times \ldots \times t_n \mid \mu \chi \cdot t_1 + \ldots + t_n \mid \chi$ 

 $\chi$  ranges over a set of type variables. In  $\mu\chi$ .  $t_1 + \ldots + t_n$ ,  $\chi$  is bound in  $t_1, \ldots, t_n$ . A type is closed if it has no free type variables.  $Type_0$  is the set of closed types. We write  $t[t'/\chi]$  for the substitution of t' for the free occurrences of  $\chi$  in t.

Let unit and void denote the empty product type and empty sum type, respectively,

unit 
$$\stackrel{\text{def}}{=} t_1 \times \ldots \times t_n$$
, when  $n = 0$   
void  $\stackrel{\text{def}}{=} \mu \chi \cdot t_1 + \ldots + t_n$ , when  $n = 0$ 

The notation for unary product types is ambiguous, but we shall never explicitly use unary product types, so a type t should never be read as the unary product type with component type t—we might exclude unary product types altogether as in Standard ML.

A type environment  $\Gamma$  is of the form  $\vec{x} : \vec{t}$  and associates each variable  $x_i$  in  $\vec{x}$  with the corresponding type  $t_i$  in  $\vec{t}$ . We write  $\vec{x} : \vec{t} \vdash a : t$  to mean that  $a \in Exp_{\vec{x}}$  has the closed type t in the type environment  $\vec{x} : \vec{t}$ . We write just a : t if  $\emptyset \vdash a : t$  where  $\emptyset$  denotes the empty type environment. The rules for type assignment are given in Table 2.5. One can check that  $\vec{x} : \vec{t} \vdash a : t$  implies  $a \in Exp_{\vec{x}}$  by induction on the derivation of  $\vec{x} : \vec{t} \vdash a : t$ .

Typings are not unique, e.g., for the identity function we have that  $\lambda x. x: t \rightarrow t$  for every closed type t.

#### **Proposition 2.3.1** Type assignment is closed under

- 1. weakening,  $\Gamma \vdash a : t$  and  $\Gamma \subseteq \Gamma'$  imply  $\Gamma' \vdash a : t$ , where  $\Gamma \subseteq \Gamma'$  means that all variable bindings  $x_i : t_i$  in  $\Gamma$  occur in  $\Gamma'$ , in any order; and
- 2. value substitution,  $\Gamma, x : t \vdash a : t'$  and  $\Gamma \vdash u : t$  imply  $\Gamma \vdash a[u/x] : t'$ .

(Type var) 
$$\Gamma, x: t, \Gamma' \vdash x: t$$

(Type let) 
$$\frac{\Gamma \vdash a_1 : t_1 \quad \Gamma, x : t_1 \vdash a_2 : t_2}{\Gamma \vdash \mathbf{let} \ x = a_1 \ \mathbf{in} \ a_2 : t_2}$$

(Type fun) 
$$\frac{\Gamma, x: t_1 \vdash a: t_2}{\Gamma \vdash \lambda x. a: t_1 \rightharpoonup t_2}$$

(Type apply) 
$$\frac{\Gamma \vdash u : t_1 \rightharpoonup t_2 \quad \Gamma \vdash v : t_1}{\Gamma \vdash u v : t_2}$$

(Type product) 
$$\frac{\Gamma \vdash u_1 : t_1 \quad \dots \quad \Gamma \vdash u_n : t_n}{\Gamma \vdash \langle u_1, \dots, u_n \rangle : t_1 \times \dots \times t_n}$$

(Type case 
$$\times$$
) 
$$\frac{\Gamma \vdash u : t_1 \times \ldots \times t_n \quad \Gamma, x_1 : t_1, \ldots, x_n : t_n \vdash a : t}{\Gamma \vdash \mathbf{case} \ u \ \mathbf{of} \ \langle x_1, \ldots, x_n \rangle. a : t}$$

(Type sum) 
$$\frac{\Gamma \vdash u : t_i[t/\chi]}{\Gamma \vdash \mathbf{inj}_i \, u : t} \quad \text{if } t = \mu \chi. t_1 + \ldots + t_n \text{ and } i \in 1..n$$

$$(\text{Type case } +) \quad \frac{\Gamma \vdash u: t \quad \Gamma, x_1: t_1[t/\chi] \vdash a_1: t' \quad \dots \quad \Gamma, x_n: t_n[t/\chi] \vdash a_n: t'}{\Gamma \vdash \text{case } u \text{ of inj}_1 x_1. a_1 \parallel \dots \parallel \text{inj}_n x_n. a_n: t'} \text{ if } t = \mu \chi. t_1 + \dots + t_n \text{ and } n \ge 1$$

Table 2.5: Type system

## 2.3.2 Examples

Some useful type abbreviations are:

bool 
$$\stackrel{\text{def}}{=} \mu \chi$$
. unit + unit  
nat  $\stackrel{\text{def}}{=} \mu \chi$ . unit +  $\chi$   
t list  $\stackrel{\text{def}}{=} \mu \chi$ . unit +  $(t \times \chi)$ 

It is easy to check that the derived notation in §2.1 for Boolean constants, conditional expressions, natural numbers, and lists enjoys the expected typing properties.

Recall the combinators  $\Omega$  and **Y** from §2.1.

$$\begin{split} \mathbf{\Omega} &\stackrel{\text{def}}{=} & (\lambda \mathbf{inj} \ f. \ f(\mathbf{inj} \ f)) \ (\mathbf{inj} \ \lambda \mathbf{inj} \ f. \ f(\mathbf{inj} \ f)) \\ \mathbf{Y} &\stackrel{\text{def}}{=} & \lambda g. \ \text{fix}[g] \\ \text{fix}[u] &\stackrel{\text{def}}{=} & (\lambda \mathbf{inj} \ y. \ u \ (\lambda x. \ y \ (\mathbf{inj} \ y) \ x)) \ (\mathbf{inj} \ \lambda \mathbf{inj} \ y. \ u \ (\lambda x. \ y \ (\mathbf{inj} \ y) \ x)) \end{split}$$

The diverging expression  $\Omega$  can be assigned any type t, and  $g : t \rightarrow t \vdash \operatorname{fix}[g] : t$  and  $\mathbf{Y} : (t \rightarrow t) \rightarrow t$  hold for every function type t. For illustration, let us work out the derivation of the type judgment  $\Omega$  : unit. We show the derivation tree for  $\lambda \operatorname{inj} f \cdot f(\operatorname{inj} f) : \mu \rightarrow \operatorname{unit}$ , where  $\mu \stackrel{\text{def}}{=} \mu \chi \cdot \chi \rightarrow \operatorname{unit}$ , from which the reader is invited to derive that  $\Omega$  : unit.

$$\begin{array}{c} \underline{f:\mu \rightharpoonup \mathrm{unit} \vdash f:\mu \rightharpoonup \mathrm{unit}} \\ \underline{f:\mu \rightharpoonup \mathrm{unit} \vdash f:\mu \rightharpoonup \mathrm{unit}} \\ \hline f:\mu \rightharpoonup \mathrm{unit} \vdash f(\mathbf{inj}\ f):\mathrm{unit}} \\ \hline \frac{f:\mu \vdash \mathrm{case}\ f'\ \mathrm{of}\ \mathrm{inj}\ f.\ f(\mathrm{inj}\ f):\mathrm{unit}}{\lambda \mathrm{inj}\ f.\ f(\mathrm{inj}\ f):\mu \rightarrow \mathrm{unit}} \end{array}$$

If we consider the simpler combinators

$$\begin{split} \mathbf{\Omega}' &\stackrel{\text{def}}{=} & (\lambda f.\,f\,f)\,(\lambda f.\,f\,f) \\ \mathbf{Y}' &\stackrel{\text{def}}{=} & \lambda g.\,(\lambda y.\,g\,(\lambda x.\,y\,y\,x))\,(\lambda y.\,g\,(\lambda x.\,y\,y\,x)) \end{split}$$

they cannot be assigned any types because in the self applications f f and y y the operator and the operand must have the same type which makes it impossible to instantiate the typing rule (Type apply) for application.

#### 2.3.3 Soundness

The type system satisfies type preservation (subject reduction) and type soundness (well typed programs don't go wrong) properties with respect to the evaluation and transition relations from §2.2. We outline the proofs because they illustrate the advantage of specifying the two semantic relations in terms of the primitive reduction relation: the common parts of the two proofs can be formulated as the following type preservation lemma for the primitive reduction relation.

**Lemma 2.3.4** If a : t and  $a \to b$  then b : t.

**Proof** By inspection of the primitive reduction rules, Table 2.2, using Proposition 2.3.1.  $\Box$ 

**Proposition 2.3.5** If a : t and  $a \rightsquigarrow v$  then v : t.

**Proof** By induction on the derivation of  $a \rightsquigarrow v$ , using Lemma 2.3.4 and Proposition 2.3.1.

**Proposition 2.3.6** If a : t and  $a \rightarrow b$  then b : t.

**Proof** By induction on the derivation of  $a \rightarrow v$ , using Lemma 2.3.4 and Proposition 2.3.1.

The transition relation also satisfies a type soundness property. This can be established via the following property of the primitive reduction relation.

**Lemma 2.3.7** If a:t, either a is a value or a is a let expression or  $a \rightarrow b$  for some b.

**Proof** By inspection of the derivation of a: t.

The type soundness property is that well typed programs don't go wrong, that is, a well typed program can always do a transition unless it is already a value:

**Proposition 2.3.8** If a: t, either a is a value or  $a \rightarrow b$  for some b.

**Proof** By induction on the derivation of a: t, using Lemma 2.3.7.

## 2.4 Contexts

This chapter closes with extensions of the syntax and types for expressions to term contexts. Intuitively, a context is a term containing holes that may be filled by other terms. Contexts will play an important role in formulating and proving the results in the next chapters. Although we shall generally avoid working with explicit representations of contexts, we now give a formal definition of contexts.

#### 2.4.1 Evaluation contexts

A simple form of contexts are *evaluation contexts* (Felleisen and Friedman 1987). An evaluation context, E, is a closed expression with a hole,  $\bullet$ , at redex position.

$$(EvCtx)$$
  $E$  ::= • | let  $x = E$  in  $b$ ,  $b \in Exp_{a}$ 

We write  $E[\![a]\!]$  for the expression obtained by filling in a closed expression *a* for the occurrence of  $\bullet$  in *E*.

The definition of evaluation contexts embodies the call-by-value evaluation strategy of the language: a let expression, let x = a in b, first evaluates a. This is reflected by the facts that evaluation factors through evaluation contexts,

$$E\llbracket a \rrbracket \rightsquigarrow v \quad \text{iff} \quad \exists u. \ a \rightsquigarrow u \ \& \ E\llbracket u \rrbracket \rightsquigarrow v \tag{2.1}$$

and transitions are closed under evaluation contexts, in the sense that

$$a \mapsto b$$
 implies  $E[\![a]\!] \mapsto E[\![b]\!]$  (2.2)

for all evaluation contexts E.

Evaluation contexts are particularly simple examples of contexts because they are closed and the holes do not occur under variable binders.

#### 2.4.2 Abstractions and variable capturing contexts

We now proceed to consider variable capturing contexts. In order to give a well behaved definition of these, we introduce some notation which will also serve as the basis of our formalisation of relations between expressions in Chapter 3.

We first introduce a new meta-syntactic category of *abstractions*. (They are meta-syntax because they do not occur in the ordinary syntax of the language.) A term of the form  $(\vec{x})a$ is an abstraction of  $a \in Exp_{\vec{x}}$ ; the  $(\vec{x})$  prefix is a binder and  $\vec{x}$  is subject to  $\alpha$ -renaming. Correspondingly, we introduce *abstraction types* of the form  $(\vec{t})t$ . An abstraction  $(\vec{x})a$  has abstraction type  $(\vec{t})t$ , written  $(\vec{x})a: (\vec{t})t$ , if  $\vec{x}: \vec{t} \vdash a: t$ .

We already saw an example of an abstraction,

$$\mathsf{fix} = (g) (\lambda \mathbf{inj} y. g (\lambda x. y (\mathbf{inj} y) x)) (\mathbf{inj} \lambda \mathbf{inj} y. g (\lambda x. y (\mathbf{inj} y) x))$$

and fix :  $(t \rightarrow t) t$  for all function types t.

We identify every closed expression a and type t with the 0-ary abstraction ()a and the 0-ary abstraction type ()t.

A variable capturing context C is an expression with occurrences of pseudo-expressions  $\xi[\vec{x}]$  where  $\xi$  is an abstraction variable,  $\xi : (\vec{t})t$ ; the pseudo-expression  $\xi[\vec{x}]$  plays the role of an expression of type t and with free variables  $\vec{x} : \vec{t}$ . Let  $\phi$  range over abstractions and  $\theta$  range over abstraction types. If C has occurrences of abstraction variables  $\vec{\xi} : \vec{\theta}$  and we have abstractions  $\vec{\phi} : \vec{\theta}$ , then  $C[[\vec{\phi}/\vec{\xi}]]$  denotes the expression obtained by filling  $\phi_i[\vec{x}]$  in for every pseudo-expression  $\xi_i[\vec{x}]$  in C.

The hole • from §2.4.1 is a distinguished 0-ary abstraction variable; we reserve the notation C[a] as abbreviation for  $C[a] \cdot [a]$  whenever C is a context with • as the only abstraction variable.

## 2.4.3 Substituting contexts

A more general class of contexts, which we call *substituting contexts*, is obtained if we allow pseudo-expressions of the form  $\xi[\vec{U}]$  where, recursively,  $U_1 \dots U_n$  are substituting value contexts (substituting contexts with a value constructor as outermost syntactic constructor).

When an abstraction  $(\vec{x})a$  is filled in for an abstraction variable  $\xi$  in a substituting context A, written,  $A[[(\vec{x})a/\xi]]$ , every occurrence of pseudo-expressions of the form  $\xi[\vec{U}]$  in A is replaced by  $a[\vec{U}[[(\vec{x})a/\xi]]/\vec{x}]$ . So a substituting context A may substitute arbitrary values for the free variables  $\vec{x}$  of a when  $(\vec{x})a$  is filled in for an abstraction variable  $\xi$  in A, and these values may again have occurrences of  $\xi$  which are also filled by  $(\vec{x})a$ . This is more general than variable capturing contexts C which only bind or rename the variables  $\vec{x}$  when  $(\vec{x})a$  is filled in for  $\xi$  in C.

Occasionally, typing judgments of the form  $\vec{\xi} : \vec{\theta}, \Gamma \vdash A : t$  will be used to mean that A would have type t in the type environment  $\Gamma$  were every  $\xi_i$  in A an abstraction of abstraction type  $\theta_i$ ; so  $\Gamma \vdash A[\vec{\phi},\vec{\xi}]$  : t holds whenever  $\vec{\phi} : \vec{\theta}$ .

The definition of substituting contexts reflects the call-by-value nature of the language in the way that the arguments  $\vec{U}$  of pseudo-expressions  $\xi[\vec{U}]$  are constrained to produce values upon instantiation of their abstraction variables. Otherwise substituting contexts and abstraction variables correspond to 'meta-terms' and 'meta-variables' in Klop, van Oostrom, and van Raamsdonk (1993) or 'extended expressions' and 'function variables' in Pitts (1994b); see also Martin-Löf's theory of arities (Nordström, Petersson, and Smith 1990), the higherorder syntax of Pfenning and Elliott (1988), and Talcott's binding structures (Talcott 1993; Agha, Mason, Smith, and Talcott 1997; Mason 1996). Pitts (1994b) advocates a form of substituting contexts as a generalised notion of contexts in place of conventional variable capturing contexts because the latter cannot be identified up to  $\alpha$ -renaming of bound variables. However, the elaborate definition in  $\S2.4.2$  of variable capturing contexts and hole filling in terms of abstraction variables and abstractions does not suffer from this deficiency. Nevertheless, the notion of substituting contexts, in some form or another, turns up in many syntactic operational arguments when one wants to trace the structure of composite terms across  $\beta$ -reductions that perform syntactic substitutions; see, e.g., Klop, van Oostrom, and van Raamsdonk (1993), Talcott (1998), and Sands (1998a). In the relational reasoning of Chapters 4, 6 and 8 this phenomenon appears frequently, albeit implicitly in terms of "substituting context closure" of relations. In general, the relational theory in the remainder of the dissertation eschews explicit term contexts by operating with more convenient relational representations. But it is instructive to compare the latter with the notions of variable capturing contexts and substituting contexts as we go along.

# Chapter 3

# Relations

This chapter introduces our notation for relations on terms and defines various operations on relations. Compatible refinement and various forms of context closure are of particular importance. Their precise definitions are key to the relational proofs in later sections. The relations and relational operators that we introduce satisfy a rich collection of useful properties connected to term substitution, variable capturing contexts, and substituting contexts. These elements are specific to variable-binding terms and take us beyond the classical calculus of relations of de Morgan and Pierce (cf. Tarski 1941; Pratt 1992), and the term relations considered in most of the literature about algebraic specifications (Wirsing 1990) and term rewriting (Klop 1992). Rather, our relational algebra builds on the syntactic methods of Howe (1996), Sangiorgi (1994), Gordon (1994, 1995a), and Pitts (1995) for reasoning about bisimulation in process calculi and functional calculi.

Relations and terms are quite simple entities and an effort will be made to develop the theory with mathematical rigour but without difficult proofs. Everything is based on simple inductive definitions and proofs by induction. The proof steps are kept simple by avoiding unwieldy notions of syntactic occurrences and explicit term contexts. This should make the proof steps simpler and it appears to encourage a high degree of mathematical rigour and attention to detail, whereas arguments about explicit contexts more often relies on (sometimes deceptive) intuitions. In many cases we argue algebraically in a "point-free" fashion, e.g., we do induction by checking that a relation is a pre-fixed point of a monotone operator by means of algebraic laws of the relational operators, instead of "pointwise" proofs where the terms being related are made explicit and inductions are on the derivation of relationships between terms. Our choice between the two is motivated by the fact that the point-free style is a good benchmark for the pragmatic "completeness" of the relational algebra.

For the sake of concreteness, the theory given below is developed for terms of the particular functional language from the preceding chapter, and the treatment of values and substitutions are tailored to the call-by-value nature of the language. It is relatively straightforward to see how the operations extend to other language constructs and general term substitutions. We discuss these and other extensions of the framework at the end of the chapter. Most of the theory is developed for binary relations but at the end of the chapter §3.9 outlines extensions to relations of higher arity.

The exposition introduces a certain amount of relational concepts and notation, and the reader may find the symbol index on page 125 helpful for keeping track of them all. It is not easy to come up with lucid, concise, and consistent terminology and notation. Several new relational operations are introduced but for existing notions an attempt has been made to adhere to existing notation in the literature on operationally-based term relations, e.g., Howe (1996), Gordon (1995c) and Pitts (1997a). However, one finds many conflicting notations for all the relational constants and operators, in particular, none of our notation coincides with that used in the mathematical literature on the calculus of binary relations (see Pratt 1992).

## 3.1 Binary relations

A binary relation  $\mathcal{R}$  is a set of pairs,  $\mathcal{R} \subseteq \mathcal{D} \times \mathcal{D}'$ , where  $\mathcal{D}$  and  $\mathcal{D}'$  are the domain and the co-domain of  $\mathcal{R}$ , respectively. We use infix notation,  $q \mathcal{R} q'$ , to mean  $(q,q') \in \mathcal{R}$ . The *reciprocal* relation of  $\mathcal{R}$ , written  $\mathcal{R}^{\text{op}}$ , is defined by  $q \mathcal{R}^{\text{op}} q' \stackrel{\text{def}}{\Leftrightarrow} q' \mathcal{R} q$ . Relation composition is written by juxtaposition,  $q \mathcal{RS} q' \stackrel{\text{def}}{\Leftrightarrow} \exists q'' \land q \mathcal{R} q'' \land q'' \mathcal{S} q'$ .

Suppose  $\mathcal{R}$  has identical domain and co-domain. Then  $\mathcal{R}$  is *reflexive* if  $q \mathcal{R} q$  for all q in the domain. We let  $\mathcal{R}^n$  denote the *n*-fold composition of  $\mathcal{R}$  with itself,  $\mathcal{R}^+ \stackrel{\text{def}}{=} \bigcup_{n \ge 1} \mathcal{R}^n$  is its transitive closure, and  $\mathcal{R}^* \stackrel{\text{def}}{=} \bigcup_{n \ge 0} \mathcal{R}^n$  is its reflexive transitive closure.  $\mathcal{R}$  is *transitive* if  $\mathcal{R}\mathcal{R} \subseteq \mathcal{R}$  or, equivalently, if  $\mathcal{R}^+ \subseteq \mathcal{R}$ . We call  $\mathcal{R}$  symmetric if  $q \mathcal{R} q'$  whenever  $q' \mathcal{R} q$ . The symmetrisation of any relation  $\mathcal{R}$  is the largest symmetric relation contained in  $\mathcal{R}$ , i.e.,  $\mathcal{R} \cap \mathcal{R}^{\text{op}}$  (we choose not to call it "symmetric closure" because the latter terminology might be taken to mean the smallest symmetric relation which contains  $\mathcal{R}$ , i.e.,  $\mathcal{R} \cup \mathcal{R}^{\text{op}}$ ).

## **3.2** Open and closed relations

Let  $Rel|_{(\vec{t})t}$  be the universal relation on abstractions of type  $(\vec{t})t$ ,

$$Rel|_{(\vec{t})t} = \left\{ \left( (\vec{x})a, (\vec{x})a' \right) \mid \vec{x}: \vec{t} \vdash a: t \text{ and } \vec{x}: \vec{t} \vdash a': t \right\}$$

*Rel* is the abstraction-type indexed family of all these:

$$Rel = \{Rel|_{(\vec{t})t}\}_{(\vec{t})t \in AbstrType}$$

The usual operations on sets and relations extend pointwise to abstraction-type indexed families of relations, e.g.,  $X \subseteq Rel$  if X is an abstraction type indexed family of relations and  $X|_{(\vec{t})t} \subseteq Rel|_{(\vec{t})t}$  for every abstraction type  $(\vec{t})t$ . Let  $\emptyset$  denote the family of empty relations and Id the family of identity relations. We write  $\vec{x} : \vec{t} \vdash a X a' : t$  or  $(\vec{x})a X (\vec{x})a' : (\vec{t})t$  whenever  $((\vec{x})a, (\vec{x})a') \in X|_{(\vec{t})t}$ . When  $(\vec{x})a$  and  $(\vec{x})a'$  are 0-ary (i.e., they are just the closed expressions a and a') we write a X a' : t.

An open relation is any  $R \subseteq Rel$  which is closed under weakening (cf. Proposition 2.3.1):

$$\Gamma \vdash a \ R \ a' : t \quad \text{and} \quad \Gamma \subseteq \Gamma' \quad \text{imply} \quad \Gamma' \vdash a \ R \ a' : t$$

$$(3.1)$$

For instance, Rel,  $\emptyset$  and Id are open relations. Let  $REL \subseteq \mathcal{P}(Rel)$  be the set of all open relations, ranged over by R, S. We call open relations  $R \in REL$  reflexive if  $Id \subseteq R$ .

A closed relation R is an open relation which satisfies that

$$\Gamma \vdash a \mathbf{R} a' : t \quad \text{iff} \quad \vdash a \mathbf{R} a' : t$$

for all  $\Gamma$ , a, a', and t. Hence, closed relations only relate closed expressions.  $\emptyset$  is a closed relation. Let  $REL_0 \subseteq \mathcal{P}(Rel)$  be the set of all closed relations, ranged over by  $\mathbf{R}, \mathbf{S}$ .

(Match var) 
$$\Gamma, x: t, \Gamma' \vdash x \overline{R} x: t$$

(Match fun) 
$$\frac{\Gamma, x: t_1 \vdash a \ R \ a': t_2}{\Gamma \vdash \lambda x. \ a \ \overline{R} \ \lambda x. \ a': t_1 \rightharpoonup t_2}$$

$$(\text{Match product}) \quad \frac{\Gamma \vdash u_1 \ \overline{R} \ u'_1 : t_1 \quad \dots \quad \Gamma \vdash u_n \ \overline{R} \ u'_n : t_n}{\Gamma \vdash \langle u_1, \dots, u_n \rangle \ \overline{R} \ \langle u'_1, \dots, u'_n \rangle : t_1 \times \dots \times t_n}$$

(Match sum) 
$$\frac{\Gamma \vdash u \ R \ u' : t_i[t/\chi]}{\Gamma \vdash \mathbf{inj}_i \ u \ \overline{R} \ \mathbf{inj}_i \ u' : t} \quad \text{if} \ t = \mu\chi. \ t_1 + \ldots + t_n \ \text{and} \ i \in 1..n$$

Table 3.1: Matching values

REL and  $REL_0$  are each closed under relation composition, intersections and unions and both form complete lattices ordered by subset inclusion.

Because of the type preservation results from  $\S2.3.3$  we shall, whenever it is convenient, tacitly regard the operational relations from  $\S2.2$  as the closed relations given by

| $\Gamma \vdash a \to b: t,$              | if | a:t | and | $a \rightarrow b$     |
|--|----|-----|-----|-----------------------|
| $\Gamma \vdash a \rightsquigarrow v: t,$ | if | a:t | and | $a \leadsto v$        |
| $\Gamma \vdash a \rightarrowtail b: t,$  | if | a:t | and | $a \rightarrowtail b$ |

It is easy to see that these closed relations are closed under weakening. All the operations on open and closed relations that are presented in the sequel preserve closure under weakening, as can easily be checked.

We define a projection  $(\cdot)_0 : REL \to REL_0$  that maps any open relation  $R \in REL$  into the closed relation  $R_0 \in REL_0$  given by

$$\Gamma \vdash a \ S_{\mathbf{0}} \ a' : t \quad \text{iff} \quad a \ S \ a' : t \tag{3.2}$$

It satisfies  $R_0 \subseteq R$  and  $S_0 = S$ , for all  $R \in REL$  and  $S \in REL_0$ , as expected.

# 3.3 Matching values

Open relations are relations between *expressions*. In some cases we are interested in relations on *values* only. Since values are special cases of expressions, every open relation R immediately induces a relation on values by restriction. Nonetheless, we shall use another construction instead, akin to Gordon's 'matching values' (Gordon 1995b), which turns out to be more convenient. For every open relation R, let  $\overline{R} \in REL$  relate 'matching' values built from identical value constructors and with function bodies pairwise related by R. This is defined inductively by the rules in Table 3.1.

The matching values operator is monotone and preserves reflexivity, transitivity, and symmetry. In fact, it commutes with relation composition and reciprocation:

$$\overline{R}\,\overline{S} = \overline{R}\,\overline{S} \tag{3.3}$$

$$\overline{R}^{\rm op} = \overline{R^{\rm op}} \tag{3.4}$$

Let t be a ground type if it is built from products and recursive sums only, e.g., unit, nat and nat list are ground types. For arbitrary open relations R and ground types  $t, \overline{R}|_t$  is the identity relation on values of type t,

$$\Gamma \vdash v \ \overline{R} \ v': t \quad \text{iff} \quad v = v', \quad \text{if} \ \Gamma \vdash v: t \text{ and } \Gamma \vdash v': t \tag{3.5}$$

This follows by induction on the derivation of  $\Gamma \vdash v : t$ .

To motivate the definition of matching values, a few words about the use of relations in later chapters are appropriate. The relations R that we shall be working with are, generally, semantic preorders or equivalences of some kind which satisfy that (closed) values are related if and only if they have the same observable outermost value constructors and the subterms underneath are again related, i.e.,  $\overline{R}$  is the same as R on (closed) values. It is often convenient to anticipate this and refer to  $\overline{R}$ , where this structure is made explicit, rather than to R itself. For instance, we do this in the definition of relation substitution below.

## 3.4 Substitution

For open relations R and S, the relation substitution of S into R, written R[S], is an open relation between expressions obtained by simultaneous substitution of  $\overline{S}$  related values into R related expressions,

$$\frac{\Gamma, \vec{x}: \vec{t} \vdash a \ R \ a': t \quad \Gamma \vdash \vec{u} \ \overline{S} \ \vec{u}': \vec{t}}{\Gamma \vdash a[\vec{u}/\vec{x}] \ R[S] \ a'[\vec{u}'/\vec{x}]: t}$$

where  $\Gamma \vdash \vec{u} \, \overline{S} \, \vec{u}' : \vec{t}$  is shorthand for  $\Gamma \vdash u_i \, \overline{S} \, u'_i : t_i$ , for all  $i = 1 \dots n$ , if  $\vec{u} : \vec{t}, \, \vec{u}' : \vec{t}$ , and  $\vec{t} = t_1 \dots t_n$  for some  $n \ge 0$ .

Relation substitution is associative; we have that  $R \subseteq R[S]$ , for arbitrary  $R, S \in REL$ ; and  $\mathbf{R} = \mathbf{R}[S]$ , whenever  $\mathbf{R} \in REL_0$ .

We call  $R \in REL$  substitutive if  $R[R] \subseteq R$  (i.e., R[R] = R because the reverse inclusion holds for arbitrary R), and R is closed under substitutions if  $R[Id] \subseteq R$  (i.e., R[Id] = R). Any substitutive and reflexive open relation is closed under substitutions. Each of these properties is preserved by relation composition.

The following lemma addresses the behaviour of substitution with respect to relation composition and matching values.

**Lemma 3.4.1** For all  $R, R', S, S' \in REL$ ,

(1)  $(R R')[S S'] \subseteq (R[S])(R'[S']),$ 

$$(2) \ \overline{R}[S] \subseteq \overline{R[S] \cup S}$$

(3)  $\overline{R}[R] = \overline{R}$ , if R is substitutive

**Proof** (1) Suppose  $\Gamma \vdash a[\vec{u}/\vec{x}] (RR')[SS'] a'[\vec{u}'/\vec{x}] : t$  because  $\Gamma \vdash a RR' a' : t$  and  $\Gamma \vdash \vec{u} SS' \vec{u}' : t$ . They must be derived from  $\Gamma \vdash a Ra'' R' a' : t$  and  $\Gamma \vdash \vec{u} S \vec{u}'' S' \vec{u}' : t$ , for some a'' and  $\vec{u}''$ . Hence  $\Gamma \vdash a[\vec{u}/\vec{x}] R[S] a''[\vec{u}''/\vec{x}] : t$  and  $\Gamma \vdash a''[\vec{u}''/\vec{x}] R'[S'] a'[\vec{u}'/\vec{x}] : t$ , and we conclude that  $\Gamma \vdash a[\vec{u}/\vec{x}] (R[S])(R'[S']) a'[\vec{u}'/\vec{x}] : t$ , as required.

(2) follows if

$$\Gamma, \vec{x}: \vec{t} \vdash v \ \overline{R} \ v': t \ \& \ \Gamma \vdash \vec{u} \ \overline{S} \ \vec{u}': \vec{t} \ \Rightarrow \ \Gamma \vdash v[\vec{u}/\vec{x}] \ \overline{R[S] \cup S} \ v'[\vec{u}'/\vec{x}]: t$$
which we prove by induction on the derivation of  $\Gamma, \vec{x} : \vec{t} \vdash v \, \overline{R} \, v' : t$  from the rules in Table 3.1. We show one case for illustration, and we pick (Match fun) in order to illustrate the role of weakening. So, in this case  $v = \lambda x. a, v' = \lambda x. a'$  and  $t = t' \rightarrow t''$  such that  $\Gamma, \vec{x}: \vec{t}, x: t' \vdash a \ R \ a': t''$ . Both R and  $\overline{S}$  are closed under weakening, as they are open relations. Therefore  $\Gamma, x: t', \vec{x}: \vec{t} \vdash a \ R \ a': t''$  and  $\Gamma, x: t' \vdash \vec{u} \ \overline{S} \ \vec{u}': \vec{t}$ . Hence  $\Gamma, x: t' \vdash$  $a[\vec{u}/\vec{x}] R[S] a'[\vec{u}'/\vec{x}] : t''$ . By monotonicity, (Match fun), and substitution under  $\lambda$ , we conclude that  $\Gamma \vdash (\lambda x. a)[\vec{u}/\vec{x}] \ \overline{R[S] \cup S} \ (\lambda x. a')[\vec{u}'/\vec{x}] : t$ , as required. 

(3) is immediate from (2) and the definition of substitutivity.

Given any relation  $R \in REL$ , its open extension,  $R^{\circ} \in REL$ , is given by

$$rac{orall ec u:ec t.\ a[ec u/ec x]\ R\ a'[ec u/ec x]:t}{ec x:ec tec a\ R^\circ\ a':t}$$

For example, Rel is the open extension of  $Rel_0$  and the open identity relation Id is the open extension of  $Id_0$ .

Open extension is monotone, closed under substitutions, and satisfies that

$$R \subseteq S^{\circ} \quad \text{iff} \quad R[Id]_{\mathbf{0}} \subseteq S \tag{3.6}$$

These properties together with Lemma 3.4.1(1) imply that

$$R^{\circ} S^{\circ} \subseteq (R S)^{\circ} \tag{3.7}$$

The reverse inclusion does not hold in general.

Open extension on closed relations preserves and reflects transitivity as well as symmetry, and  $\mathbf{R}^{\circ}$  is reflexive if and only if  $Id_{\mathbf{0}} \subseteq \mathbf{R}$  (i.e., if  $\mathbf{R}$  is reflexive relative to  $Rel_{\mathbf{0}}$ ).

Let us also introduce a binary operation  $S \gg R$ , on open relations S and R. (It will only be used in the calculations with relation substitution in the remainder of this chapter; the reader may want to skip its definition on first reading.) It is defined as the greatest open relation  $S \gg R$  such that  $(S \gg R)[S] \subseteq R$ . That is,  $\Gamma \vdash a (S \gg R) a' : t$  holds if and only if  $\Gamma_0 \vdash a[\vec{u}/\vec{x}] \ (S \gg R) \ a'[\vec{u}'/\vec{x}] : t \text{ holds whenever } \Gamma = \Gamma_0, \vec{x} : \vec{t} \text{ and } \Gamma_0 \vdash \vec{u} \ \overline{S} \ \vec{u}' : \vec{t}.$ 

We notice that  $S \gg R \subseteq R$ . A relation R is substitutive iff  $R \subseteq R \gg R$  (i.e.,  $R = R \gg R$ ), and R is closed under substitutions iff  $R \subseteq Id \gg R$  (i.e.,  $R = Id \gg R$ ).

The operation  $S \gg R$  is anti-monotone in its first argument, S, and monotone in its second, R.

The following adjunction with relation substitution is very useful in calculations.

$$R[S] \subseteq R' \quad \text{iff} \quad R \subseteq S \gg R' \tag{3.8}$$

The "only if" implication is immediate because  $S \gg R'$  is defined as the greatest R such that  $R[S] \subseteq R'$ . For the reverse implication, supposing  $R \subseteq S \gg R'$  then, by monotonicity,  $R[S] \subseteq (S \gg R')[S]$  and the result follows because  $(S \gg R')[S] \subseteq R'$ , by the definitions.

In passing we note that, as a general property of adjunctions, (3.8) is equivalent to the conjunction of

$$R \subseteq (S \gg R[S]) \tag{3.9}$$

and

$$(S \gg R)[S] \subseteq R \tag{3.10}$$

(Comp value) 
$$\frac{\Gamma \vdash u \ \overline{R} \ u' : t}{\Gamma \vdash u \ \widehat{R} \ u' : t}$$

(Comp let) 
$$\frac{\Gamma \vdash a_1 \ R \ a'_1 : t_1 \quad \Gamma, x : t_1 \vdash a_2 \ R \ a'_2 : t_2}{\Gamma \vdash \mathbf{let} \ x = a_1 \ \mathbf{in} \ a_2 \ \widehat{R} \ \mathbf{let} \ x = a'_1 \ \mathbf{in} \ a'_2 : t_2}$$

(Comp apply) 
$$\frac{\Gamma \vdash u \ R \ u' : t_1 \rightharpoonup t_2 \quad \Gamma \vdash v \ R \ v' : t_1}{\Gamma \vdash u \ v \ \widehat{R} \ u' \ v' : t_2}$$

$$(\text{Comp case } \times) \qquad \qquad \frac{\Gamma \vdash u \ R \ u : t_1 \times \ldots \times t_n \quad \Gamma, x_1 : t_1, \ldots, x_n : t_n \vdash a \ R \ a' : t}{\Gamma \vdash \text{case } u \text{ of } \langle x_1, \ldots, x_n \rangle. \ a \ \widehat{R} \text{ case } u' \text{ of } \langle x_1, \ldots, x_n \rangle. \ a' : t}$$

$$(\text{Comp case +}) \quad \frac{\Gamma \vdash u \ \overline{R} \ u': t \quad \Gamma, x_1 : t_1[t/\chi] \vdash a_1 \ R \ a'_1 : t' \ \dots \ \Gamma, x_n : t_n[t/\chi] \vdash a_n \ R \ a'_n : t'}{\Gamma \vdash \begin{pmatrix} \text{case } u \ \text{of inj}_1 \ x_1 . \ a_1 \\ & \parallel \ \dots \\ & \parallel \ \text{inj}_n \ x_n . \ a_n \end{pmatrix} \ \widehat{R} \begin{pmatrix} \text{case } u' \ \text{of inj}_1 \ x_1 . \ a'_1 \\ & \parallel \ \dots \\ & \parallel \ \text{inj}_n \ x_n . \ a'_n \end{pmatrix} : t'} \\ \text{if } t = \mu \chi. \ t_1 + \dots + t_n \text{ and } n \ge 1$$



(see e.g. Crole 1994).

By easy calculations, using the adjunction (3.8), the following properties follows from the corresponding properties of relation substitution in Lemma 3.4.1.

**Lemma 3.4.2** For all  $R, R', S, S' \in REL$ ,

- $(1) \ (S \gg R)(S' \gg R') \subseteq (S S') \gg (R R')$
- $(2) \ \overline{S \gg R} \subseteq S \gg \overline{R \cup S}$
- (3)  $\overline{R} = R \gg \overline{R}$ , if R is substitutive

#### 3.5 Compatible refinement

For every open relation R, its compatible refinement  $\widehat{R} \in REL$  (Gordon 1994) relates expressions with identical outermost syntactic constructor and immediate subterms pairwise related by R. Table 3.2 makes this definition precise for our language; it is obtained quite mechanically from the type system in Table 2.5 and the distinction between values and general expressions in the syntax, Table 2.1. (The distinction between expressions and values and the use of matching values is similar to the treatment of actions and yielders in action notation in Lassen (1997).)

The restriction of compatible refinement to values coincides with matching values, i.e.,  $\overline{R} \subseteq \widehat{R}$  and whenever  $R \in REL$ ,  $\Gamma \vdash u : t$  and  $\Gamma \vdash u' : t$ ,

$$\Gamma \vdash u \ \widehat{R} \ u' : t \quad \text{iff} \quad \Gamma \vdash u \ \overline{R} \ u' : t \tag{3.11}$$

Moreover, due to the reduced syntax, compatible refinement only relates values to values and non-values to non-values:

**Lemma 3.5.1** If  $R \in REL$  and  $\Gamma \vdash a \hat{R} a' : t$  then a is a value if and only if a' is a value.

The properties of compatible refinement are very similar to those of matching values. Compatible refinement is monotone and preserves reflexivity, transitivity, and symmetry. It commutes with relation composition and reciprocation:

$$\widehat{RS} = \widehat{RS} \tag{3.12}$$

$$\widehat{R}^{\rm op} = \widehat{R^{\rm op}} \tag{3.13}$$

The following lemma is the analogue of (2)–(3) in Lemmas 3.4.1 and 3.4.2. The proof is analogous as well and is omitted.

**Lemma 3.5.2** For all  $R, S \in REL$ ,

(1)  $\widehat{R}[S] \subseteq \widehat{R[S] \cup S}$ 

$$(2) \ \widehat{S} \gg \widehat{R} \subseteq S \gg \widehat{R} \cup \widehat{S}$$

(3)  $\widehat{R}[R] = \widehat{R} = R \gg \widehat{R}$ , if R is substitutive

#### 3.6 Compatibility

An open relation R is compatible if  $\hat{R} \subseteq R$ . Compatibility can also be expressed in terms of contexts: a relation R is compatible if whenever  $\phi_i$  and  $\phi'_i$  are abstractions related by  $R|_{\theta_i}$ , for  $i = 1 \dots n$ , so are  $C[\![\vec{\phi}/\vec{\xi}]\!]$  and  $C[\![\vec{\phi}'/\vec{\xi}]\!]$ , for all contexts C with abstraction variables  $\vec{\xi} : \vec{\theta}$ . But compatible refinement provides a tractable, indirect notation for contexts that is easier to work with. We will use it extensively for constructing and reasoning about relations and contexts.

If a compatible relation is transitive, hence a preorder, we call it a *pre-congruence*. If it is also symmetric, i.e., it is a compatible equivalence relation, we call it a *congruence*.

Notice that the definition of compatibility says that the compatible relations are the prefixed points of compatible refinement. It is easy to see that the open identity relation, *Id*, is compatible and one can show, by induction on typings, that it is the least pre-fixed point of compatible refinement, that is, every compatible relation is reflexive. This is called the *binary induction principle* in Jacobs and Rutten (1997). We also observe that compatibility is closed under compatible refinement.

**Lemma 3.6.1** Compatibility is preserved by relation composition and transitive closure.

**Proof** First we observe that compatibility is preserved by relation composition: if R and S are compatible, so is their composition, RS, because

$$\widehat{RS} = \widehat{RS} \subseteq RS.$$

Next, suppose R is compatible and  $\Gamma \vdash a \widehat{R^+} a' : t$ . If this is derived by (Comp var) or (Comp unit), a = a' and we conclude that  $\Gamma \vdash a R^+ a' : t$  because R is compatible and thus R

and  $R^+$  are reflexive. Otherwise each immediate subterm  $a_i$  of a is related to a corresponding subterm  $a'_i$  of a',  $\Gamma$ ,  $\Gamma_i \vdash a_i \ R^+ \ a'_i : t_i$ , for some  $\Gamma_i$  and  $t_i$ . This means that there exists  $m_i \ge 1$ such that  $\Gamma$ ,  $\Gamma_i \vdash a_i \ R^{m_i} \ a'_i : t_i$ , where  $R^{m_i}$  is the  $m_i$ -fold composition of R with itself. Let m be the greatest of these  $m_i$ , for all pairs of subterms. Since R is compatible it is also reflexive. Hence  $\Gamma$ ,  $\Gamma_i \vdash a'_i \ R^{m-m_i} \ a'_i : t_i$  and then  $\Gamma$ ,  $\Gamma_i \vdash a_i \ R^m \ a'_i : t_i$ , for all corresponding subterms  $a_i$  and  $a'_i$ . Hence  $\Gamma \vdash a \ \widehat{R^m} \ a'$ , by definition of compatible refinement, and then  $\Gamma \vdash a \ R^m \ a' : t$  because compatibility is preserved by relation composition. So  $\Gamma \vdash a \ R^+ \ a' : t$ and we conclude that  $R^+$  is compatible.  $\Box$ 

The universal open relation, Rel, is the largest compatible relation: it contains every open relation, in particular, it contains Rel and thus it is compatible. Compatibility is closed under arbitrary intersections; the empty intersection is Rel. Therefore compatible relations form a complete lattice, ordered by subset inclusion. The least compatible relation is Id. The least upper bound is not just set union, because compatibility is not closed under unions; rather, it is the intersection of all compatible upper bounds.

There is an associated closure operator, called context closure. For any open relation  $R \in REL$ , its *context closure*,  $R^{\mathsf{C}} \in REL$ , relates expressions with matching outermost variable capturing context C and subterms  $\vec{\phi}$  and  $\vec{\phi'}$  related by R

$$\Gamma \vdash C[\![\vec{\phi}\!\vec{\xi}]\!] \ R^{\mathsf{C}} \ C[\![\vec{\phi}'\!/\vec{\xi}]\!] : t \quad \text{whenever} \quad \vec{\xi} \, : \vec{\theta}, \Gamma \vdash C : t \quad \text{and} \quad \vec{\phi} \ R \ \vec{\phi}' : \vec{\theta}$$

In other words,  $R^{\mathsf{C}}$  is the closure of R under variable capturing contexts.

Context closure can be defined inductively by means of compatible refinement as the smallest open relation closed under the rules:

(Ctx R) 
$$\frac{\Gamma \vdash a \ R \ a': t}{\Gamma \vdash a \ R^{\mathsf{C}} \ a': t}$$
  
(Ctx Comp) 
$$\frac{\Gamma \vdash a \ \widehat{R^{\mathsf{C}}} \ a': t}{\Gamma \vdash a \ R^{\mathsf{C}} \ a': t}$$

Context closure is monotone and idempotent,  $(R^{\mathsf{C}})^{\mathsf{C}} = R^{\mathsf{C}}$ . By definition,  $R^{\mathsf{C}}$  is the least compatible relation that contains R, so context closure could more accurately be called 'compatible closure' (as in Barendregt 1984). An open relation R is compatible if and only if  $R = R^{\mathsf{C}}$ .

Closure under substitutions is preserved by context closure but substitutivity is not, in general.

In several cases, we will encounter relations of the form Id[R]. It is useful to note that Id[R] is included in the context closure of R. More precisely and more generally,

$$Id[R^{\mathsf{C}}] \subseteq \widehat{R^{\mathsf{C}}} \tag{3.14}$$

To see this, use the adjunction (3.8) to rewrite (3.14) to  $Id \subseteq R^{\mathsf{C}} \gg \widehat{R^{\mathsf{C}}}$ . This follows if  $R^{\mathsf{C}} \gg \widehat{R^{\mathsf{C}}}$  is compatible which can be deduced as follows.

$$\widehat{R^{\mathsf{C}} \gg \widehat{R^{\mathsf{C}}}} \subseteq R^{\mathsf{C}} \gg \widehat{\widehat{R^{\mathsf{C}}} \cup R^{\mathsf{C}}} \text{ by Lemma 3.5.2(1)}$$
$$\subseteq R^{\mathsf{C}} \gg \widehat{\widehat{R^{\mathsf{C}}} \cup R^{\mathsf{C}}} \text{ since } \widehat{R^{\mathsf{C}}} \subseteq R^{\mathsf{C}}$$

#### 3.7 Contextual equivalence

The relations that we shall study in the remainder are mainly semantic preorders and equivalences between expressions. The most important of these is contextual equivalence (Morris 1968; Plotkin 1975). It is a standard and generally accepted syntactic definition of semantic equivalence based on suitable observations of the operational behaviour of programs. Conventionally, two expressions are defined to be contextually equivalent if they exhibit comparable operational behaviour in all program contexts (a suitable class of closed variable capturing contexts). One can then prove that this definition yields the largest compatible relation (and congruence) between expressions with comparable operational behaviour. This characterisation is really the motivation for contextual equivalence and it is key to many of the relational proofs about contextual equivalence in the sequel. In keeping with our attempts to avoid dealing explicitly with term contexts, here we will even define contextual equivalence as the largest compatible relation between expressions with comparable operational behaviour. This shifts the proof obligation from proving properties based on a definition phrased in terms of variable capturing contexts to proving the existence of a largest relation with the desired properties. This is accomplished by means of the following lemma.

**Lemma 3.7.1** Suppose  $\mathbb{P}$  is a predicate on open relations,  $\mathbb{P} \subseteq REL$ . If  $Id \in \mathbb{P}$  and  $\mathbb{P}$  is closed under non-empty unions and relation composition, there exists a largest compatible relation S in  $\mathbb{P}$ . The relation S is transitive. It is also symmetric if  $\mathbb{P}$  is closed under reciprocation, *i.e.*, if  $R^{\mathrm{op}} \in \mathbb{P}$  whenever  $R \in \mathbb{P}$ .

**Proof** Suppose that  $\mathbb{P}$  satisfies the conditions of the lemma. Let S be the union of all compatible relations in  $\mathbb{P}$ ,

$$S \stackrel{\text{def}}{=} \bigcup \{ R \mid \widehat{R} \subseteq R \in \mathbb{P} \}.$$

Since Id is compatible and  $Id \in \mathbb{P}$ , this is a non-empty union, hence  $S \in \mathbb{P}$ .

By definition, S is larger than any compatible relation in  $\mathbb{P}$ . We are going to show that S is itself compatible, i.e.,  $\widehat{S} \subseteq S$ . So suppose that  $\Gamma \vdash a \widehat{S} a' : t$ . If this is derived by (Comp var) or (Comp unit), a = a' and  $\Gamma \vdash a S a' : t$  holds because  $Id \subseteq S$ . Otherwise, a and a' have  $n \ge 1$  subterms  $a_1 \ldots a_n$  and  $a'_1 \ldots a'_n$ , pairwise related by S,  $\Gamma, \Gamma_i \vdash a_i S a'_i : t_i$  for  $i \in 1..n$ . By the definition of S we see that for each  $i \in 1..n$  there is a compatible  $R_i$  in  $\mathbb{P}$  with  $\Gamma, \Gamma_i \vdash a_i R_i a'_i : t_i$ . Now let R denote their composition,

$$R \stackrel{\text{def}}{=} R_1 \cdots R_n.$$

We note that R is compatible and in  $\mathbb{P}$  because compatibility and  $\mathbb{P}$  are closed under composition. Compatibility implies reflexivity, so  $R_1 \ldots R_n$  are all reflexive and, for each  $i \in 1..n$ , we get that  $\Gamma, \Gamma_i \vdash a_i R a'_i : t_i$  by the calculation:

$$\Gamma, \Gamma_i \vdash a_i \ R_1 \cdots R_{i-1} \ a_i \ R_i \ a'_i \ R_{i+1} \cdots R_n \ a'_i : t_i$$

Hence  $\Gamma \vdash a \ \widehat{R} \ a' : t$ . But  $\widehat{R} \subseteq R \subseteq S$ , since R is compatible and in  $\mathbb{P}$ , and thus  $\Gamma \vdash a \ S \ a' : t$ , as required.

We conclude that S is the largest compatible relation in  $\mathbb{P}$ .

The composition SS is also compatible and in  $\mathbb{P}$ , because these properties are closed under relation composition, therefore  $SS \subseteq S$ , i.e., S is transitive and thus it is a pre-congruence.

If  $\mathbb{P}$  is also closed under reciprocation,  $R \in \mathbb{P} \Rightarrow R^{\mathrm{op}} \in \mathbb{P}$ , then S is a congruence because compatibility is closed under reciprocation, so  $S^{\mathrm{op}}$  is compatible and in  $\mathbb{P}$  and thus  $S^{\mathrm{op}} \subseteq S$ , i.e., S is symmetric.

The properties  $\mathbb{P}$  that we shall be working with are various forms of "adequacy". In general, a relation is adequate with respect to some basic observation on expressions if it only relates expressions that are observationally identical. Our basic observation is going to be termination of closed expressions of type unit. So we say that an open relation R is *adequate* if, for all a, a': unit,

$$a \ R \ a' : \text{unit} \quad \text{implies} \quad a \rightsquigarrow \langle \rangle \iff a' \rightsquigarrow \langle \rangle$$

$$(3.15)$$

Let ADEQ denote the set of all adequate open relations,

$$ADEQ = \{ R \in REL \mid \forall a, a'. \ a \ R \ a' : unit \Rightarrow (a \rightsquigarrow \langle \rangle \Leftrightarrow a' \rightsquigarrow \langle \rangle) \}$$

It satisfies the conditions of Lemma 3.7.1 and it is closed under reciprocation. We define *contextual equivalence*,  $\cong \in REL$ , to be the largest compatible and adequate relation. According to the lemma, such a relation exists and it is a congruence.

Contextual equivalence is sometimes called "observational congruence" (Meyer 1988). Here, we effectively *define* contextual equivalence to be the largest congruence which respects "observation" of termination of closed expressions of type unit.

Our definition is equivalent to the more conventional definitions in terms of variable capturing contexts:

**Proposition 3.7.2** Whenever  $\vec{x} : \vec{t} \vdash a : t$  and  $\vec{x} : \vec{t} \vdash a' : t$ ,

$$\vec{x}: \vec{t} \vdash a \cong a': t \quad i\!f\!f \quad \forall \xi, C. \quad \xi: (\vec{t})t \vdash C: \text{unit} \ \Rightarrow \ (C\llbracket(\vec{x})a\!/\!\xi\rrbracket \rightsquigarrow \langle \, \rangle \ \Leftrightarrow \ C\llbracket(\vec{x})a'\!/\!\xi\rrbracket \rightsquigarrow \langle \, \rangle)$$

**Proof** If  $\vec{x} : \vec{t} \vdash a \cong a' : t$  and  $\xi : (\vec{t})t \vdash C :$  unit then  $C[[(\vec{x})a/\xi]] \cong C[[(\vec{x})a'/\xi]] :$  unit, by compatibility. Since  $\cong$  is adequate we get that  $C[[(\vec{x})a/\xi]] \rightsquigarrow \langle \rangle$  iff  $C[[(\vec{x})a'/\xi]] \rightsquigarrow \langle \rangle$ .

Conversely, supposing a and a' satisfy the right hand side, let R be the "singleton" open relation between them (i.e., the smallest open relation such that  $\vec{x} : \vec{t} \vdash a R a' : t$ ). We observe that the fact that a and a' satisfy the right hand side is equivalent to that  $R^{\mathsf{C}}$  is adequate. As  $R^{\mathsf{C}}$  is also compatible, it is included in contextual equivalence. By (Ctx Comp) it follows that  $\vec{x} : \vec{t} \vdash a \cong a' : t$ , as required.

Contextual equivalence was introduced in this form by Morris (1968) for untyped  $\lambda$ calculus. He called it "extensional equivalence". His notion of observation was reducibility to  $\alpha\beta\eta$ -normal form for arbitrary open terms. Our observation of termination of closed expressions according to an operational semantics is closer to the definition in Plotkin (1975).

The alternative definition of contextual equivalence here, as the greatest compatible and adequate relation, offers two advantages. One is that it can be formulated without any reference to explicit contexts. The other is that it yields a co-induction-like proof rule for contextual equivalence:

$$\frac{\widehat{R} \subseteq R \quad R \in ADEQ}{R \subseteq \cong} \tag{3.16}$$

The arguments for transitivity and symmetry at the end of the proof of Lemma 3.7.1 were examples of this co-induction-like reasoning.

We get that  $R \subseteq \cong$  if and only if  $R^{\mathsf{C}} \in ADEQ$ . In particular,  $\Gamma \vdash a \cong a' : t$  if and only if the singleton open relation R, which relates a and a' at type t,

 $\Gamma' \vdash a \ R \ a' : t$ , whenever  $\Gamma \subseteq \Gamma'$ 

and is empty otherwise, satisfies  $R^{\mathsf{C}} \in ADEQ$ . Indeed, this could be taken as the definition of contextual equivalence as in Gordon (1998) and Gordon, Hankin, and Lassen (1997b).

#### 3.8 Compatibility and substitutivity

Now, let us look at compatible relations which are also substitutive. Everything that we said about the class of compatible relations can also be said about relations that are both compatible and substitutive. The conjunct property of compatibility and substitutivity is preserved by arbitrary intersections, by relation composition, by transitive closure, and by compatible refinement. The latter holds because compatibility is preserved by compatible refinement and because of Lemma 3.5.2.

The compatible and substitutive relations form a complete lattice with Rel as top element and Id as bottom element. The associated closure operation, substitutive context closure,  $R^{SC} \in REL$ , is the closure of  $R \in REL$  under substituting contexts,

 $\Gamma \vdash A[\![\vec{\phi}\!/\!\vec{\xi}\,]\!] \ R^{\mathsf{SC}} \ A[\![\vec{\phi}'\!/\!\vec{\xi}\,]\!] : t \quad \text{whenever} \quad \vec{\xi} \, : \vec{\theta}, \Gamma \vdash A : t \quad \text{and} \quad \vec{\phi} \ R \ \vec{\phi}' : \vec{\theta}$ 

It has a succinct inductive definition:

(SC Subst) 
$$\frac{\Gamma \vdash a \ R[R^{SC}] \ a': t}{\Gamma \vdash a \ R^{SC} \ a': t}$$
  
(SC Comp) 
$$\frac{\Gamma \vdash a \ \widehat{R^{SC}} \ a': t}{\Gamma \vdash a \ R^{SC} \ a': t}$$

c.c.

In other words,  $R^{SC}$  is the least solution to the recursive equation:

$$R^{\mathsf{SC}} = R[R^{\mathsf{SC}}] \cup \widehat{R^{\mathsf{SC}}}$$
(3.17)

Substitutive context closure is monotone and idempotent, and  $R^{SC}$  is compatible and substitutive. Compatibility is direct from (SC Comp). Substitutivity is equivalent to that  $R^{SC} \subseteq R^{SC} \gg R^{SC}$ . We prove the latter by induction as follows. Since  $R^{SC}$  is the least solution to (3.17), the result follows by induction if we can show that  $R^{SC} \gg R^{SC}$  is a pre-fixed point (of the implicit monotone operator):

$$R[R^{\mathsf{SC}} \gg R^{\mathsf{SC}}] \cup R^{\widehat{\mathsf{SC}} \gg R^{\mathsf{SC}}} \subseteq R^{\mathsf{SC}} \cong R^{\mathsf{SC}}$$

It follows from the calculations:

$$\begin{split} R[R^{\mathsf{SC}} \gg R^{\mathsf{SC}}] &\subseteq R^{\mathsf{SC}} \gg R[R^{\mathsf{SC}} \gg R^{\mathsf{SC}}][R^{\mathsf{SC}}] & \text{by (3.9)} \\ &\subseteq R^{\mathsf{SC}} \gg R[(R^{\mathsf{SC}} \gg R^{\mathsf{SC}})[R^{\mathsf{SC}}]] & \text{associativity of substitution} \\ &\subseteq R^{\mathsf{SC}} \gg R[R^{\mathsf{SC}}] & (3.10) \\ &\subseteq R^{\mathsf{SC}} \gg R^{\mathsf{SC}} & (\mathrm{SC \ Subst}) \end{split}$$
$$\begin{split} &\widehat{R^{\mathsf{SC}} \gg R^{\mathsf{SC}}} & \subseteq R^{\mathsf{SC}} \gg R^{\mathsf{SC}} & Lemma \ 3.5.2(2) \\ &\subseteq R^{\mathsf{SC}} \gg R^{\mathsf{SC}} & (\mathrm{SC \ Comp}) \end{split}$$

An open relation R is compatible and substitutive if and only if  $R = R^{SC}$ . Since  $R^{SC}$  is compatible and substitutive, it is also reflexive and closed under substitutions.

Clearly  $R^{\mathsf{C}} \subseteq R^{\mathsf{SC}}$ , for every open relation R. Furthermore,  $R^{\mathsf{C}} = R^{\mathsf{SC}}$  if and only if  $R[R^{\mathsf{C}}] \subseteq R^{\mathsf{C}}$ . In other words  $R^{\mathsf{C}}$  is substitutive if and only if  $R[R^{\mathsf{C}}] \subseteq R^{\mathsf{C}}$ . In particular, if  $\mathbf{R} \in REL_0$ ,  $\mathbf{R}^{\mathsf{C}}$  is substitutive because  $\mathbf{R}[S] = \mathbf{R}$  for arbitrary  $S \in REL$ .

An analogue of Lemma 3.7.1 holds for the space of compatible and substitutive relations. It tells us that there exists a largest compatible and *substitutive* adequate relation. This corresponds to the way Pitts defines observational congruence in some of his expositions (1994a, 1998). We shall see in Chapter 4 that contextual equivalence is substitutive, so it is the same relation as one would obtain in this fashion.

Let us mention a useful property of substitutive context closure.

**Lemma 3.8.1** If  $R \in REL$  is a preorder and is closed under substitutions, then  $R^{SC} \subseteq \widehat{R^{SC}} R$ and, symmetrically,  $R^{SC} \subseteq R \widehat{R^{SC}}$ .

**Proof** We are going to show that  $R^{SC} \subseteq \widehat{R^{SC}} R[Id]^*$  for arbitrary open relations R. Then the first inclusion of the lemma follows because  $R[Id]^* \subseteq R$  whenever R is closed under substitutions and is reflexive and transitive. The second inclusion follows symmetrically.

Since  $R^{\mathsf{SC}} = R[R^{\mathsf{SC}}] \cup \widehat{R^{\mathsf{SC}}}$  we must show that  $R[R^{\mathsf{SC}}]$  and  $\widehat{R^{\mathsf{SC}}}$  are included in  $\widehat{R^{\mathsf{SC}}} R[Id]^*$ . For  $R[R^{\mathsf{SC}}]$  this follows by calculation,

$$R[R^{\mathsf{SC}}] \subseteq Id[R^{\mathsf{SC}}] R[Id] \text{ by Lemma 3.4.1(1)}$$
$$\subseteq \widehat{R^{\mathsf{SC}}} R[Id] \text{ by (3.14)}$$
$$\subseteq \widehat{R^{\mathsf{SC}}} R[Id]^*$$

Secondly,  $\widehat{R^{SC}} \subseteq \widehat{R^{SC}} R[Id]^*$  is immediate because  $R[Id]^*$  is reflexive.

A commonly used construction of compatible and substitutive relations is that of Howe's pre-congruence candidate relation (Howe 1996), called *compatible extension* by Gordon (1998). For any closed relation  $\mathbf{R}$ , its compatible extension  $\mathbf{R}^{\bullet}$  is the smallest open relation closed under the rule

(Cand Def) 
$$\frac{\Gamma \vdash a \ \widehat{\mathbf{R}^{\bullet}} \ a'': t \quad \Gamma \vdash a'' \ \mathbf{R}^{\circ} \ a': t}{\Gamma \vdash a \ \mathbf{R}^{\bullet} \ a': t}$$

That is,  $\mathbf{R}^{\bullet}$  is the least solution to the recursive equation:

$$\boldsymbol{R}^{\bullet} = \widehat{\boldsymbol{R}^{\bullet}} \boldsymbol{R}^{\circ} \tag{3.18}$$

Furthermore, it is the least solution to the inequality:

$$\widehat{\boldsymbol{R}^{\bullet}} \, \boldsymbol{R}^{\circ} \subseteq \boldsymbol{R}^{\bullet} \tag{3.19}$$

The following lemma lists some properties of compatible extension.

#### **Lemma 3.8.2** For every $\mathbf{R} \in REL_0$ ,

- (1)  $\mathbf{R}^{\bullet}$  is substitutive
- (2) if  $\mathbf{R}^{\circ}$  is reflexive, then  $\mathbf{R}^{\bullet}$  is compatible and  $\mathbf{R}^{\circ} \subseteq \mathbf{R}^{\bullet}$

- (3) if  $\mathbf{R}^{\circ}$  is transitive, then  $\mathbf{R}^{\bullet} \mathbf{R}^{\circ} \subseteq \mathbf{R}^{\bullet}$
- (4) if  $\mathbf{R}^{\circ}$  is reflexive and symmetric, then  $\mathbf{R}^{\bullet*}$  is symmetric

**Proof** (1)  $\mathbf{R}^{\bullet}$  is substitutive if  $\mathbf{R}^{\bullet} \subseteq \mathbf{R}^{\bullet} \gg \mathbf{R}^{\bullet}$ . Recalling that  $\mathbf{R}^{\bullet}$  is the least solution to (3.19), we get the required result if  $\mathbf{R}^{\bullet} \gg \mathbf{R}^{\bullet}$  is another solution to (3.19). We see this by calculating

$$\widehat{\mathbf{R}^{\bullet} \gg \mathbf{R}^{\bullet}} \ \widehat{\mathbf{R}^{\circ}} \subseteq (\mathbf{R}^{\bullet} \gg \widehat{\mathbf{R}^{\bullet}}) \ \mathbf{R}^{\circ} \qquad \text{by Lemma 3.5.2(2)} \\
= (\mathbf{R}^{\bullet} \gg \widehat{\mathbf{R}^{\bullet}})(Id \gg \mathbf{R}^{\circ}) \qquad \mathbf{R}^{\circ} \text{ is closed under substitutions} \\
\subseteq (\mathbf{R}^{\bullet} Id) \gg (\widehat{\mathbf{R}^{\bullet}} \ \mathbf{R}^{\circ}) \qquad \text{Lemma 3.4.2(1)} \\
= \mathbf{R}^{\bullet} \gg \mathbf{R}^{\bullet} \qquad (3.18)$$

(2) Suppose  $\mathbf{R}^{\circ}$  is reflexive. Then  $\widehat{\mathbf{R}^{\bullet}} \subseteq \widehat{\mathbf{R}^{\bullet}} \mathbf{R}^{\circ}$ . Hence  $\widehat{\mathbf{R}^{\bullet}} \subseteq \mathbf{R}^{\bullet}$  by (3.18) which means that  $\mathbf{R}^{\bullet}$  is compatible. Hence  $\widehat{\mathbf{R}^{\bullet}}$  is compatible too and therefore  $\widehat{\mathbf{R}^{\bullet}}$  is reflexive. We obtain that  $\mathbf{R}^{\circ} \subseteq \widehat{\mathbf{R}^{\bullet}} \mathbf{R}^{\circ}$  and conclude  $\mathbf{R}^{\circ} \subseteq \mathbf{R}^{\bullet}$  by (3.18).

(3) Suppose  $\mathbf{R}^{\circ}$  is transitive. Then  $\widehat{\mathbf{R}^{\bullet}} \mathbf{R}^{\circ} \mathbf{R}^{\circ} \subseteq \widehat{\mathbf{R}^{\bullet}} \mathbf{R}^{\circ}$  and hence  $\mathbf{R}^{\bullet} \mathbf{R}^{\circ} \subseteq \mathbf{R}^{\bullet}$  by (3.18).

(4) Suppose  $\mathbf{R}^{\circ}$  is reflexive and symmetric. We get that  $\mathbf{R}^{\bullet*}$  is symmetric if  $\mathbf{R}^{\bullet} \subseteq \mathbf{R}^{\bullet*\circ \mathrm{p}}$ . This follows by the definiton of  $\mathbf{R}^{\bullet}$  if  $\mathbf{R}^{\bullet*\circ \mathrm{p}}$  is a solution to (3.19), which we calculate as follows.

$$(\widehat{\mathbf{R}^{\bullet}})^{*^{\mathrm{op}}} \mathbf{R}^{\circ} = (\widehat{\mathbf{R}^{\bullet}})^{*^{\mathrm{op}}} \mathbf{R}^{\circ \mathrm{op}} \text{ by (3.13) and symmetry}$$

$$\subseteq (\mathbf{R}^{\bullet})^{*^{\mathrm{op}}} \mathbf{R}^{\bullet \mathrm{op}} \text{ from (2) above and Lemma 3.6.1}$$

$$= (\mathbf{R}^{\bullet})^{\mathrm{op}*} \mathbf{R}^{\bullet \mathrm{op}}$$

$$= (\mathbf{R}^{\bullet})^{\mathrm{op}*}$$

$$= (\mathbf{R}^{\bullet})^{*^{\mathrm{op}}}$$

#### 3.9 Relations of higher arity

So far, we have only considered binary relations. Indeed, we shall mostly use binary relations but it is worthwhile investigating how the theory generalises to relations of arbitrary countable arity.

For any countable ordinal  $\alpha$ , let  $Rel(\alpha)$  be the universal abstraction-type indexed  $\alpha$ -ary relation on abstractions, i.e.,

$$\operatorname{Rel}(\alpha)|_{(\vec{t})t} = \left\{ ((\vec{x}_{\iota})a_{\iota})_{\iota < \alpha} \mid \vec{x}_{\iota} : \vec{t} \vdash a_{\iota} : t \text{ for all } \iota < \alpha \right\}$$

Let  $R \subseteq Rel(\alpha)$  be an  $\alpha$ -ary open relation if it is closed under weakening, analogously to (3.1), and let  $REL(\alpha) \subseteq \mathcal{P}(Rel(\alpha))$  be the set of  $\alpha$ -ary open relations. Of course, Rel(2) = Rel and REL(2) = REL.

The bulk of the relational theory presented in the preceding sections carry over unchanged to relations of arbitrary arity (including the notions of reflexivity, closed relations, projection, matching values, relation substitution, substitutivity, closure under substitutions, open extension, compatible refinement, compatibility, context closure, substitutive context closure).

Exceptions are reciprocation and relation composition (and derived notions: symmetry, symmetrisation, transitivity, transitive closure, compatible extension). They do not generalise

to arbitrary arity in an obvious way. Note also that the proof of Lemma 3.7.1 makes essential use of transitivity—it is not clear which properties that the predicate  $\mathbb{P}$  should possess, in general, to ensure that there is a largest compatible relation S in  $\mathbb{P}$ .

Relations of higher arities will be used in arguments about sequentiality in §4.4 and §6.8.

#### 3.10 Structural reduction

There is a useful link between compatible refinement and the operational semantics from Chapter 2, via the primitive reduction relation,  $\rightarrow$ .

**Lemma 3.10.1** Suppose  $R \in REL$  is substitutive. Whenever  $a \hat{R} a' : t \text{ and } a \to b$ , there exists b' such that  $a' \to b'$  and b R b' : t.

**Proof** For any arbitrary open relation R we prove that

$$a \widehat{R} a' : t \& a \to b \Rightarrow \exists b'. a' \to b' \& b R[R] b' : t$$

$$(3.20)$$

by case analysis of the derivation of  $a \rightarrow b$  by the rules in Table 2.2. We show only one case for illustration.

**Case (Redex apply)**  $a \to b$  because  $a = (\lambda x. a_0) v$  and  $b = a_0[v/x]$  for some function  $\lambda x. a_0$ and value v. Since  $a \hat{R} a' : t$  we have that  $a' = (\lambda x. a'_0) v'$  with  $x : t' \vdash a_0 R a'_0 : t$  and  $v \overline{R} v' : t'$ . Let  $b' = a'_0[v'/x]$  then  $a' \to b'$ , by (Redex apply), and b R[R] b' : t, by (Comp apply) and (Match fun).

The remaining cases are similar.

From (3.20) the result follows if R is substitutive.

It is not difficult to see that the proof generalises to relations of arbitrary arity:

**Lemma 3.10.2** Suppose  $R \in REL(\alpha)$  is substitutive. Whenever  $(a_{\iota})_{\iota < \alpha} \in \widehat{R}|_{\iota}$  and  $a_{\iota_0} \to b_{\iota_0}$ , for some  $\iota_0 < \alpha$ , there exists  $(b_{\iota})_{\iota < \alpha} \in R|_{\iota}$  such that  $a_{\iota} \to b_{\iota}$ , for all  $\iota < \alpha$ .

The lemmas assert that primitive reduction is determined by the outermost syntactic structure of expressions (in accordance with the definition of compatible refinement) and primitive reduction operates "uniformly" on the immediate subexpressions under this outermost syntactic structure. In other words, the lemmas formalise the intuitions from §2.2.1 about the "structural" nature of the primitive reduction relation. The ramifications of this property and its relational formalisation will become apparent in the next chapter in the proofs by induction on derivations of evaluations or transitions where the (Eval redex) and (Trans redex) cases can, in most cases, be resolved uniformly be reference to Lemma 3.10.1 or 3.10.2.

The structural property of reductions characterised by Lemmas 3.10.1 and 3.10.2 is very strong and depends rather heavily on the exact syntax. For instance, suppose we add projections,  $\mathbf{proj}_i u$ , into the syntax, equipped with the following rules for compatible refinement and reduction.

(Comp proj) 
$$\frac{\Gamma \vdash u \ R \ u' : t_1 \times \ldots \times t_n}{\Gamma \vdash \mathbf{proj}_i \ u \ \widehat{R} \ \mathbf{proj}_i \ u' : t_i} \quad \text{if } 1 \le i \le n$$

(Redex proj) **proj**<sub>i</sub>  $\langle u_1, \ldots, u_n \rangle \to u_i$ , if  $1 \le i \le n$ 

Then we would have to add to the lemmas the requirement that R contains its matching values,  $\overline{R} \subseteq R$ . More generally, if we require that R is compatible and substitutive in the two lemmas, they appear to be robust with respect to most language extensions. The extra requirement that R must be compatible should not impair the use of the lemmas significantly.

#### 3.11 Future work

The relational algebra given here suffices for the development in the remainder of the dissertation of operationally-based relational reasoning techniques for the functional language from Chapter 2 and its nondeterministic extensions in Part II. Nonetheless, it would be interesting to further develop the relational theory in this chapter in its own right.

First of all, one might generalise the presentation of the theory to general higher-order syntax (see §2.4.3) rather than just the terms of our particular functional language. This would involve a clarification of the underlying principles of matching values, substitutions, compatible refinement, and the structural property of simple reductions reflected by Lemma 3.10.1. Perhaps the most productive approach would be to develop a paradigmatic relational theory for Moggi's monadic meta-language (Moggi 1991) thus encompassing both call-by-value and call-by-name variables and reductions.

Another task is to complete the relational algebra with further operations that would support algebraic manipulations and calculations with relations. For instance, there are several operators in the calculus of relations (see Pratt 1992) that we have omitted but which can be handy for defining or reasoning about relations. (The relational apparatus of the Squiggol school (see Bird and de Moor 1997) does not seem directly relevant for our purposes—it is mainly concerned with relations between program inputs and outputs, an issue which is largely orthogonal to our study of relations between programs.)

Recent work by Pitts and others (Pitts 1997b; Birkedal and Harper 1997; Pitts 1998) employs the versatile notion of logical relations (surveyed in Mitchell 1990, 1996) to reason about operational semantics and operationally-based equivalence relations. The central notions of matching values and substitutivity in our development are reminiscent of the qualities of logical relations, and it would be interesting to see if our relational algebra can contribute to the research on syntactic logical relations. Sometimes logical relations relate terms of *related* type, e.g., when used as implementation relations involving some sort of data refinement, as in Birkedal and Harper (1997). This is more general than the term relations presented in this chapter which only relate identically typed terms, and it suggests a generalisation of our framework that might be worth pursuing.

As mentioned at the beginning of the chapter, efforts have been made to achieve mathematical rigour and simplicity in the development of the theory. The theory is also quite versatile, as the applications in the remainder of the dissertation will show, so it could be both feasible and useful to put the theory into an automated theorem prover to verify that the goals of mathematical rigour and simplicity have been achieved and to provide tool support for relational reasoning.

### Chapter 4

## **Relational Reasoning**

In this chapter the relational algebra from Chapter 3 is used to develop the theory of contextual equivalence for the deterministic language from Chapter 2. Novel proofs of the basic laws, sequentiality and continuity properties, induction rules, and the CIU Theorem are presented together with proof rules for simulation up to context. Finally, operational extensionality is proved by Howe's method.

The phrase "relational reasoning" refers broadly to syntactic arguments that make use of the relational algebra from the previous chapter in an essential way. More specifically, operational properties can often be proved relationally by constructing a "suitable" relation which is shown to be "preserved by evaluation" by induction on derivations of evaluations—being "suitable" will normally include substitutivity in order to obtain "preservation by evaluation" which will be some form of simulation property.

The theory of contextual equivalence is basically developed in three stages, and it is established which properties can be shown at each stage of sophistication.

First, in §4.3, it is shown that the open extension of Kleene equivalence is contained in contextual equivalence; this basically means that the evaluation relation is contained in contextual equivalence and that two open expressions are contextually equivalent if all their closed instances can be shown to be contextually equivalent by means of Kleene equivalence. This result allows us to verify basic  $\beta_v$ -equivalences and reason about open terms, it entails that contextual equivalence is substitutive, and it entails accurate characterisations of contextual equivalence on closed expressions.

Secondly, in §4.6–4.7, new simulation rules for contextual equivalence are introduced. They are akin to "bisimulation up to context" proof rules for process calculi (Sangiorgi 1994) and applicative bisimulation (Pitts 1995; Sands 1998b; Lassen 1998). They enable us to establish a recursion induction principle and to prove that contextual equivalence is closed under open extension. The latter validates  $\eta_v$ -equivalence and entails the CIU Theorem.

Thirdly, in §4.8, Howe's method is used to prove an Operational Extensionality Theorem which is a stronger result than the CIU Theorem.

In fact, it would have been less laborious to start with the Operational Extensionality Theorem and therefrom derive the results of the previous stages. But the unorthodox presentation of the theory of deterministic contextual equivalence here clarifies how strong assumptions each of the key properties of contextual equivalence makes about the language. This is relevant because the CIU Theorem and operational extensionality do not hold in various "impure" language extensions. In particular, operational extensionality does not extend to any of the nondeterministic extensions in Part II. In the presence of fairness in Chapter 8, even the CIU Theorem fails.

#### 4.1 Contextual approximation and equivalence

Let us recall the definitions of adequacy and contextual equivalence from §3.7.

The set of all adequate open relations is

$$ADEQ = \{ R \in REL \mid \forall a, a'. \ a \ R \ a' : unit \Rightarrow (a \rightsquigarrow \langle \rangle \Leftrightarrow a' \rightsquigarrow \langle \rangle) \}$$

Contextual equivalence,  $\cong \in REL$ , is the largest compatible and adequate relation. It is well defined by reference to Lemma 3.7.1 and it is symmetric because adequacy is closed under inversion,  $R^{\text{op}} \in ADEQ$  whenever  $R \in ADEQ$ .

In many cases it is useful to operate with a *contextual approximation* preorder whose symmetrisation is the contextual equivalence relation. We first define the associated notion of *pre-adequacy* as the set of open relations

$$PREADEQ = \{ R \in REL \mid \forall a, a'. a \ R \ a' : unit \Rightarrow (a \rightsquigarrow \langle \rangle \Rightarrow a' \rightsquigarrow \langle \rangle) \}$$

The identity relation is pre-adequate,  $Id \in PREADEQ$ , and pre-adequacy is closed under non-empty unions and relation composition. Let contextual approximation,  $\subseteq REL$ , be the the largest compatible and pre-adequate relation. Again, this exists according to Lemma 3.7.1.

Our definitions of contextual approximation and equivalence are equivalent to the more conventional definitions in terms of variable capturing contexts: whenever  $\vec{x} : \vec{t} \vdash a : t$  and  $\vec{x} : \vec{t} \vdash a' : t$ ,

$$\vec{x}: \vec{t} \vdash a \sqsubseteq a': t \quad \text{iff} \\ \forall \xi, C. \quad \xi: (\vec{t})t \vdash C: \text{unit} \quad \Rightarrow \quad (C\llbracket(\vec{x})a/\!\!/\xi\rrbracket \rightsquigarrow \langle \rangle \Rightarrow \quad C\llbracket(\vec{x})a'/\!\!/\xi\rrbracket \rightsquigarrow \langle \rangle)$$
(4.1)  
$$\vec{x}: \vec{t} \vdash a \cong a': t \quad \text{iff}$$

$$\forall \xi, C. \ \xi : (\vec{t})t \vdash C : \text{unit} \ \Rightarrow \ (C\llbracket(\vec{x})a/\!\!/\xi\rrbracket \rightsquigarrow \langle \rangle \ \Leftrightarrow \ C\llbracket(\vec{x})a'/\!\!/\xi\rrbracket \rightsquigarrow \langle \rangle) \tag{4.2}$$

See Proposition 3.7.2.

#### 4.2 Simulation

Most relational arguments about operational semantics are of the form that one term simulates the operational behaviour of another. In order to formulate such operational simulations, we introduce a simulation operator,  $\langle \cdot \rangle$  that maps any open relation R into the closed relation  $\langle R \rangle$  given by:

$$a \langle R \rangle a' : t \stackrel{\text{def}}{\Leftrightarrow} \forall u. \ a \rightsquigarrow u \ \Rightarrow \ \exists u'. \ a' \rightsquigarrow u' \ \& \ u \ \overline{R} \ u' : t$$

$$(4.3)$$

for all a, a': t. The simulation operator is monotone.

We call  $\mathbf{R} \in REL_0$  a (closed) simulation if it is a post-fixed point of  $\langle \cdot \circ \rangle$ , i.e.,  $\mathbf{R} \subseteq \langle \mathbf{R}^{\circ} \rangle$ , and  $S \in REL$  is an open simulation if it is a post-fixed point of  $\langle \cdot \rangle^{\circ}$ .

Every simulation is pre-adequate.

Suppose **R** is a simulation,  $\mathbf{R} \subseteq \langle \mathbf{R}^{\circ} \rangle$ . To see the role of matching values and open extension in the definition of  $\langle \mathbf{R}^{\circ} \rangle$ , it is instructive to expand the definition at function types  $t_1 \rightharpoonup t_2,$ 

$$\begin{array}{ll} a \langle \mathbf{R}^{\circ} \rangle \ a' : t_1 \rightharpoonup t_2 & \text{iff} \\ \forall (x)b. \ a \rightsquigarrow \lambda x. b \ \Rightarrow \ \exists (x)b'. \ a' \rightsquigarrow \lambda x. b' \ \& \ \forall u : t_1. \ b[u/x] \ \mathbf{R} \ b'[u/x] : t_2 \end{array}$$

hence, related functions at type  $t_1 \rightarrow t_2$  are related by R on all closed values u of the argument type  $t_1$ . At ground types t, we observe that

$$a \langle \mathbf{R}^{\circ} \rangle a' : t \quad \text{iff} \quad \forall u. \ a \rightsquigarrow u \Rightarrow a' \rightsquigarrow u$$

because  $\overline{\mathbf{R}^{\circ}}$  is the identity relation on values at ground types, cf. (3.5). At compound types, e.g.,  $(t_1 \rightarrow t_2)$  list, we have

$$\begin{array}{ll} a \langle \boldsymbol{R}^{\circ} \rangle \ a' : (t_1 \rightharpoonup t_2) \ \text{list} & \text{iff} \\ \forall n \ge 0. \ \forall (x_1) b_1, \dots, (x_n) b_n. \ a \rightsquigarrow \mathbf{cons} \langle \lambda x_1. \ b_1, \dots \mathbf{cons} \langle \lambda x_n. \ b_n, \mathbf{nil} \rangle \dots \rangle \\ \exists (x_1) b'_1, \dots, (x_n) b'_n. \ a \rightsquigarrow \mathbf{cons} \langle \lambda x_1. \ b'_1, \dots \mathbf{cons} \langle \lambda x_n. \ b'_n, \mathbf{nil} \rangle \dots \rangle \\ \forall u : t_1. \ b_1[u/x] \ \boldsymbol{R} \ b'_1[u/x] : t_2 \ \& \ \dots \ \& \ b_n[u/x] \ \boldsymbol{R} \ b'_n[u/x] : t_2 \end{array}$$

hence, related lists have the same length and their elements are pairwise related.

#### 4.3Kleene approximation and equivalence

We will now show a number of basic results about contextual equivalence which we derive from a simple-minded equivalence relation, called Kleene equivalence; it is included in contextual equivalence by a straightforward relational proof. The proofs make no reference to the CIU Theorem and operational extensionality.

Let Kleene approximation,  $\preceq$ , and equivalence,  $\asymp$ , be the closed relations defined by

$$\begin{array}{lll} a \preceq a':t & \stackrel{\mathrm{def}}{\Leftrightarrow} & \forall u. \ a \rightsquigarrow u \ \Rightarrow \ a' \rightsquigarrow u \\ a \asymp a':t & \stackrel{\mathrm{def}}{\Leftrightarrow} & \forall u. \ a \rightsquigarrow u \ \Leftrightarrow \ a' \rightsquigarrow u \end{array}$$

for all a, a': t. Notice that  $\leq \langle Id \rangle$  and that  $\approx$  is the symmetrisation of  $\leq$ .

As a paradigmatic example of "relational reasoning" let us prove that the open extension of Kleene approximation is included in contextual approximation.

**Proposition 4.3.1**  $\leq^{\circ} \subseteq \Box$ .

**Proof** We are going to show that  $\preceq^{\circ SC}$  is an open simulation. Then  $\preceq^{\circ SC}$  is pre-adequate and, as it is compatible, it is included in contextual approximation, the largest compatible and pre-adequate relation, and the result follows because  $\preceq^{\circ} \subseteq \preceq^{\circ SC}$ . By (3.6),  $\preceq^{\circ SC}$  is an open simulation,  $\preceq^{\circ SC} \subseteq \langle \preceq^{\circ SC} \rangle^{\circ}$ , if and only if

$$\preceq^{\circ \mathsf{SC}}[Id]_{\mathbf{0}} \subseteq \langle \preceq^{\circ \mathsf{SC}} \rangle$$

Furthermore, since  $\preceq^{\circ SC}$  is closed under substitutions,  $\preceq^{\circ SC}[Id] = (\preceq^{\circ SC})$ , we have that

$$\preceq^{\circ \mathsf{SC}} [Id]_{\mathbf{0}} = (\preceq^{\circ \mathsf{SC}})_{\mathbf{0}}$$

So, expanding the definition of the simulation operator (4.3), we find that we must show that

$$a \preceq^{\circ \mathsf{SC}} a' : t \& a \rightsquigarrow v \Rightarrow \exists v'. a' \rightsquigarrow v' \& v \preceq^{\circ \mathsf{SC}} v' : t$$

The proof is by induction on the derivation of  $a \rightsquigarrow v$ .

First observe that, by Lemma 3.8.1, there exists a'' : t such that  $a \leq \circ SC a'' : t$  and  $a'' \leq \circ a' : t$ . Since a'' and a' are closed, the latter holds if and only if  $a'' \leq a' : t$ . By the definition of Kleene approximation, it suffices to show that there exists v' such that  $a'' \rightsquigarrow v'$  and  $v \leq \circ SC v' : t$ . Then  $a' \rightsquigarrow v'$  too, as required.

We proceed by analysis of the derivation of  $a \rightsquigarrow v$ .

#### **Case (Eval value)** $a \rightsquigarrow v$ because a = v.

Since  $a \stackrel{\frown}{\preceq} \circ^{\mathsf{SC}} a'' : t$ , we get that a'' is a value and  $a \stackrel{\frown}{\preceq} \circ^{\mathsf{SC}} a'' : t$  from Lemma 3.5.1 and (3.11). So let v' = a''.

**Case (Eval redex)**  $a \rightsquigarrow v$  because  $a \rightarrow b$  and  $b \rightsquigarrow v$ .

From  $a \preceq \circ^{\mathsf{SC}} a'' : t$  and Lemma 3.10.1, we get that  $a'' \to b'$  with  $b \preceq \circ^{\mathsf{SC}} b' : t$ . By the induction hypothesis,  $b' \rightsquigarrow v'$  for some v' such that  $v \preceq \circ^{\mathsf{SC}} v' : t$ . By (Eval redex),  $a'' \rightsquigarrow v'$  too.

**Case (Eval let)**  $a \rightsquigarrow v$  because  $a = \text{let } x = a_1 \text{ in } b_2, a_1 \rightsquigarrow u$ , and  $b_2[u/x] \rightsquigarrow v$ .

Since  $a \stackrel{\frown}{\leq} \overset{\frown}{sC} a''$ : t, the latter must be of the form  $a'' = \operatorname{let} x = a'_1$  in  $b'_2$  where  $a_1 \stackrel{\frown}{\leq} \overset{\circ}{sC} a'_1$ : t' and  $x : t' \vdash b_2 \stackrel{\frown}{\leq} \overset{\circ}{sC} b'_2$ : t. By the induction hypothesis,  $a'_1 \rightsquigarrow u'$  with  $u \stackrel{\frown}{\leq} \overset{\circ}{sC} u'$ : t'. Since  $\stackrel{\frown}{\leq} \overset{\circ}{sC}$  is substitutive,  $b_2[u/x] \stackrel{\frown}{\leq} \overset{\circ}{sC} b'_2[u'/x]$ : t, so, by the induction hypothesis,  $b'_2[u'/x] \rightsquigarrow v'$  for some v' such that  $v \stackrel{\frown}{\leq} \overset{\circ}{sC} v'$ : t. Finally,  $a'' \rightsquigarrow v'$ , by (Eval let).

Since Kleene equivalence and contextual equivalence are both symmetrisations of the corresponding approximation preorders, it follows that  $\simeq^{\circ} \subseteq \simeq$  as well.

The following example shows how contextual equivalence reflects that the language is deterministic and history-insensitive (outcomes are reproducible).

**Example 4.3.2** Let  $u, u' : ((unit \rightarrow bool) \rightarrow bool)$  be the functions

 $u = \lambda f.$  if  $f \langle \rangle$  then true else true  $u' = \lambda f.$  if  $f \langle \rangle$  then  $f \langle \rangle$  else true

Let us show that they are contextually equivalent. By compatibility, this holds if

f: unit  $\rightarrow$  bool  $\vdash$  if  $f \langle \rangle$  then true else true  $\cong$  if  $f \langle \rangle$  then  $f \langle \rangle$  else true : bool

By Proposition 4.3.1, it suffices to show that

 $\mathbf{if} \,\, v \, \langle \, \rangle \,\, \mathbf{then} \,\, \mathbf{true} \,\, \mathbf{else} \,\, \mathbf{true} \rightsquigarrow v' \,\, \Leftrightarrow \,\, \mathbf{if} \,\, v \, \langle \, \rangle \,\, \mathbf{then} \,\, v \, \langle \, \rangle \,\, \mathbf{else} \,\, \mathbf{true} \rightsquigarrow v'$ 

for all functions  $v : (unit \rightarrow bool)$  and Boolean values v'. This follows from the observations that (1) each evaluates to **true** if and only if  $v \langle \rangle$  evaluates to any Boolean value, **true** or **false**; and (2) neither evaluates to **false**, in the latter case because the evaluation of  $v \langle \rangle$  is deterministic, Proposition 2.2.4.

Proposition 4.3.1 is quite a useful result about contextual approximation and equivalence. It leads to easy proofs of many commonly used equational and inequational laws, e.g., that any diverging expression is a *syntactic bottom* element in the contextual approximation order,

$$\Gamma \vdash a \sqsubset a' : t, \quad \text{if } \neg \exists v. \ a \rightsquigarrow v, \ \Gamma \vdash a : t \text{ and } \Gamma \vdash a' : t \tag{4.4}$$

and that the evaluation and transition relations and their open extensions are *sound* with respect to contextual equivalence:

$$\rightsquigarrow^{\circ} \subseteq \cong$$
 (4.5)

$$\rightarrow^{\circ} \subseteq \cong$$
 (4.6)

In particular,  $\beta_v$ -equivalence is valid for contextual equivalence,

$$\Gamma \vdash (\lambda x. a) v \cong a[v/x] : t, \text{ if } \Gamma, x : t' \vdash a : t \text{ and } \Gamma \vdash v : t'$$

$$(4.7)$$

with the important consequence that contextual approximation and equivalence are substitutive:

**Proposition 4.3.3** Contextual approximation and equivalence are substitutive.

**Proof** Suppose  $\Gamma, \vec{x}: \vec{t} \vdash a \sqsubset a': t$  and  $\Gamma \vdash \vec{u} \boxdot \vec{u}': \vec{t}$ . We must show that

$$\Gamma \vdash a[\vec{u}/\vec{x}] \sqsubset a'[\vec{u}'/\vec{x}] : t \tag{4.8}$$

By weakening,  $\Gamma, x_1, \ldots, x_{i-1} \vdash u_i \overline{\sqsubset} u'_i : t_i$ , for  $i \in 1..n$ , and by compatibility,

$$\Gamma \vdash (\lambda x_1. \ldots (\lambda x_n. a) u_n \ldots) u_1 \sqsubset (\lambda x_1. \ldots (\lambda x_n. a') u'_n \ldots) u'_1 : t$$

Now (4.8) follows by  $\beta_v$ -equivalence and transitivity.

Substitutivity of contextual equivalence follows easily.

Contextual approximation satisfies a "canonical freeness" property (Gordon 1994), here formulated in terms of matching values:

**Proposition 4.3.4**  $u \sqsubset u' : t$  if and only if  $u \overleftarrow{\subseteq} u' : t$ .

**Proof** By compatibility,  $u \sqsubseteq u' : t$  implies  $u \sqsubseteq u' : t$ . We prove the converse by induction on the typing derivation of u : t.

**Case (Type fun)**  $t = t_1 \rightarrow t_2$ ,  $u = \lambda x. a$ , and  $x : t_1 \vdash a : t_2$ . Then u' must also be a function  $u' = \lambda x. a'$  with  $x : t_1 \vdash a' : t_2$ . Now  $x : t_1 \vdash a \sqsubset a' : t_2$  follows by the calculation:

$$\begin{array}{rcl} x:t_1 \vdash a &\asymp^{\circ} & \operatorname{let} y = u \text{ in } y \, x \\ & \sqsubseteq & \operatorname{let} y = u' \text{ in } y \, x \\ & \stackrel{\scriptstyle \sim}{\scriptstyle \sim} & a':t_2 \end{array} \text{ by compatibility}$$

because  $\asymp^{\circ} \subseteq \Box$ , by Proposition 4.3.1, and  $\Box$  is transitive. Hence  $u \,\overline{\Box} \, u' : t$ , by (Match fun).

**Case (Type product)**  $t = t_1 \times \ldots \times t_n$ ,  $u = \langle u_1, \ldots, u_n \rangle$ , and  $u_1 : t_1, \ldots, u_n : t_n$ . Then u' must also be of the form  $u' = \langle u'_1, \ldots, u'_n \rangle$  with  $u'_1 : t_1, \ldots, u'_n : t_n$ . For each  $i \in 1..n$ ,

$$u_i \ \asymp \ \mathbf{let} \ y = u \ \mathbf{in} \ \mathbf{case} \ y \ \mathbf{of} \ \langle x_1, \dots, x_n \rangle. \ x_i$$
$$\ \sqsubseteq \ \ \mathbf{let} \ y = u' \ \mathbf{in} \ \mathbf{case} \ y \ \mathbf{of} \ \langle x_1, \dots, x_n \rangle. \ x_i$$
$$\ \asymp \ \ u'_i : t_i$$

We deduce that  $u_i \sqsubset u'_i : t_i$  and then, by the induction hypothesis,  $u_i \boxdot u'_i : t_i$ . Hence we obtain that  $u \sqsubseteq u' : t$ , by (Match product).

**Case (Type sum)**  $t = \mu \chi \cdot t_1 + \ldots + t_n, u = \mathbf{inj}_i v, i \in 1..n, \text{ and } v : t_i[t/\chi]$ . Then

 $\begin{array}{rcl} v &\asymp & \operatorname{let} y = u \text{ in case } y \text{ of } \operatorname{inj}_1 x_1 . \ \Omega & [] \ \dots \ [] \ \operatorname{inj}_i x_i . x_i \ [] \ \dots \ [] \ \operatorname{inj}_n x_n . \ \Omega \\ & \sqsubset & \operatorname{let} y = u' \text{ in case } y \text{ of } \operatorname{inj}_1 x_1 . \ \Omega \ [] \ \dots \ [] \ \operatorname{inj}_i x_i . x_i \ [] \ \dots \ [] \ \operatorname{inj}_n x_n . \ \Omega : t_i[t/\chi] \end{array}$ 

Since  $\sqsubset$  is pre-adequate, the latter terminates, so u' must be of the form  $u' = \mathbf{inj}_i v'$  with  $v' : t_j[t/\chi]$ . We see that  $v \asymp \sqsubset \varkappa' : t_i[t/\chi]$  and thus  $v \sqsubset v' : t_i[t/\chi]$ . By the induction hypothesis, this implies  $v \bar{\sqsubseteq} v' : t_i[t/\chi]$ . Hence  $u \bar{\sqsubseteq} u' : t$ , by (Match sum).

The proposition does not hold for open values. For instance,  $x : \text{unit} \vdash x \subseteq \langle \rangle : \text{unit but}$  not  $x : \text{unit} \vdash x \subseteq \langle \rangle : \text{unit.}$ 

Proposition 4.3.4 characterises contextual approximation between closed values. We can also characterise contextual approximation and equivalence between closed non-values in various ways. For instance, the equivalence class of  $\Omega$  is the set of all diverging expressions,

$$a \cong \mathbf{\Omega} : t \quad \text{iff} \quad \neg \exists v. \ a \rightsquigarrow v, \quad \text{if} \quad a : t$$

$$(4.9)$$

This property of  $\cong$  is called "complete adequacy" in Meyer (1988).

Suppose we define the *outcome* of a well typed closed expression a to be its value if evaluation terminates (recall that evaluation is deterministic) and  $\Omega$  if it diverges. Then complete adequacy (4.9) and the soundness of evaluation (4.5) assert that a is contextually equivalent to its outcome. Gordon (1994) calls this "Strachey's property":

either 
$$a \cong \mathbf{\Omega} : t$$
 or  $\exists u. \ a \cong u : t$ , if  $a : t$  (4.10)

Two useful characterisations of closed contextual approximation can be derived from syntactic bottom (4.4) and the soundness of evaluation (4.5).

**Proposition 4.3.5** If a, a' : t then (i), (ii) and (iii) are equivalent,

- (i)  $a \sqsubset a' : t$
- $(ii) \ \forall u. \ a \rightsquigarrow u \ \Rightarrow \ \exists u'. \ a' \rightsquigarrow u' \ \& \ u \sqsubseteq u': t$
- $(iii) \ \forall E. \ \bullet : t \vdash E : unit \ \Rightarrow \ (E[\![a]\!] \rightsquigarrow \langle \rangle \ \Rightarrow \ E[\![a']\!] \rightsquigarrow \langle \rangle)$

**Proof** That (i) implies (iii) is immediate from (4.1).

Next, suppose (*iii*) holds and let us prove (*ii*). If  $a \rightsquigarrow u$  then  $(a; \langle \rangle)$  terminates, so, by assumption,  $(a'; \langle \rangle)$  terminates as well. Hence there is a value u' such that  $a' \rightsquigarrow u'$ . Now, let C be any closed unit context of type  $\bullet : t \vdash C$ : unit and suppose that C[[u]] terminates. Then

 $E[\![a]\!]$  terminates, where  $E = \text{let } x = \bullet$  in  $C[\![x]\!]$ . By assumption,  $E[\![a']\!]$  terminates because E is an evaluation context. As evaluation is deterministic and factors through evaluation contexts (2.1), this means that  $C[\![u']\!]$  terminates. As C was chosen arbitrarily, (4.1) tells us that  $u \sqsubset u' : t$ .

Finally, to see that (ii) implies (i), suppose a and a' satisfies (ii). If a diverges, (i) holds by syntactic bottom (4.4). Otherwise,  $a \rightsquigarrow u$  for some u and then we know that  $a' \rightsquigarrow u'$  for some u' such that  $u \sqsubset u' : t$ . By the soundness of evaluation (4.5), we get that  $a \cong u \sqsubset u' \cong a' : t$ . We conclude (i) by transitivity.  $\Box$ 

Propositions 4.3.4 and 4.3.5 imply that  $\subseteq_{\mathbf{0}} = \langle \subseteq \rangle$ . Furthermore,  $\subseteq \subseteq (\subseteq_{\mathbf{0}})^{\circ}$  because  $\subseteq$  is closed under substitutions, by reflexivity and substitutivity. Therefore contextual approximation is an open simulation,

$$\Box \subseteq \left\langle \Box \right\rangle^{\circ} \tag{4.11}$$

It follows that  $\leq^{\circ}$  is an exact characterisation of contextual approximation for ground types (i.e., types built from products and recursive sums only).

**Lemma 4.3.6**  $\Gamma \vdash a \subseteq a' : t$  if and only if  $\Gamma \vdash a \preceq^{\circ} a' : t$ , whenever t is a ground type.

**Proof** The backward implication holds for arbitrary types t by Proposition 4.3.1. If t is a ground type,  $\overline{\subseteq}|_t$  is the identity relation on values of type t, by (3.5), and therefore  $\langle \subseteq \rangle$  coincides with Kleene approximation at type t (i.e.,  $\langle \subseteq \rangle|_t = \preceq|_t$ ). Now the forward implication of the lemma follows because  $\subseteq$  is an open simulation, (4.11).

Of course, the same holds for contextual equivalence and Kleene equivalence in place of contextual approximation and Kleene approximation, respectively.

#### 4.4 Sequentiality

For some purposes it is useful to operate with relations of arity greater than two, as introduced in §3.9. In this section we illustrate how this approach can be used to prove program equivalences that depend on the sequential nature of the language. The examples are originally due to Dana Scott and Pierre-Louis Curien and were rephrased for call-by-value by Riecke and Sandholm (1997). Our proofs are adapted from those in Sieber (1992) and Riecke and Sandholm (1997), based on logical relations. The purpose of recasting the proofs syntactically here is to give another illustration of how our relational framework makes it possible to manage complex syntactic arguments, technically and notationally. The reader is encouraged to compare our proof of the first example with that of Plotkin (1977), elaborated in Mitchell (1996), which is phrased in terms of explicit contexts. A further motivation for the material in this section is that it will be important for assessing and comparing different operational equivalence relations in the nondeterministic setting of Part II.

The first example shows that there is no parallel convergence tester function, of type  $(unit \rightarrow unit) \times (unit \rightarrow unit) \rightarrow unit$ , which terminates if either of its function arguments does and diverges if they both diverge. This is the simplest parallel function one can think of for our language and it plays the same role as the parallel or function for PCF (Plotkin 1977). It is well known that the non-definability of such parallel functions is reflected by contextual equivalence and, moreover, that relations of higher arity are useful for reasoning about sequentiality (Sieber 1992; O'Hearn and Riecke 1995; Riecke and Sandholm 1997).

**Example 4.4.1** Let  $u, u' : ((unit \rightarrow unit) \times (unit \rightarrow unit) \rightarrow unit) \rightarrow unit be the two functions$ 

$$\begin{array}{rcl} u & = & \lambda f. \left( f \left< \mathbf{I}, \mathbf{U} \right>; \ f \left< \mathbf{U}, \mathbf{I} \right> \right) \\ u' & = & \lambda f. f \left< \mathbf{U}, \mathbf{U} \right> \end{array}$$

where  $\mathbf{I} = \lambda x. x$  and  $\mathbf{U} = \lambda x. \Omega$ . They are contextually equivalent,

$$u \cong u' : ((\text{unit} \rightarrow \text{unit}) \times (\text{unit} \rightarrow \text{unit}) \rightarrow \text{unit}) \rightarrow \text{unit}$$

It is easy to see that u' approximates u, by Proposition 4.3.1 and compatibility of contextual approximation.

The other direction, that u approximates u', also holds by Proposition 4.3.1 and compatibility if, for all functions  $v : ((unit \rightarrow unit) \times (unit \rightarrow unit)) \rightarrow unit)$ ,

$$(v \langle \mathbf{I}, \mathbf{U} \rangle; v \langle \mathbf{U}, \mathbf{I} \rangle) \rightsquigarrow \langle \rangle \Rightarrow v \langle \mathbf{U}, \mathbf{U} \rangle \rightsquigarrow \langle \rangle$$

This is equivalent to

$$v \langle \mathbf{I}, \mathbf{U} \rangle \rightsquigarrow \langle \rangle \quad \& \quad v \langle \mathbf{U}, \mathbf{I} \rangle \rightsquigarrow \langle \rangle \implies v \langle \mathbf{U}, \mathbf{U} \rangle \rightsquigarrow \langle \rangle \tag{4.12}$$

Intuitively, v cannot apply any of the two function components of its input tuple—that would force one of  $v \langle \mathbf{I}, \mathbf{U} \rangle$  and  $v \langle \mathbf{U}, \mathbf{I} \rangle$  to diverge—therefore v must terminate on any input. We can make this argument precise by means of the ternary relation  $\mathbf{S}^{\mathsf{C}}$ , where  $\mathbf{S}$  is the closed ternary relation defined by

$$oldsymbol{S}ert_{(ec{t})t} \;\; \stackrel{ ext{def}}{=} \;\; \{(\langle\,
angle, oldsymbol{\Omega}, oldsymbol{\Omega}), (oldsymbol{\Omega}, \langle\,
angle, oldsymbol{\Omega}) \;|\; t = ext{unit}\}$$

for every abstraction type  $(\vec{t})t$  (i.e., S is a two-point closed relation, relating  $\langle \rangle, \Omega, \Omega$  and relating  $\Omega, \langle \rangle, \Omega$  at type unit, and is empty at all types other than unit). As S is closed,  $S^{\mathsf{C}}$  is substitutive. We are going to prove that

$$(a_1, a_2, a') \in \mathbf{S}^{\mathsf{C}}|_t \& a_1 \rightsquigarrow u_1 \& a_2 \rightsquigarrow u_2 \Rightarrow \\ \exists u'. a' \rightsquigarrow u' \& (u_1, u_2, u') \in \overline{\mathbf{S}^{\mathsf{C}}}|_t$$

$$(4.13)$$

which implies (4.12) because  $(v \langle \mathbf{I}, \mathbf{U} \rangle, v \langle \mathbf{U}, \mathbf{I} \rangle, v \langle \mathbf{U}, \mathbf{U} \rangle) \in \mathbf{S}^{\mathsf{C}}|_{\text{unit}}$  and  $(\langle \rangle, \langle \rangle, \langle \rangle) \in \overline{\mathbf{S}^{\mathsf{C}}}|_{\text{unit}}$ .

The proof of (4.13) is by induction on the derivation of  $a_1 \rightsquigarrow u_1$  and by analysis of the derivation of  $(a_1, a_2, a') \in \mathbf{S}^{\mathsf{C}}|_t$ .

If the latter is derived by (Ctx S), then  $a_1 = \langle \rangle$  and  $a_2 = U$  or  $a_1 = U$  and  $a_2 = \langle \rangle$ ; in any case, one of  $a_1 \rightsquigarrow u_1$  and  $a_2 \rightsquigarrow u_2$  is false, so (4.13) holds vacuously.

Otherwise,  $(a_1, a_2, a') \in \widehat{S}^{\hat{C}}|_t$  and the argument proceeds by analysis of the derivation of  $a_1 \rightsquigarrow u_1$ . The argument for each case is similar to that in the proof of Proposition 4.3.1 but with some extra bookkeeping for the extra component in the relation.

**Case (Eval value)** We use the obvious generalisation of Lemma 3.5.1 to relations of arbitrary arity.

**Case (Eval redex)** An easy application of Lemma 3.10.2 and the induction hypothesis.

**Case (Eval let)**  $a_1 \rightsquigarrow u_1$  because  $a_1$  is of the form  $a_1 = \text{let } x = a_{10}$  in  $b_{10}$  and  $a_{10} \rightsquigarrow v_1$  and  $b_{10}[v_1/x] \rightsquigarrow u_1$  for some value  $v_1$ .

Since  $(a_1, a_2, a') \in \widehat{\mathbf{S}^{\mathsf{C}}}|_t$ , we have that  $a_2$  and a' are of the form  $a_2 = \mathbf{let} \ x = a_{20}$  in  $b_{20}$ and  $a' = \mathbf{let} \ x = a'_0$  in  $b'_0$  with  $(a_{10}, a_{20}, a'_0) \in \mathbf{S}^{\mathsf{C}}|_{t'}$  and  $((x)b_{10}, (x)b_{20}, (x)b'_0) \in \mathbf{S}^{\mathsf{C}}|_{(t')t}$ . Then  $a_2 \rightsquigarrow u_2$  must be derived by (Eval let) from  $a_{20} \rightsquigarrow v_2$  and  $b_{20}[v_2/x] \rightsquigarrow u_2$  for some value  $v_2$ .

We can now apply the induction hypothesis and get that  $a'_0 \rightsquigarrow v'$  for some v' such that  $(v_1, v_2, v') \in \overline{S^{\mathsf{C}}}|_{t'}$ .

By substitutivity,  $(b_{10}[v_1/x], b_{20}[v_2/x], b'_0[v'/x]) \in \mathbf{S}^{\mathsf{C}}|_t$ . Finally, another application of the induction hypothesis gives us that  $b'_0[v'/x] \rightsquigarrow u'$  with  $(u_1, u_2, u') \in \overline{\mathbf{S}^{\mathsf{C}}}|_t$ , and by (Eval let) we get that  $a' \rightsquigarrow u'$ , as required.

**Example 4.4.2** Let  $t_0 = (\text{unit} \rightarrow \text{bool})$  and let  $a_1, a_2, a_3, a_4$  be the expressions defined by

$$\begin{array}{rcl} a_1 &=& \mathbf{if} \ h_2 \left< \right> \mathbf{then} \ \left< \right> \mathbf{else} \ \mathbf{if} \ h_1 \left< \right> \mathbf{then} \ \left< \right> \mathbf{else} \ \mathbf{\Omega} \\ a_2 &=& \mathbf{if} \ h_1 \left< \right> \mathbf{then} \ \mathbf{\Omega} \ \mathbf{else} \ \left< \right> \\ a_3 &=& \mathbf{if} \ h_2 \left< \right> \mathbf{then} \ \mathbf{\Omega} \ \mathbf{else} \ \left< \right> \\ a_4 &=& \mathbf{\Omega} \end{array}$$

and for each  $i \in 1..4$ , let  $g_i : (t_0 \times t_0) \rightarrow$  unit denote the function  $g_i = \lambda \langle h_1, h_2 \rangle$ .  $a_i$ .

We will prove the contextual equivalence:

$$f: (t_0 \times t_0 \to \text{unit}) \to \text{unit} \vdash (f g_1; f g_2; f g_3) \cong f g_4: \text{unit}$$

$$(4.14)$$

By Lemma 4.3.6 and inspection of evaluations we see that this holds if and only if, for all functions  $u : (t_0 \times t_0 \rightarrow \text{unit}) \rightarrow \text{unit}$ ,

$$(u g_1 \rightsquigarrow \langle \rangle \& u g_2 \rightsquigarrow \langle \rangle \& u g_3 \rightsquigarrow \langle \rangle) \Leftrightarrow u g_4 \rightsquigarrow \langle \rangle$$

$$(4.15)$$

The right-to-left implication follows easily from the theory of Kleene approximation and contextual approximation in §4.3. In the other direction, the intuition is that  $g_1$ ,  $g_2$ , and  $g_3$  are constructed so that for any argument  $v : t_0 \times t_0$  at least one of  $g_1 v$ ,  $g_2 v$ , and  $g_3 v$  will diverge. Hence, if u terminates on all of  $g_1$ ,  $g_2$  and  $g_3$ , it must be because it never applies its argument and it terminates on any input. To prove this, we define R as the smallest 4-ary open relation such that

$$h_1: t_0, h_2: t_0 \vdash (a_1, a_2, a_3) R a_4:$$
 unit

The grouping of the first three components into a tuple is just for notational convenience; the notation means that

$$((h_1, h_2)a_1, (h_1, h_2)a_2, (h_1, h_2)a_3, (h_1, h_2)a_4) \in R|_{(t_0, t_0)$$
unit

We also define binary relations  $R_{12}$  and  $R_{13}$ , each obtained from R by projections onto two of the four components; they are the smallest open relations such that

$$h_1: t_0, h_2: t_0 \vdash a_1 \ R_{12} \ a_2:$$
 unit  
 $h_1: t_0, h_2: t_0 \vdash a_1 \ R_{13} \ a_3:$  unit

We prove that their substitutive context closures satisfy that

The proof is by induction on the derivation of  $b_1 \rightsquigarrow u_1$ . First we consider the derivation of  $(b_1, b_2, b_3) R^{\mathsf{SC}} b_4 : t$ .

**Case (SC Subst)** Then  $b_1 = a_1[v_{11}/h_1, v_{12}/h_2], \ldots, b_4 = a_4[v_{41}/h_1, v_{42}/h_2]$ , where  $(v_{1i}, v_{2i}, v_{3i}) \overline{R^{SC}} v_{4i} : t_0 \text{ for } i = 1, 2.$  The evaluation  $b_1 \rightsquigarrow u_1$  must go either via  $v_{12} \langle \rangle \rightsquigarrow \text{true}$ , or via  $v_{12} \langle \rangle \rightsquigarrow \text{false and } v_{11} \langle \rangle \rightsquigarrow \text{true}.$ 

(i) If  $b_2 \rightsquigarrow u_2$  then  $v_{21} \langle \rangle \rightsquigarrow$  false. By compatibility,  $(v_{11} \langle \rangle, v_{21} \langle \rangle, v_{31} \langle \rangle) \overline{R^{\mathsf{SC}}} v_{41} \langle \rangle : t$ , so by part (i) of the induction hypothesis we conclude that the evaluation of  $b_1$  cannot go via  $v_{11} \langle \rangle \rightsquigarrow$  true.

- (*ii*) By a similar argument we see that the evaluation of  $b_1$  cannot go via  $v_{12} \langle \rangle \rightsquigarrow$ true.
- (*iii*) Because of the contradictory conclusions from parts (*i*) and (*ii*), it is evident that not all of  $b_1$ ,  $b_2$  and  $b_3$  can terminate. Therefore (*iii*) holds vacuously.
- **Case (SC Comp)** The argument for this case is by analysis of the derivation of  $b_1 \rightsquigarrow u_1$  and proceeds analogously to the previous example. We omit the details.

This concludes the proof of (4.16).

Since  $(u g_1, u g_2, u g_3) R^{\mathsf{SC}} u g_4$ : unit, we conclude from (4.16)(iii) that  $u g_4$  terminates if all three of  $u g_1, u g_2$  and  $u g_3$  terminate, as required to establish (4.15) and (4.14).

The proofs by induction on evaluations in the two examples are tedious, of course. It might be possible to obtain the examples as instances of a general syntactic proof rule for reasoning about sequentiality. The proof rule could combine Sieber's sequentiality relations with a notion of simulation up to context for relations of arbitrary arity.

### 4.5 Unwinding and syntactic continuity

In this section, we give a relational proof of the Unwinding Theorem and we derive a syntactic continuity property of contextual approximation. Similar results appear already in the work of Morris (1968) and Wadsworth (1971). One motivation for repeating them here is to provide yet another illustration of how our relational framework assists us in managing complex syntactic arguments. Another reason for going over these proofs here is to prepare the ground for our investigations into continuity properties in nondeterministic settings in Part II.

The Unwinding Theorem says that an expression with occurrences of the  $\mathbf{Y}$  combinator terminates if and only if it terminates with some finite unwinding of  $\mathbf{Y}$  in place of  $\mathbf{Y}$ . Let  $\mathbf{Y}^{(n)}$  denote the *n*'th unwinding of  $\mathbf{Y}$ ,

$$\mathbf{Y}^{(n)} \stackrel{\text{def}}{=} \lambda g. \operatorname{fix}^{(n)}[g]$$

where fix<sup>(n)</sup> is defined by induction on  $n < \omega$  as

$$\begin{aligned} & \operatorname{fix}^{(0)}[u] \stackrel{\text{def}}{=} & \mathbf{\Omega} \\ & \operatorname{fix}^{(n+1)}[u] \stackrel{\text{def}}{=} & u\left(\lambda x.\operatorname{fix}^{(n)}[u]x\right) \end{aligned}$$

**Theorem 4.5.1 (Unwinding)** Suppose  $y : (t \rightarrow t) \rightarrow t \vdash a : t'$  and t is a function type. Then  $a[\mathbf{Y}/y]$  terminates if and only if  $a[\mathbf{Y}^{(n)}/y]$  terminates for some  $n < \omega$ .

There is an equational variant of the theorem, called rational openness (Braüner 1996),

 $\Gamma \vdash a[\mathbf{Y}/y] \cong \mathbf{\Omega} : t' \quad \text{iff} \quad \forall n < \omega. \ \Gamma \vdash a[\mathbf{Y}^{(n)}/y] \cong \mathbf{\Omega} : t'$ 

It is equivalent to the Unwinding Theorem because, by (4.9), a closed expression terminates if and only if it is not contextually equivalent to  $\Omega$ .

We split the proof of the theorem into two lemmas. The first lemma establishes the 'if' direction of the theorem which, in effect, asserts that  $\mathbf{Y}$  is an upper bound of its finite unwindings.

**Lemma 4.5.2** Y is a  $\sqsubset$ -upper bound of its finite unwindings  $\{\mathbf{Y}^{(i)} \mid i < \omega\}$ .

**Proof** Suppose  $\Gamma \vdash u : t \to t$  and t is a function type. Then  $\Gamma \vdash fix^{(0)}[u] \sqsubset fix[u] : t$ , by (4.4). Moreover, if  $\Gamma \vdash fix^{(i)}[u] \sqsubset fix[u] : t$ , then

By induction we see that fix[u] is an upper bound of  $\{fix^{(i)}[u] \mid i < \omega\}$  and, by compatibility, **Y** is an upper bound of  $\{\mathbf{Y}^{(i)} \mid i < \omega\}$ .

The other half of the Unwinding Theorem is more challenging. For each  $n < \omega$ , we are going to construct a relation  $W^{(n)}$  which satisfies

$$a[\mathbf{Y}/y] W^{(n)} a[\mathbf{Y}^{(n)}/y] : t'$$
(4.17)

whenever  $y: (t \rightarrow t) \rightarrow t \vdash a: t'$  and t is a function type.

In the course of the proof of the next lemma, this family of relations must be preserved by evaluation in an appropriate sense. To this end we generalise  $W^{(n)}$  to relate expressions with matching occurrences of fix and fix<sup>(m)</sup>, where  $m \ge n$ . We will also need that  $W^{(n)}$  is substitutive. So we take  $W^{(n)}$  to be the substitutive context closure,

$$W^{(n)} \stackrel{\text{def}}{=} \left(\bigcup_{i\geq n} R^{(i)}\right)^{\text{SC}}$$

where each  $R^{(i)}$  is given by

fix 
$$R^{(i)}$$
 fix<sup>(i)</sup> :  $(t \rightarrow t) t$ 

for all function types t. Note that  $\mathbf{Y} \overline{R^{(n)}} \mathbf{Y}^{(n)} : (t \rightarrow t) \rightarrow t$ .

We see that (4.17) holds because  $a[\mathbf{Y}/y] Id[R^{(n)}] a[\mathbf{Y}^{(n)}/y] : t'$ , and because  $W^{(n)}$  includes  $R^{(n)}$  and is reflexive and substitutive.

We observe that the  $W^{(n)}$  relations form a decreasing sequence,

$$W^{(n)} \subseteq W^{(n')} \quad \text{iff} \quad n' \le n \tag{4.18}$$

**Lemma 4.5.3** Whenever a:t and  $a \rightsquigarrow u$ , there exists  $n < \omega$  such that for all  $m < \omega$ ,

$$\forall a': t. \ a \ W^{(m+n)} \ a': t \ \Rightarrow \ \exists u'. \ a' \rightsquigarrow u' \ \& \ u \ \overline{W^{(m)}} \ u': t$$

**Proof** By induction on the derivation of  $a \rightsquigarrow u$ .

#### Cases (Eval value) and (Eval let)

a cannot be a fix expression, so whenever  $a W^{(m+n)} a' : t$ , this must be derived from  $a \widetilde{W^{(m+n)}} a' : t$ . The arguments are similar to those in the proof of Proposition 4.3.1, but extended with appropriate bookkeeping for calculating n. We omit the details.

**Case (Eval redex)**  $a \rightsquigarrow u$  because  $b \rightsquigarrow u$  for some b such that  $a \rightarrow b$ .

We split the argument in two cases according to the form of a.

**Case** a = fix[v] for some  $v: t \rightarrow t$ . Then

$$b = \operatorname{case} (\operatorname{inj} \lambda \operatorname{inj} y. v (\lambda x. y (\operatorname{inj} y) x)) \text{ of inj } y. v (\lambda x. y (\operatorname{inj} y) x)$$

and  $b \rightsquigarrow u$  via (Eval redex) and  $b \rightarrow v (\lambda x. a x) \rightsquigarrow u$ .

By the induction hypothesis for  $v(\lambda x. a x) \rightsquigarrow u$ , there is an  $n_0$  such that

$$\forall m < \omega. \ \forall b'': t. \ v (\lambda x. a x) \ W^{(m+n_0)} \ b'': t \ \Rightarrow \ \exists u'. \ b'' \rightsquigarrow u' \ \& \ u \ \overline{W^{(m)}} \ u': t \ (4.19)$$

Let  $n = n_0 + 1$ . Suppose a  $W^{(m+n)} a' : t$ , i.e., either

$$a R^{(i)}[W^{(m+n)}] a': t$$
 (4.20)

for some  $i \ge m + n$ , or

$$a \widehat{W^{(m+n)}} a': t \tag{4.21}$$

In the first case  $a' = \operatorname{fix}^{(i)}[v']$  with  $v \overline{W^{(m+n)}} v' : t \to t$ . Note that  $i \ge m+n \ge 1$ , so  $a' = v' (\lambda x. \operatorname{fix}^{(i-1)}[v']x)$ . Since  $v \overline{W^{(m+n_0)}} v' : t \to t$ , by (4.18), and  $i-1 \ge m+n_0$ , we get that  $v (\lambda x. ax) W^{(m+n_0)} a' : t$ . By (4.19), the desired conclusion follows.

In the second case, an inspection of the derivation of (4.21) reveals that  $a' = \operatorname{fix}[v']$ with  $v \overline{W^{(m+n)}} v' : t \to t$ . We see that  $a' \rightsquigarrow u'$  whenever  $v'(\lambda x. a' x) \rightsquigarrow u'$ , by two applications of (Eval redex). Moreover,  $v(\lambda x. a x) W^{(m+n_0)} v'(\lambda x. a' x) : t$ . By (4.18) and (4.19), the desired conclusion follows.

**Case** *a* is not of the form fix[*v*] for any *v*. By the induction hypothesis for  $b \rightsquigarrow u$  there exists  $n < \omega$  such that, for all  $m < \omega$ ,  $b W^{(m+n)} b' : t$  implies  $b' \rightsquigarrow u'$  with  $u \overline{W^{(m)}} u' : t$ . Supposing that  $a W^{(m+n)} a' : t$  for some  $m < \omega$  and a' : t, since *a* is not of the form fix[*v*],  $a W^{(m+n)} a' : t$  must be derived from  $a W^{(m+n)} a' : t$  by (SC Comp). Now the result follows easily by Lemma 3.10.1.

**Proof of Theorem 4.5.1** If  $a[\mathbf{Y}/y]$  terminates, according to Lemma 4.5.3 there exists an  $n < \omega$  such that whenever  $a[\mathbf{Y}/y] W^{(n)} a' : t'$  then a' terminates too. In particular,  $a[\mathbf{Y}^{(n)}/y]$  terminates because of (4.17). The reverse implication follows from Lemma 4.5.2.

An important consequence of the Unwinding Theorem is a so-called syntactic continuity property of contextual approximation (Pitts 1997a). We should first remark that the contextual approximation ordering is not  $\omega$ -complete, i.e., there are  $\omega$ -chains of expressions Ain the contextual approximation ordering such that A does not have an upper bound. At function types, this is the case where the least upper bound in a domain-theoretic semantics is not a computable function. Moreover, there are operators in the language, e.g., function application, which are not continuous with respect to the  $\omega$ -chains that do have least upper bounds: there are  $\omega$ -chains  $\{v_i \mid i < \omega\}$  with a least upper bound v but such that u v is not the least upper bound of  $\{(u v_i) \mid i < \omega\}$ ; see Mason, Smith, and Talcott (1996).

Syntactic continuity is a more restricted form of  $\omega$ -completeness and continuity principle enjoyed by contextual approximation and the language operators. It asserts both that **Y** is the *least* upper bound of its finite unwindings and that these least upper bounds are preserved by all language constructs, i.e., by substitution into arbitrary contexts.

**Proposition 4.5.4 (Syntactic continuity)** If  $y : (t \rightarrow t) \rightarrow t \vdash a : t'$  and t is a function type,  $a[\mathbf{Y}/y] \sqsubset a' : t'$  if and only if  $\forall n < \omega$ .  $a[\mathbf{Y}^{(n)}/y] \sqsubset a' : t'$ .

**Proof** The forward implication is immediate from Lemma 4.5.2.

The argument for the reverse implication is straightforward in terms of explicit contexts, by reference to the Unwinding Theorem and (4.1).<sup>1</sup> Nevertheless, let us give a relational proof without reference to explicit contexts. This proof is closer to the argument that is needed to prove syntactic continuity for applicative simulation preorders, as we shall see in Chapter 7.

We must show that  $a[\mathbf{Y}/y] \sqsubset a' : t'$  under the assumption that  $a[\mathbf{Y}^{(n)}/y] \sqsubset a' : t'$  for all  $n < \omega$ . To this end we construct the open relation

$$T \stackrel{\text{def}}{=} \bigcap_{n < \omega} (W^{(n)} \, \varsigma)$$

Observe that  $a[\mathbf{Y}/y] T a' : t'$  if  $\forall n < \omega$ .  $a[\mathbf{Y}^{(n)}/y] \sqsubset a' : t'$ . We will show that T is compatible and pre-adequate, then  $T \subseteq \sqsubset$  and the result follows.

Each  $W^{(n)}$  is compatible, and so is  $\subseteq$ . Since compatibility is closed under relation composition and arbitrary intersections, T is compatible.

Now let us prove that T is pre-adequate. Suppose b T b': unit. If  $b \rightsquigarrow \langle \rangle$  then we know from Lemma 4.5.3 that there exists an  $n < \omega$  such that whenever  $b W^{(n)} b''$ : unit then  $b'' \rightsquigarrow \langle \rangle$ . By the construction of T, we have that  $b W^{(n)} b'' \sqsubset b'$ : unit for some b''. But now we know that  $b'' \rightsquigarrow \langle \rangle$  and, by pre-adequacy of  $\sqsubset$ , we get that  $b' \rightsquigarrow \langle \rangle$  too, as required.  $\Box$ 

By means of syntactic continuity we can prove a recursion induction principle for the **Y** combinator (also known as "Park induction").

Proposition 4.5.5 (Recursion induction)  $\frac{u v' \sqsubset v' : t_1 \rightharpoonup t_2}{\mathbf{Y} u \sqsubset v' : t_1 \rightharpoonup t_2}$ 

<sup>&</sup>lt;sup>1</sup>Although here it is a bit awkward to relate the formulation of the Unwinding Theorem in terms of variable substitution to contexts because we haven't provided the means to go from filling contexts to variable substitution, i.e., we lack a converse to (3.14).

**Proof** Assume that the premise holds. By syntactic continuity it suffices to prove the conclusion for all finite unwindings  $\mathbf{Y}^{(0)}, \mathbf{Y}^{(1)}, \ldots$  in place of  $\mathbf{Y}$ ,

$$\mathbf{Y}^{(n)} u \sqsubset v' : t_1 \rightharpoonup t_2, \text{ for all } n < \omega$$

We argue by induction on *n*. The base case,  $\mathbf{Y}^{(0)} u \sqsubset v' : t_1 \rightharpoonup t_2$ , is immediate by Kleene approximation because  $\mathbf{Y}^{(0)} u$  diverges. The induction step follows by calculation:

(the last  $\beta_v$ -equivalence holds because v' must be of the form  $\lambda x. b$ ).

#### 

#### 4.6 Compatible simulations

This section introduces new simulation rules for contextual equivalence, akin to "bisimulation up to context" proof rules for process calculi (Sangiorgi 1994) and applicative bisimulation (Pitts 1995; Sands 1998b; Lassen 1998). They enable us to establish a recursion induction principle and, in the next section, to derive the CIU Theorem.

The results about sequentiality, unwinding, and syntactic continuity with respect to contextual approximation in the preceding sections were proved by variations of the basic proof principle for contextual approximation used in the proof of Proposition 4.3.1, that is, by (1) constructing a compatible and substitutive relation, (2) showing it to be preserved by evaluation in an appropriate sense (using substitutivity), and (3) concluding that it is pre-adequate and, hence (by compatibility), included in contextual approximation. Although alleviated by the lemmas about primitive reduction from  $\S3.10$ , the proofs of preservation by evaluation involve tedious inductions on derivations of evaluations. In many cases, when preservation of evaluation can be expressed in terms of the simulation operator, the proofs are instances of the following lemma.

Suppose R is a compatible and substitutive relation, then the lemma says that R is included in contextual approximation if

$$R_{\mathbf{0}} \subseteq (\langle R \rangle \cup \widehat{R}) \sqsubset$$

This inclusion combines two common situations:

- (1) R is of the form  $S \cup \widehat{R}$  and  $S_0 \subseteq \langle R \rangle$ , or
- (2)  $R_0 \subseteq \widehat{R} S$  holds for some S such that  $S \subseteq \Box$  is derivable by the results from §4.3.

For illustration, let us sketch how Proposition 4.3.1,  $\leq^{\circ} \subseteq \subset$ , can be derived by two successive applications of the lemma. First, consider  $\leq^{\mathsf{C}}$ . It is compatible and also substitutive because  $\leq$  is closed. Since  $\leq^{\mathsf{C}} = \leq \cup \widehat{\leq^{\mathsf{C}}}$  and  $\leq = \langle Id \rangle \subseteq \langle \leq^{\mathsf{C}} \rangle$  (because  $\leq^{\mathsf{C}}$  is reflexive and  $\langle \cdot \rangle$  is monotone), we have situation (1) with  $R = \leq^{\mathsf{C}}$  and  $S = \leq$ , so according to the lemma contextual approximation includes  $\leq^{\mathsf{C}}$  and thus also  $\leq$ . Secondly, we can use this fact to prove that contextual approximation includes  $\leq^{\circ\mathsf{SC}}$  and thus also  $\leq^{\circ:\mathsf{C}}$  as in the proof of Proposition 4.3.1, we use Lemma 3.8.1 to get that  $(\leq^{\circ\mathsf{SC}})_{\mathbf{0}} \subseteq \widehat{\leq^{\circ\mathsf{SC}}} \cup \leq$ , i.e., we have situation (2) by taking  $R = \leq^{\circ\mathsf{SC}}$  and  $S = \leq$ , and the lemma yields the desired conclusion.

**Lemma 4.6.1** 
$$\frac{R_0 \subseteq (\langle R \rangle \cup R) \sqsubset}{R \subseteq \sqsubset} \quad if \ R \in REL \ is \ compatible \ and \ substitutive.$$

**Proof** Assume that the premise holds. We are going to prove that  $R_0 \subseteq \langle R \rangle \sqsubset$  holds if  $R \in REL$  is substitutive. It implies that R is pre-adequate. Hence, if R is also compatible, it is included in operational approximation,  $R \subseteq \sqsubset$ .

So, assuming that  $R \in REL$  is substitutive, we show that

$$a R a': t \& a \rightsquigarrow v \Rightarrow \exists v': t. v \overline{R} v': t \& v' \sqsubset a': t$$

$$(4.22)$$

by induction on the derivation of  $a \rightsquigarrow v$ . By the premise of the lemma,  $a \ R \ a' : t$  implies that there is an a'' : t such that  $a'' \sqsubset a' : t$  and either  $a \langle R \rangle a'' : t$  or  $a \ \widehat{R} \ a'' : t$ .

If  $a \langle R \rangle a'' : t$ , then  $a \rightsquigarrow v$  implies that  $a'' \rightsquigarrow v'$  for some v' : t such that  $v \overline{R} v' : t$ . Hence  $v' \cong a'' : t$ , and we conclude that  $v' \sqsubset a' : t$ , by transitivity.

If  $a \widehat{R} a'' : t$ , we proceed by analysis of the derivation of  $a \rightsquigarrow v$ .

**Case (Eval value)**  $a \rightsquigarrow v$  because a = v.

Since  $a \widehat{R} a'' : t$ , we get that a'' is a value and  $a \overline{R} a'' : t$ , from Lemma 3.5.1 and (3.11). So let v' = a''.

**Case (Eval redex)**  $a \rightsquigarrow v$  because  $a \rightarrow b$  and  $b \rightsquigarrow v$ .

From  $a \hat{R} a'' : t$  and Lemma 3.10.1, we get that  $a'' \to b'$  with b R b' : t. By the induction hypothesis,  $v \overline{R} v' : t$  and  $v' \subseteq b' : t$ , for some v' : t. The primitive reduction relation is included in contextual equivalence, by (Trans redex) and (4.6), so  $b' \cong a'' : t$ , and we obtain that  $v' \subseteq a' : t$ , by transitivity.

**Case (Eval let)**  $a \rightsquigarrow v$  because  $a = \text{let } x = a_1 \text{ in } b_2, a_1 \rightsquigarrow u$ , and  $b_2[u/x] \rightsquigarrow v$ .

Since  $a \ \widehat{R} \ a'' : t$ , a'' must be of the form  $a'' = \mathbf{let} \ x = a'_1 \ \mathbf{in} \ b'_2$  where  $a_1 \ R \ a'_1 : t'$  and  $x : t' \vdash b_2 \ R \ b'_2 : t$ . By the induction hypothesis, there exists u' : t' such that  $u \ \overline{R} \ u' : t'$  and  $u' \sqsubset a'_1 : t'$ . By substitutivity,  $b_2[u/x] \ R \ b'_2[u'/x] : t$ . So, by the induction hypothesis, there is a v' : t such that  $v \ \overline{R} \ v' : t$  and  $v' \sqsubset b'_2[u'/x] : t$ . Finally, we calculate

$$b'_2[u'/x] \sqsubseteq \operatorname{let} x = u' \operatorname{in} b'_2$$
 by (Trans let beta) and (4.6)  
 $\sqsubset \operatorname{let} x = a'_1 \operatorname{in} b'_2$  because  $\sqsubset$  is compatible  
 $= a''$ 

Hence  $v' \sqsubset a' : t$ , by transitivity.

We derive the following "simulation up to context and contextual approximation" proof rule,

$$\frac{\boldsymbol{S} \subseteq \langle \boldsymbol{S}^{\mathsf{C}} \rangle \sqsubset}{\boldsymbol{S} \subseteq \sqsubset}$$

$$(4.23)$$

if we let  $R = S^{\mathsf{C}}$  in Lemma 4.6.1.

**Example 4.6.2** The rule (4.23) yields an alternative proof of recursion induction, Proposition 4.5.5, for fixed points of "total" functionals of the form  $\lambda f. v.$  (Recursion in call-by-value languages is commonly of this restricted form.)

$$\frac{(\lambda f. v) v' \sqsubset v' : t_1 \rightarrow t_2}{\mathbf{Y} (\lambda f. v) \sqsubset v' : t_1 \rightarrow t_2}$$

Assume that the premise holds. It suffices to prove that  $fix[\lambda f. v] \sqsubset v' : t_1 \rightharpoonup t_2$  because  $\mathbf{Y}(\lambda f. v)$  is  $\beta_v$ -equivalent to  $fix[\lambda f. v]$ .

We construct the singleton closed relation S given by

$$\operatorname{fix}[\lambda f. v] \boldsymbol{S} v' : t_1 \rightharpoonup t_2$$

We need to prove that  $S \subseteq \Box$ . We calculate

$$\begin{aligned} \mathsf{fix}[\lambda f. v] & \rightsquigarrow & v[\lambda x. \mathsf{fix}[\lambda f. v] \, x/f] \\ & \overline{\mathbf{S}^{\mathsf{C}}} & v[\lambda x. v' \, x/f] & \text{by (3.14) and (3.11)} \\ & \cong & (\lambda f. v) (\lambda x. v' \, x) & \beta_v \\ & \cong & (\lambda f. v) \, v' & \beta_v \\ & & \bigtriangledown & v': t_1 \rightharpoonup t_2 & \text{by assumption} \end{aligned}$$

(the last  $\beta_v$ -equivalence holds because v' must be of the form  $\lambda x. b$ ). The calculation shows that  $\mathbf{S} \subseteq \langle \mathbf{S}^{\mathsf{C}} \rangle \sqsubset$ . By (4.23),  $\mathbf{S} \subseteq \sqsubset$  and hence fix $[\lambda f. v] \sqsubset v' : t_1 \rightharpoonup t_2$ , as required.  $\Box$ 

The inability to establish the full recursion induction rule from Proposition 4.5.5 in the previous example indicates a limitation suffered by proof rules based on evaluation semantics. They force us to reason about computations in terms of entire evaluations while, in some cases, we only know the first steps of the computation and need to draw conclusions from those only. An alternative is to base the proof rules on the transition semantics. This is indeed a very expressive formalism and many of the ideas presented here were first developed in terms of transitions. Nevertheless, in order to simplify presentation, proof rules based on transitions are not included in the present exposition. However, let us mention without proof just one rule which seems difficult to mimic with evaluation semantics:

$$\frac{\boldsymbol{S} \subseteq \rightarrowtail^* \widehat{\boldsymbol{S}^{\mathsf{C}}} \boldsymbol{\Xi}}{\boldsymbol{S} \subseteq \boldsymbol{\Xi}}$$

It generalises (4.23) because a little calculation shows that  $(\langle S^{\mathsf{C}} \rangle \Box) \subseteq (\rightarrow^* \widehat{S^{\mathsf{C}}} \Box)$ . With this transition-based proof rule in place of (4.23), it is straightforward to amend the argument in the previous example to validate full recursion induction in the form of Proposition 4.5.5.

#### 4.7 The CIU Theorem

We end this chapter by proving some well-known extensionality properties of contextual approximation and equivalence, the CIU theorem and an operational extensionality theorem.

**Lemma 4.7.1** Contextual approximation is closed under open extension,  $(\sqsubseteq_0)^\circ \subseteq \sqsubseteq$ .

**Proof** By definition of open extension,  $(\sqsubseteq_0)^\circ$  is closed under substitutions. Therefore  $(\eqsim_0)^{\circ SC} \subseteq (\eqsim_0)^{\circ SC} (\eqsim_0)^\circ$ , by Lemma 3.8.1, and hence  $(\eqsim_0)^{\circ SC} \subseteq \eqsim$ , by Lemma 4.6.1. The result follows because  $(\eqsim_0)^\circ \subseteq (\eqsim_0)^{\circ SC}$ .

In fact, we have an equality,  $(\sqsubseteq_0)^\circ = \sqsubset$ , because contextual approximation is closed under substitutions, being substitutive and reflexive. Consequently, the reverse inclusion of (4.11) holds, that is, contextual approximation is a fixed point of  $\langle \cdot \circ \rangle$ ,

But at this stage we are not able to prove that it is the largest fixed point. This we postpone till §4.8.

**Example 4.7.2** We can now prove that  $\eta_v$ -equivalence is valid for contextual equivalence,

$$\vec{x}: \vec{t} \vdash v \cong \lambda y. v y: t \rightarrow t', \text{ if } \vec{x}: \vec{t} \vdash v: t \rightarrow t'$$

By the previous proposition it suffices to prove every closed instance of the law, that is, for arbitrary closed values  $\vec{u}: \vec{t}$  substituted for  $\vec{x}$ . Then  $v[\vec{u}/\vec{x}] = \lambda y. b$ , for some b of type  $y: t \vdash b: t'$ , and  $(\lambda y. v y)[\vec{u}/\vec{x}] = \lambda y. (v[\vec{u}/\vec{x}]) y = \lambda y. (\lambda y. b) y$ . We get that  $y: t_1 \vdash b \cong (\lambda y. b) y: t_2$ , by  $\beta_v$ -equivalence, and the result follows by compatibility.

Proposition 4.3.5 and Lemma 4.7.1 yield a CIU Theorem for contextual approximation. The acronym stands for Closed Instantiations (two expressions are related if all their closed instantiations are) of Uses (termination is observed only in evaluation contexts rather than arbitrary variable capturing contexts). We will call a relation R closed under evaluation contexts if

$$\Gamma \vdash a \ R \ a' : t \quad \text{implies} \quad \Gamma \vdash E\llbracket a \rrbracket \ R \ E\llbracket a' \rrbracket : t'$$

whenever E is an evaluation context and  $\bullet: t \vdash E: t'$ .

**Theorem 4.7.3 (CIU)** Contextual approximation is the largest pre-adequate open relation which is closed under evaluation contexts and substitutions.

**Proof** Contextual approximation is pre-adequate and closed under evaluation context by definition. It is closed under substitutions because it is reflexive and substitutive.

Let R be any other pre-adequate open relation which is closed under evaluation contexts and substitutions. It is easy to see that all closed a and a', related by a R a' : t, satisfy Proposition 4.3.5(*iii*) and hence that  $a \sqsubset a' : t$ . That is,  $R_0 \subseteq \sqsubset_0$ . The result follows by calculation:

$$\begin{array}{rcl} R & \subseteq & (R_{\mathbf{0}})^{\circ} & \text{ because } R \text{ is closed under substitutions} \\ & \subseteq & (\Box_{\mathbf{0}})^{\circ} & \text{ open extension is monotone} \\ & \subseteq & \Box & \text{ Lemma 4.7.1} \end{array}$$

Let us rephrase the theorem to see that it is equivalent to the formulation by Mason and Talcott (1991). We define a *CIU context*,  $\mathcal{E}$ , to be any one-holed closed unit context,  $\xi : (\vec{t})t \vdash \mathcal{E}$ : unit, of the form  $E[\![\xi[\vec{u}]]\!]$ , that is,  $\mathcal{E}$  substitutes closed values  $\vec{u}$  for the hole's parameters and places it in an evaluation context E of type unit. The CIU Theorem asserts that, whenever  $\vec{x}: \vec{t} \vdash a: t$  and  $\vec{x}: \vec{t} \vdash a': t$ ,

$$\vec{x}: \vec{t} \vdash a \sqsubset a': t \quad \text{iff} \\ \forall \text{ CIU contexts } \mathcal{E}. \quad \xi: (\vec{t})t \vdash \mathcal{E}: \text{ unit } \Rightarrow (\mathcal{E}\llbracket(\vec{x})a/\!\!/\xi\rrbracket \rightsquigarrow \langle \rangle \Rightarrow \mathcal{E}\llbracket(\vec{x})a'/\!\!/\xi\rrbracket \rightsquigarrow \langle \rangle)$$

$$(4.25)$$

This is a more tractable characterisation than (4.1). For example, Propositions 4.3.1 and 4.3.5 and Lemma 4.7.1 all follow from (4.25).

#### 4.8 Similarity

In this section we define an applicative similarity preorder, prove that it is a pre-congruence by Howe's method, and derive an operational extensionality property of contextual approximation. Finally, the interrelationship with the CIU Theorem is discussed. Howe's proof is well-known but we include it because it is one of the main sources of inspiration for the relational approach in this dissertation. Moreover, we will extend the proof to various forms of nondeterminism in Chapters 7 and 8, and it may be instructive first to see the proof here in a deterministic setting.

Let similarity,  $\leq \in REL_0$ , be defined co-inductively as the greatest fixed point of  $\langle \cdot \circ \rangle$ .

$$\lesssim \stackrel{\mathrm{def}}{=} \nu \boldsymbol{R}. \langle \boldsymbol{R}^{\circ} \rangle$$

The compound operator  $\langle \cdot \circ \rangle$  is a monotone function on the complete lattice of closed relations, so a greatest fixed point exists and it is also the greatest post-fixed point. This means that similarity is the greatest simulation.

A simple co-inductive argument shows that open similarity,  $\leq^{\circ}$ , is the greatest fixed point of  $\langle \cdot \rangle^{\circ}$ , that is,  $\leq^{\circ} = \nu R \cdot \langle R \rangle^{\circ}$ . Therefore open similarity is the greatest open simulation. For instance, open similarity includes contextual approximation because the latter is an open simulation, (4.11). The reverse inclusion holds if open similarity is compatible, because we know that it is pre-adequate as it is itself an open simulation. Let us prove that open similarity is a pre-congruence, by means of Howe's method (Howe 1996).

The main lemma says that the compatible extension of the reflexive transitive closure of any simulation is itself a simulation.

**Lemma 4.8.1** If  $\mathbf{R} \subseteq \langle \mathbf{R}^{\circ} \rangle$  then  $\mathbf{R}^{*\bullet} \subseteq \langle \mathbf{R}^{*\bullet} \rangle^{\circ}$ .

**Proof** The proof resembles that of Proposition 4.3.1.

Assume that R is a simulation,  $R \subseteq \langle R^{\circ} \rangle$ . Since  $R^{*\bullet}$  is closed under substitutions it suffices to show that

$$a \mathbf{R}^{*\bullet} a' : t \& a \rightsquigarrow v \Rightarrow \exists v' : t. a' \rightsquigarrow v' \& v \overline{\mathbf{R}^{*\bullet}} v' : t$$

The proof is by induction on the derivation of  $a \rightsquigarrow v$ .

First observe that, by the definition of compatible extension, there exists a'': t such that  $a \widehat{\mathbf{R}^{*\bullet}} a'': t$  and  $a'' \mathbf{R}^* a': t$ . It suffices to show that

$$\exists v'': t. \ a'' \rightsquigarrow v'' \quad \& \quad v \ \overline{R^{*\bullet}} \ v'': t \tag{4.26}$$

The proof of (4.26) proceeds by analysis of the derivation of  $a \rightsquigarrow v$ . All the cases are identical to those in the proof of Proposition 4.3.1 (but with  $\mathbf{R}^{*\bullet}$  in place of  $\leq^{\circ \mathsf{SC}}$  and v'' in place of v').

From this lemma we derive that open similarity is a pre-congruence.

**Proposition 4.8.2**  $\leq^{\circ}$  is a pre-congruence.

**Proof** Since  $\leq$  is a simulation, Lemma 4.8.1 entails that  $\leq^{*\bullet}$  is an open simulation and is thus included in  $\leq^{\circ}$ . The reverse inclusion holds by the definition of compatible extension, so we get that  $\leq^{\circ}$  is compatible since  $\leq^{*\bullet}$  is. Finally, the calculation  $\leq^{\circ^*} = \leq^{*\circ} \subseteq \leq^{*\bullet} \subseteq \leq^{\circ}$  shows that  $\leq^{\circ}$  is reflexive and transitive.

**Theorem 4.8.3 (Operational extensionality)** Contextual approximation coincides with open similarity, i.e., it is the greatest fixed point of  $\langle \cdot \rangle^{\circ}$ .

This is a stronger characterisation than the CIU Theorem, (4.25). The CIU Theorem does not seem to be of any assistance for proving the operational extensionality theorem (essentially, it is no easier to show that similarity is closed under evaluation contexts than to show that open similarity is closed under arbitrary contexts).<sup>2</sup> Conversely, the CIU Theorem was derived from Proposition 4.3.5 and Lemma 4.7.1 which, in turn, can be derived from the operational extensionality theorem as follows. First, from the operational extensionality theorem we easily derive syntactic bottom (4.4) and the soundness of evaluation (4.5) which were used to prove Proposition 4.3.5. Second, the operational extensionality theorem implies that closed contextual approximation coincides with closed similarity, hence the open extension of closed contextual approximation coincides with open similarity which, again by the operational extensionality theorem, coincides with contextual equivalence, so in particular Lemma 4.7.1 holds.

Essentially, the CIU Theorem says that contextual approximation is a fixed point of the simulation operator  $\langle \cdot \rangle^{\circ}$ , (4.24), whereas the operational extensionality theorem says that it is the greatest fixed point. The latter leads to a co-induction principle for reasoning about infinite behaviour of higher-order functions, as illustrated in the next example.

**Example 4.8.4** Let a and a' be the recursive functions

$$a = \mathbf{Y} \lambda f. \lambda y. \mathbf{inj} f$$
  
$$a' = \mathbf{Y} \lambda f. \lambda y. \mathbf{inj} \lambda z. \mathbf{inj} f$$

of type  $a, a': t \rightarrow t'$  for any closed type t and  $t' \stackrel{\text{def}}{=} \mu \chi. t \rightarrow \chi$ . When applied, each function returns a recursive function with the same behaviour as itself. They are contextually equivalent,

$$a \cong a': t \rightharpoonup t' \tag{4.27}$$

<sup>&</sup>lt;sup>2</sup>The proof that the CIU Theorem implies the operational extensionality theorem in Mason, Smith, and Talcott (1996, Lemma 3.8) is incorrect. (Scott Smith, personal communication, June 1997.)

By the operational extensionality theorem we can prove this by exhibiting simulations that relate a to a' and a' to a. Let R be the smallest open relation such that

$$a \ R \ a': t \rightarrow t'$$

$$x: t \vdash \ \mathbf{inj} \ \lambda x. \operatorname{fix}[\lambda f. \ \lambda y. \mathbf{inj} \ f] \ x \ R \ \mathbf{inj} \ \lambda z. \mathbf{inj} \ \lambda x. \operatorname{fix}[\lambda f. \ \lambda y. \mathbf{inj} \ \lambda z. \mathbf{inj} \ f] \ x: t'$$

$$x: t \vdash \ \operatorname{fix}[\lambda f. \ \lambda y. \mathbf{inj} \ f] \ x \ R \ \mathbf{inj} \ \lambda x. \operatorname{fix}[\lambda f. \ \lambda y. \mathbf{inj} \ \lambda z. \mathbf{inj} \ f] \ x: t'$$

$$x: t \vdash \ \operatorname{fix}[\lambda f. \ \lambda y. \mathbf{inj} \ f] \ x \ R \ \operatorname{fix}[\lambda f. \ \lambda y. \mathbf{inj} \ \lambda z. \mathbf{inj} \ f] \ x: t'$$

It is straightforward to check that R and  $R^{\text{op}}$  are open simulations. So, by co-induction, R is included in contextual equivalence. Since a and a' are related by R, we conclude that they are contextually equivalent, (4.27).

The CIU theorem does not seem to be of any use for proving (4.27), so this example illustrates how the CIU Theorem is weaker than operational extensionality. However, (4.27) can be proved by other proof rules developed in §4.5 and §4.6. First, observe that (4.27) is an instance of a more general law about recursive functions:

$$\mathbf{Y}\,\lambda f.\,v \cong \mathbf{Y}\,\lambda f.\,v[v/f]: t_1 \rightharpoonup t_2, \quad \text{if} \quad f: t_1 \rightharpoonup t_2 \vdash v: t_1 \rightharpoonup t_2 \tag{4.28}$$

One direction, that  $\mathbf{Y} \lambda f. v$  contextually approximates  $\mathbf{Y} \lambda f. v[v/f]$ , is easily proved by recursion induction and the other direction by syntactic continuity. The law can also be established by the simulation up to context rule (4.23). Interestingly, operational extensionality does not seem applicable for proving (4.28). So, although operational extensionality subsumes the CIU Theorem and is useful for reasoning about infinite behaviour of higher-order functions, it is no replacement for the induction principles and simulation up to context principles developed in §4.5 and §4.6 which, in general, seem to be better suited for reasoning about recursion.

# Part II

# Nondeterminism

### Chapter 5

# A Nondeterministic Functional Language

In the remainder of the dissertation, the relational techniques and results about deterministic operational equivalences in Chapter 4 are extended to different kinds of nondeterminism. First, in Chapters 6 and 7, the finitely branching and countable branching sequential nondeterminism arising from erratic choice and countable choice combinators is considered.

This chapter defines the syntax and operational semantics of erratic and countable choice. The evaluation semantics comes in two flavours: a *may* evaluation relation which specifies possible outcomes and a *must* relation which relates a program with its set of outcomes if and only if all the program's computation paths terminate.

The interrelationships between the may and the must operational semantics and the deterministic operational semantics from Chapter 2 will be made precise, and the properties of the deterministic operational semantics with respect to typing and relations are generalised to the nondeterministic counterparts. All the results are straightforward to establish and the proofs are omitted.

#### 5.1 Syntax

We shall consider two nondeterministic extensions of our functional language. One is erratic choice,  $a_1$  or  $a_2$ , between two expressions. The other is countable choice, ?, of an arbitrary natural number (in the unary representation of numbers from §2.1.1).

$$\begin{array}{cccc} a,b & ::= & \dots \\ & & | & a_1 \text{ or } a_2 \\ & & | & ? \end{array}$$

The two constructs introduce bounded (finitely branching) and unbounded (countably branching) nondeterminism, respectively. The latter is more general, of course; erratic choice,  $a_1$  or  $a_2$ , can be encoded using countable choice as

let 
$$x = ?$$
 in case  $x$  of  $0. a_1$  or succ  $x. a_2$  (5.1)

where x is not free in  $a_1$  and  $a_2$ .

The most faithful encoding of countable choice, ?, using erratic choice is

$$(\mathbf{Y} \lambda f. \lambda \langle \rangle. \mathbf{0} \text{ or } (\mathbf{let} \ y = f \langle \rangle \mathbf{in} \mathbf{succ} \ y)) \langle \rangle$$
(5.2)

but it differs from ? with respect to termination behaviour: the encoding admits an infinite transition sequence (by always choosing the right hand side of **or**). This difference is significant for one of the semantic equivalences which we will look at.

#### 5.2 Operational semantics

The operational semantics comes in two flavours, a may modality,  $\diamondsuit$ , and a must modality,  $\Box$ .

The may modality is the most obvious extension of the deterministic operational semantics. Whereas the deterministic primitive reduction, evaluation and transition relations,  $\rightarrow$ ,  $\rightsquigarrow$ , and  $\rightarrowtail$ , are partial functions, their nondeterministic may counterparts, written  $\rightarrow_{\Diamond}$ ,  $\rightsquigarrow_{\Diamond}$ , and  $\succ_{\Diamond}$ , are proper relations that relate some expressions to multiple expressions and values. They are given in Tables 5.1–5.3.

The first rule (Redex $\diamond$ !) defines the may primitive reduction relation  $\rightarrow \diamond$  as a superset of the deterministic primitive reduction relation  $\rightarrow$ . The only differences from the deterministic relations are the other rules, (Redex $\diamond$  or) and (Redex $\diamond$  choose), for may primitive reduction of erratic and countable choice. The definitions of the may evaluation and transition relations,  $\rightsquigarrow_{\diamond}$  and  $\rightarrowtail_{\diamond}$ , are the same as those of the deterministic relations,  $\rightsquigarrow$  and  $\succ_{\diamond}$ , but with may primitive reduction  $\rightarrow_{\diamond}$  in place of deterministic primitive reduction  $\rightarrow$ .

The may evaluation and transition relations are related like the deterministic ones (Proposition 2.2.6),

$$a \rightsquigarrow_{\Diamond} u \quad \text{iff} \quad a \rightarrowtail_{\Diamond}^{*} u \tag{5.3}$$

We say that a closed expression a may terminate if there exists u such that  $a \rightsquigarrow_{\Diamond} u$ . Otherwise a must diverge.

The operational relations of the must modality relate expressions to sets of expressions and values. The must primitive reduction, evaluation and transition relations, written  $\rightarrow_{\Box}$ ,  $\rightsquigarrow_{\Box}$ , and  $\succ_{\Box}$ , are given in Tables 5.4–5.6; A and B range over sets of closed expressions; U and V range over sets of closed values.

The must primitive reduction and transition relations,  $\rightarrow_{\Box}$  and  $\succ_{\Box}$ , relate expressions to the sets of successor expressions determined by the corresponding may relations,

$$a \to_{\Box} B \quad \text{iff} \quad \emptyset \neq B = \{b \mid a \to_{\Diamond} b\}$$

$$(5.4)$$

$$a \rightarrowtail_{\Box} B \quad \text{iff} \quad \emptyset \neq B = \{b \mid a \rightarrowtail_{\Diamond} b\} \tag{5.5}$$

So the must relations are deterministic,

$$a \to_{\Box} B$$
 and  $a \to_{\Box} B'$  imply  $B = B'$  (5.6)

$$a \mapsto_{\Box} B$$
 and  $a \mapsto_{\Box} B'$  imply  $B = B'$  (5.7)

In other words, they are partial functions.

So far, the must relations are just a convenient reformulation of the corresponding may relations. But the must evaluation relation,  $\rightsquigarrow_{\Box}$ , defined inductively by the rules in Table 5.5,
$$(\operatorname{Redex}_{\diamond} \, !) \qquad \qquad a \to_{\diamond} b, \text{ if } a \to b$$

(Redex<sub>$$\diamond$$</sub> or)  $a_1$  or  $a_2 \rightarrow_{\diamond} a_i$ , if  $i = 1, 2$ 

 $(\operatorname{Redex}_{\diamond} \operatorname{choose}) \qquad ? \rightarrow_{\diamond} \ulcorner i \urcorner, \text{ if } i < \omega$ 

Table 5.1: May primitive reduction relation

let x = a in  $b \rightsquigarrow_{\Diamond} v$ 

# Table 5.2: May evaluation relation

$$\begin{array}{ll} (\mathrm{Trans}_{\Diamond} \ \mathrm{redex}) & a \rightarrowtail_{\Diamond} b, \ \mathrm{if} \ a \rightarrow_{\Diamond} b \\ (\mathrm{Trans}_{\Diamond} \ \mathrm{let} \ \mathrm{beta}) & \mathbf{let} \ x = u \ \mathbf{in} \ b \rightarrowtail_{\Diamond} b[u\!/\!x] \end{array}$$

$$(\text{Trans}_{\Diamond} \text{ let left}) \quad \frac{a \rightarrowtail_{\Diamond} a'}{\text{let } x = a \text{ in } b \rightarrowtail_{\Diamond} \text{let } x = a' \text{ in } b}$$

#### Table 5.3: May transition relation

$$(\operatorname{Redex}_{\Box} !) \qquad a \to_{\Box} \{b\}, \text{ if } a \to b$$

$$(\text{Redex}_{\Box} \text{ or}) \qquad a_1 \text{ or } a_2 \rightarrow_{\Box} \{a_1, a_2\}$$

 $(\operatorname{Redex}_\square \text{ choose}) \quad \ ? \to_\square \{ \ulcorner i \urcorner \mid i < \omega \}$ 

Table 5.4: Must primitive reduction relation

$$\begin{array}{ll} (\operatorname{Eval}_{\Box} \ \operatorname{value}) & v \rightsquigarrow_{\Box} \{v\} \\ \\ (\operatorname{Eval}_{\Box} \ \operatorname{redex}) & \frac{\forall b \in B. \ b \rightsquigarrow_{\Box} V_b}{a \rightsquigarrow_{\Box} \bigcup_{b \in B} V_b} \ \operatorname{if} \ a \rightarrow_{\Box} B \\ \\ (\operatorname{Eval}_{\Box} \ \operatorname{let}) & \frac{a \rightsquigarrow_{\Box} U \quad \forall u \in U. \ b[u/x] \rightsquigarrow_{\Box} V_u}{\operatorname{let} x = a \ \operatorname{in} \ b \rightsquigarrow_{\Box} \bigcup_{u \in U} V_u} \end{array}$$

$$\begin{array}{ll} (\operatorname{Trans}_{\Box} \operatorname{redex}) & a \rightarrowtail_{\Box} B, \text{ if } a \rightarrow_{\Box} B \\ (\operatorname{Trans}_{\Box} \operatorname{let} \operatorname{beta}) & \mathbf{let} \ x = u \ \mathbf{in} \ b \rightarrowtail_{\Box} \{ b[u/x] \} \end{array}$$

$$(\operatorname{Trans}_{\Box} \operatorname{let} \operatorname{left}) \quad \frac{a \rightarrowtail_{\Box} A}{\operatorname{let} x = a \text{ in } b \rightarrowtail_{\Box} \{\operatorname{let} x = a' \text{ in } b \mid a' \in A\}}$$

# Table 5.6: Must transition relation

is complementary to the may evaluation relation,  $\rightsquigarrow_{\Diamond}$ . The two cannot be derived from each other. The meaning of  $a \rightsquigarrow_{\Box} U$  is that a must terminate (every  $\diamond$ -transition sequence from a terminates) and U is the set of values that a may evaluate to. We have that

$$a \rightsquigarrow_{\Box} U$$
 implies  $\emptyset \neq U = \{u \mid a \rightsquigarrow_{\Diamond} u\}$  (5.8)

The reverse implication is false because a may evaluate to a non-empty set of values U even though it may diverge, in which case the must evaluation relation is undefined on a.

So the must evaluation relation is deterministic in the sense that there is at most one set U of outcomes that a must evaluate to,

$$a \rightsquigarrow_{\Box} U$$
 and  $a \rightsquigarrow_{\Box} U'$  imply  $U = U'$  (5.9)

We say that a closed expression a must terminate if there exists a set U such that  $a \rightsquigarrow_{\Box} U$ . Otherwise a may diverge. From (5.8) we get that if a must terminate then it may terminate.

Corresponding to (5.3) we have that the must evaluation relation is the smallest relation closed under (Eval<sub> $\Box$ </sub> value) and the following generalisation of (Eval<sub> $\Box$ </sub> redex)

$$\frac{\forall b \in B. \ b \rightsquigarrow_{\Box} V_b}{a \rightsquigarrow_{\Box} \bigcup_{b \in B} V_b} \text{ if } a \rightarrowtail_{\Box} B \tag{5.10}$$

#### Determinism and bounded nondeterminism

For expressions a: t without occurrences of the two nondeterministic constructs, we observe that

$$a \to_{\Diamond} b \quad \text{iff} \quad a \to b \tag{5.11}$$

$$a \rightsquigarrow_{\Diamond} u \quad \text{iff} \quad a \rightsquigarrow u \tag{5.12}$$

$$a \mapsto_{\Diamond} b \quad \text{iff} \quad a \mapsto b \tag{5.13}$$

for all b and u. In particular, we see that the may relations are deterministic for such a, and

$$a \to_{\Box} B$$
 iff  $a \to b \& B = \{b\}$  (5.14)

$$a \rightsquigarrow_{\Box} U$$
 iff  $a \rightsquigarrow u \& U = \{u\}$  (5.15)

$$a \rightarrowtail_{\Box} B$$
 iff  $a \rightarrowtail b \& B = \{b\}$  (5.16)

for all b and u and sets B and U.

If a:t does not contain any occurrences of countable choice, but possibly contains occurrences of erratic choice, then a has only a finite set of outcomes in any of the must relations (i.e., the sets B and U above are finite rather than just singletons). For the primitive reduction relation and the transition relation this is immediate by inspection of the reduction and transition rules. For the evaluation relation, we see that any derivation tree is finitely branching and hence the tree is finite, by König's lemma. But a can have an infinite number of outcomes in the may evaluation relation, i.e., the set  $\{u \mid a \rightsquigarrow_{\Diamond} u\}$  can be infinite, as in (5.2), for example. In that case, a may diverge, according to (5.8). In other words, both erratic choice and countable choice generate unbounded nondeterminism. An accurate way of describing the difference between them is that countable choice exhibits unbounded nondeterminism arising in finite time whereas erratic choice can only exhibit unbounded nondeterminism arising in infinite time; cf. Kumar and Pandya (1993).

# 5.3 Types

The typing rules for erratic choice and countable choice are:

(Type or) 
$$\frac{\Gamma \vdash a_1 : t \quad \Gamma \vdash a_2 : t}{\Gamma \vdash a_1 \text{ or } a_2 : t}$$
(Type choose) 
$$\Gamma \vdash ? : \text{nat}$$

The type system's weakening and value substitution properties from §2.3, Proposition 2.3.1, are preserved.

The nondeterministic operational semantics satisfy type preservation and soundness properties corresponding to those of the deterministic operational semantics from Chapter 2. Firstly, for the may operational relations, Lemmas 2.3.4 and 2.3.7 are easily extended to the primitive may reduction relation, by inspection of reductions and type derivations. Then Propositions 2.3.5, 2.3.6 and 2.3.8 carry over to the may evaluation and transition relations; the proofs from §2.3.3 need no modifications other than the replacement of the may operational relations for the deterministic ones. Finally, the results for the may operational relations together with (5.4), (5.5) and (5.8) give us the appropriate type preservation and soundness properties for the must operational relations.

# 5.4 Relations

The definition of compatible refinement, Table 3.2, is extended by two extra rules to cover the new syntactic constructs.

(Comp or) 
$$\frac{\Gamma \vdash a_1 \ R \ a'_1 : t \quad \Gamma \vdash a_2 \ R \ a'_2 : t}{\Gamma \vdash a_1 \ \mathbf{or} \ a_2 \ \widehat{R} \ a'_1 \ \mathbf{or} \ a'_2 : t}$$

(Comp choose)  $\Gamma \vdash ? \hat{R} ? : t$ 

Lemma 3.10.1 holds for the may primitive reduction relation and for relations which are both reflexive and substitutive. Reflexivity is needed for the case (Redex $\diamond$  choose).

**Lemma 5.4.1** Suppose  $R \in REL$  is reflexive and substitutive. Whenever  $a \hat{R} a' : t$  and  $a \rightarrow_{\Diamond} b$  there exists b' : t such that  $a' \rightarrow_{\Diamond} b'$  and b R b' : t.

The lemma generalises to relations of arbitrary arity as in Lemma 3.10.2.

An analogous lemma holds for must primitive reduction. First notice that the set B is countable whenever  $a \to_{\Box} B$ , so we can always write B in the form  $\{b_i \mid i < \omega\}$ . For instance, if  $a \to b$  then  $a \to_{\Box} \{b_i \mid i < \omega\}$  with  $b_i = b$  for all  $i < \omega$ .

**Lemma 5.4.2** Suppose  $R \in REL$  is reflexive and substitutive. Whenever  $a \widehat{R} a' : t$  and  $a \to_{\Box} \{b_i \mid i < \omega\}$  there exists  $b'_0, b'_1, \cdots : t$  such that  $a' \to_{\Box} \{b'_i \mid i < \omega\}$  and  $b_i R b'_i : t$  for all  $i < \omega$ .

Lemma 5.4.2 entails Lemma 5.4.1 because of the correspondence (5.4) between the may and must primitive reduction relations.

# Chapter 6

# Contexts

The may and must modalities of the nondeterministic operational semantics from the previous chapter give rise to two different nondeterministic generalisations of contextual approximation and equivalence. This chapter shows interrelationship between the different contextual relations, and generalises results from the deterministic case in Chapter 4 to the may and must contextual relations. Most of the deterministic theory is shown to carry over in appropriately modified form. Each form of contextual approximation enjoys a CIU Theorem, a recursion induction principle, and simulation up to context proof rules. Furthermore, syntactic continuity is valid for may contextual approximation and, in the absence of countable choice, it is valid for must contextual approximation as well. In the presence of countable nondeterminism, syntactic  $\omega$ -continuity is invalid for must contextual approximation, as one would expect. This is addressed here by a novel transfinite syntactic continuity principle. Finally, the chapter gives results about the interplay between sequentiality and bounded and countable nondeterminism, relative to must contextual equivalence.

# 6.1 Contextual approximation and equivalence

Given the may and the must extensions of the deterministic operational semantics, there are two natural formulations of adequacy for nondeterminism. We say that an open relation Ris *may-adequate* if, for all a, a': unit,

$$a \ R \ a' : \text{unit} \quad \text{implies} \quad a \rightsquigarrow_{\Diamond} \langle \rangle \iff a' \rightsquigarrow_{\Diamond} \langle \rangle \tag{6.1}$$

We say that an open relation R is *must-adequate* if, for all a, a': unit,

$$a \ R \ a' :$$
 unit implies  $a \rightsquigarrow_{\Box} \{\langle \rangle\} \Leftrightarrow a' \rightsquigarrow_{\Box} \{\langle \rangle\}$  (6.2)

Let  $ADEQ_{\Diamond}$  and  $ADEQ_{\Box}$  denote the sets of all may- and must-adequate open relations,

$$\begin{aligned} ADEQ_{\diamond} &= \{R \in REL \mid \forall a, a' : \text{unit. } a \ R \ a' : \text{unit} \Rightarrow (a \rightsquigarrow_{\diamond} \langle \rangle \Leftrightarrow a' \rightsquigarrow_{\diamond} \langle \rangle) \} \\ ADEQ_{\Box} &= \{R \in REL \mid \forall a, a' : \text{unit. } a \ R \ a' : \text{unit} \Rightarrow (a \rightsquigarrow_{\Box} \{\langle \rangle\} \Leftrightarrow a' \rightsquigarrow_{\Box} \{\langle \rangle\}) \} \end{aligned}$$

We also define corresponding notions of pre-adequacy, may-pre-adequacy  $PREADEQ_{\diamond}$ and must-pre-adequacy  $PREADEQ_{\diamond}$ ,

$$\begin{aligned} PREADEQ_{\diamond} &= \{R \in REL \mid \forall a, a' : \text{unit. } a \; R \; a' : \text{unit} \Rightarrow (a \rightsquigarrow_{\diamond} \langle \rangle \Rightarrow a' \rightsquigarrow_{\diamond} \langle \rangle) \} \\ PREADEQ_{\Box} &= \{R \in REL \mid \forall a, a' : \text{unit. } a \; R \; a' : \text{unit} \Rightarrow (a \rightsquigarrow_{\Box} \{\langle \rangle\} \Rightarrow a' \rightsquigarrow_{\Box} \{\langle \rangle\}) \} \end{aligned}$$

The two forms of pre-adequacy and adequacy satisfy the conditions of Lemma 3.7.1. Therefore we may define corresponding contextual approximation and equivalence relations as follows. We define may contextual approximation,  $\Box_{\diamond}$ , to be the largest compatible and may-pre-adequate open relation. The symmetrisation, may contextual equivalence,  $\cong_{\diamond}$ , is the largest compatible and may-adequate open relation. Analogously, let must contextual approximation,  $\Box_{\Box}$ , be the largest compatible and must-pre-adequate open relation, and let must contextual equivalence,  $\cong_{\Box}$ , be the symmetrisation; it is also the largest compatible and must-adequate open relation.

The approximation relations are both pre-congruences (compatible preorders) and the equivalence relations are congruences. As in the deterministic case, our definitions are equivalent to the more conventional definitions in terms of contexts: whenever  $\vec{x} : \vec{t} \vdash a : t$  and  $\vec{x} : \vec{t} \vdash a' : t$ ,

$$\vec{x}: \vec{t} \vdash a \sqsubset_{\Diamond} a': t \quad \text{iff} \\ \forall \xi, C. \quad \xi: (\vec{t})t \vdash C: \text{unit} \Rightarrow (C[[(\vec{x})a/\xi]] \rightsquigarrow_{\Diamond} \langle \rangle \Rightarrow C[[(\vec{x})a'/\xi]] \rightsquigarrow_{\Diamond} \langle \rangle)$$

$$\vec{x}: \vec{t} \vdash a \cong_{\Diamond} a': t \quad \text{iff}$$

$$(6.3)$$

$$\forall \xi, C. \quad \xi : (\vec{t})t \vdash C : \text{unit} \quad \Rightarrow \quad (C\llbracket (\vec{x})a/\!\!/ \xi \rrbracket \rightsquigarrow_{\Diamond} \langle \rangle \iff C\llbracket (\vec{x})a'/\!\!/ \xi \rrbracket \rightsquigarrow_{\Diamond} \langle \rangle) \tag{6.4}$$
$$\vec{x} : \vec{t} \vdash a \sqsubset_{\Box} a' : t \quad \text{iff}$$

$$\forall \xi, \vec{C}. \quad \xi : (\vec{t})t \vdash C : \text{unit} \quad \Rightarrow \quad (C\llbracket (\vec{x})a/\!\! \xi \rrbracket \rightsquigarrow_{\Box} \{\langle \rangle\} \quad \Rightarrow \quad C\llbracket (\vec{x})a'/\!\! \xi \rrbracket \rightsquigarrow_{\Box} \{\langle \rangle\}) \quad (6.5)$$
$$\vec{x} : \vec{t} \vdash a \cong_{\Box} a' : t \quad \text{iff}$$

$$\forall \xi, C. \quad \xi : (\vec{t})t \vdash C : \text{unit} \Rightarrow (C\llbracket (\vec{x})a/\xi \rrbracket \rightsquigarrow_{\Box} \{\langle \rangle\} \Leftrightarrow C\llbracket (\vec{x})a'/\xi \rrbracket \rightsquigarrow_{\Box} \{\langle \rangle\}) \quad (6.6)$$

The may and must contextual approximation preorders and contextual equivalences are natural extensions of contextual approximation and equivalence to nondeterministic languages. Similar definitions appear elsewhere, e.g., in the work of Jagadeesan and Panangaden (1990) on Boudol's  $\gamma$ -calculus (Boudol 1989) and in Moran (1994). The may and must modalities appear in work on testing theories for processes (DeNicola and Hennessy 1984; Hennessy 1988). We may regard may and must contextual approximation as testing preorders in the following way. For every abstraction type  $(\vec{t})t$ , let a  $(\vec{t})t$ -test be a context C of type  $\xi : (\vec{t})t \vdash C$ : unit. Let us say that an abstraction  $(\vec{x})a$  of type  $(\vec{t})t$  may pass a  $(\vec{t})t$ -test, C, if  $C[[(\vec{x})a|\xi]] \rightsquigarrow_{\Diamond} \langle \rangle$ ; and  $(\vec{x})a$  must pass C, if  $C[[(\vec{x})a|\xi]] \rightsquigarrow_{\Box} \{\langle \rangle\}$ . Then two abstractions  $(\vec{x})a$  and  $(\vec{x})a'$  of type  $(\vec{t})t$  are may contextually approximate,  $\vec{x} : \vec{t} \vdash a \cong_{\Diamond} a' : t$ , if and only if  $(\vec{x})a'$  may pass all the  $(\vec{t})t$ -tests that  $(\vec{x})a$  may pass; and they are must contextually approximate,  $\vec{x} : \vec{t} \vdash a \cong_{\Box} a' : t$ , if and only if  $(\vec{x})a'$  must pass all the  $(\vec{t})t$ -tests that  $(\vec{x})a$  must pass.

To provide some intuition about the two approximation relations, let us here mention a few of their properties which will be established in the remainder of the chapter.

May contextual approximation is insensitive to divergence, and it orders nondeterministic programs above less nondeterministic ones. These facts are expressed in the laws for erratic choice: it is a least upper bound operator, and its unit is  $\Omega$  which is a least element in the may approximation preorder.

In the must approximation preorder, every program that may diverge is a least element (so all divergent programs are equated by must contextual equivalence). Nondeterministic programs are ordered below more deterministic ones, and erratic choice is a greatest lower bound operator. The "nondeterministic" contextual relations (may and must contextual approximation and equivalence) are all refinements of the corresponding "deterministic" contextual relations (contextual approximation,  $\subseteq$ , and equivalence,  $\cong$ ) with regard to "deterministic" terms: for all *a* and *a'* without syntactic occurrences of **or** and ?,

$$\Gamma \vdash a \sqsubset_{\diamond} a' : t \quad \text{implies} \quad \Gamma \vdash a \sqsubset a' : t \\ \Gamma \vdash a \sqsubset_{\Box} a' : t \quad \text{implies} \quad \Gamma \vdash a \sqsubset a' : t \\ \end{cases}$$

and similarly for the contextual equivalence relations. This is immediate from the definitions and the fact that the may and must evaluation relations coincide with the deterministic evaluation relation for deterministic terms, (5.12) and (5.15). Furthermore, the nondeterministic contextual relations are strict refinements of the deterministic contextual relations: there are (deterministic) contextually equivalent terms a and a' that are distinguished by all the nondeterministic contextual relations. For instance, recall u and u' from Example 4.3.2,

> $u = \lambda f.$  if  $f \langle \rangle$  then true else true  $u' = \lambda f.$  if  $f \langle \rangle$  then  $f \langle \rangle$  else true

They are (deterministically) contextually equivalent,  $u \cong u'$ : (unit  $\rightarrow$  bool)  $\rightarrow$  bool, but they are not may contextually equivalent: let v: (unit  $\rightarrow$  bool) denote the function  $v = \lambda \langle \rangle$ . true or false and define  $C = \text{let } f = \bullet$  in if f v then  $\Omega$  else  $\langle \rangle$ , then C[[u']] may terminate whereas C[[u]] must diverge. Furthermore, u and u' are not must contextually equivalent either: letting  $C' = \text{let } f = \bullet$  in if f v then  $\langle \rangle$  else  $\Omega$ , then C[[u]] must terminate whereas C[[[u']]] may diverge.

May contextual equivalence is insensitive to the difference between finitely brancing nondeterminism, generated by the binary erratic choice combinator **or**, and countably branching nondeterminism, generated by the countable choice primitive ?. In fact, we shall see in §6.4 that  $a_1$  **or**  $a_2$  is may contextually equivalent to the encoding (5.1), and that ? is may contextually equivalent to the encoding (5.2).

This is not the case for must contextual equivalence. Countable choice is not must contextual equivalent to (5.2) because

let 
$$x = ?$$
 in  $\langle \rangle \rightsquigarrow_{\Box} \{\langle \rangle \}$ 

whereas with (5.2) in place of ? the evaluation may diverge.

In fact, bounded nondeterminism and countable nondeterminism each introduce increasingly discriminative contexts that can distinguish more expressions with respect to must contextual equivalence. In our discussion of Example 4.3.2, above, we saw that bounded nondeterminism alone makes must contextual equivalence more discriminative than deterministic contextual equivalence. The next example shows how the introduction of countable nondeterminism distinguishes even more expressions.<sup>1</sup>

**Example 6.1.1** Let v and v' denote the functions

$$v = \lambda f. \text{ let } x = f \langle \rangle \text{ in } \langle \rangle$$
  

$$v' = \lambda f. \mathbf{Y} (\lambda g. \lambda y. \text{ let } x = f \langle \rangle \text{ in if } eq x y \text{ then } g (\operatorname{succ} y) \text{ else } \langle \rangle) \mathbf{0}$$

<sup>&</sup>lt;sup>1</sup>The observation that countable nondeterminism adds further discriminative power is due to Corin Pitcher. (Personal communication, October 1997).

where eq : nat  $\rightarrow$  nat  $\rightarrow$  bool is an appropriate recursive function that tests for equality.

Without countable nondeterminism, the two functions are must contextually equivalent,

$$v \cong_{\Box} v' : (unit \rightarrow nat) \rightarrow unit$$

We can prove this by an analogue of Proposition 4.3.1, which we shall establish later in Proposition 6.5.5. It tells us that the equivalence holds if

$$\forall u: t. \ v \ u \rightsquigarrow_{\Box} \{\langle \rangle\} \Leftrightarrow v' \ u \rightsquigarrow_{\Box} \{\langle \rangle\} \tag{6.7}$$

Now, if  $v u \rightsquigarrow_{\Box} \{\langle \rangle\}$  then, according to (Eval<sub> $\Box$ </sub> let),  $u \langle \rangle \rightsquigarrow_{\Box} V$  for some non-empty set of natural numbers V. The set V is finite (because we exclude countable nondeterminism). Let  $\lceil i \rceil$  be the first natural number not in V. Then v' u is guaranteed to terminate no later than by the *i*'th iteration of the loop, i.e.,  $v' u \rightsquigarrow_{\Box} \{\langle \rangle\}$ . This establishes the forward implication. The reverse is easy to see, regardless of the presence of countable nondeterminism.

But v is not contextually equivalent to v' if we admit countable choice because then we can take  $u = \lambda \langle \rangle$ . ? in (6.7). We see that  $v u \rightsquigarrow_{\Box} \{\langle \rangle\}$  but not  $v' u \rightsquigarrow_{\Box} \{\langle \rangle\}$  because  $u \langle \rangle$  may always evaluate to the number that it is being tested against so that the loop keeps unfolding, leading to divergence.

Our discussion of the relative discriminative power of the deterministic language and its extensions with bounded and countable nondeterminism can be turned into formal arguments about relative expressive power in accordance with Felleisen (1991); see also Mitchell (1993). But we shall not pursue this issue here.

### 6.2 Lower and upper simulation

We define lower and upper simulation operators,  $\langle \cdot \rangle_{\diamond}$  and  $\langle \cdot \rangle_{\Box}$ , that map any open relation R into the closed relations  $\langle R \rangle_{\diamond}$  and  $\langle R \rangle_{\Box}$  given by

$$\begin{array}{lll} a \ \langle R \rangle_{\diamond} \ a':t & \stackrel{\mathrm{def}}{\Leftrightarrow} & \forall u:t. \ a \rightsquigarrow_{\diamond} u \ \Rightarrow \ \exists u':t. \ a' \rightsquigarrow_{\diamond} u' \ \& \ u \ \overline{R} \ u':t \\ a \ \langle R \rangle_{\Box} \ a':t & \stackrel{\mathrm{def}}{\Leftrightarrow} & \forall U. \ a \rightsquigarrow_{\Box} U \ \Rightarrow \ \exists U'. \ a' \rightsquigarrow_{\Box} U' \ \& \ \forall u' \in U'. \ \exists u \in U. \ u \ \overline{R} \ u':t \end{array}$$

We call  $\mathbf{R} \in REL_0$  a (closed) *lower simulation* if it is a post-fixed point of  $\langle \cdot^{\circ} \rangle_{\diamond}$ , i.e.,  $\mathbf{R} \subseteq \langle \mathbf{R}^{\circ} \rangle_{\diamond}$ , and  $S \in REL$  is an open lower simulation if it is a post-fixed point of  $\langle \cdot \rangle_{\diamond}^{\circ}$ . Analogously,  $\mathbf{R} \in REL_0$  is a (closed) *upper simulation* if it is a post-fixed point of  $\langle \cdot^{\circ} \rangle_{\Box}$ , and  $S \in REL$  is an open upper simulation if it is a post-fixed point of  $\langle \cdot^{\circ} \rangle_{\Box}$ .

Note that every lower simulation is may-pre-adequate and every upper simulation is mustpre-adequate.

## 6.3 Lower relational reasoning

In this section we generalise the first two stages of the theory for contextual equivalence from Chapter 4 (i.e., §4.3 and §4.6–4.7) to may contextual equivalence. The presentation is a bit more economical here because we start with the simulation up to context proof rules where from all the other results follow, including the CIU Theorem.

**Lemma 6.3.1** 
$$\frac{R_{\mathbf{0}} \subseteq (\langle R \rangle_{\diamond} \cup R) \sqsubset_{\diamond}}{R \subseteq \sqsubset_{\diamond}} \quad if \ R \in REL \ is \ compatible \ and \ substitutive}$$

**Proof** First we construct a closed relation  $\mapsto_{\diamondsuit} \stackrel{\text{def}}{=} \mapsto_{\diamondsuit} \cup \sqsubset_{\diamondsuit}^{\text{op}}$ . We shall think of  $\mapsto_{\diamondsuit}$  as a generalised transition relation. Since  $\sqsubset_{\diamondsuit}$  and  $\mapsto_{\diamondsuit}$  are closed under evaluation contexts—by compatibility and (Trans $\diamond$  let left), respectively—so is  $\mapsto_{\diamondsuit}$ . Furthermore,  $\sqsubset_{\diamondsuit}$  and  $\mapsto_{\diamondsuit}^{\text{op}}$  are both may-pre-adequate and therefore  $\mapsto_{\blacklozenge}^{\text{op}}$  is may-pre-adequate.

Under the assumption that the premise of the rule holds and that R is reflexive and substitutive, we will prove that

$$a R a': t \& a \rightsquigarrow_{\Diamond} v \Rightarrow \exists v': t. v \overline{R} v': t \& a' \rightarrowtail_{\blacklozenge}^{*} v': t$$

$$(6.8)$$

Since  $\mapsto_{\blacklozenge}^{\text{op}}$  is may-pre-adequate, (6.8) implies that R is may-pre-adequate, so if R is compatible, it is included in may contextual approximation,  $R \subseteq \Box_{\diamondsuit}$ , as required.

The proof of (6.8) is similar to that of (4.22) in the proof of Lemma 4.6.1. It proceeds by induction on the derivation of  $a \rightsquigarrow_{\Diamond} v$  and exploits that, whenever a R a' : t, the premise of the lemma asserts that there exists an a'' : t such that  $a' \rightarrowtail_{\blacklozenge} a'' : t$  and either  $a \langle R \rangle_{\Diamond} a'' : t$  or  $a \hat{R} a'' : t$ . In the first case, the result follows easily from the definition of the lower simulation operator and the fact that the may evaluation relation is included in  $\rightarrowtail_{\blacklozenge}$ , by (5.3). In the second case, when  $a \hat{R} a'' : t$ , the argument is straightforward by analysis of the derivation of  $a \rightsquigarrow_{\Diamond} v$ .

From Lemma 6.3.1 we deduce a number of results.

**Lemma 6.3.2** May contextual approximation is closed under open extension,  $(\sqsubseteq_{\diamond \mathbf{0}})^{\circ} \subseteq \sqsubset_{\diamond}$ .

**Proof** As Lemma 4.7.1.

Corresponding to (4.23), we have a "lower simulation up to context and may contextual approximation" proof rule,

$$\frac{\boldsymbol{S} \subseteq \langle \boldsymbol{S}^{\mathsf{C}} \rangle_{\Diamond} \boxtimes_{\Diamond}}{\boldsymbol{S} \subseteq \boxtimes_{\Diamond}} \tag{6.9}$$

Suppose we define lower Kleene approximation and equivalence as the closed relations such that, for all t and a, a': t,

 $\begin{array}{rcl} a \preceq_{\Diamond} a': t & \stackrel{\mathrm{def}}{\Leftrightarrow} & \forall u. \; a \rightsquigarrow_{\Diamond} u \; \Rightarrow \; a' \rightsquigarrow_{\Diamond} u \\ a \asymp_{\Diamond} a': t & \stackrel{\mathrm{def}}{\Leftrightarrow} & \forall u. \; a \rightsquigarrow_{\Diamond} u \; \Leftrightarrow \; a' \rightsquigarrow_{\Diamond} u \end{array}$ 

that is,  $\leq_{\diamond} = \langle Id \rangle_{\diamond}$  and  $\approx_{\diamond}$  is the symmetrisation of  $\leq_{\diamond}$ . Then we get that

$$\preceq_{\Diamond} \subseteq \sqsubset_{\Diamond} \tag{6.10}$$

from (6.9) by letting  $S = \preceq_{\diamond}$ , because  $\preceq_{\diamond} = \langle Id \rangle_{\diamond}$  and  $(\preceq_{\diamond})^{\mathsf{C}}$  is reflexive.

**Proposition 6.3.3**  $\preceq^{\circ}_{\Diamond} \subseteq \sqsubset_{\diamond}$ .

**Proof** From (6.10) and Lemma 6.3.2.

By means of (6.10) we see that the reciprocals of the may evaluation and transition relations are included in may contextual approximation,

$$\rightsquigarrow^{\mathrm{op}}_{\Diamond} \subseteq \Box_{\Diamond}$$
 (6.11)

$$\mapsto^{\mathrm{op}}_{\Diamond} \subseteq \sqsubset_{\Diamond} \tag{6.12}$$

They are not included in may contextual equivalence, in general. For instance,  $(\mathbf{U} \text{ or } \mathbf{I}) \rightsquigarrow_{\Diamond} \mathbf{U}$  but not  $(\mathbf{U} \text{ or } \mathbf{I}) \sqsubset_{\Diamond} \mathbf{U}$ : unit  $\rightarrow$  unit, because  $(\mathbf{U} \text{ or } \mathbf{I}) \langle \rangle$  may terminate whereas  $\mathbf{U} \langle \rangle$  must diverge. However, deterministic evaluations and transitions are included in may contextual equivalence, that is, if a: t then

$$a \leadsto_{\Box} \{u\} \quad \Rightarrow \quad a \cong_{\Diamond} u : t \tag{6.13}$$

$$a \rightarrowtail_{\Box} \{b\} \quad \Rightarrow \quad a \cong_{\Diamond} b : t \tag{6.14}$$

In particular, when we combine this with Lemma 6.3.2, we get that  $\beta_v$ -equivalence is valid for may contextual equivalence,

$$\Gamma \vdash (\lambda x. a) v \cong_{\Diamond} a[v/x] : t, \text{ if } \Gamma, x : t' \vdash a : t \text{ and } \Gamma \vdash v : t'$$

$$(6.15)$$

As in Chapter 4, it follows that may contextual approximation is substitutive and satisfies a canonical freeness property.

Proposition 6.3.4 May contextual approximation and equivalence are substitutive.

**Proposition 6.3.5**  $u \sqsubset_{\diamond} u' : t$  if and only if  $u \overleftarrow{\sqsubset_{\diamond}} u' : t$ .

From Lemma 6.3.1 and (6.11) we can derive two characterisations of closed may contextual approximation, corresponding to those of Proposition 4.3.5.

**Proposition 6.3.6** If a, a' : t then (i), (ii) and (iii) are equivalent,

- (i)  $a \sqsubset_{a} a' : t$
- $(ii) \ \forall u. \ a \leadsto_{\Diamond} u \ \Rightarrow \ u \sqsubseteq_{\Diamond} a' : t$
- $(iii) \ \forall E. \ \bullet: t \vdash E: \text{unit} \ \Rightarrow \ (E\llbracket a \rrbracket \rightsquigarrow_{\Diamond} \langle \rangle \ \Rightarrow \ E\llbracket a' \rrbracket \rightsquigarrow_{\Diamond} \langle \rangle)$

**Proof** (i) implies (iii) by (6.3).

To see that (*iii*) implies (*ii*), suppose that  $a \rightsquigarrow_{\Diamond} u$  and let us prove that  $u \sqsubset_{\Diamond} a' : t$  by reference to (6.3). So let C be any closed context of type  $\bullet : t \vdash C :$  unit. If  $C[\![u]\!] \rightsquigarrow_{\Diamond} \langle \rangle$  then  $E[\![a]\!] \rightsquigarrow_{\Diamond} \langle \rangle$  as well, where  $E = (\text{let } x = \bullet \text{ in } C[\![x]\!])$ . Now (*iii*) entails that  $E[\![a']\!] \rightsquigarrow_{\Diamond} \langle \rangle$ , that is, there is u' : t such that  $a' \rightsquigarrow_{\Diamond} u'$  and  $C[\![u']\!] \rightsquigarrow_{\Diamond} \langle \rangle$ . By (6.11) and (6.3),  $u' \sqsubset_{\Diamond} a' : t$  and  $C[\![a']\!] \rightsquigarrow_{\Diamond} \langle \rangle$ , as required to show that  $u \sqsubset_{\Diamond} a' : t$ .

Finally, the implication  $(ii) \Rightarrow (i)$  follows by a slightly simpler argument than the proof of the corresponding implication in Proposition 4.3.5.

The equivalence between (i) and (ii) means that  $a \sqsubset_{\diamond} a' : t$  if and only if a' is an upper bound of the set of outcomes of a. Hence every well typed closed expression is the least upper bound of its set of outcomes, up to may contextual equivalence. We might call this a generalisation of Strachey's property (4.10) to nondeterminism.

The implication  $(ii) \Rightarrow (i)$  together with (6.11) and Lemma 6.3.2 implies that may contextual approximation is a pre-fixed point of  $\langle \cdot \rangle^{\circ}_{\diamond}$ ,

$$\left\langle \boldsymbol{\Xi}_{\diamond} \right\rangle_{\diamond}^{\circ} \subseteq \boldsymbol{\Xi}_{\diamond} \tag{6.16}$$

The reverse inclusion is false. It would amount to saying that may contextual approximation is an open lower simulation, but the next chapter, §7.1, shows that the *greatest* open lower simulation is *strictly* included in may contextual approximation. So may contextual approximation is not a fixed point of  $\langle \cdot \rangle_{\diamond}^{\circ}$ .

The equivalence between (i) and (iii) in Proposition 6.3.6 together with Lemma 6.3.2 gives us the following CIU Theorem.

**Theorem 6.3.7 (CIU)** May contextual approximation is the largest may-pre-adequate open relation which is closed under evaluation contexts and substitutions.

That is, whenever  $\vec{x}: \vec{t} \vdash a: t$  and  $\vec{x}: \vec{t} \vdash a': t$ ,

$$\vec{x}: \vec{t} \vdash a \sqsubset_{\Diamond} a': t \quad \text{iff} \quad \forall \text{ CIU contexts } \mathcal{E}. \quad \xi: (\vec{t})t \vdash \mathcal{E}: \text{ unit } \Rightarrow \qquad (6.17)$$
$$(\mathcal{E}\llbracket(\vec{x})a/\xi\rrbracket \rightsquigarrow_{\Diamond} \langle \rangle \Rightarrow \mathcal{E}\llbracket(\vec{x})a'/\xi\rrbracket \rightsquigarrow_{\Diamond} \langle \rangle)$$

# 6.4 May equational theory

In this section we present a small selection of equational and inequational laws for the may contextual relations. Most are straightforwardly derived via lower Kleene approximation and equivalence.

May contextual equivalence is closed under open extension and  $\beta_v$ -equivalence, so  $\eta_v$ -equivalence is valid by the same derivation as in Example 4.7.2.

$$\Gamma \vdash v \cong_{\Diamond} \lambda y. v y : t \rightharpoonup t', \text{ if } \Gamma \vdash v : t \rightharpoonup t'$$

By lower Kleene equivalence, we see that the inter-translation between erratic choice and countable choice in (5.1) and (5.2) is correct up to may contextual equivalence:

$$a_1 \text{ or } a_2 \cong_{\Diamond} \text{ let } x = ? \text{ in case } x \text{ of } \mathbf{0}. a_1 \text{ or succ } x. a_2 : t, \text{ if } a_1, a_2 : t$$
 (6.18)

$$? \cong_{\Diamond} (\mathbf{Y} \lambda f, \lambda \langle \rangle, \mathbf{0} \text{ or } (\mathbf{let} \ y = f \langle \rangle \mathbf{in} \ \mathbf{succ} \ y)) \langle \rangle : \mathbf{nat}$$
(6.19)

By unfolding the fixed point we get another formulation of the correspondence between erratic and countable choice:

$$? \cong_{\Diamond} \mathbf{0} \text{ or } (\mathbf{let } y = ? \mathbf{in succ } y) \tag{6.20}$$

Erratic choice is idempotent, symmetric, and associative: whenever  $a, a_1, a_2, a_3 : t$ ,

$$\Gamma \vdash a \text{ or } a \cong_{\Diamond} a : t \tag{6.21}$$

$$\Gamma \vdash a_1 \text{ or } a_2 \cong_{\diamond} a_2 \text{ or } a_1 : t \tag{6.22}$$

$$\Gamma \vdash a_1 \text{ or } (a_2 \text{ or } a_3) \cong_{\Diamond} (a_1 \text{ or } a_2) \text{ or } a_3 : t$$
(6.23)

Furthermore, erratic choice is the least upper bound operator:

$$\Gamma \vdash a_1 \sqsubset_{\diamond} (a_1 \text{ or } a_2) : t \tag{6.24}$$

$$\Gamma \vdash a_2 \sqsubset_{\diamond} (a_1 \text{ or } a_2) : t \tag{6.25}$$

$$\frac{\Gamma \vdash a_1 \boxtimes_{\Diamond} a : t \quad \Gamma \vdash a_2 \boxtimes_{\Diamond} a : t}{\Gamma \vdash (a_1 \text{ or } a_2) \boxtimes_{\Diamond} a : t}$$
(6.26)

Every expression that must diverge is a syntactic bottom element:

$$\Gamma \vdash a \sqsubset_{\diamond} a' : t, \text{ if } \neg \exists v. \ a \rightsquigarrow_{\diamond} v, \ \Gamma \vdash a : t \text{ and } \Gamma \vdash a' : t$$

$$(6.27)$$

The **Y** combinator is a least pre-fixed point combinator.

**Proposition 6.4.1 (Recursion induction)** 
$$\frac{(\lambda f. v) u \sqsubset_{\Diamond} u : t_1 \rightharpoonup t_2}{\mathbf{Y} (\lambda f. v) \sqsubset_{\Diamond} u : t_1 \rightharpoonup t_2}$$

The may evaluation relation and may contextual approximation satisfy the same unwinding and syntactic continuity properties as their deterministic counterparts:

**Theorem 6.4.2 (May Unwinding)** Suppose  $y : (t \rightarrow t) \rightarrow t \vdash a : t'$  and t is a function type. Then  $a[\mathbf{Y}/y]$  may terminate if and only if  $a[\mathbf{Y}^{(n)}/y]$  may terminate for some  $n < \omega$ .

**Proposition 6.4.3 (Syntactic may continuity)** If  $y : (t \rightarrow t) \rightarrow t \vdash a : t'$  and t is a function type,  $a[\mathbf{Y}/y] \sqsubset_{\diamond} a' : t'$  if and only if  $\forall n < \omega$ .  $a[\mathbf{Y}^{(n)}/y] \sqsubset_{\diamond} a' : t'$ .

The proofs are exactly the same as for the deterministic case in  $\S4.5$ .

Example 6.4.4 Let

$$a = \lambda \langle \rangle.?$$
  
 
$$a' = \mathbf{let} \ y = ? \ \mathbf{in} \ \lambda \langle \rangle. \ min(?, y)$$

where min(?, y) is a suitable fixed point expression that computes the minimum of the outcome of ? and y.

The two expressions a and a' are may contextually equivalent,  $a \cong_{\Diamond} a'$ : unit  $\rightarrow$  nat.

One direction,  $a' \sqsubset_{\diamond} a$ : unit  $\rightarrow$  nat, is easy: whenever  $a' \rightsquigarrow_{\diamond} u'$ , the outcome is of the form  $u' = \lambda \langle \rangle$ .  $min(?, \lceil n \rceil)$ ; by lower Kleene approximation,  $min(?, \lceil n \rceil) \sqsubset_{\diamond} ?$ : nat holds for all  $n < \omega$ ; hence  $a \sqsubset_{\diamond} a'$ : unit  $\rightarrow$  nat, by Proposition 6.3.6.

The difficult part is whether  $a \sqsubset_{\diamond} a' : \text{unit} \rightarrow \text{nat.}$  Intuitively, this holds because in any context C such that  $C[\![a]\!] \rightsquigarrow_{\diamond} \langle \rangle$ , there is an upper bound n on the outputs of the function a during the evaluation; therefore  $C[\![a']\!] \rightsquigarrow_{\diamond} \langle \rangle$  too by always choosing y in a' to be greater than n. We can prove this most easily by means of the syntactic continuity proof rule and the fact that, according to (6.19),  $a \cong_{\diamond} \mathbf{Y} u : \text{unit} \rightarrow \text{nat}$ , where u is the functional

$$u = \lambda f \cdot \lambda \langle \rangle \cdot \mathbf{0} \text{ or } (\text{let } y = f \langle \rangle \text{ in succ } y)$$

From (6.27) we get that

$$\mathbf{Y}^{(0)} u \sqsubset_{\diamond} a' : \text{unit} \rightarrow \text{nat}$$

and we calculate, via lower Kleene equivalence and approximation,

$$\mathbf{Y}^{(n+1)} \, u \cong_{\Diamond} \lambda \langle \, \rangle. \, \min(\ref{mn}) \sqsubseteq_{\Diamond} a' : \text{unit} \rightharpoonup \text{nat}, \ \text{ for all } n < \omega$$

By syntactic may continuity, Proposition 6.4.3, we obtain  $\mathbf{Y} u \sqsubset_{\diamond} a'$ : unit  $\rightarrow$  nat, as required to show that  $a \sqsubset_{\diamond} a'$ : unit  $\rightarrow$  nat.

The example was used in Lassen (1997) to distinguish may contextual approximation from an applicative simulation preorder, and it serves the same purpose in the next chapter, §7.1. Variations of this example has been discovered and studied elsewhere in different contexts. For instance, a similar example with streams, due to Boudol (1980), illustrated some problems with certain choices of operational equivalence between nondeterministic recursive program schemes. The programs in the example are equated in the continuous powerdomain model considered by Winskel (1983), and he argued that the identification is perfectly natural from an appropriate view of the operational semantics. That conclusion is corroborated here by the fact that they are *contextually* equivalent (our proof in the preceding example carries over to Boudol's example). Abramsky (1983) discussed the same example, but argued that such programs should be distinguished, and that a semantics like ours fails to give an exact agreement with the operational semantics. But the semantics of finitely branching nondeterminism on non-discrete data domains, e.g., streams or function spaces, is discontinuous in partial-order fixed-point models that makes such distinctions. Therefore he gave a categorytheoretic fixed-point semantics, using Lehmann's categorical powerdomain construction (see §6.9 below), which is continuous and makes the desired distinctions. However, as these distinctions are not warranted from the viewpoint of observational behaviour underlying the definition of contextual equivalence, one may instead argue that Abramsky's semantics is not fully abstract.

## 6.5 Upper relational reasoning

In this section the theory of must contextual equivalence is developed along the same lines as in §6.3. The main difference is that some extra definitions are needed to set up the proof of the fundamental lemma:

**Lemma 6.5.1** 
$$\frac{R_0 \subseteq (\langle R \rangle_{\square} \cup \widehat{R}) \sqsubset_{\square}}{R \subseteq \sqsubset_{\square}} \quad if \ R \ is \ compatible \ and \ substitutive.}$$

In order to prove the lemma, we introduce a generalised must evaluation relation,  $\rightsquigarrow_{\blacksquare}$ , and a corresponding generalised upper simulation operator,  $\langle \cdot \rangle_{\blacksquare}$ .

We define  $\rightsquigarrow \blacksquare$  inductively as the smallest type indexed family of relations between closed expressions a and sets of closed values U such that  $\rightsquigarrow \blacksquare$  is closed under the rules in Table 6.1.

Modulo types, the last three rules are the defining rules of the must evaluation relation, Table 5.5. By rule induction,  $\rightsquigarrow_{\Box}$  is included in  $\rightsquigarrow_{\blacksquare}$ ,

$$a \rightsquigarrow_{\Box} U$$
 implies  $a \rightsquigarrow_{\blacksquare} U: t$ , if  $a:t$  (6.28)

**Lemma 6.5.2** If  $a \rightsquigarrow U$ : t then a must terminate.

(Eval approx) 
$$\frac{b \rightsquigarrow U: t}{a \rightsquigarrow U: t} \text{ if } b \sqsubset_{\Box} a: t$$

(Eval value)  $v \rightsquigarrow \{v\} : t, \text{ if } v : t$ 

(Eval<sub>$$\blacksquare$$</sub> redex) 
$$\frac{\forall b \in B. \ b \rightsquigarrow \blacksquare V_b : t}{a \rightsquigarrow \blacksquare \bigcup_{b \in B} V_b : t} \text{ if } a \rightarrow_{\Box} B \text{ and } a : t$$

(Eval let) 
$$\frac{a \rightsquigarrow U: t' \quad \forall u \in U. \ b[u/x] \rightsquigarrow V_u: t}{\text{let } x = a \text{ in } b \rightsquigarrow U_u: t} \text{ if } x: t' \vdash b: t$$



**Proof** The proof is by induction on the derivation of  $a \rightsquigarrow \square U : t$ . In order to carry out the induction step for (Eval<sub> $\blacksquare$ </sub> let), it is necessary to strengthen the goal of the induction hypothesis from that a must terminate to

 $\forall t'. \forall$  evaluation contexts  $E. \bullet: t \vdash E: t' \& (\forall u \in U. E[\![u]\!] \Downarrow_{\Box}) \Rightarrow E[\![a]\!] \Downarrow_{\Box}$ 

where  $b \Downarrow_{\Box} \stackrel{\text{def}}{\Leftrightarrow} \exists V. b \rightsquigarrow_{\Box} V$ . Then all cases run smoothly. (5.10) is used for the cases (Eval redex) and (Eval let).

The generalised upper simulation operator maps any open relation R into a closed relation  $\langle R \rangle_{\blacksquare}$  given by

$$a \langle R \rangle_{\blacksquare} a' : t \quad \stackrel{\text{def}}{\Leftrightarrow} \quad \forall U. \ a \rightsquigarrow_{\square} U \ \Rightarrow \ \exists U'. \ a' \rightsquigarrow_{\blacksquare} U' : t \quad \& \quad \forall u' \in U'. \ \exists u \in U. \ u \ \overline{R} \ u' : t$$

Clearly,  $\langle R \rangle_{\Box} \subseteq \langle R \rangle_{\blacksquare}$  and, moreover,  $\langle R \rangle_{\blacksquare} \subset_{\Box} \subseteq \langle R \rangle_{\blacksquare}$ , by (Eval<sub>■</sub> approx). From Lemma 6.5.2, we get that  $\langle R \rangle_{\blacksquare}$  is must-pre-adequate.

Generalised upper simulation satisfies the following proof rule.

**Lemma 6.5.3** 
$$\frac{R_{\mathbf{0}} \subseteq \langle R \rangle_{\mathbf{B}} \cup (\widehat{R} \sqsubset_{\Box})}{R_{\mathbf{0}} \subseteq \langle R \rangle_{\mathbf{B}}} \quad if \ R \ is \ reflexive \ and \ substitutive.$$

**Proof** Assume that R is reflexive and substitutive and that the premise holds. We prove the conclusion  $R_0 \subseteq \langle R \rangle_{\blacksquare}$ , i.e.,

$$a \ R \ a': t \ \& \ a \rightsquigarrow_{\Box} U \ \Rightarrow \ \exists U'. \ a' \rightsquigarrow_{\blacksquare} U': t \ \& \ \forall u' \in U'. \ \exists u \in U. \ u \ \overline{R} \ u': t$$

by induction on the derivation of  $a \rightsquigarrow_{\Box} U$ .

Since  $a \ R \ a' : t$ , according to the premise either  $a \ \langle R \rangle_{\blacksquare} \ a' : t$  or there is an a'' : t such that  $a \ \widehat{R} \ a'' : t$  and  $a'' \sqsubset_{\Box} a' : t$ .

If  $a \langle R \rangle_{\blacksquare} a' : t$ , the result is immediate.

If  $a \ \widehat{R} \ a'' : t$  and  $a'' \sqsubset_{\Box} a' : t$ , we proceed by analysis of the derivation of  $a \rightsquigarrow_{\Box} U$ .

Case (Eval $\square$  value)

$$a$$
 is a value  
 $U = \{a\}$ 

Since  $a \hat{R} a'' : t$  Lemma 3.5.1 implies that a'' is a value. Let  $U' = \{a''\}$ , then  $a'' \rightsquigarrow U'$ , by (Eval value), and  $a' \rightsquigarrow U'$ , by (Eval approx).

Case (Eval<sub> $\Box$ </sub> redex)

$$a \to_{\Box} \{b_i \mid i < \omega\}$$
  
$$b_i \rightsquigarrow_{\Box} U_i \text{ for all } i < \omega$$
  
$$U = \bigcup_{i < \omega} U_i$$

From  $a \ \widehat{R} \ a'' : t$  and Lemma 3.10.1 we get that  $a'' \to_{\Box} \{b'_i \mid i < \omega\}$  with  $b_i \ R \ b'_i : t$  for all  $i < \omega$ . By the induction hypothesis for each  $i < \omega$  there exists  $U'_i$  such that  $b'_i \rightsquigarrow_{\blacksquare} U'_i$  and

$$\forall u' \in U'_i. \exists u \in U_i. u \ \overline{R} \ u' : t$$

Let  $U' = \bigcup_{i < \omega} U'_i$ , then

$$\forall u' \in U'. \exists u \in U. \ u \ \overline{R} \ u' : t$$

Finally, by (Eval<sub> $\blacksquare$ </sub> redex) and (Eval<sub> $\blacksquare$ </sub> approx), we get that  $a'' \rightsquigarrow_{<math>\blacksquare} U'$  and  $a' \rightsquigarrow_{<math>\blacksquare} U'$ , as required.

Case (Eval<sub> $\Box$ </sub> let)

$$a = \mathbf{let} \ x = a_0 \ \mathbf{in} \ b_0$$
$$a_0 \rightsquigarrow_{\Box} V$$
$$b_0[v/x] \rightsquigarrow_{\Box} U_v \ \text{ for all } v \in V$$
$$U = \bigcup_{v \in V} U_v$$

Since  $a \ \widehat{R} \ a'' : t$ , a'' must be of the form  $a'' = \text{let } x = a'_0 \text{ in } b'_0$  with  $a_0 \ R \ a'_0 : t'$  and  $x : t' \vdash b_0 \ R \ b'_0 : t$ . By the induction hypothesis,  $a'_0 \rightsquigarrow_{\Box} V'$  for some V' such that for every  $v' \in V'$  there is  $v \in V$  with  $v \ \overline{R} \ v' : t'$  and hence  $b_0[v/x] \ R \ b'_0[v'/x] : t$ . Another application of the induction hypothesis yields that  $b'_0[v'/x] \rightsquigarrow_{\Box} U'_{v'}$  such that

$$\forall u' \in U'_{v'}. \ \exists u \in U_v. \ u \ \overline{R} \ u' : t$$

Letting  $U' = \bigcup_{v' \in V'} U'_{v'}$ , we have that

$$\forall u' \in U'. \exists u \in U. \ u \ \overline{R} \ u': t$$

By (Eval  $\blacksquare$  let) and (Eval  $\blacksquare$  approx), we get that  $a'' \rightsquigarrow \blacksquare U'$  and  $a' \rightsquigarrow \blacksquare U'$ , as required.  $\Box$ 

Lemma 6.5.1 now follows because  $\langle R \rangle_{\square} \sqsubset_{\square} \subseteq \langle R \rangle_{\blacksquare}$  and because  $R_0 \subseteq \langle R \rangle_{\blacksquare}$  implies that R is must-pre-adequate so that  $R \subseteq \sqsubset_{\square}$ , if R is compatible.

From Lemma 6.5.1 the theory of must contextual approximation unfolds in much the same way as for may contextual approximation in §6.3.

**Lemma 6.5.4** *Must contextual approximation is closed under open extension,*  $(\sqsubseteq_{\Box 0})^{\circ} \subseteq \sqsubseteq_{\Box}$ .

We have an "upper simulation up to context and must contextual approximation" proof rule,

$$\frac{\boldsymbol{S} \subseteq \langle \boldsymbol{S}^{\mathsf{C}} \rangle_{\square} \boxtimes_{\square}}{\boldsymbol{S} \subseteq \boxtimes_{\square}}$$
(6.29)

which we can derive either from Lemma 6.5.1 or from a "generalised upper simulation up to context" proof rule

$$\frac{S \subseteq \langle S^{\mathsf{C}} \rangle_{\blacksquare}}{S \subseteq \sqsubset_{\square}} \tag{6.30}$$

which follows from Lemma 6.5.3.

We define upper Kleene approximation and equivalence as the closed relations such that, for all t and a, a': t,

$$\begin{array}{lll} a \preceq_{\Box} a': t & \stackrel{\mathrm{def}}{\Leftrightarrow} & \forall U. \; a \rightsquigarrow_{\Box} U \; \Rightarrow \; \exists U' \subseteq U. \; a' \rightsquigarrow_{\Box} U' \\ a \asymp_{\Box} a': t & \stackrel{\mathrm{def}}{\Leftrightarrow} & \forall U. \; a \rightsquigarrow_{\Box} U \; \Leftrightarrow \; a' \rightsquigarrow_{\Box} U \end{array}$$

That is,  $\preceq_{\Box} = \langle Id \rangle_{\Box}$  and  $\asymp_{\Box}$  is the symmetrisation of  $\preceq_{\Box}$ .

# **Proposition 6.5.5** $\preceq^{\circ}_{\Box} \subseteq \sqsubseteq_{\Box}$ .

**Proof** From Lemma 6.5.4 and (6.29). See the proof of Proposition 6.3.3.  $\Box$ 

The may evaluation and transition relations are included in must contextual approximation,

$$\rightsquigarrow_{\Diamond} \subseteq \sqsubset_{\Box} \tag{6.31}$$

$$\rightarrowtail_{\Diamond} \subseteq \sqsubset_{\Box} \tag{6.32}$$

Deterministic evaluations and transitions are included in must contextual equivalence, that is, if a: t then

$$a \leadsto_{\Box} \{u\} \quad \Rightarrow \quad a \cong_{\Box} u : t \tag{6.33}$$

$$a \rightarrowtail_{\Box} \{b\} \quad \Rightarrow \quad a \cong_{\Box} b : t \tag{6.34}$$

Lemma 6.5.4 and (6.34) imply that  $\beta_v$ -equivalence is valid for must contextual equivalence,

$$\Gamma \vdash (\lambda x. a) v \cong_{\Box} a[v/x] : t, \text{ if } \Gamma, x : t' \vdash a : t \text{ and } \Gamma \vdash v : t'$$
(6.35)

and entails that must contextual approximation is substitutive and satisfies a canonical freeness property.

Proposition 6.5.6 Must contextual approximation and equivalence are substitutive.

**Proposition 6.5.7**  $u \sqsubset_{\Box} u' : t$  if and only if  $u \overleftarrow{\sqsubset_{\Box}} u' : t$ .

From Lemma 6.5.3 and (6.31) we derive two characterisations of closed must contextual approximation.

**Proposition 6.5.8** If a, a' : t then (i), (ii) and (iii) are equivalent,

(i)  $a \sqsubset_{\Box} a' : t$ 

$$(ii) \ (\exists U. \ a \leadsto_{\square} U) \ \Rightarrow \ \exists U'. \ a' \leadsto_{\square} U' \ \& \ \forall u' \in U'. \ a \sqsubseteq_{\square} u' : t$$

 $(iii) \ \forall E. \ \bullet: t \vdash E: \text{unit} \ \Rightarrow \ (E\llbracket a \rrbracket \rightsquigarrow_{\Box} \{\langle \, \rangle \} \ \Rightarrow \ E\llbracket a' \rrbracket \rightsquigarrow_{\Box} \{\langle \, \rangle \})$ 

**Proof** (i) implies (iii) because of (6.5).

(*iii*) implies (*ii*) because a must terminate if and only if  $(a; \langle \rangle) \rightsquigarrow_{\Box} \{\langle \rangle\}$  and, according to (*iii*),  $(a; \langle \rangle) \rightsquigarrow_{\Box} \{\langle \rangle\}$  implies  $(a'; \langle \rangle) \rightsquigarrow_{\Box} \{\langle \rangle\}$  which again holds if and only if a' must terminate. So  $a' \rightsquigarrow_{\Box} U'$  for some set U'. From (5.8) and (6.31) it follows that  $a' \sqsubset_{\Box} u' : t$  for all  $u' \in U'$ . By transitivity, we conclude that (*ii*) holds.

It remains to show that (ii) implies (i). If a and a' satisfy (ii) then  $a \langle Id \rangle_{\blacksquare} a' : t$ . (Recall that  $\langle Id \rangle_{\square}$  is upper Kleene approximation, so we might call  $\langle Id \rangle_{\blacksquare}$  "generalised upper Kleene approximation".) By a similar argument as in the proof of Proposition 6.5.5, but with (6.30) in place of (6.29), we see that  $\langle Id \rangle_{\blacksquare} \subseteq \Box_{\square}$  and hence that  $a \equiv_{\square} a' : t$ , as required.  $\square$ 

The first characterisation tells us that  $a \sqsubset_{\Box} a' : t$  if and only if a is a lower bound of the set of outcomes of a' (if we say that the set of outcomes contains  $\Omega$  when a' may diverge). Hence every well typed closed expression is the greatest lower bound of its set of outcomes, up to must contextual equivalence. This complements Proposition 6.3.6 as a generalisation of Strachey's property (4.10) to nondeterminism.

The implication  $(ii) \Rightarrow (i)$  together with (6.31) implies that must contextual approximation is a pre-fixed point of  $\langle \cdot \rangle_{\Box}^{\circ}$ ,

$$\left\langle \boldsymbol{\boldsymbol{\sqsubset}}_{\Box} \right\rangle_{\Box}^{\circ} \subseteq \boldsymbol{\boldsymbol{\sqsubseteq}}_{\Box} \tag{6.36}$$

The reverse inclusion is false, analogously to the situation for may contextual approximation in  $\S6.3$ . This follows from the results in the next chapter,  $\S7.2$ .

If we combine the last characterisation of closed must contextual approximation in Proposition 6.5.8 with Lemma 6.5.4 we obtain the following CIU Theorem.

**Theorem 6.5.9 (CIU)** Must contextual approximation is the largest must-pre-adequate open relation which is closed under evaluation contexts and substitutions.

That is, whenever  $\vec{x}: \vec{t} \vdash a: t$  and  $\vec{x}: \vec{t} \vdash a': t$ ,

$$\vec{x}: \vec{t} \vdash a \sqsubset_{\Box} a': t \quad \text{iff} \quad \forall \text{ CIU contexts } \mathcal{E}. \quad \xi: (\vec{t})t \vdash \mathcal{E}: \text{ unit } \Rightarrow \qquad (6.37)$$
$$(\mathcal{E}[\![(\vec{x})a/\!\!/\xi]\!] \rightsquigarrow_{\Box} \{\langle\rangle\} \Rightarrow \mathcal{E}[\![(\vec{x})a'/\!\!/\xi]\!] \rightsquigarrow_{\Box} \{\langle\rangle\})$$

# 6.6 Must equational theory

Most of the equational laws for may contextual approximation carry over to the must modality. One exception is the equivalence between countable choice and the encoding (5.2),

$$? \cong_{\Diamond} (\mathbf{Y} \lambda f. \lambda \langle \rangle. \mathbf{0} \text{ or } (\mathbf{let } y = f \langle \rangle \mathbf{in succ } y)) \langle \rangle : \mathbf{nat}$$

The encoding may diverge, so it is strictly below ? with respect to must contextual approximation. But the correspondence between erratic and countable choice in (6.20) is valid for must contextual equivalence,

$$? \cong_{\Box} \mathbf{0}$$
 or (let  $x = ?$  in succ  $x$ )

Erratic choice is the greatest lower bound operator for must contextual approximation:

$$\Gamma \vdash (a_1 \text{ or } a_2) \sqsubset_{\Box} a_1 : t \tag{6.38}$$

$$\Gamma \vdash (a_1 \text{ or } a_2) \sqsubset_{\Box} a_2 : t \tag{6.39}$$

$$\frac{\Gamma \vdash a \sqsubset_{\Box} a_1 : t \quad \Gamma \vdash a \sqsubset_{\Box} a_2 : t}{\Gamma \vdash a \sqsubset_{\Box} (a_1 \text{ or } a_2) : t}$$
(6.40)

Every expression that may diverge is a syntactic bottom element:

$$\Gamma \vdash a \sqsubset_{\Box} a' : t, \text{ if } \neg \exists V. a \rightsquigarrow_{\Box} V, \ \Gamma \vdash a : t \text{ and } \Gamma \vdash a' : t$$

$$(6.41)$$

From (6.29) we can derive a recursion induction rule for the Y combinator.

# Proposition 6.6.1 (Recursion induction) $\frac{(\lambda f. v) u \sqsubset_{\Box} u : t_1 \rightharpoonup t_2}{\mathbf{Y} (\lambda f. v) \sqsubset_{\Box} u : t_1 \rightharpoonup t_2}$

When we generalise the unwinding and continuity properties of evaluation and contextual approximation from §4.5 to must evaluation and must contextual approximation, it makes a significant difference whether we restrict ourselves to bounded (finitely branching) nondeterminism, generated by erratic choice, or we admit countable (countably branching) nondeterminism, generated by countable choice. Let us postpone countable nondeterminism to the next section, and first consider the case for bounded nondeterminism.

The next theorem holds for the finitely branching sublanguage without countable choice.

**Theorem 6.6.2 (Must**  $\omega$ -Unwinding) Suppose  $y : (t \rightarrow t) \rightarrow t \vdash a : t'$  and t is a function type. Then  $a[\mathbf{Y}/y]$  must terminate if and only if  $a[\mathbf{Y}^{(n)}/y]$  must terminate for some  $n < \omega$ .

The proof is similar to those of the deterministic Unwinding Theorem 4.5.1 and the Must  $\omega_1^{CK}$ -Unwinding Theorem 6.7.4, given below. It employs the following two lemmas. We omit the proofs.

**Lemma 6.6.3 Y** is a  $\sqsubset_{\Box}$ -upper bound of its finite unwindings  $\{\mathbf{Y}^{(i)} \mid i < \omega\}$ .

From the two lemmas we also obtain a syntactic continuity property of must contextual approximation for the sublanguage without countable choice.

**Proposition 6.6.5 (Syntactic must**  $\omega$ -continuity) If  $y : (t \rightarrow t) \rightarrow t \vdash a : t'$  and t is a function type,  $a[\mathbf{Y}/y] \sqsubset_{\Box} a' : t'$  if and only if  $\forall n < \omega$ .  $a[\mathbf{Y}^{(n)}/y] \sqsubset_{\Box} a' : t'$ .

The proof is the same as for Proposition 4.5.4.

# 6.7 Transfinite unwinding and syntactic $\omega_1^{CK}$ -continuity

The Must  $\omega$ -Unwinding Theorem reflects that every derivation tree for must evaluation is of finite height in the case of bounded nondeterminism. Formally, the closure ordinal of the rules in Table 5.5 that define the must evaluation relation is  $\omega$ , that is, the height of every derivation tree is an ordinal less than  $\omega$ . For infinitely branching nondeterminism, it is wellknown that the height of must evaluation trees may be transfinite. In this case the closure ordinal of the must evaluation rules is  $\omega_1^{CK}$  which is the smallest non-recursive ordinal. This is proved in Lassen and Pitcher (1998) for a variant of PCF with countable nondeterminism; see also Apt and Plotkin (1986). In other words, for any recursive ordinal  $\alpha < \omega_1^{CK}$ , there are programs with must evaluation trees of height greater than or equal to  $\alpha$ .

#### Example 6.7.1 Let

$$\begin{array}{ll} u &=& \lambda f. \, \lambda \langle x, y \rangle. \, \text{if eq } 0 \, x \, \text{then} \, \left( \text{if eq } 0 \, y \, \text{then} \, \left\langle \right\rangle \\ & \quad \text{else} \, f \, \langle x, \text{pred} \, y \rangle \right) \\ & \quad \text{else let} \, \, y' = ? \, \text{in} \, f \, \left\langle \text{pred} \, x, y' \right\rangle \end{array}$$

where  $eq : nat \rightarrow nat \rightarrow bool$  and  $pred : nat \rightarrow nat$  are appropriate recursive functions for comparing and decrementing natural numbers.

For any pair of closed values m, n: nat, the program  $\mathbf{Y} u \langle m, n \rangle$  will first count down n to zero, then it will decrement m and choose an arbitrary natural number  $n_1$  for the second component and start counting down  $n_1$ , and so on. The program eventually stops when m has been decremented to zero and the m'th arbitrarily chosen number has been counted down to zero as well.

If m is non-zero, we cannot give a finite bound on the number of iterations of u, so in that case  $a = y u \langle m, n \rangle$  is a counterexample to the Must  $\omega$ -Unwinding Theorem.

Since  $\mathbf{Y} u \langle m, n \rangle \rightsquigarrow_{\Box} \{ \langle \rangle \}$  for all  $\langle m, n \rangle$ : nat  $\times$  nat, we see that

$$\mathbf{Y} u \cong_{\Box} \lambda z. \langle \rangle : \operatorname{nat} \times \operatorname{nat} \rightarrow \operatorname{unit}$$

But with any finite unwinding  $\mathbf{Y}^{(i)}$  in place of  $\mathbf{Y}$  we get that

$$\mathbf{Y}^{(i)} u \models_{\Box} \lambda \langle x, y \rangle. \text{ (if } \mathbf{eq} \, \mathbf{0} \, x \text{ then } \langle \rangle \text{ else } \mathbf{\Omega} \text{) : nat} \times \text{nat} \rightarrow \text{unit}$$

because if the first argument is non-zero there is no finite upper bound on the number of iterations of u. So, on such input, there may be more than i iterations of u and  $\mathbf{Y}^{(i)} u$  may diverge. This gives us a counterexample to syntactic must  $\omega$ -continuity, by taking a = y u and  $a' = \lambda \langle x, y \rangle$ . (if eq 0 x then  $\langle \rangle$  else  $\Omega$ ).

It is possible to formulate an  $\omega_1^{CK}$ -unwinding property for must termination that holds for countable nondeterminism. To this end we must first extend the syntax so that we can express transfinite unwindings of the fixed point combinator.

In this section  $\alpha$ ,  $\beta$  and  $\gamma$  range over arbitrary recursive ordinals, and  $\lambda$  ranges over recursive limit ordinals,  $\alpha, \beta, \gamma, \lambda < \omega_1^{CK}$ . We introduce a family of "approximate" fixed point expressions indexed by the recursive ordinals,

$$\begin{array}{rrrr} (Exp) & a,b & ::= & \dots \\ & & & | & \mathbf{fix}^{(\alpha)} \, u \end{array}$$

with the typing rule:

(Type fix) 
$$\frac{\Gamma \vdash u : (t_1 \rightharpoonup t_2) \rightharpoonup t_1 \rightharpoonup t_2}{\Gamma \vdash \mathbf{fix}^{(\alpha)} u : t_1 \rightharpoonup t_2}$$

We call the new constructs 'transfinite'.

We extend the must primitive reduction relation to the new transfinite syntax as follows.

(Redex\_{\sigma} fix zero) 
$$\mathbf{fix}^{(0)} u \to_{\sigma} \{\mathbf{fix}^{(0)} u\}$$
  
(Redex\_{\sigma} fix succ)  $\mathbf{fix}^{(\alpha+1)} u \to_{\sigma} \{u (\lambda x. (\mathbf{fix}^{(\alpha)} u) x)\}$ 

(Redex<sub>□</sub> fix limit) 
$$\mathbf{fix}^{(\lambda)} u \to_{\Box} {\mathbf{fix}^{(\alpha)} u}$$
 if  $\alpha < \lambda$ 

This indirectly extends the must evaluation and transition relations via ( $\text{Eval}_{\Box}$  redex) and ( $\text{Trans}_{\Box}$  redex). We observe that ( $\text{Redex}_{\Box}$  fix limit) introduces nondeterminacy in all three must relations.

It does not seem possible to extend the may operational relations from Tables 5.1-5.3 to the extended syntax. So the characterisations (5.4), (5.5), and (5.8) of the must operational relations are lost.

Compatible refinement extends to the new syntax by the following rule.

(Comp fix) 
$$\frac{\Gamma \vdash u \ \overline{R} \ u': t}{\Gamma \vdash \mathbf{fix}^{(\alpha)} \ u \ \widehat{R} \ \mathbf{fix}^{(\alpha)} \ u': t}$$

Each of the rules for primitive reduction, (Redex<sub> $\Box$ </sub> fix zero), (Redex<sub> $\Box$ </sub> fix succ), and (Redex<sub> $\Box$ </sub> fix limit), violates Lemma 5.4.2. However, the lemma holds for compatible relations:

**Lemma 6.7.2** Suppose  $R \in REL$  is compatible and substitutive. Whenever  $a \hat{R} a' : t$  and  $a \rightarrow_{\Box} \{b_i \mid i < \omega\}$  there exist  $b'_0, b'_1, \cdots : t$  such that  $a' \rightarrow_{\Box} \{b'_i \mid i < \omega\}$  and  $b_i R b'_i : t$  for all  $i < \omega$ .

The proof is straightforward and omitted.

The definitions of must contextual approximation and equivalence from §6.1 are still meaningful in the extended language. But it is not clear whether the extension is *conservative*, that is, whether the instances of must contextual approximation and equivalence that one proves in the original language are true with respect to the definition of must contextual approximation and equivalence for the extended language.

For many purposes, including proofs by transfinite induction over approximate fixed point expressions, it is not important that the indexes range over recursive ordinals rather than, e.g., countable ordinals. However, one consequence of the restriction to recursive ordinals is that we can exploit the fact that these are expressible as  $\lambda$ -terms (Church and Kleene 1937) to encode the transfinite fixed point expressions in various other extensions of the language, e.g., using Boudol's parallel combinator (Boudol 1994), or by means of control operators and a global queue of continuations—the details of these two encodings can be found in Appendix A. Unfortunately, it does not seem possible to encode the transfinite fixed point expressions within the language (without transformation into continuation and state passing style or the like) which would be the easy way to show that the extension is conservative.

Of the results about must contextual approximation and equivalence from §6.5–6.6, (6.31), (6.32) and the implication  $(iii) \Rightarrow (ii)$  in Propostion 6.5.8 depend on the correspondence between the must operational relations and the may operational relations that holds only in the absence of the transfinite language extension. (The equivalence between (i) and (iii) in Propostion 6.5.8 remains valid; it can be proved by a suitable modification of Lemma 6.5.3.) The rules (6.33) and (6.34) are valid in the extended language only for must evaluations and must transitions that are deterministic.

All concrete instances of must contextual approximation and equivalence in the original language that are proved by the techniques in §6.5–6.6 carry over to the extended language.

The next lemma interrelates the transfinite fixed point expressions; (3) asserts that  $\mathbf{fix}^{(\lambda)} u$ , for a limit ordinal  $\lambda$ , is the least upper bound of the set of all  $\mathbf{fix}^{(\alpha)} u$  at ordinals  $\alpha < \lambda$ .

**Lemma 6.7.3** Whenever t is a function type and  $u: t \rightarrow t$ ,

- (1)  $\mathbf{fix}^{(0)} u \cong_{\Box} \mathbf{\Omega} : t$
- (2)  $\mathbf{fix}^{(\alpha+1)} u \cong_{\Box} u (\lambda x. (\mathbf{fix}^{(\alpha)} u) x) : t$
- $(3) \ \mathbf{fix}^{(\lambda)} \, u \sqsubseteq_{\sqcap} b: t \ \Leftrightarrow \ \forall \alpha < \lambda. \ \mathbf{fix}^{(\alpha)} \, u \sqsubseteq_{\sqcap} b: t$

**Proof** Straightforward, by reference to the CIU Theorem and the reduction rules for the transfinite fixed point expressions.  $\Box$ 

Recall from §4.5 the definition of the finite unwindings of **Y**. We are going to use the transfinite fixed point expressions to express transfinite unwindings of **Y**. For every  $\alpha < \omega_1^{CK}$ , we let  $\mathbf{Y}^{(\alpha)} \stackrel{\text{def}}{=} \lambda g$ . fix<sup>(\alpha)</sup> g. For finite ordinals we get from (1) and (2) in the previous lemma, by induction, that this definition is must contextually equivalent to that in §4.5.

It is now possible to formulate transfinite versions of the Unwinding Theorem and syntactic continuity.

**Theorem 6.7.4 (Must**  $\omega_1^{CK}$ -**Unwinding)** Suppose  $y: (t \rightarrow t) \rightarrow t \vdash a: t'$  and t is a function type. Then  $a[\mathbf{Y}/y]$  must terminate if and only if  $a[\mathbf{Y}^{(\alpha)}/y]$  must terminate for some  $\alpha < \omega_1^{CK}$ .

The proof is similar to those of the unwinding theorems for the deterministic language, Theorem 4.5.1, and the finitely branching sublanguage without countable choice, Theorem 6.6.2. The proof of the "only if" direction, given below, omits the argument for why the ordinal  $\alpha$  that is constructed is recursive; an account of this fact can be found in Lassen and Pitcher (1998) and Apt and Plotkin (1986).

For every  $\alpha < \omega_1^{CK}$ , let  $W_2^{(\alpha)}$  be the relation

$$W_?^{(\alpha)} \stackrel{\text{def}}{=} \left( \bigcup_{\gamma \ge \alpha} R_?^{(\gamma)} \right)^{\mathsf{SC}}.$$

where each  $R_{?}^{(\gamma)}$  is given by

$$g: t \rightarrow t \vdash \mathsf{fix}[g] \ R_?^{(\beta)} \ \mathbf{fix}^{(\beta)} \ g: t$$

for all function types t. In other words,  $W_{?}^{(\alpha)}$  is the smallest substitutive and compatible relation which satisfies

$$g: t \to t \vdash \mathsf{fix}[g] \ W_?^{(\alpha)} \ \mathbf{fix}^{(\beta)} \ g: t \tag{6.42}$$

and hence

$$\mathbf{Y} \ \overline{W_{?}^{(\alpha)}} \ \mathbf{Y}^{(\beta)} : (t \rightharpoonup t) \rightharpoonup t \tag{6.43}$$

for all  $\beta \ge \alpha$  and all function types t. In particular, whenever  $y : (t \rightharpoonup t) \rightharpoonup t \vdash a : t'$  and t is a function type,

$$a[\mathbf{Y}/y] W_{?}^{(\alpha)} a[\mathbf{Y}^{(\alpha)}/y] : t'$$
(6.44)

The  $W_{?}^{(\alpha)}$  relations form a decreasing sequence,

$$W_{?}^{(\alpha)} \subseteq W_{?}^{(\alpha')} \quad \text{iff} \quad \alpha' \le \alpha$$

$$(6.45)$$

The Must  $\omega_1^{CK}$ -Unwinding Theorem follows from (6.44) and the next two lemmas.

**Lemma 6.7.5 Y** is a  $\sqsubset_{\Box}$ -upper bound of its transfinite unwindings  $\{\mathbf{Y}^{(\alpha)} \mid \alpha < \omega_1^{CK}\}$ .

**Proof** As the proof of Lemma 4.5.2. The induction step for limit ordinals is immediate from Lemma 6.7.3(3).

**Proof** By induction on the derivation of  $a \rightsquigarrow_{\Box} U$ . (It is easy to see that the ordinal  $\alpha$  that is constructed in the course of the proof is countable, but the argument for why it is recursive is omitted.)

#### Case (Eval<sub> □</sub> value)

$$a$$
 is a value  
 $U = \{a\}$ 

Let  $\alpha = 0$ . Suppose  $a W_{?}^{(\beta)} a' : t$  for some  $\beta < \omega_{1}^{CK}$  and some a' : t. Since a is a value, a and a' are not related by any  $R_{?}^{(\gamma)}$ , so it must be the case that  $a \widetilde{W_{?}^{(\beta)}} a' : t$ . By Lemma 3.5.1, a' is a value too. Hence  $a' \rightsquigarrow_{\Box} \{a'\}$  and, by (3.11),  $a W_{?}^{(\beta)} a' : t$ , as required.

Case (Eval<sub> $\Box$ </sub> redex)

$$\begin{aligned} a \to_{\Box} \{ b_i \mid i < \omega \} \\ b_i \rightsquigarrow_{\Box} U_i, \text{ for all } i < \omega \\ U = \bigcup_{i < \omega} U_i \end{aligned}$$

We split the argument in two cases according to the form of a.

**Case** a = fix[v] for some  $v: t \rightarrow t$ . Then  $a \rightarrow \Box \{b_0\}$  where

$$b_0 = \operatorname{case} (\operatorname{inj} \lambda \operatorname{inj} y. v (\lambda x. y (\operatorname{inj} y) x)) \text{ of inj } y. v (\lambda x. y (\operatorname{inj} y) x)$$

and  $b_0 \rightsquigarrow_{\Box} U$  via (Eval<sub> $\Box$ </sub> redex),  $b_0 \rightarrow_{\Box} \{b\}$ , and  $b = v (\lambda x. ax) \rightsquigarrow_{\Box} U$ . By the induction hypothesis for  $b \rightsquigarrow_{\Box} U$ , there is an  $\alpha_0$  such that

$$\forall \beta < \omega_1^{CK}. \ \forall b': t. \ b \ W_?^{(\beta + \alpha_0)} \ b': t \ \Rightarrow \\ \exists U'. \ b' \rightsquigarrow_{\Box} U' \ \& \ \forall u' \in U'. \ \exists u \in U. \ u \ \overline{W_?^{(\beta)}} \ u': t$$

$$(6.46)$$

Let  $\alpha = \alpha_0 + 1$ . Suppose  $a W_?^{(\beta+\alpha)} a' : t$ , i.e., either

$$a R_{?}^{(\gamma)}[W_{?}^{(\beta+\alpha)}] a':t$$
 (6.47)

for some  $\gamma \geq \beta + \alpha$ , or

$$a \ \widehat{W_{?}^{(\beta+\alpha)}} \ a': t \tag{6.48}$$

In the first case  $a' = \mathbf{fix}^{(\gamma)} v'$  with  $v \overline{W_?^{(\beta+\alpha)}} v' : t \to t$ . Note that  $\gamma \ge \beta + \alpha \ge 1$ . If  $\gamma$  is a successor ordinal,  $\gamma = \gamma' + 1$  for some  $\gamma'$ , let  $b' = v'(\lambda x. \mathbf{fix}^{(\gamma')} v' x)$ . Then  $a' \to_{\Box} \{b'\}$  so that  $a' \rightsquigarrow_{\Box} U'$  whenever  $b' \rightsquigarrow_{\Box} U'$ . Since  $v \overline{W_?^{(\beta+\alpha_0)}} v' : t \to t$ , by (6.45), and  $\gamma' \ge \beta + \alpha_0$ , we get that  $b W_?^{(\beta+\alpha_0)} b' : t$ . By (6.46), the desired conclusion follows. If  $\gamma$  is a limit ordinal,  $\gamma \ge \beta + \alpha$  implies that  $\gamma > \beta + \alpha$  because  $\beta + \alpha$  is a successor ordinal. Therefore  $a' \to_{\Box} \{\mathbf{fix}^{(\beta+\alpha)} v'\}$ . The argument proceeds as above with  $\beta + \alpha_0$  in place of  $\gamma'$ .

In the second case, an inspection of the derivation of (6.48) reveals that  $a' = \operatorname{fix}[v']$  with  $v \overline{W_{?}^{(\beta+\alpha)}} v': t \rightarrow t$ . Let  $b' = v' (\lambda x. a' x)$ . We see that  $a' \rightsquigarrow_{\Box} U'$  whenever  $b' \rightsquigarrow_{\Box} U'$ , by two applications of (Eval<sub>\Bar[\top]</sub> redex). Moreover,  $b W_{?}^{(\beta+\alpha_0)} b': t$ . By (6.45) and (6.46), the desired conclusion follows.

**Case** *a* is not of the form fix[*v*] for any *v*. Then we know that  $a W_{?}^{(\beta)} a' : t$  if and only if  $a \widetilde{W_{?}^{(\beta)}} a' : t$ , for all  $\beta < \omega_{1}^{CK}$  and all a' : t. By the induction hypothesis, there is an  $\alpha_{i}$  for each  $i < \omega$  such that

$$\forall \beta < \omega_1^{CK}. \ \forall b'_i : t. \ b_i \ W_?^{(\beta + \alpha_i)} \ b'_i : t \Rightarrow \\ \exists U'_i. \ b'_i \rightsquigarrow_{\Box} U'_i \ \& \ \forall u' \in U'_i. \ \exists u \in U_i. \ u \ \overline{W_?^{(\beta)}} \ u' : t \end{cases}$$
(6.49)

Let  $\alpha = \bigcup_{i < \omega} \alpha_i$ . Now suppose that  $a W_?^{(\beta+\alpha)} a' : t$ , for some  $\beta < \omega_1^{CK}$  and some a' : t. Then  $a W_?^{(\beta)} a' : t$  and from Lemma 6.7.2 we obtain that  $a' \to_{\Box} \{b'_i \mid i < \omega\}$  for some  $b'_0, b'_1, \cdots : t$  such that  $b_i W_?^{(\beta+\alpha)} b'_i : t$  for all  $i < \omega$ . By (6.45) and (6.49), for each  $i < \omega$  there exists  $U'_i$  such that  $b'_i \rightsquigarrow_{\Box} U'_i$  and  $\forall u' \in U'_i$ .  $\exists u \in U_i$ .  $u W_?^{(\beta)} u' : t$ . Let  $U' = \bigcup_{i < \omega} U'_i$ . Then  $a' \rightsquigarrow_{\Box} U'$ , by (Eval\_{\Box} redex), and  $\forall u' \in U'$ .  $\exists u \in U$ .  $u W_?^{(\beta)} u' : t$ , as required. Case (Eval<sub> $\Box$ </sub> let)

$$a = \mathbf{let} \ x = a_1 \ \mathbf{in} \ b_2$$
$$a_1 : t' \quad \text{and} \quad x : t' \vdash b_2 : t$$
$$a_1 \rightsquigarrow_{\Box} V$$
$$b_2[v/x] \rightsquigarrow_{\Box} U_v, \text{ for all } v \in V$$
$$U = \bigcup_{v \in V} U_v$$

By the induction hypothesis, there is an  $\alpha_1$  such that

$$\forall \beta < \omega_1^{CK}. \ \forall a_1': t'. \ a_1 \ W_?^{(\beta+\alpha_1)} \ a_1': t' \Rightarrow$$
  
$$\exists V'. \ a_1' \rightsquigarrow_{\Box} V' \ \& \ \forall v' \in V'. \ \exists v \in V. \ v \ \overline{W_?^{(\beta)}} \ v': t'$$
 (6.50)

and for each  $v \in V$  there is a  $\alpha_v$  such that

$$\forall \beta < \omega_1^{CK}. \ \forall b': t. \ b_2[v]x] \ W_?^{(\beta+\alpha_v)} \ b': t \Rightarrow$$
  
$$\exists U'_{b'}. \ b' \rightsquigarrow_{\Box} U'_{b'} \ \& \ \forall u' \in U'_{b'}. \ \exists u \in U_v. \ u \ \overline{W_?^{(\beta)}} \ u': t$$
 (6.51)

Let  $\alpha_2 = \bigcup_{v \in V} \alpha_v$  and let  $\alpha = \alpha_2 + \alpha_1$ . Now suppose that  $a W_?^{(\beta+\alpha)} a' : t$ , for some  $\beta < \omega_1^{CK}$  and some a' : t. Since a is not of the form fix[...], a and a' are not related by any  $R_?^{(\beta')}$ , so it must be the case that  $a W_?^{(\beta+\alpha)} a' : t$ . Thus  $a' = \text{let } x = a'_1 \text{ in } b'_2$  with  $a_1 W_?^{(\beta+\alpha)} a'_1 : t'$  and  $x : t' \vdash b_2 W_?^{(\beta+\alpha)} b'_2 : t$ . Now we use (6.50) to deduce that there exists V' such that for every  $v' \in V'$ , there exists  $v \in V$  such that  $v W_?^{(\beta+\alpha_2)} v' : t'$ . By (6.45),  $b_2[v/x] W_?^{(\beta+\alpha_v)} b'_2[v'/x] : t'$  because  $W_?^{(\beta+\alpha_v)}$  is substitutive. From (6.51) we get a set  $U'_{v'}$  such that  $b'_2[v'/x] \rightsquigarrow_{\Box} U'_{v'}$  and for all  $u' \in U'_{v'}$ , there exists  $u \in U_v$  with  $u W_?^{(\beta)} u' : t$ .

Finally, the desired conclusion follows with  $U' = \bigcup_{v' \in V'} U'_{v'}$ .  $\Box$ 

**Proof of the Must**  $\omega_1^{CK}$ -Unwinding Theorem 6.7.4 Consider the two implications separately.

If  $a[\mathbf{\tilde{Y}}/y]$  must terminate, according to Lemma 6.7.6 there exists an  $\alpha < \omega_1^{CK}$  such that whenever  $a[\mathbf{Y}/y] W_?^{(\alpha)} a' : t'$  then a' must terminate too. In particular,  $a[\mathbf{Y}^{(\alpha)}/y]$  must terminate because of (6.44).

Conversely, suppose  $a[\mathbf{Y}^{(\alpha)}/y]$  must terminate. By the facts (6.44) and (6.45), we see that  $a[\mathbf{Y}/y] W_{?}^{(0)} a[\mathbf{Y}^{(\alpha)}/y] : t'$ . We conclude that  $a[\mathbf{Y}/y]$  must terminate because  $W_{?}^{(0)^{\text{op}}}$  is must-pre-adequate, by Lemma 6.7.5.

**Proposition 6.7.7 (Syntactic must**  $\omega_1^{CK}$ -continuity) If  $y : (t \rightarrow t) \rightarrow t \vdash a : t'$  and t is a function type,  $a[\mathbf{Y}/y] \sqsubset_{\Box} a' : t'$  if and only if  $\forall \alpha < \omega_1^{CK} . a[\mathbf{Y}^{(\alpha)}/y] \sqsubset_{\Box} a' : t'$ .

The proof is analogous to that of Proposition 4.5.4.

As an application of the syntactic  $\omega_1^{CK}$ -continuity rule for must contextual approximation, we extend the proof of recursion induction in Proposition 4.5.5 to must contextual approximation. Proposition 6.7.8 (Recursion induction)  $\frac{u v' \sqsubset_{\Box} v' : t_1 \rightharpoonup t_2}{\mathbf{Y} u \sqsubset_{\Box} v' : t_1 \rightharpoonup t_2}$ 

**Proof** Assume that the premise holds. By syntactic  $\omega_1^{CK}$ -continuity, it suffices to show that  $\mathbf{Y}^{(\alpha)} u \leq_{\Box} v' : t_1 \rightarrow t_2$ , for all  $\alpha < \omega_1^{CK}$ . We prove this by transfinite induction on  $\alpha$ . If  $\alpha = 0$  or  $\alpha$  is a successor ordinal, we argue as in the induction over natural numbers in the proof of Proposition 4.5.5. If  $\alpha$  is a recursive limit ordinal, the induction step is immediate from Lemma 6.7.3(3).

The unresolved problem, mentioned earlier, of whether the transfinite fixed point operators are a conservative extension of the language, is relevant if we want to use syntactic  $\omega_1^{CK}$ continuity for reasoning about must contextual approximation and equivalence in the original language. For instance, the proof of recursion induction, given above, requires that the premise holds for the must contextual approximation relation in the extended language. Clearly, it would be nice if this coincides with the original must contextual approximation relation on expressions from the original language because then the recursion induction proof rule can immediately be used to reason about the latter.

# 6.8 Sequentiality

Recall the contextual equivalence between u and u' from Example 4.4.1,

$$u = \lambda f. (f \langle \mathbf{I}, \mathbf{U} \rangle; f \langle \mathbf{U}, \mathbf{I} \rangle)$$
  
$$u' = \lambda f. f \langle \mathbf{U}, \mathbf{U} \rangle$$

where  $\mathbf{I} = \lambda x. x$  and  $\mathbf{U} = \lambda x. \Omega$ . It shows that the language is sequential by the non-existence of a parallel convergence tester function.

This equivalence is not valid for may contextual equivalence: when applied to  $f \stackrel{\text{def}}{=} \lambda \langle g, h \rangle . g \langle \rangle$  or  $h \langle \rangle$ , one may terminate,  $u f \rightsquigarrow_{\Diamond} \langle \rangle$ , whereas the other, u' f, must diverge. In effect, we cannot observe the difference between the parallel convergence tester function and the function that nondeterministically applies one or the other argument function.

But the equivalence holds for must contextual equivalence:

$$u \cong_{\Box} u' : ((\text{unit} \to \text{unit}) \times (\text{unit} \to \text{unit}) \to \text{unit}) \to \text{unit}$$
(6.52)

The proof from Example 4.4.1 generalises to the nondeterministic case if, instead of (4.13), we prove

$$(a_1, a_2, a') \in R^{\mathsf{C}}|_t \& a_1 \rightsquigarrow_{\Box} U_1 \& a_2 \rightsquigarrow_{\Box} U_2 \Rightarrow \exists U'. a' \rightsquigarrow_{\Box} U' \& \forall u' \in U'. \exists u_1 \in U_1, u_2 \in U_2. (u_1, u_2, u') \in \overline{R^{\mathsf{C}}}|_t$$

The proof is by induction on the derivation of  $a_1 \rightsquigarrow_{\Box} U_1$  and proceeds quite analogous to the proof from Example 4.4.1.

The next example is a nondeterministic variant of this equivalence.

**Example 6.8.1** Let  $t = \text{unit} \rightarrow \text{unit}$  and let  $a, a' : t \times t$  be the expressions

$$a = \langle \mathbf{I}, \mathbf{U} \rangle \text{ or } \langle \mathbf{U}, \mathbf{I} \rangle$$
  
 $a' = \langle \mathbf{U}, \mathbf{U} \rangle$ 

They are must contextually equivalent,  $a \cong_{\Box} a' : t \times t$ .

It is easy to calculate that a' approximates a. First,  $a' \cong_{\Box} (a' \text{ or } a') : t \times t$  because or is idempotent. Next,  $(a' \text{ or } a') \sqsubset_{\Box} a : t \times t$ , because  $\Omega \sqsubset_{\Box} \langle \rangle :$  unit, and by compatibility; and then  $a' \sqsubset_{\Box} a : t \times t$ , by transitivity.

The converse,  $a \sqsubset_{\square} a' : t \times t$ , is more interesting. According to Proposition 6.5.8 it suffices to show that  $E[\![a]\!] \rightsquigarrow_{\square} \langle \rangle$  implies  $E[\![a']\!] \rightsquigarrow_{\square} \langle \rangle$  for all evaluation contexts E such that  $\bullet : t \times t \vdash E :$  unit. From (Redex<sub>\u03c0</sub> or), (Trans<sub>\u03c0</sub> redex), and the fact that transitions are closed under evaluation contexts, we get

$$E\llbracket a \rrbracket \rightarrowtail_{\Box} \{ E\llbracket \langle \mathbf{I}, \mathbf{U} \rangle \rrbracket, E\llbracket \langle \mathbf{U}, \mathbf{I} \rangle \rrbracket \}$$

and then, by (5.10), we get that  $E[\![a]\!] \rightsquigarrow_{\Box} \langle \rangle$  if and only if both  $E[\![\langle \mathbf{I}, \mathbf{U} \rangle]\!] \rightsquigarrow_{\Box} \langle \rangle$  and  $E[\![\langle \mathbf{U}, \mathbf{I} \rangle]\!] \rightsquigarrow_{\Box} \langle \rangle$ . Let  $v : (t \times t) \rightarrow t$  be the function  $\lambda x. E[\![x]\!]$ , and consider (6.52). Notice that  $u v \rightsquigarrow_{\Box} \{\langle \rangle\}$ . Since must contextual equivalence is compatible and must adequate, we get that  $u' v \rightsquigarrow_{\Box} \{\langle \rangle\}$  too. This means that  $E[\![a']\!] \rightsquigarrow_{\Box} \langle \rangle$ , as we had to show.  $\Box$ 

The fact that Example 4.4.1 is valid for must contextual equivalence means that sequentiality is observable for this modality. In fact, sequentiality is more observable in the presence of countable nondeterminism. The next example shows that we can observe the absence of a function  $\exists_v : ((\text{unit} \rightarrow \text{nat}) \rightarrow \text{bool}) \rightarrow \text{bool}$ , with the functionality:

 $\begin{array}{ll} \exists_{v} v \rightsquigarrow_{\Box} \{ \mathbf{false} \} & \text{if } v \left( \lambda \langle \rangle . \, \mathbf{\Omega} \right) \rightsquigarrow_{\Box} \{ \mathbf{false} \} \\ \exists_{v} v \rightsquigarrow_{\Box} \{ \mathbf{true} \} & \text{if } \exists i < \omega. \ v \left( \lambda \langle \rangle . \, \ulcorner i \urcorner \right) \rightsquigarrow_{\Box} \{ \mathbf{true} \} \\ \exists_{v} v \text{ may diverge} & \text{otherwise} \end{array}$ 

This is a call-by-value version of the existential quantifier function in Plotkin (1977). It is computable, but it needs to apply its argument v to many inputs in parallel.

#### Example 6.8.2 Let

$$\begin{array}{lll} a & = & \mbox{if } g \left( \lambda h. \, \mbox{false} \right) \\ & \mbox{then } \Omega \\ & \mbox{else let } x = \mbox{? in if } g \left( \lambda h. \, \mbox{test}[x,h] \right) \, \mbox{then } \langle \, \rangle \, \mbox{else } \Omega \\ \\ & \mbox{where } & \mbox{test}[n,v] \, = \, \mbox{let } y = v \, \langle \, \rangle \, \mbox{in if } \mbox{eq} \, n \, y \, \mbox{then true else } \Omega \end{array}$$

Then a is must contextually equivalent to  $\Omega$ ,

$$g: ((\text{unit} \rightarrow \text{nat}) \rightarrow \text{bool}) \rightarrow \text{bool} \vdash a \cong_{\Box} \Omega: \text{unit}$$

To prove this, suppose  $u : ((\text{unit} \rightarrow \text{nat}) \rightarrow \text{bool}) \rightarrow \text{bool}$ . Then we observe that  $a[u/g] \rightsquigarrow_{\Box} \{\langle \rangle \}$  if and only if  $u(\lambda h. \text{false}) \rightsquigarrow_{\Box} \{\text{false}\}$  and  $u(\lambda h. \text{test}[\ulcorneri\urcorner, h]) \rightsquigarrow_{\Box} \{\text{true}\}$  for all  $i < \omega$ . We will now prove that no such u exists. Then we can conclude that a[u/g] may diverge for every u and thus, by Proposition 6.5.5, that a and  $\Omega$  are must contextually equivalent.

The desired result follows if, for every  $u : ((unit \rightarrow nat) \rightarrow bool) \rightarrow bool$ ,

$$(\forall i < \omega. \ u \ (\lambda h. \mathsf{test}[\ulcorneri\urcorner, h]) \rightsquigarrow_{\Box} \{\mathsf{true}\} \implies u \ (\lambda h. \mathsf{false}) \rightsquigarrow_{\Box} \{\mathsf{true}\}$$
(6.53)

We can prove this by an analogous relational argument to that in Example 6.8.1, this time with a relation of arity  $\omega + 1$ .

Let R be the smallest  $(\omega + 1)$ -ary open relation such that

 $h : \text{unit} \rightarrow \text{nat} \vdash (\text{test}[\ulcorneri\urcorner, h])_{i < \omega} R \text{ false} : \text{bool}$ 

(As in Example 4.4.2,  $\vec{x} : \vec{t} \vdash (a_i)_{i < \omega} S a_\omega : t$  is notation for  $(a_i)_{i < \omega+1} \in S|_{(\vec{t})t}$ .) Then

$$\begin{aligned} &(a_i)_{i < \omega} R^{\mathsf{SC}} a: t \& (\forall i < \omega. \ a_i \rightsquigarrow_{\Box} U_i) \Rightarrow \\ &\exists U. \ a \rightsquigarrow_{\Box} U \& \forall u \in U. \ \exists u_0 \in U_0, u_1 \in U_1, \dots \ (u_i)_{i < \omega} \overline{R^{\mathsf{SC}}} u: t \end{aligned}$$

$$(6.54)$$

can be proved by induction on the derivation of  $a_0 \rightsquigarrow_{\Box} u_0$ , as in Example 6.8.1.

Finally, we note that (6.54) implies (6.53), as required.

The example shows that the abovementioned function  $\exists_v$  is not definable in our language because  $(\lambda g. a) \exists_v \rightsquigarrow_{\Box} \{\langle \rangle\}$  whereas  $(\lambda g. \Omega) \exists_v$  diverges.

# 6.9 Related work

The results in this chapter illustrate the versatility of our relational approach. The only other extensive study of contextual equivalences from first principles in a nondeterministic context is that of Agha, Mason, Smith, and Talcott (1997) for an actor language. They also find it necessary to develop sophisticated syntactic methods, involving generalised forms of contexts akin to our substituting contexts from §2.4.3, in order to deal with the syntactic complexity of the task. The actor language has primitives for fair, asynchronous communication and dynamic process creation and, thus, provides more challenges for the semantic models. Consequently, the results about contextual equivalences that are obtained in the cited paper are rather limited.

In programming languages with unbounded nondeterminism the  $\omega$ -continuity properties required by conventional domain theory fail. Apt and Plotkin (1986) showed that this is inevitable, and it complicates the models needed for a denotational semantics; either one turns to  $\omega_1$ -complete partial orders for modelling countable nondeterminism (Plotkin 1982; Di Gianantonio, Honsell, and Plotkin 1995) or category-theoretic fixed-point semantics (see below). Operational approaches are generally not based on continuity properties and are not complicated by the presence of countable nondeterminism. Specifically, the CIU Theorem remains valid and applicative similarity remains a congruence (Lassen 1997; Lassen and Pitcher 1998).

The transfinite Unwinding Theorem and syntactic continuity proof rule as well as the associated transfinite syntax and its operational semantics in §6.7 appear to be new. I have not been able to make useful connections between this work and similar notions of infinite terms in the literature, e.g., in term rewriting (Kennaway, Klop, Sleep, and de Vries 1995) or recursion theory (Schwichtenberg 1996), although the fixed point operators bounded by well-orderings in Schwichtenberg and Wainer (1995) bear some resemblance to our transfinite fixed point expressions.

An alternative approach to dealing with the discontinuity induced by unbounded nondeterminism is the category-theoretic fixed-point semantics of (Lehmann 1976). The approximation ordering between elements is expressed by morphisms between objects in a category, rather than by a partial order on a set as in domain theory, and fixed points are obtained as co-limits of  $\omega$ -diagrams. By operating with a 'more detailed' notion of approximation than

that of a partial order, it is possible to construct an  $\omega$ -complete categorical powerdomain which can be used to model unbounded nondeterminism. Abramsky (1983) and Panangaden and Russell (1989) use this approach to give  $\omega$ -continuous, least fixed-point semantics of simple languages with unbounded nondeterminism. But the catch is that the models are not fully abstract. Panangaden and Russell show how their model collapses via a discontinuous abstraction function to a fully abstract semantics. Apt and Plotkin (1986) showed that it is impossible to have a fully abstract  $\omega$ -continuous, least fixed-point semantics of unbounded nondeterminism, so we cannot eschew discontinuity. However, it would be interesting to see if ideas from Lehmann's category-theoretic approach can be used in an operational setting.

# Chapter 7

# Simulation

In this chapter, we are going to investigate co-inductively defined simulation preorders and equivalences that are related to the may and must contextual relations from the previous chapter. One might expect this to be a straightforward exercise, given the theory that we have developed for the contextual relations (Chapter 6), and given the well developed theory of applicative similarity and its relationship with contextual approximation in the deterministic case (Chapter 4). But it turns out that things are not so easy. The simulation preorders that we introduce, lower and upper similarity, are contained in but not identical to the corresponding contextual approximation preorders. Furthermore, lower similarity does not enjoy the syntactic continuity property. In the countably nondeterministic case, it is not clear whether upper similarity satisfies the syntactic  $\omega_1^{CK}$ -continuity property of must contextual approximation.

Even so, the similarity preorders provide sound co-inductive reasoning principles which can be used to reason about the contextual approximation preorders as well. The key results of this chapter are that lower and upper similarity are pre-congruences and hence are contained in may and must contextual approximation, respectively. Moreover, syntactic continuity is shown to be valid for upper similarity in the case of bounded nondeterminism.

At the end of the chapter, some open problems are presented, followed by a discussion of related work.

# 7.1 Lower similarity

Let *lower similarity*,  $\leq_{\diamond} \in REL_0$ , be defined co-inductively as the greatest fixed point of  $\langle \cdot \circ \rangle_{\diamond}$ .

$$\lesssim_{\diamond} \stackrel{\text{def}}{=} \nu \boldsymbol{R}. \langle \boldsymbol{R}^{\circ} \rangle_{\diamond}$$

Lower similarity is the greatest lower simulation and its open extension,  $\lesssim^{\circ}_{\diamond}$ , is the greatest open lower simulation.

The open extension of lower similarity is a pre-congruence. The proof is virtually identical to that of deterministic similarity from §4.8. The main lemma uses Lemma 5.4.1 in place of Lemma 3.10.1.

**Lemma 7.1.1** If  $\mathbf{R} \subseteq \langle \mathbf{R}^{\circ} \rangle_{\Diamond}$  then  $\mathbf{R}^{*\bullet} \subseteq \langle \mathbf{R}^{*\bullet} \rangle_{\Diamond}^{\circ}$ .

**Proposition 7.1.2**  $\lesssim^{\circ}_{\Diamond}$  is a pre-congruence.

Lower similarity is may-pre-adequate, because it is a lower simulation. As it is also compatible, it is included in may contextual approximation. The inclusion is strict. To see this, recall from Example 6.4.4 that the following two expressions are may contextually equivalent.

$$\begin{array}{rcl} a & = & \lambda \langle \, \rangle . \, ? \\ a' & = & \operatorname{let} \, y = ? \, \operatorname{in} \, \lambda \langle \, \rangle . \, min(?,y) \end{array}$$

But  $a \not\leq_{\diamond} a'$ : unit  $\rightarrow$  nat, because the latter evaluates to some bounded choice function  $\lambda\langle\rangle$ .  $min(?, \lceil n \rceil)$ , and ? is not lower similar to  $min(?, \lceil n \rceil)$ , for any n. For instance, ?  $\rightsquigarrow_{\diamond} \lceil n + 1 \rceil$  whereas  $min(?, \lceil n \rceil) \rightsquigarrow_{\diamond} \lceil m \rceil$  only if  $m \leq n$ , but  $\lceil n + 1 \rceil \leq_{\diamond}^{\diamond} \lceil m \rceil$  only if m = n + 1.

From Example 6.4.4 we can also derive that syntactic continuity is invalid for lower similarity. Recall that a is may contextual equivalent to  $\mathbf{Y} u$ , where

$$u = \lambda f. \lambda \langle \rangle. 0 \text{ or } (\text{let } y = f \langle \rangle \text{ in succ } y)$$

They are also lower similar,  $a \leq \mathbf{v} \mathbf{Y} u$ : unit  $\rightarrow$  nat, and hence  $\mathbf{Y} u$  is not lower similar to a'. But, by inspection of the proof of Example 6.4.4, we see that all its finite unwindings are.

## 7.2 Upper similarity

Upper similarity,  $\leq_{\Box} \in REL_0$ , is the greatest fixed point of  $\langle \cdot \circ \rangle_{\Box}$ .

$$\lesssim_{\Box} \stackrel{\text{def}}{=} \nu \boldsymbol{R}. \langle \boldsymbol{R}^{\circ} \rangle_{\Box}$$

Upper similarity is the greatest upper simulation and its open extension,  $\leq_{\Box}^{\circ}$ , is the greatest open upper simulation.

We can prove that  $\leq_{\Box}^{\circ}$  is a pre-congruence by Howe's method. Firstly, the compatible extension of the reflexive transitive closure of any upper simulation is itself an upper simulation:

**Lemma 7.2.1** If  $\mathbf{R} \subseteq \langle \mathbf{R}^{\circ} \rangle_{\Box}$  then  $\mathbf{R}^{*\bullet} \subseteq \langle \mathbf{R}^{*\bullet} \rangle_{\Box}^{\circ}$ .

**Proof** Analogous to the proof of Lemma 4.8.1.

From this lemma we derive that open upper similarity is a pre-congruence:

**Proposition 7.2.2**  $\leq_{\Box}^{\circ}$  is a pre-congruence.

**Proof** Analogous to that of Proposition 4.8.2.

Upper similarity is an upper simulation, so it is must-pre-adequate, and the fact that it is a pre-congruence implies that it is included in must contextual approximation. The inclusion is strict. To see this, recall Example 6.8.1,

$$egin{array}{rcl} a&=&\langle {f I},{f U}
angle \; {f or} \; \langle {f U},{f I}
angle \ a'&=&\langle {f U},{f U}
angle \end{array}$$

where  $\mathbf{I} = \lambda x. x$  and  $\mathbf{U} = \lambda x. \Omega$ .

They are not upper similar, because  $a \rightsquigarrow_{\Box} \{ \langle \mathbf{I}, \mathbf{U} \rangle, \langle \mathbf{U}, \mathbf{I} \rangle \}$  and  $a' \rightsquigarrow_{\Box} \{ \langle \mathbf{U}, \mathbf{U} \rangle \}$  but neither of the following hold

since  $\langle \rangle \not\leq_{\Box} \mathbf{\Omega}$  : unit.

In Example 6.8.1 we showed that a and a' are must contextually equivalent. Hence must contextual approximation is not included in upper similarity.

### 7.3 Other simulation and bisimulation relations

Suppose we define *lower bisimilarity*,  $\sim_{\diamond} \in REL_0$ , as the greatest fixed point of  $\langle \cdot \circ \rangle_{\diamond} \cap \langle \cdot \circ \circ \rangle_{\diamond}^{\text{op}}$ .

$$\sim_{\Diamond} \stackrel{\text{def}}{=} \nu \boldsymbol{R}. \langle \boldsymbol{R}^{\circ} \rangle_{\Diamond} \cap \langle \boldsymbol{R}^{\mathrm{op}\circ} \rangle_{\Diamond}^{\mathrm{op}}$$

Then we find that it is the greatest symmetric lower simulation but, contrary to the deterministic case, it is strictly smaller than the symmetrisation of lower similarity, as illustrated by the following example from Ong (1993).

$$(\lambda x. \Omega)$$
 or  $(\lambda x. \lambda y. \Omega) \quad \mathbf{R} \quad \lambda x. (\Omega \text{ or } \lambda y. \Omega) : \text{unit} \rightarrow \text{unit} \rightarrow \text{unit}$  (7.1)

This holds if  $\mathbf{R}$  is  $\leq_{\diamond}$  or  $\leq_{\diamond}^{\mathrm{op}}$  but is false if  $\mathbf{R}$  is  $\sim_{\diamond}$  because the left hand side may evaluate to  $\lambda x$ .  $\mathbf{\Omega}$  and this cannot be matched by a lower bisimilar outcome of the right hand side.

Since lower bisimilarity is not the symmetrisation of lower similarity, the fact that lower similarity is a pre-congruence does not entail that lower bisimilarity is a congruence. A separate proof is needed.

**Proposition 7.3.1**  $\sim^{\circ}_{\Diamond}$  is a congruence.

**Proof**  $\sim_{\diamond}$  is a simulation so  $\sim_{\diamond}^{*\bullet}$  is a simulation, by Lemma 7.1.1. Since lower simulations are closed under composition,  $(\sim_{\diamond}^{*\bullet})^*$  is a simulation too. By Lemma 3.8.2,  $(\sim_{\diamond}^{*\bullet})^*$  is symmetric. Hence it is included in  $\sim_{\diamond}^{\circ}$ . The reverse inclusion also follows from Lemma 3.8.2. We conclude that  $\sim_{\diamond}^{\circ}$  is compatible, since  $\sim_{\diamond}^{\bullet}$  is. Finally, the calculation

$$\sim^{\circ}_{\diamondsuit}^{*} = \sim^{*}_{\diamondsuit}^{\circ} \subseteq (\sim^{*}_{\diamondsuit}^{\bullet})^{*} \subseteq \sim^{\circ}_{\diamondsuit}$$

shows that  $\sim^{\circ}_{\Diamond}$  is reflexive and transitive.

Upper bisimilarity,  $\sim_{\Box} \in REL_0$ , is the greatest fixed point of  $\langle \cdot \circ \rangle_{\Box} \cap \langle \cdot \circ \rangle_{\Box}^{\circ p} \rangle_{\Box}^{\circ p}$ . From the example (7.1), one can see that it is strictly smaller than the symmetrisation of upper similarity.

By the same argument as in the proof of Proposition 7.3.1, it follows from Lemma 7.2.1 that upper bisimilarity a congruence:

**Proposition 7.3.2**  $\sim_{\Box}^{\circ}$  is a congruence.

By different combinations of the lower and upper simulation operators, one can define other notions of similarity and bisimilarity. The greatest fixed point of the disjunction of the lower and upper simulation operators,  $\langle \cdot \circ \rangle_{\diamond} \cap \langle \cdot \circ \rangle_{\Box}$ , is *convex similarity*; and the greatest symmetric fixed point is *convex bisimilarity*; see Lassen and Pitcher (1998). Convex similarity corresponds to the bisimulation preorder studied by Ong (1993) and convex bisimilarity corresponds to the bisimulation equivalence considered by Howe (1996). By analogous arguments to those in the proofs of Propositions 7.3.1 and 7.3.2, we can derive from Lemmas 7.1.1 and 7.2.1 that convex similarity and bisimilarity are compatible.

By definition, convex similarity is both a lower and an upper simulation. Hence it is included in the lower and upper similarity preorders. But convex similarity is not just their intersection because convex similarity is more discriminative with respect to the higher order nondeterministic branching structure of functions. As we noted earlier, the example from Ong (1993),

$$(\lambda x. \Omega)$$
 or  $(\lambda x. \lambda y. \Omega)$  **R**  $\lambda x. (\Omega \text{ or } \lambda y. \Omega) : unit \rightarrow unit \rightarrow unit$ 

holds for lower and upper similarity in place of R but Ong shows that it fails for convex similarity.

Then what about convex similarity versus the intersection of lower and upper bisimilarity? They are again different. For instance,  $\Omega$  is convex similar to  $\langle \rangle$  but these are neither lower bisimilar nor upper bisimilar. Conversely, ( $\Omega$  or  $\lambda x. \langle \rangle$ ) and ( $\Omega$  or ( $\lambda x. \Omega$  or  $\langle \rangle$ )) are lower and upper bisimilar but they are not convex similar.

Convex bisimilarity is included in the symmetrisation of convex similarity. The following example shows that the inclusion is strict.

$$\begin{array}{rcl}
a &=& (\lambda\langle\rangle, \mathbf{\Omega}) \ \mathbf{or} \ (\lambda\langle\rangle, \langle\rangle) \\
a' &=& a \ \mathbf{or} \ (\lambda\langle\rangle, \mathbf{\Omega} \ \mathbf{or} \ \langle\rangle)
\end{array}$$
(7.2)

a and a' are mutually convex similar but not convex bisimilar. Convex bisimilarity is both a lower and an upper bisimulation and hence included in the intersection of lower and upper bisimilarity. The counterexample for convex similarity above shows that this is a strict inclusion.

Yet another similarity preorder, which we call refinement similarity and denote by  $\leq_{\mathsf{R}} \in REL_0$ , is obtained as the greatest fixed point of  $\langle \cdot {}^{\mathrm{op}} \circ \rangle_{\diamond}^{\mathrm{op}} \cap \langle \cdot \circ \rangle_{\Box}$ . Refinement similarity is a pre-congruence. This can be derived from Lemmas 7.1.1 and 7.2.1 and an additional argument involving the transitive closure of compatible extension.<sup>1</sup> We omit the details.

Refinement similarity is an upper simulation and its reciprocal is a lower simulation. Hence refinement similarity is included in  $\leq^{\text{op}}_{\Diamond} \cap \leq_{\Box}$ . The inclusion is strict; e.g., Ong's example (7.1) fails for refinement similarity.

# 7.4 Syntactic continuity

At the end of §7.1 we saw that the syntactic continuity property is invalid for lower similarity. The situation is different for upper similarity in the case of bounded nondeterminism ordinary  $\omega$ -continuity holds for the finitely branching sublanguage without countable choice.

<sup>&</sup>lt;sup>1</sup>The proof is due to Corin Pitcher (Personal communication, September 1997). He also provided the example (7.2).

**Proposition 7.4.1 (Syntactic upper**  $\omega$ -continuity) If  $y : (t \rightarrow t) \rightarrow t \vdash a : t'$  and t is a function type,  $a[\mathbf{Y}/y] \lesssim_{\Box} a' : t'$  if and only if  $\forall n < \omega$ .  $a[\mathbf{Y}^{(n)}/y] \lesssim_{\Box} a' : t'$ .

**Proof** We employ the lemmas from the proof of the Must  $\omega$ -Unwinding Theorem from §6.6 to give a co-inductive proof. The proof is similar to that for the deterministic case; see Lassen (1998) and Pitts (1997a).

The forward implication follows from Lemma 6.6.3 (one can check that the lemma holds for upper similarity as well as must contextual approximation).

The backward implication is more interesting. As in our proof of syntactic continuity for contextual approximation, Proposition 4.5.4, we construct the relation

$$T \stackrel{\text{def}}{=} \bigcap_{n < \omega} (W^{(n)} \lesssim_{\Box}^{\circ})$$

Observe that  $a[\mathbf{Y}/y] T a' : t'$  if  $\forall n < \omega$ .  $a[\mathbf{Y}^{(n)}/y] \lesssim_{\Box} a' : t'$ . We show that T is an open upper simulation, then  $T \subseteq \lesssim_{\Box}^{\circ}$  and the result follows.

So suppose a T a' : t and  $a \sim_{\Box} U$ . By Lemma 6.6.4, there exists an  $n < \omega$  such that

$$\forall m < \omega. \ \forall b: t. \ a \ W^{(m+n)} \ b: t \Rightarrow \exists V. \ b \rightsquigarrow_{\Box} V \ \& \ \forall v \in V. \ \exists u \in U. \ u \ \overline{W^{(m)}} \ v: t$$

$$(7.3)$$

By definition of T, for all  $m < \omega$ ,  $a W^{(m+n)} b_m : t$ , for some  $b_m \leq_{\Box} a' : t$ . From (7.3) we get  $b_m \rightsquigarrow_{\Box} V_m$  such that

$$\forall v_m \in V_m. \ \exists u \in U. \ u \ \overline{W^{(m)}} \ v_m : t$$
(7.4)

Since  $b_m \leq_{\Box} a' : t$  also  $a' \rightsquigarrow_{\Box} U'_m$  such that

$$\forall u' \in U'_m. \ \exists v_m \in V_m. \ v_m \ \overline{\lesssim_{\square}^{\circ}} \ u' : t$$
(7.5)

The must evaluation relation is deterministic. Therefore all  $U'_m$  are identical. Let U' denote this set (i.e.,  $U' = U'_0 = U'_1 = ...$ ) and fix any  $u' \in U'$ .

By (7.5) there is an  $\omega$ -sequence of values,  $(v_m)_{m < \omega}$ , such that, for all  $m < \omega$ ,  $v_m \in V_m$ and  $v_m \leq_{\Box}^{\circ} u' : t$  and, by (7.4),  $u_m \overline{W^{(m)}} v_m : t$ , where  $(u_m)_{m < \omega}$  is an  $\omega$ -sequence of values in U.

But U is finite, by the assumption that nondeterminism is bounded. Therefore there is some  $u \in U$  that occurs infinitely often in the sequence  $(u_m)_{m < \omega}$ . This ensures that the  $\omega$ -sequence  $(v'_m)_{m < \omega}$  where

$$v'_m = \begin{cases} v_m & \text{if } u_m = u \\ v'_{m+1} & \text{otherwise} \end{cases}$$

is well defined. For every  $m < \omega, v'_m = v_n$  for some  $n \ge m$  such that  $u \overline{W^{(n)}} v_n : t$ ; hence  $u \overline{W^{(m)}} v'_m : t$ , by (4.18), and  $u \overline{W^{(m)}} \lesssim_{\Box}^{\circ} u' : t$ , by (3.3). That is, we obtain that  $u \overline{T} u' : t$ .

Since u' was chosen arbitrarily from U', we conclude that

$$\forall u' \in U'. \exists u \in U. \ u \ \overline{T} \ u' : t$$

Hence T is an open upper simulation, as it is closed under substitutions.

# 7.5 Open problems

The negative results and the omissions in this chapter prompt several open questions.

First of all, the mismatch between the similarity preorders and the contextual approximation preorders could be investigated further. I conjecture that lower similarity and may contextual approximation coincide on "finite elements", i.e., expressions that are built without the use of recursion and countable choice—they could be defined and studied operationally along the lines of Mason, Smith, and Talcott (1996). In other words, lower similarity and may contextual approximation differ only "at the limit".

The situation is different for upper similarity and must contextual approximation. In the spirit of Plotkin's solution to the full abstraction problem for PCF (Plotkin 1977)—where he added a parallel-or function to the language to obtain a complete match between equality in a domain model and contextual equivalence—we may ask: will adding a parallel convergence tester to the language make contextual approximation as discriminative as upper similarity?

Finally, it is an open problem whether a syntactic  $\omega_1^{CK}$ -continuity principle holds for upper similarity. Syntactic continuity properties of convex similarity could also be investigated. This is left for future work.

# 7.6 Related work

The definitions of lower and upper similarity appeared in the context of action semantics in Lassen (1997). They were inspired by Ong's convex similarity (Ong 1992; Ong 1993) and Ulidowski's copy+refusal testing for processes (Ulidowski 1992). Lower, upper, and convex similarity correspond to the different constructions on preorders that are used to characterise the lower, upper and convex powerdomains (see, e.g., Gunter and Scott 1990).

Howe (1989) describes a method for establishing the compatibility of (essentially) lower similarity for a nondeterministic call-by-name language. He shows that lower similarity does not coincide with may contextual approximation, but his example is specific for call-by-name parameter passing and is unrelated to the mismatch that we discuss in §7.1. Moran (1994) also proves that a lower similarity preorder is a pre-congruence for a lazy  $\lambda$ -calculus equipped with an ambiguous choice combinator (which behaves like erratic choice with respect to lower similarity). Our argument for the mismatch between lower similarity and may contextual approximation carries over to his language.

Howe (1996) and Ong (1992) independently extended Howe's method to prove compatibility of convex bisimilarity and convex similarity, respectively, for  $\lambda$ -calculi with bounded nondeterminism. An extension of Ong's proof to countable nondeterminism was found independently by Lassen and Pitcher (1998). In Lassen (1997) it was used to prove compatibility of an upper similarity preorder for action notation. The present formulation of upper simulation, in terms of an inductively defined must evaluation relation, and the resulting compatibility proof for upper similarity, by induction on must evaluations, appear to be new.

Direct proofs of syntactic continuity for applicative similarity appear in Pitts (1997a) and Lassen (1998), but our syntactic continuity result for upper similarity is the first such proof in a nondeterministic context.

# Chapter 8

# Fairness

Up till now we have been looking at erratic choice and countable choice. Another form of nondeterministic choice that has been studied for higher-order languages is McCarthy's ambiguous choice (McCarthy 1963). It introduces a notion of fairness. This chapter discusses the difficulties of reasoning about ambiguous choice and extends some of the results from the previous chapters to fairness.

In general, fairness is more difficult to model, by domain-theoretic or operational means, than bounded and countable nondeterminism. This is apparent from existing work on the semantics of ambiguous choice. Broy (1986) has given a non-standard fixed-point semantics for a stream-processing language with ambiguous choice, using multiple fixed points over multiple powerdomains with different notions of approximation. These ideas have also been applied to derive fixed-point principles for a version of Dijkstra's guarded command language extended with ambiguous choice (Broy and Nelson 1994). Moran (1994) investigates contextual equivalences and applicative similarity for a call-by-name  $\lambda$ -calculus with ambiguous choice. He highlights the difficulties of reasoning about must contextual equivalence, and shows that convex similarity is not a pre-congruence.

In fact, no context lemma of any form is known for must contextual equivalence in the presence of fairness. Divergence is not a least element with respect to must contextual approximation, recursive functions are not least fixed points, and basic reasoning principles such as recursion induction, the Unwinding Theorem, and syntactic continuity fail. Despite these obstacles, our relational techniques can be used to establish some basic results about must contextual equivalence. The development includes a theory of improvement simulation and a version of Sands' improvement theorem (Sands 1998b). This is interesting as otherwise conventional approaches for reasoning about recursion break down in the presence of fairness. Nonetheless, important open problems about must contextual equivalence and applicative bisimulation for fairness remain unsolved. These are summarised at the end of the chapter.

## 8.1 Syntax and operational semantics

We extend the syntax with a binary combinator **amb** for representing ambiguous choice.

$$egin{array}{cccc} a,b & ::= & \dots & \ & & | & \mathbf{amb}(a_1,a_2) \end{array}$$

The typing rule is the same as for erratic choice.

(Type amb) 
$$\frac{\Gamma \vdash a_1 : t \quad \Gamma \vdash a_2 : t}{\Gamma \vdash \mathbf{amb}(a_1, a_2) : t}$$

The meaning of  $\operatorname{amb}(a_1, a_2)$  is that it may evaluate to anything that  $a_1$  or  $a_2$  may evaluate to. Furthermore, it must terminate if and only if one or both of  $a_1$  and  $a_2$  must terminate.

Fair nondeterminism is at least as general as countable nondeterminism: countable choice is faithfully encoded with **amb** in place of **or** in (5.2),

$$(\mathbf{Y}_{\text{unit} \rightarrow \text{nat}} \lambda f. \lambda \langle \rangle. \operatorname{amb}(\mathbf{0}, \operatorname{let} y = f \langle \rangle \operatorname{in succ} y)) \langle \rangle$$

$$(8.1)$$

It is rather tricky to extend the may and must transition relations and the must evaluation relation to ambiguous choice. Following Moran (1994) one can add resource attributes to the syntax, define the transition relations for the extended syntax, and then define the must evaluation relation on basis of the must transition relation. However, we will just consider the may evaluation relation,  $\rightsquigarrow_{\diamond}$ , and a must termination predicate,  $\Downarrow_{\Box}$ , and these both have simple inductive definitions; cf. Moran (1994). We define the may evaluation relation, Table 8.1, and the must termination relation, Table 8.2, on basis of the may and must primitive reduction relations from Chapter 5.

The operational semantics still enjoys the property that every a that must terminate may terminate:

$$a \Downarrow_{\Box} \quad \text{implies} \quad \exists v. \ a \rightsquigarrow_{\Diamond} v \tag{8.2}$$

For expressions a without occurrences of ambiguous choice, the must termination predicate  $a \Downarrow_{\Box}$  holds if and only if a must terminate in the sense of Chapter 5, that is,  $a \rightsquigarrow_{\Box} U$  for some set U. For such expressions we also see that the may evaluation relation is the same as in Chapter 5.

By inspection of the may and must primitive reduction relations one can see that ambiguous choice and erratic choice have identical may evaluation behaviour, but that they differ with respect to must termination. Erratic choice must terminate only if both of its branches must terminate.

One can check that the encoding (8.1) and countable choice, ?, behave identically with respect to may evaluation as well as must termination.

We extend the definition of compatible refinement with a rule for ambiguous choice, similar to the rule for erratic choice (Comp or) in Chapter 5.

(Comp amb) 
$$\frac{\Gamma \vdash a_1 \ R \ a'_1 : t \quad \Gamma \vdash a_2 \ R \ a'_2 : t}{\Gamma \vdash \mathbf{amb}(a_1, a_2) \ \widehat{R} \ \mathbf{amb}(a'_1, a'_2) : t}$$

We notice that Lemmas 5.4.1 and 5.4.2 remain valid in the presence of ambiguous choice.

# 8.2 Generalising the sequential theory

Since ambiguous choice and erratic choice behave identically with respect to may evaluation, the theory of may contextual approximation and similarity for sequential nondeterminism
$(\text{Term}_{\Box} \text{ value})$ 



 $v \Downarrow_\square$ 

| $(\mathrm{Term}_\square \ \mathrm{redex})$     | $\frac{\forall b \in B. \ b \Downarrow_{\square}}{a \Downarrow_{\square}} \ \text{ if } a \rightarrow_{\square} B$  |
|--|---|
| $(\mathrm{Term}_{\Box}   \mathrm{let})$        | $\frac{a \Downarrow_{\Box}  \forall u. \ a \rightsquigarrow_{\Diamond} u \ \Rightarrow \ b[u/x] \Downarrow_{\Box}}{\operatorname{let} \ x = a \ \operatorname{in} \ b \Downarrow_{\Box}}$ |
| $(\mathrm{Term}_{\Box} \mathrm{ amb \ left})$  | $\frac{a_1\Downarrow_{\square}}{\mathbf{amb}(a_1,a_2)\Downarrow_{\square}}$   |
| $(\mathrm{Term}_{\Box} \mathrm{ amb \ right})$ | $\frac{a_2\Downarrow_{\square}}{\textbf{amb}(a_1,a_2)\Downarrow_{\square}}$   |

Table 8.2: Must termination predicate

from the preceding chapters is unaffected—all that is needed is to include cases for (Eval $\diamond$  amb left) and (Eval $\diamond$  amb right) in the proofs of Lemmas 6.3.1 and 7.1.1, and they are essentially identical to the cases for erratic choice. In particular, the open extension of lower Kleene equivalence is still included in may contextual equivalence. It is easy to see that  $\operatorname{amb}(a_1, a_2)$  is lower Kleene equivalent to  $(a_1 \text{ or } a_2)$ ,

$$\Gamma \vdash \mathbf{amb}(a_1, a_2) \asymp_{\Diamond}^{\circ} (a_1 \text{ or } a_2) : t, \text{ if } \Gamma \vdash a_i : t \text{ for } i = 1, 2$$

so ambiguous choice is equivalent to erratic choice up to may contextual equivalence.

It is more challenging to study must termination behaviour for fair nondeterminism. First of all, since we do not have a must evaluation relation for fairness, we must reformulate the definitions of must-(pre-)adequacy at the beginning of Chapter 2.4 in the obvious way, by substituting  $a \downarrow_{\Box}$  and  $a' \downarrow_{\Box}$  for  $a \rightsquigarrow_{\Box} \{\langle \rangle\}$  and  $a' \rightsquigarrow_{\Box} \{\langle \rangle\}$ .

As discussed in §6.1, the contextual approximations and equivalences are examples of testing preorders and equivalences. Agha, Mason, Smith, and Talcott (1997) have shown that must testing equivalence is included in may testing equivalence in the presence of fairness. More precisely, the must testing preorder is included in the reciprocal may testing preorder. In our case this means that must contextual approximation is included in the reciprocal may contextual approximation in the presence of ambiguous choice. The proof of this result is based on the fact that there is a context D,

$$D \stackrel{\text{def}}{=} \mathbf{let} \ f = \mathbf{amb}(\lambda \langle \rangle, \langle \rangle, \mathbf{let} \ y = \bullet \ \mathbf{in} \ \lambda \langle \rangle, \mathbf{\Omega}) \ \mathbf{in} \ f \langle \rangle$$

of type • : unit  $\vdash D$  : unit, with the property that  $a \rightsquigarrow_{\Diamond} \langle \rangle$  if and only if  $\neg(D[\![a]\!] \Downarrow_{\Box})$ , for all a : unit.

**Proposition 8.2.1**  $\sqsubset_{\Box} \subseteq \sqsubset_{\Diamond}^{\operatorname{op}}$ .

. .

**Proof** Since must contextual approximation is compatible it suffices to show that its reciprocal is may-pre-adequate, then it is included in reciprocal may contextual approximation. So suppose that  $a \\[-2mm] \square_{\square} a'$ : unit and that  $a' \\[-2mm] \rightsquigarrow_{\Diamond} \langle \rangle$ . We must show that  $a \\[-2mm] \rightsquigarrow_{\Diamond} \langle \rangle$ . Firstly,  $a' \\[-2mm] \rightsquigarrow_{\Diamond} \langle \rangle$  holds if and only if  $\neg (D[\![a']\!] \Downarrow_{\square})$ . Since  $\\[-2mm] \square_{\square}$  is compatible,  $D[\![a]\!] \\[-2mm] \square_{\square} D[\![a']\!]$ : unit. Hence  $\neg (D[\![a]\!] \Downarrow_{\square})$  which means that  $a \\[-2mm] \rightsquigarrow_{\Diamond} \langle \rangle$ , as required.

The inclusion of must contextual approximation in may contextual approximation fails in the absence of fairness, e.g., then  $\Omega \sqsubset_{\Box} (\Omega \text{ or } \langle \rangle)$ : unit but not  $(\Omega \text{ or } \langle \rangle) \sqsubset_{\Diamond} \Omega$ : unit, (both are false in the presence of fairness). Hence we observe that the introduction of fairness makes must contextual approximation and equivalence more discriminating.

For bounded and countable nondeterminism,  $\Omega$  is a minimum element in the must contextual approximation preorder, but in the presence of ambiguous choice it becomes a maximal element, as we are going to show in Example 8.3.2 below. Consequently, some order-theoretic properties of must contextual approximation from Chapter 6 are invalid for fairness, including recursion induction. For instance,  $\lambda \langle \rangle$ . *a* is a fixed point for the functional  $\lambda f$ .  $\lambda \langle \rangle$ .  $f \langle \rangle$ ,

$$(\lambda f. \lambda \langle \rangle. f \langle \rangle) \lambda \langle \rangle. a \cong_{\Box} \lambda \langle \rangle. a : \text{unit} \rightharpoonup t$$

if a : t, but it is not necessarily the case that  $\mathbf{Y} \lambda f. \lambda \langle \rangle. f \langle \rangle \sqsubset_{\Box} \lambda \langle \rangle. a : \text{unit} \rightarrow t$ , e.g., if  $a = \langle \rangle$  then  $D[[(\mathbf{Y} \lambda f. \lambda \langle \rangle. f \langle \rangle) \langle \rangle]] \Downarrow_{\Box}$  and  $\neg (D[[(\lambda \langle \rangle. a) \langle \rangle]] \Downarrow_{\Box})$ , because  $(\mathbf{Y} \lambda f. \lambda \langle \rangle. f \langle \rangle) \langle \rangle$  must diverge and  $(\lambda \langle \rangle. a) \langle \rangle \rightsquigarrow_{\Diamond} \langle \rangle$ .

Upper similarity from Chapter 7 can be defined for fairness by replacing the assertion  $a \rightsquigarrow_{\Box} U$  in the definition of upper simulation in §6.2 by  $(a \Downarrow_{\Box} \& U = \{u \mid a \rightsquigarrow_{\Diamond} u\})$ ; and similarly for  $a' \rightsquigarrow_{\Box} U'$ . But upper similarity is not a pre-congruence nor is it included in must contextual approximation in the presence of ambiguous choice. For instance,  $\Omega \leq_{\Box} \langle \rangle$  : unit but this relationship is not preserved by the context D from above,  $D[\![\Omega]\!] \not\leq_{\Box} D[\![\langle \rangle]\!]$  : unit, nor is it contained in must contextual approximation,  $\Omega \not\equiv_{\Box} \langle \rangle$  : unit, since  $D[\![\Omega]\!] \Downarrow_{\Box}$  and  $\neg(D[\![\langle \rangle]\!] \Downarrow_{\Box})$ .

### 8.3 Refinement

The relationship between must and may contextual approximation in Proposition 8.2.1 suggests that we should consider refinement similarity as a tool for reasoning about must contextual approximation. Let  $\langle \cdot \rangle_{\mathsf{R}}$  be the refinement simulation operator that maps every open relation R into the closed relation  $\langle R \rangle_{\mathsf{R}}$  given by

It is the appropriate reformulation of the operator  $\langle \cdot {}^{op} \circ \rangle_{\Diamond}^{op} \cap \langle \cdot \circ \rangle_{\Box}$  in terms of may evaluation and must termination.

We call post-fixed points of  $\langle \cdot \circ \rangle_{\mathsf{R}}$  refinement simulations. Refinement similarity is the greatest refinement simulation. Notice that  $\mathbf{R}$  is a refinement simulation if and only if  $\mathbf{R}^{\mathrm{op}}$  is a lower simulation and  $\mathbf{R}$  is must-pre-adequate. We see that refinement similarity is included in reciprocal lower similarity (in accordance with Proposition 8.2.1). It is an open problem whether refinement similarity is compatible, and whether it is included in must contextual approximation, but every compatible refinement simulation is included in must contextual approximation.

Let refinement Kleene approximation,  $\leq_{\mathsf{R}}$ , be the relation  $\langle Id \rangle_{\mathsf{R}}$ , that is,

for all a, a' : t. Convex Kleene equivalence,  $\asymp_{\mathsf{R}}$ , is the symmetrisation of  $\preceq_{\mathsf{R}}$ . (The names "refinement" and "convex" are chosen in accordance with the taxonomy from §7.3.)

Examples of convex Kleene equivalences are the correctness of the encoding of countable choice,

$$? \asymp_{\mathsf{R}} (\mathbf{Y}_{\text{unit} \rightharpoonup \text{nat}} \lambda f. \lambda \langle \rangle. \operatorname{\mathbf{amb}}(\mathbf{0}, \operatorname{\mathbf{let}} y = f \langle \rangle \operatorname{\mathbf{in}} \operatorname{\mathbf{succ}} y)) \langle \rangle : t$$

$$(8.3)$$

and  $\beta_v$ -equivalence,

$$\Gamma \vdash (\lambda x. a) v \asymp_{\mathsf{R}}^{\circ} a[v/x] : t, \text{ if } \Gamma, x : t' \vdash a : t \text{ and } \Gamma \vdash v : t'$$

$$(8.4)$$

**Proposition 8.3.1**  $\leq_{\mathsf{R}}^{\circ} \subseteq \sqsubseteq_{\Box}$ .

**Proof** We are going to show that  $\preceq_{\mathsf{R}}^{\circ \mathsf{SC}}$  is an open refinement simulation, i.e., (*i*) that  $(\preceq_{\mathsf{R}}^{\circ \mathsf{SC}})^{\mathrm{op}}$  is an open lower simulation and (*ii*) that  $\preceq_{\mathsf{R}}^{\circ \mathsf{SC}}$  is must-pre-adequate.

Part (i) follows by the same argument as in the proof of Proposition 4.3.1. We show

 $a' \preceq^{\circ \mathsf{SC}}_{\mathsf{R}} a: t \And a \rightsquigarrow_{\Diamond} v \ \Rightarrow \ \exists v': t. \ a' \rightsquigarrow_{\Diamond} v' \And v' \ \overleftarrow{\preceq^{\circ \mathsf{SC}}_{\mathsf{R}}} v: t$ 

by induction on the derivation of  $a \rightsquigarrow_{\Diamond} v$ . By Lemma 3.8.1, there exists a'' : t such that  $a'' \stackrel{\frown}{\leq_{\mathsf{R}}^{\circ}} \widehat{\mathsf{SC}} a : t$  and  $a' \preceq_{\mathsf{R}} a'' : t$ . It suffices to show that there exists v' : t such that  $a'' \rightsquigarrow v'$  and  $v' \stackrel{\frown}{\leq_{\mathsf{R}}^{\circ}} \widehat{\mathsf{SC}} v : t$ ; then  $a' \rightsquigarrow v'$  too, by the definition of  $\preceq_{\mathsf{R}}$ . We proceed by analysis of the derivation of  $a \rightsquigarrow v$ .

Let us just consider the case (Eval<sub> $\diamond$ </sub> amb left) where  $a = \mathbf{amb}(a_1, a_2)$  and  $a_1 \rightsquigarrow_{\diamond} v$ . Since  $a'' \stackrel{\frown}{\leq_{\mathsf{R}}^{\circ}\mathsf{SC}} a : t$ , there are  $a'_1, a'_2 : t$  such that  $a'' = \mathbf{amb}(a'_1, a'_2)$  and  $a'_1 \stackrel{\frown}{\leq_{\mathsf{R}}^{\circ}\mathsf{SC}} a_1 : t$ . By the induction hypothesis,  $a'_1 \rightsquigarrow_{\diamond} v'$  with  $v' \stackrel{\frown}{\leq_{\mathsf{R}}^{\circ}\mathsf{SC}} v : t$ , and by (Eval<sub> $\diamond$ </sub> amb left),  $a'' \rightsquigarrow_{\diamond} v'$ , as required.

The (Eval<sub> $\diamond$ </sub> amb right) case is symmetrical. The remaining cases are exactly as in the proof of Proposition 4.3.1, but with  $\rightsquigarrow_{\diamond}$  in place of  $\rightsquigarrow$  and Lemma 5.4.1 in place of Lemma 3.10.1.

Part (*ii*), that  $\preceq_{\mathsf{R}}^{\circ \mathsf{SC}}$  is must-pre-adequate,

$$a \preceq^{\circ SC}_{\mathsf{R}} a' : t \& a \Downarrow_{\square} \Rightarrow a' \Downarrow_{\square}$$

$$(8.5)$$

holds by induction on the derivation of  $a \Downarrow_{\Box}$ . As in the proof of part (*i*) above, we first employ Lemma 3.8.1, this time to get an a'' : t such that  $a \preceq_{\mathsf{R}} \widehat{\mathsf{SC}} a'' : t$  and  $a'' \preceq_{\mathsf{R}} a' : t$ . It suffices to show that  $a'' \Downarrow_{\Box}$ ; then  $a' \Downarrow_{\Box}$  too, by the definition of  $\preceq_{\mathsf{R}}$ . We proceed by analysis of the derivation of  $a \Downarrow_{\Box}$ .

Case (Term<sub> $\Box$ </sub> value)

a is a value

From  $a \stackrel{\leq}{\leq_{\mathsf{R}}} \circ \overset{\circ}{\mathsf{SC}} a'' : t$  and Lemma 3.5.1 follows that a'' is also a value. Hence  $a'' \Downarrow_{\Box}$  by (Term<sub> $\Box$ </sub> value) too.

Case (Term<sub> $\Box$ </sub> redex)

$$\begin{array}{c} a \to_{\Box} B\\ b \Downarrow_{\Box}, \text{ for all } b \in B \end{array}$$

The set *B* is countable so  $B = \{b_i\}_{i < \omega}$  for some  $b_0, b_1, \dots : t$ . From  $a \preceq_{\mathsf{R}}^{\circ \mathsf{SC}} a'' : t$ and Lemma 5.4.2 we get another countable set  $B' = \{b'_i\}_{i < \omega}$  such that  $a'' \to_{\Box} B'$  and  $b_i \preceq_{\mathsf{R}}^{\circ \mathsf{SC}} b'_i : t$  for all  $i < \omega$ . For each  $i < \omega$  we get that  $b'_i \Downarrow_{\Box}$  by the induction hypothesis. We conclude  $a'' \Downarrow_{\Box}$ , by (Term<sub> $\Box$ </sub> redex).

Case (Term<sub> $\Box$ </sub> let)

$$egin{aligned} a = \mathbf{let} \,\, x = a_0 \,\, \mathbf{in} \,\, b_0 \ a_0 \Downarrow_\square \ U = \{ u \mid a_0 \rightsquigarrow_\diamondsuit u \} \ b_0 [u\!/\!x] \Downarrow_\square, \,\, \mathrm{for} \,\, \mathrm{all} \,\, u \in U \end{aligned}$$

Then  $a \xrightarrow{\leq_{\mathsf{R}}^{\circ}\mathsf{SC}} a'' : t$  implies that a'' is of the form  $a'' = \operatorname{let} x = a'_0 \operatorname{in} b'_0$  with  $a_0 \preceq_{\mathsf{R}}^{\circ}\mathsf{SC} a'_0 : t'$ and  $x : t' \vdash b_0 \preceq_{\mathsf{R}}^{\circ}\mathsf{SC} b'_0 : t$ . By the induction hypothesis, we get that  $a'_0 \Downarrow_{\Box}$ . Let  $U' = \{u' \mid a'_0 \xrightarrow{\sim} u'\}$  and fix any  $u' \in U'$ . Since  $(\preceq_{\mathsf{R}}^{\circ}\mathsf{SC})^{\operatorname{op}}$  is a lower simulation, there is  $u \in U$  with  $u \preceq_{\mathsf{R}}^{\circ}\mathsf{SC} u' : t'$ . We obtain  $b_0[u/x] \preceq_{\mathsf{R}}^{\circ}\mathsf{SC} b'_0[u'/x] : t$  because  $\preceq_{\mathsf{R}}^{\circ}\mathsf{SC}$  is compatible and substitutive. Again by the induction hypothesis,  $b'_0[u'/x] \Downarrow_{\Box}$ . We conclude  $a'' \Downarrow_{\Box}$ , by (Term<sub> $\Box$ </sub> let).

Case (Term<sub> $\Box$ </sub> amb left)

$$a = \mathbf{amb}(a_1, a_2)$$
$$a_1 \Downarrow_{\square}$$

As  $a \preceq_{\mathsf{R}}^{\circ \mathsf{SC}} a'' : t, a'' = \operatorname{\mathbf{amb}}(a'_1, a'_2)$  with  $a_1 \preceq_{\mathsf{R}}^{\circ \mathsf{SC}} a'_1 : t$  and  $a_2 \preceq_{\mathsf{R}}^{\circ \mathsf{SC}} a'_2 : t$ . By the induction hypothesis, we get that  $a'_1 \Downarrow_{\square}$ . Hence  $a'' \Downarrow_{\square}$ , by (Term<sub> $\square$ </sub> amb left).

**Case (Term** $_{\Box}$  **amb right)** Symmetrical to the previous case.

Consequently, (8.3) asserts that the encoding (8.1) of countable choice is correct up to must contextual equivalence.

Similarly, (8.4) entails that  $\beta_v$ -equivalence is valid for must contextual equivalence. Therefore must contextual approximation and equivalence are substitutive, by the same argument as in §4.3.

**Example 8.3.2**  $\Omega$  is maximal with respect to both refinement Kleene approximation,

$$\Gamma \vdash \mathbf{\Omega} \preceq^{\circ}_{\mathsf{R}} a: t \quad \text{implies} \quad \Gamma \vdash \mathbf{\Omega} \asymp^{\circ}_{\mathsf{R}} a: t$$

$$(8.6)$$

and must contextual approximation,

$$\Gamma \vdash \mathbf{\Omega} \sqsubseteq_{\Box} a : t \quad \text{implies} \quad \Gamma \vdash \mathbf{\Omega} \cong_{\Box} a : t \tag{8.7}$$

Let us first prove (8.6). By definition of refinement Kleene approximation and open extension,  $\preceq^{\circ}_{\mathsf{R}}$  is closed under substitutions and its reciprocal is may-pre-adequate. Therefore, since  $\Omega$  neither may evaluate to anything nor must terminate,  $\Gamma \vdash \Omega \preceq^{\circ}_{\mathsf{R}} a : t$  if and only if  $a[\vec{u}/\vec{x}]$  may not evaluate to anything for any substitution of closed values  $\vec{u}: \vec{t}$  for  $\vec{x}$ , supposing  $\Gamma = \vec{x}: \vec{t}$ . But if  $a[\vec{u}/\vec{x}]$  may not evaluate to anything, it must not terminate either, by (8.2). From the definition of refinement Kleene approximation and open extension, it follows that  $\Gamma \vdash a \preceq^{\circ}_{\mathsf{R}} \Omega : t$ . In conjunction with the antecedent of (8.6), this implies the conclusion of (8.6), as required.

We deduce (8.7) by a similar argument: since  $\sqsubset_{\Box}$  is also closed under substitutions, as it is reflexive and substitutive, and its reciprocal is may-pre-adequate, by Proposition 8.2.1,  $\Gamma \vdash \Omega \sqsubset_{\Box} a : t$  implies  $\Gamma \vdash a \preceq_{\mathsf{R}}^{\circ} \Omega : t$  by the same argument as above. Proposition 8.3.1 entails that  $\Gamma \vdash a \sqsubset_{\Box} \Omega : t$ , which in conjunction with the antecedent of (8.7) implies  $\Gamma \vdash$  $\Omega \cong_{\Box} a : t$ , as required.  $\Box$ 

**Example 8.3.3** Moran (1994) lists some algebraic properties of ambiguous choice: idempotency, commutativity, associativity, and  $\Omega$  is unit. They are all easily seen to hold for open

convex Kleene equivalence:

$$\Gamma \vdash \mathbf{amb}(a, a) \asymp_{\mathsf{R}}^{\circ} a : t$$
  

$$\Gamma \vdash \mathbf{amb}(a_1, a_2) \asymp_{\mathsf{R}}^{\circ} \mathbf{amb}(a_2, a_1) : t$$
  

$$\Gamma \vdash \mathbf{amb}(a_1, \mathbf{amb}(a_2, a_3)) \asymp_{\mathsf{R}}^{\circ} \mathbf{amb}(\mathbf{amb}(a_1, a_2), a_3) : t$$
  

$$\Gamma \vdash \mathbf{amb}(\mathbf{\Omega}, a) \asymp_{\mathsf{R}}^{\circ} \mathbf{amb}(a, \mathbf{\Omega}) \asymp_{\mathsf{R}}^{\circ} a : t$$

whenever  $\Gamma \vdash a : t$  and  $\Gamma \vdash a_i : t$  for  $i \in 1..3$ . Proposition 8.3.1 implies that the equations hold for must contextual equivalence as well.  $\Box$ 

**Example 8.3.4** The first three laws for ambiguous choice also hold for erratic choice,

$$\Gamma \vdash a \text{ or } a \asymp_{\mathsf{R}}^{\circ} a : t$$
$$\Gamma \vdash a_1 \text{ or } a_2 \asymp_{\mathsf{R}}^{\circ} a_2 \text{ or } a_1 : t$$
$$\Gamma \vdash a_1 \text{ or } (a_2 \text{ or } a_3) \asymp_{\mathsf{R}}^{\circ} (a_1 \text{ or } a_2) \text{ or } a_3 : t$$

Furthermore, erratic choice is the greatest lower bound operator,

$$\frac{\Gamma \vdash (a_1 \text{ or } a_2) \preceq^{\circ}_{\mathsf{R}} a_1 : t}{\Gamma \vdash (a_1 \text{ or } a_2) \preceq^{\circ}_{\mathsf{R}} a_2 : t} \\
\frac{\Gamma \vdash a \preceq^{\circ}_{\mathsf{R}} a_1 : t \quad \Gamma \vdash a \preceq^{\circ}_{\mathsf{R}} a_2 : t}{\Gamma \vdash a \preceq^{\circ}_{\mathsf{R}} (a_1 \text{ or } a_2) : t}$$

By Proposition 8.3.1, these laws also hold for must contextual approximation and equivalence. Erratic choice is also the greatest lower bound operator for must contextual approximation because the last rule, which says that any lower bound of  $a_1$  and  $a_2$  is below ( $a_1$  or  $a_2$ ), extends easily to must contextual approximation by the calculation

$$\Gamma \vdash a \sqsubset_{\Box} (a \text{ or } a) \sqsubset_{\Box} (a_1 \text{ or } a_2) : t$$

where we use that erratic choice is idempotent, that must contextual approximation is compatible, and the assumption that  $\Gamma \vdash a \sqsubset_{\Box} a_1 : t$  and  $\Gamma \vdash a \sqsubset_{\Box} a_2 : t$ .  $\Box$ 

Example 8.3.5 Erratic choice must contextually approximates ambiguous choice, i.e.,

$$(a_1 \text{ or } a_2) \sqsubset_{\Box} \mathbf{amb}(a_1, a_2) : t \tag{8.8}$$

if  $a_1, a_2 : t$ . This is easily established by refinement Kleene approximation and Proposition 8.3.1. The reverse is false in general; e.g.,  $\operatorname{amb}(\Omega, \langle \rangle)$  must terminate whereas  $(\Omega \text{ or } \langle \rangle)$  may diverge. In fact,  $(a_1 \text{ or } a_2)$  is must contextually equivalent to  $\operatorname{amb}(a_1, a_2)$  if and only if  $a_1$  and  $a_2$  have the same must termination behaviour,

$$(a_1 \text{ or } a_2) \cong_{\square} \mathbf{amb}(a_1, a_2) : t \quad \text{iff} \quad (a_1 \Downarrow_{\square} \Leftrightarrow a_2 \Downarrow_{\square})$$

$$(8.9)$$

The forward implication is immediate because must contextual equivalence is must-adequate. The converse follows by convex Kleene equivalence and Proposition 8.3.1.  $\Box$ 

Kleene equivalence is useful for practical equational reasoning based on evaluation of expressions, but it does not provide the power of co-inductive methods for reasoning about infinite behaviour of higher order functions. For want of more general results about refinement simulation, we shall now develop "improvement" simulation proof tools for establishing must contextual equivalences between expressions which must terminate at equal "cost".

### 8.4 Improvement

Following Sands (1998b) we introduce a fine-grained "intensional" operational ordering that takes computational cost into account, in our case the number of function applications in the derivation of must termination. Improvement theory has independent interest as a formal approach to the study of program efficiency but Sands has also demonstrated that it is a powerful tool for reasoning about conventional semantic equivalences. Here we are interested in the latter use of improvement. The motivation for our definition of improvement below is that it gives us a co-inductively defined simulation preorder which is a pre-congruence in the presence of fairness.

Let us first define a cost measure for must termination. We write  $a \Downarrow_{\Box}^{\alpha}$  to mean that a must terminate and that the recursive ordinal  $\alpha$  is the maximal number of function application steps—i.e., (Term<sub> $\Box$ </sub> redex) nodes for function application—on any path in the derivation tree for  $a \Downarrow_{\Box}$ . The height of every derivation tree for must termination is bounded by a recursive ordinal—this follows from the same property of derivation trees for must evaluation for countable nondeterminism, see §6.7, and the fact that ambiguous choice can be interpreted by means of countable choice, see Moran (1994).

To formalise the definition of  $a \Downarrow_{\Box}^{\alpha}$  we first split the must primitive reduction relation,  $\rightarrow_{\Box}$ , into  $\beta_v$ -reduction for function application,  $\stackrel{\checkmark}{\rightarrow}$ , and all other primitive reductions,  $\stackrel{\tau}{\rightarrow}_{\Box}$ ; see Tables 8.3 and 8.4. They satisfy

$$a \to_{\Box} B$$
 iff either  $a \xrightarrow{\tau}_{\Box} B$  or  $\exists b. a \xrightarrow{\vee} b \& \{b\} = B$ 

Then we assign recursive ordinals to derivations of must termination by introducing a family of must termination predicates,  $\Downarrow^{\alpha}_{\Box}$ , for  $\alpha < \omega_1^{CK}$ . They are defined inductively by the rules in Table 8.5. The new family of must termination predicates satisfies

$$a \Downarrow_{\Box} \quad \text{iff} \quad \exists \alpha < \omega_1^{CK}. \ a \Downarrow_{\Box}^{\alpha}$$

$$\tag{8.10}$$

The cost measure is closely linked with our particular inductive definition of must termination. A more sensible cost measure would be the maximal number of function applications in any evaluation or in any terminating transition sequence. We refrain from this approach because the details of the definition are more complicated. We shall see that our cost measure serves the purpose of proving some must contextual equivalences, even if it is rather artificial.

Now we can define the appropriate improvement simulation operator,  $\langle \cdot \rangle_{\mathbf{I}}$ , obtained from the refinement simulation operator by restricting the cost of must termination. Any open relation R is mapped into the closed relation  $\langle R \rangle_{\mathbf{I}}$  given by

Condition (i) says that R is a lower simulation. We shall refer to condition (ii) as improvementadequacy. It implies must-pre-adequacy because of (8.10).

The compound operator  $\langle \cdot \circ \rangle_{\mathsf{I}}$  is monotone. We call any post-fixed point an *improvement simulation*. Improvement similarity,  $\succeq$ , is the greatest fixed point

$$\succeq \stackrel{\text{def}}{=} \nu \boldsymbol{R}. \left\langle \boldsymbol{R}^{\circ} \right\rangle_{\mathsf{I}} \tag{8.11}$$

(Redex 
$$\sqrt{apply}$$
)  $(\lambda x. a) v \xrightarrow{\sqrt{apply}} a[v/x]$ 

Table 8.3:  $\sqrt{\text{primitive reduction relation}}$ 

$$(\operatorname{Redex}_{\Box} \tau \operatorname{case} \times) \qquad \qquad \mathbf{case} \ \langle \vec{u} \rangle \ \mathbf{of} \ \langle \vec{x} \rangle. \ a \xrightarrow{\tau}_{\Box} \ \{a[\vec{u}/\vec{x}]\}$$

 $(\operatorname{Redex}_{\Box} \tau \text{ case } +) \quad \operatorname{\textbf{case inj}}_{i} u \text{ of inj}_{1} x_{1}. a_{1} \ [] \ \dots \ [] \text{ inj}_{n} x_{n}. a_{n} \xrightarrow{\tau}_{\Box} \{a_{i}[u \! / \! x_{i}]\}, \quad \operatorname{if} \ i \in 1..n$ 

$$(\operatorname{Redex}_{\Box} \tau \text{ or}) \qquad \qquad a_1 \text{ or } a_2 \xrightarrow{\tau}_{\Box} \{a_1, a_2\}$$

 $(\text{Redex}_{\Box} \ \tau \text{ choose}) \qquad \qquad ? \xrightarrow{\tau}_{\Box} \{ [i] \mid i < \omega \}$ 



(Term<sup>0</sup><sub>□</sub> value)  $v \Downarrow_{□}^{0}$ (Term<sup>α+1</sup><sub>□</sub> redex  $\sqrt{}$ )  $\frac{b \Downarrow_{□}^{\alpha}}{\frac{1}{2}}$  if  $a \stackrel{\checkmark}{\to} b$ 

$$a \Downarrow_{\square}^{\alpha+1}$$

$$(\operatorname{Term}_{\Box}^{\alpha} \operatorname{redex} \tau) \qquad \qquad \frac{\forall b \in B. \ b \Downarrow_{\Box}^{\alpha_b}}{a \Downarrow_{\Box}^{\alpha}} \ \text{ if } a \xrightarrow{\tau}_{\Box} B \text{ and } \alpha = \bigcup_{b \in B} \alpha_b$$

$$(\operatorname{Term}_{\Box}^{\alpha} \operatorname{let}) \qquad \qquad \frac{a \Downarrow_{\Box}^{\alpha'} \quad \forall u \in U. \ b[u/x] \Downarrow_{\Box}^{\alpha_u}}{\operatorname{let} x = a \text{ in } b \Downarrow_{\Box}^{\alpha}} \quad \text{if } U = \{u \mid a \rightsquigarrow_{\Diamond} u\} \text{ and } \alpha = \alpha' \cup \bigcup_{u \in U} \alpha_u$$

(Term<sup>$$\alpha$$</sup> <sub>$\square$</sub>  amb left)  $\frac{a_1 \Downarrow_{\square}^{\alpha}}{\mathbf{amb}(a_1, a_2) \Downarrow_{\square}^{\alpha}}$ 

(Term<sup>$$\alpha$$</sup> <sub>$\square$</sub>  amb right)  $\frac{a_2 \Downarrow_{\square}^{\alpha}}{\mathbf{amb}(a_1, a_2) \Downarrow_{\square}^{\alpha}}$ 

#### Table 8.5: Indexed must termination predicate

#### 8.4. IMPROVEMENT

Let *cost equivalence*,  $\mathfrak{D}$ , be the symmetrisation of improvement similarity.

Improvement similarity is a preorder—it is reflexive because *Id* is an improvement simulation and it is transitive because compositions of improvement simulations are improvement simulations—and cost equivalence is an equivalence relation.

Every improvement simulation is a refinement simulation, so improvement similarity is included in refinement similarity. As for refinement we have that the reciprocal of every improvement simulation is a lower simulation and that the reciprocal of improvement similarity is included in lower similarity.

An example of improvement similarity is  $\beta_v$ -reduction,

$$(\lambda x. a) v \gtrsim a[v/x] : t, \quad \text{if } x : t' \vdash a : t \text{ and } v : t'$$

$$(8.12)$$

Since  $\succeq$  is reflexive and is a fixed point of  $\langle \cdot \circ \rangle_{\mathsf{I}}$ , (8.12) follows from the observations that  $a[v/x] \preceq_{\diamond} (\lambda x. a) v : t$  and that  $a[v/x] \Downarrow_{\Box}^{\kappa}$  if and only if  $(\lambda x. a) v \Downarrow_{\Box}^{\kappa+1}$ . Because of the last observation, the reverse of (8.12) is false when a[v/x] must terminate.

We are now going to prove that improvement similarity is a pre-congruence, by means of Howe's compatible extension. Let us first outline the new ideas before we embark on the detailed proof. The main lemma asserts that if R is an improvement simulation then an appropriate construction of a compatible relation,  $((\mathbf{R}^{op})^{*\bullet})^{op}$ , is an improvement simulation, that is, (i) its reciprocal is a lower simulation and (ii) it is itself improvement-adequate. Part (i) is direct from Lemma 7.1.1. Part (ii) is tricky because the construction by compatible extension is turned upside-down, as it were, in order to meet the requirements of part (i). (Recall that the definition of compatible extension is asymmetrical.) Therefore it is not possible just to do induction on the must termination of the expressions on the left hand side of the relation. Instead we take advantage of the fact that  $\mathbf{R}$  is improvement-adequate (i.e., an improvement simulation) and not just must-pre-adequate (i.e., a refinement simulation): when we go across the R component of compatible extension, the cost of must termination does not increase. The trick is then to do the induction both (1) on the cost of must termination and (2) on the derivation by compatible refinement, ordered lexicographically. By delaying all substitutions between function applications, part (2) of the induction hypothesis takes care of all cases but function application. Part (1) of the induction hypothesis applies to the function application case because here must termination is always derived from premises of strictly smaller cost.

To simplify the statement of the next lemma (and to emphasise the structural similarity with the proofs in  $\S4.8$ ,  $\S7.1$ , and \$7.2), we define the notation:

$$\mathbf{R}^{\S} \stackrel{\text{def}}{=} ((\mathbf{R}^{\text{op}})^{\bullet})^{\text{op}}$$

 $\mathbf{R}^{\S}$  inherits nice properties of compatible extension, in particular,  $\mathbf{R}^{\S}$  is substitutive and, if  $\mathbf{R}$  is reflexive,  $\mathbf{R}^{\S}$  is compatible and includes  $\mathbf{R}^{\circ}$ .

**Lemma 8.4.1** If  $\mathbf{R} \subseteq \langle \mathbf{R}^{\circ} \rangle_{\mathsf{I}}$  then  $\mathbf{R}^{*\S} \subseteq \langle \mathbf{R}^{*\S} \rangle_{\mathsf{I}}^{\circ}$ .

**Proof** Assume that  $\boldsymbol{R}$  is an improvement simulation,  $\boldsymbol{R} \subseteq \langle \boldsymbol{R}^{\circ} \rangle_{I}$ .

(*i*) Then  $\mathbf{R}^{\text{op}}$  is a lower simulation and, by Lemma 7.1.1,  $(\mathbf{R}^{\text{op}})^{*\bullet}$  is an open lower simulation. The latter equals  $(\mathbf{R}^{*\$})^{\text{op}}$ .

(*ii*) By (3.6) and the definition of improvement simulation, it remains to show that  $\mathbf{R}^{*\$}[Id]$  is improvement-adequate or, equivalently, that  $\mathbf{R}^{*\$}[\mathbf{R}^{*\$}]$  is improvement-adequate, because

 $\mathbf{R}^{*\S}$  is reflexive and substitutive. That is, since  $\mathbf{R}^{*\S} = ((\mathbf{R}^{*\mathrm{op}})^{\bullet})^{\mathrm{op}}$ , we must show that

$$\vec{x}: \vec{t} \vdash a' \left(\boldsymbol{R}^{* \operatorname{op}}\right)^{\bullet} a: t \& \vec{u}' \ \overline{\left(\boldsymbol{R}^{* \operatorname{op}}\right)^{\bullet}} \ \vec{u}: \vec{t} \& a[\vec{u}/\vec{x}] \Downarrow_{\Box}^{\alpha} \Rightarrow \exists \alpha' \leq \alpha. \ a'[\vec{u}'/\vec{x}] \Downarrow_{\Box}^{\alpha'}$$
(8.13)

We prove this by induction (1) on  $\alpha$  (transfinite induction) and (2) on the derivation of  $\vec{x}: \vec{t} \vdash a' (\mathbf{R}^{* \text{op}})^{\bullet} a: t$ , ordered lexicographically.

First observe that, by the definition of compatible extension, there exists an a'' such that  $\vec{x}: \vec{t} \vdash a' (\mathbf{R}^{* \text{op}})^{\bullet} a: t$  is derived from

$$\vec{x}: \vec{t} \vdash a \; \boldsymbol{R}^{*^{\circ}} \; a^{\prime\prime}: t \tag{8.14}$$

$$\vec{x}: \vec{t} \vdash a'' \ (\widehat{\boldsymbol{R}^{* \text{op}}})^{\bullet} \ a': t \tag{8.15}$$

Hence  $a[\vec{u}/\vec{x}] \mathbf{R}^n a''[\vec{u}/\vec{x}] : t$  for some  $n \ge 0$ . By n applications of the assumption that  $\mathbf{R}$  is an improvement simulation, we get  $\alpha'' \le \alpha$  such that  $a''[\vec{u}/\vec{x}] \Downarrow_{\Box}^{\alpha''}$ .

We now proceed to show that there exists  $\alpha' \leq \alpha''$  such that  $a'[\vec{u}'/\vec{x}] \Downarrow_{\Box}^{\alpha'}$ , by analysis of the derivation of  $a''[\vec{u}/\vec{x}] \Downarrow_{\Box}^{\alpha''}$ .

Case (Term<sup> $\alpha''$ </sup> value)

$$a''$$
 is a value  
 $\alpha'' = 0$ 

By (8.15) and Lemma 3.5.1, a' is also a value. Hence  $a' \Downarrow_{\Box}^{0}$  by (Term<sup>0</sup><sub> $\Box$ </sub> value) too.

Case (Term<sup> $\alpha''$ </sup><sub> $\Box$ </sub> redex  $\sqrt{}$ )

$$a'' = (\lambda y. b) v$$
$$b[v/y][\vec{u}/\vec{x}] \Downarrow_{\Box}^{\alpha_0}$$
$$\alpha'' = \alpha_0 + 1$$

By (8.15),  $a' = (\lambda y. b') v'$  with  $\vec{x} : \vec{t}, y : t' \vdash b' (\mathbf{R}^{* \operatorname{op}})^{\bullet} b : t$  and  $\vec{x} : \vec{t} \vdash v' (\mathbf{R}^{* \operatorname{op}})^{\bullet} v : t'$ . By substitutivity,  $\vec{x} : \vec{t} \vdash b' [v'/y] (\mathbf{R}^{* \operatorname{op}})^{\bullet} b[v/y] : t$ . By part (1) of the induction hypothesis, we get  $\alpha'_0 \leq \alpha_0$  such that  $b' [v'/y] [\vec{u}'/\vec{x}] \Downarrow_{\Box}^{\alpha'_0}$ . Let  $\alpha' = \alpha'_0 + 1$ . Then  $\alpha' \leq \alpha''$  and  $a' [\vec{u}'/\vec{x}] \Downarrow_{\Box}^{\alpha'}$ , by  $(\operatorname{Term}_{\Box}^{\alpha'} \operatorname{redex} \sqrt{)}$ .

Case (Term<sup> $\alpha''$ </sup> redex  $\tau$ )

$$a''[\vec{u}/\vec{x}] \xrightarrow{\tau} B$$
  
$$b \Downarrow_{\Box}^{\alpha_b}, \text{ for all } b \in B$$
  
$$\alpha'' = \bigcup_{b \in B} \alpha_b$$

We argue by cases on the derivation of  $a''[\vec{u}/\vec{x}] \xrightarrow{\tau} B$  by the rules in Table 8.4.

Case (Redex<sub> $\Box$ </sub>  $\tau$  case  $\times$ )

$$\begin{aligned} a'' &= \mathbf{case} \ v \ \mathbf{of} \ \langle \vec{y} \rangle. \ b \\ v[\vec{u}/\vec{x}] &= \langle \vec{v} \rangle \\ B &= \{ b[\vec{u}\vec{v}/\vec{x}\vec{y}] \} \\ b[\vec{u}\vec{v}/\vec{x}\vec{y}] \Downarrow_{\Box}^{\alpha''} \end{aligned}$$

By (8.15),  $a' = \operatorname{case} v'$  of  $\langle \vec{y} \rangle . b'$  with  $\vec{x} : \vec{t} \vdash v' (\overline{\mathbf{R}^{* \operatorname{op}}})^{\bullet} v : t'_1 \times \ldots \times t'_n$  and  $\vec{x} : \vec{t}, \vec{y} : \vec{t'} \vdash b' (\mathbf{R}^{* \operatorname{op}})^{\bullet} b : t$ . Then, by Lemma 3.4.1(3),  $v'[\vec{u'}/\vec{x}] (\overline{\mathbf{R}^{* \operatorname{op}}})^{\bullet} v[\vec{u}/\vec{x}] : t'_1 \times \ldots \times t'_n$ . This must be derived by (Comp product) from Table 3.2 and we see that  $v'[\vec{u'}/\vec{x}] = \langle \vec{v'} \rangle$  with  $\vec{v'} (\overline{\mathbf{R}^{* \operatorname{op}}})^{\bullet} \vec{v} : \vec{t'}$ . So  $a'[\vec{u'}/\vec{x}] \xrightarrow{\tau}_{\Box} \{b'[\vec{u'}\vec{v'}/\vec{x}\vec{y}]\}$ , by (Redex<sub>\Box</sub>  $\tau$  case  $\times$ ), and  $\vec{u'}\vec{v'} (\overline{\mathbf{R}^{* \operatorname{op}}})^{\bullet} \vec{u}\vec{v} : \vec{tt'}$ ; and then part (2) of the induction hypothesis asserts that there exists  $\alpha' \leq \alpha''$  such that  $b'[\vec{u'}\vec{v'}/\vec{x}\vec{y}] \Downarrow_{\Box}^{\alpha'}$ . We conclude that  $a'[\vec{u'}/\vec{x}] \Downarrow_{\Box}^{\alpha'}$  by (Term<sup>\alpha'</sup> redex  $\tau$ ).

Case (Redex<sub> $\Box$ </sub>  $\tau$  case +) Similar to the previous case. Case (Redex<sub> $\Box$ </sub>  $\tau$  or)

$$\begin{aligned} a'' &= a_1 \text{ or } a_2 \\ B &= \{ a_1[\vec{u}/\vec{x}], a_2[\vec{u}/\vec{x}] \} \\ a_1[\vec{u}/\vec{x}] \Downarrow_{\Box}^{\alpha_1} \\ a_2[\vec{u}/\vec{x}] \Downarrow_{\Box}^{\alpha_2} \\ \alpha'' &= \alpha_1 \cup \alpha_2 \end{aligned}$$

By (8.15),  $a' = a'_1$  or  $a'_2$  with  $\vec{x} : \vec{t} \vdash a'_1 (\boldsymbol{R}^{* \operatorname{op}})^{\bullet} a_1 : t$  and  $\vec{x} : \vec{t} \vdash a'_2 (\boldsymbol{R}^{* \operatorname{op}})^{\bullet} a_2 : t$ . Hence  $a'[\vec{u}'/\vec{x}] \xrightarrow{\tau}_{\Box} \{a'_1[\vec{u}'/\vec{x}], a'_2[\vec{u}'/\vec{x}]\}$ . By part (2) of the induction hypothesis, there are  $\alpha'_1 \leq \alpha_1$  and  $\alpha'_1 \leq \alpha_1$  such that  $a'_1[\vec{u}'/\vec{x}] \Downarrow_{\Box}^{\alpha'_1}$  and  $a'_2[\vec{u}'/\vec{x}] \Downarrow_{\Box}^{\alpha'_2}$ . Let  $\alpha' = \alpha'_1 \cup \alpha'_2$ . Then  $\alpha' \leq \alpha''$  and  $a'[\vec{u}'/\vec{x}] \Downarrow_{\Box}^{\alpha'}$  by (Term<sup> $\alpha'_1$ </sup> redex  $\tau$ ).

Case (Redex<sub> $\Box$ </sub>  $\tau$  choose)

$$a'' = ?$$
$$B = \{ \ulcorneri \urcorner | i < \omega \}$$
$$\alpha'' = 0$$

By (8.15), a' = ?. Hence  $a'[\vec{u}'/\vec{x}] = ?$  and  $a'[\vec{u}'/\vec{x}] \Downarrow_{\Box}^{0}$  by (Redex<sub> $\Box$ </sub>  $\tau$  choose) and (Term<sup>0</sup><sub> $\Box$ </sub> redex  $\tau$ ).

Case (Term<sup> $\alpha''$ </sup> let)

$$a'' = \mathbf{let} \ y = a_0 \ \mathbf{in} \ b_0$$
$$a_0[\vec{u}/\vec{x}] \Downarrow_{\Box}^{\alpha_0}$$
$$V = \{v \mid a_0[\vec{u}/\vec{x}] \rightsquigarrow_{\Diamond} v\}$$
$$b_0[\vec{u}v/\vec{x}y] \Downarrow_{\Box}^{\alpha_v}, \text{ for all } v \in V$$
$$\alpha'' = \alpha_0 \cup \bigcup_{v \in V} \alpha_v$$

By (8.15),  $a' = \operatorname{let} y = a'_0$  in  $b'_0$  with  $\vec{x} : \vec{t} \vdash a'_0 (\mathbf{R}^{*\mathrm{op}})^{\bullet} a_0 : t'$  and  $\vec{x} : \vec{t}, y : t' \vdash b'_0 (\mathbf{R}^{*\mathrm{op}})^{\bullet} b_0 : t$ . By part (2) of the induction hypothesis, we get  $\alpha'_0 \leq \alpha_0$  such that  $a'_0[\vec{u}'/\vec{x}] \Downarrow_{\Box}^{\alpha'_0}$ . Let  $V' = \{v' \mid a'_0[\vec{u}'/\vec{x}] \xrightarrow{\sim} v'\}$  and fix any  $v' \in V'$ . Since  $(\mathbf{R}^{*\mathrm{op}})^{\bullet}$  is a lower simulation, there is  $v \in V$  with  $v' (\mathbf{R}^{*\mathrm{op}})^{\bullet} v : t'$  and hence  $\vec{u}'v'(\mathbf{R}^{*\mathrm{op}})^{\bullet}\vec{u}v : \vec{t}t'$ . Again by part (2) of the induction hypothesis, there is an  $\alpha'_{v'} \leq \alpha_v$  such that  $b'_0[\vec{u}'v'/\vec{x}y] \Downarrow_{\Box}^{\alpha'_v}$ . Let  $\alpha' = \alpha'_0 \cup \bigcup_{v' \in V'} \alpha'_{v'}$ . Then  $a'[\vec{u}'/\vec{x}] \Downarrow_{\Box}^{\alpha'}$ , by (Term\_{\Box}^{\alpha'} let). Furthermore, since for all  $v' \in V'$ 

there exists some  $v \in V$  such that  $\alpha'_{v'} \leq \alpha_v$ , we see that  $\bigcup_{v' \in V'} \alpha'_{v'} \leq \bigcup_{v \in V} \alpha_v$ . As  $\alpha'_0 \leq \alpha_0$  too, we get that  $\alpha' \leq \alpha''$ .

Case (Term<sup> $\alpha''$ </sup><sub> $\Box$ </sub> amb left)

$$a'' = \mathbf{amb}(a_1, a_2)$$
$$a_1[\vec{u}/\vec{x}] \Downarrow_{\Box}^{\alpha''}$$

By (8.15),  $a' = \operatorname{amb}(a'_1, a'_2)$  with  $\vec{x} : \vec{t} \vdash a'_1 (\mathbf{R}^{* \operatorname{op}})^{\bullet} a_1 : t$  and  $\vec{x} : \vec{t} \vdash a'_2 (\mathbf{R}^{* \operatorname{op}})^{\bullet} a_2 : t$ . By part (2) of the induction hypothesis, we get  $\alpha' \leq \alpha''$  such that  $a'_1[\vec{u}'/\vec{x}] \Downarrow_{\Box}^{\alpha'}$ . Hence  $a'[\vec{u}'/\vec{x}] \Downarrow_{\Box}^{\alpha'}$ , by (Term<sup> $\alpha'$ </sup> amb left).

**Case (Term**<sup> $\alpha''$ </sup> **amb right)** Symmetrical to the previous case.

We conclude that (8.13) holds, as required to show that  $\mathbf{R}^{*\S}[Id]$  is improvement-adequate and thus that  $\mathbf{R}^{*\S}$  is an improvement simulation.

From this lemma we derive that improvement similarity is a pre-congruence.

**Proposition 8.4.2**  $\gtrsim^{\circ}$  is a pre-congruence.

**Proof** Analogous to that of Proposition 4.8.2.

Improvement similarity is must-pre-adequate, so we can conclude that it is included in must contextual approximation. As a consequence we also have that cost equivalence is included in must contextual equivalence.

Improvement and cost equivalence enjoy a rich (in)equational theory. Examples 8.3.2, 8.3.3, and 8.3.4 all hold for improvement and cost equivalence in place of must contextual approximation and equivalence. (But we should note that the facts that **amb** is idempotent and that  $\Omega$  is unit for **amb**, up to cost equivalence, are quite sensitive to the particular choice of cost measure for must termination.)

Because improvement similarity and cost equivalence are co-inductively defined relations, they support co-inductive arguments about infinite behaviour of higher order functions.

**Example 8.4.3** For every type t, let

$$t \operatorname{stream} \stackrel{\text{def}}{=} \mu \chi. \operatorname{unit} \rightarrow t \times \chi$$

be the type of potentially infinite streams of values of type t. We define the abbreviations:

$$\mathbf{strm} \ a \stackrel{\text{def}}{=} \quad \mathbf{inj} \ \lambda \langle \ \rangle. \ a$$
  
$$\mathbf{case} \ a \ \mathbf{of} \ \mathbf{strm} \langle x, s \rangle. \ b \stackrel{\text{def}}{=} \quad \mathbf{case} \ a \ \mathbf{of} \ \mathbf{inj} \ h. \ \mathbf{case} \ h \ \langle \ \rangle \ \mathbf{of} \ \langle x, s \rangle. \ b$$

where h is not free in b. For instance, strm  $\Omega$  is the empty stream that diverges when queried. The following program shuffles streams by arbitrarily reordering elements.

The ambiguous choice chooses whether the first element of the output stream should be (1) the same as the first element x' of the input stream s, or (2) some other element x'' found by taking the first element of the remainder s' of the input stream after shuffling it by a recursive call. In each case, the remainder of the output stream is obtained by recursively shuffling all the remaining elements from the input stream.

Let us prove that

&

shuffle 
$$v \gtrsim v : t$$
 stream, if  $v : (t \text{ stream})$  (8.16)

that is, (i) a possible outcome of shuffling a stream is the stream itself, and (ii) it is cheaper not to shuffle.

Let  $shuffle' \stackrel{\text{def}}{=} \lambda s. \operatorname{fix}[\lambda f. \lambda s. \operatorname{strm} sbody[f, s]] s.$  Then  $shuffle v \rightarrow^5 \operatorname{strm} sbody[shuffle', v]$ and it is easy to see that

$$shuffle \ v \gtrsim \mathbf{strm} \ sbody[shuffle', v] : t \ stream$$

As any v: (t stream) is of the form strm a, for some  $a: t \times (t \text{ stream})$ , we see that it suffices to show that the closed relation  $\mathbf{R}$ , given by

$$\Gamma \vdash sbody[shuffle', strm a] \mathbf{R} a : t \times (t \text{ stream}), \text{ if } a : t \times (t \text{ stream})$$

is included in  $\geq$ . We prove that the reflexive closure  $\mathbf{R} \cup Id_{\mathbf{0}}$  is an improvement simulation; then the desired inclusion follows by co-induction. The interesting case we need to check is, for  $a: t \times (t \text{ stream})$ ,

$$\begin{array}{ll} (i) & \forall u, b. \ a \rightsquigarrow_{\Diamond} \langle u, \operatorname{strm} b \rangle \Rightarrow \\ & sbody[shuffle', \operatorname{strm} a] \rightsquigarrow_{\Diamond} \langle u, \operatorname{strm} sbody[shuffle', \operatorname{strm} b] \rangle \\ \\ (ii) & \forall \alpha < \omega_1^{CK}. \ sbody[shuffle', \operatorname{strm} a] \Downarrow_{\Box}^{\alpha} \Rightarrow \exists \alpha' < \alpha. \ a \Downarrow_{\Box}^{\alpha'} \end{array}$$

(i) holds because we can always choose the first branch of the ambiguous choice, and we note that  $sbody[shuffle', strm b] R b : t \times (t \text{ stream})$  so that we get

 $\langle u, \mathbf{strm} \ sbody[shuffle', \mathbf{strm} \ b] \rangle \ \overline{(\mathbf{R} \cup Id_{\mathbf{0}})^{\circ}} \ \langle u, \mathbf{strm} \ b \rangle : t \times (t \ \text{stream})$ 

(*ii*) holds because the evaluation of sbody[shuffle', strm a] begins by evaluating a.

It is convenient to represent cost units (function applications) syntactically as "ticks",  $\sqrt{}$ ,

$$\sqrt{a} \stackrel{\text{def}}{=} (\lambda \langle \rangle. a) \langle \rangle$$

i.e.,  $\sqrt{}$  adds one function application step to an expression; we observe that

$$a \rightsquigarrow_{\Diamond} v \quad \text{iff} \quad \sqrt{a} \rightsquigarrow_{\Diamond} v \tag{8.17}$$

$$a \Downarrow_{\Box}^{\kappa} \quad \text{iff} \quad \sqrt{a} \Downarrow_{\Box}^{\kappa+1}$$

$$(8.18)$$

For instance,  $\beta_v$ -equivalence holds for cost equivalence modulo a tick,

$$(\lambda x. a) v \stackrel{\text{\tiny (\lambda)}}{\sim} \sqrt{a[v/x]} : t, \text{ if } v : t' \text{ and } x : t' \vdash a : t$$

A 'tick algebra' of such laws can be used for equational reasoning about computation steps; see Sands (1998b).

It is possible to enhance Lemma 8.4.1 along the lines in Lassen (1998) and Pitts (1995) to establish the following proof rule for "improvement simulation up to improvement similarity and context". We omit the proof.

# Proposition 8.4.4 $\frac{\boldsymbol{R} \subseteq \langle \boldsymbol{\Sigma}^{\circ} | \boldsymbol{R}^{\mathsf{C}} \rangle_{\mathsf{I}}}{\boldsymbol{R} \subseteq \boldsymbol{\Sigma}}$

A significant attraction of improvement similarity is that it admits a proof rule for reasoning about recursion, the so-called improvement theorem, which is not valid for the usual, more coarse-grained semantic preorders which abstract from computational cost. This seems to be particularly important here because all the usual techniques for reasoning about recursion break down in the presence of fairness. The improvement theorem is a sort of recursion co-induction rule. Approximately, it asserts that  $\mathbf{Y}(\lambda f.v)$  is the greatest fixed point of the functional  $\lambda f.v$  with respect to improvement. However,  $\mathbf{Y}(\lambda f.v)$  is not quite a fixed point of  $\lambda f.v$  up to cost equivalence. Rather,  $\mathbf{Y}(\lambda f.v) \xrightarrow{\checkmark} \text{fix}[\lambda f.v]$  and  $\text{fix}[\lambda f.v]$  is a fixed point of  $\lambda f.v$ , modulo a tick and an eta redex,

$$\operatorname{fix}[\lambda f. v] \stackrel{\text{}}{\simeq} \sqrt{(\lambda f. v)} (\lambda x. \operatorname{fix}[\lambda f. v] x) : t_1 \rightharpoonup t_2, \quad \operatorname{if} (\lambda f. v) : (t_1 \rightharpoonup t_2) \rightharpoonup t_1 \rightharpoonup t_2$$
(8.19)

The improvement theorem asserts that  $\operatorname{fix}[\lambda f, v]$  is the greatest solution of (8.19), even with  $\gtrsim$  in place of  $\mathfrak{D}$ .

**Proposition 8.4.5 (Improvement theorem)** 
$$\frac{a \gtrsim \sqrt{(\lambda f. v) (\lambda x. a x) : t_1 \rightharpoonup t_2}}{a \gtrsim \mathsf{fix}[\lambda f. v] : t_1 \rightharpoonup t_2}$$

**Proof** Assume that  $a \gtrsim \sqrt{\lambda f. v} (\lambda x. a x) : t_1 \rightarrow t_2$ . Then observe that the right hand side must terminate at cost 2,

$$\sqrt{(\lambda f. v)} (\lambda x. a x) \Downarrow_{\Box}^2$$

and that it evaluates deterministically,

$$\sqrt{(\lambda f. v) (\lambda x. a x)} \rightsquigarrow_{\Diamond} v[(\lambda x. a x)/f]$$

By the assumption and the definition of improvement similarity, (i)  $a \rightsquigarrow_{\Diamond} u$  for some u such that  $u \boxtimes^{\frown} v[(\lambda x. a x)/f] : t_1 \rightharpoonup t_2$ ; and (ii) if  $a \Downarrow_{\Box}^{\alpha}$  then  $\alpha \ge 2$ .

We prove that  $a \gtrsim fix[\lambda f. v] : t_1 \rightarrow t_2$  by means of Proposition 8.4.4. Let **R** be the singleton relation

$$\boldsymbol{R}: (t_1 \rightharpoonup t_2) = \{(a, \mathsf{fix}[\lambda f. v])\}$$

By the observations about a and u above and the fact that  $\operatorname{fix}[\lambda f. v]$  evaluates deterministically,  $\operatorname{fix}[\lambda f. v] \rightsquigarrow_{\Box} v[(\lambda x. \operatorname{fix}[\lambda f. v] x)/f]$ , it suffices to show that

$$u \stackrel{\sim}{\boxtimes}^{\circ} \mathbf{R}^{\mathsf{C}} v[(\lambda x. \operatorname{fix}[\lambda f. v] x)/f] : t_1 \rightharpoonup t_2$$

This holds because we have that  $u \stackrel{\frown}{\succeq}^{\circ} v[(\lambda x. a x)/f] : t_1 \rightarrow t_2$ , from (i) above, and

$$v[(\lambda x. a x)/f] \overline{\mathbf{R}^{\mathsf{C}}} v[(\lambda x. \operatorname{fix}[\lambda f. v] x)/f] : t_1 \rightharpoonup t_2$$

from (3.14) and (3.11).

We conclude that  $\mathbf{R} \subseteq \langle \succeq^{\circ} \mathbf{R}^{\mathsf{C}} \rangle_{\mathsf{I}}$  and hence  $\mathbf{R} \subseteq \succeq$ . That is,  $a \succeq \mathsf{fix}[\lambda f. v] : t_1 \rightharpoonup t_2$ , as required.

We remark that the improvement theorem is false with  $\mathfrak{D}$  or  $\mathfrak{D}^{\mathrm{op}}$  in place of  $\mathfrak{D}$ . In other words, fixed points are not unique with respect to cost equivalence. A counterexample for both is when  $v = (\lambda x. \Omega \text{ or } (f x))$  and  $a = \operatorname{fix}[\lambda f. \lambda x. \Omega \text{ or } \langle \rangle]$ , where we have that

$$a \ll \sqrt{(\lambda f. v)} (\lambda x. a x) : \text{unit} \rightarrow \text{unit}$$

but neither  $a \ll \text{fix}[\lambda f. v]$ : unit  $\rightarrow$  unit nor  $\text{fix}[\lambda f. v] \gtrsim a$ : unit  $\rightarrow$  unit holds. The intuition behind this counterexample is that, between expressions that may diverge, cost equivalence is more or less mutual may similarity for which fixed points are not unique; in particular,  $(\lambda x. \Omega \text{ or } \langle \rangle)$  and  $(\lambda x. \Omega)$  are not may similar but they are both fixed points of the functional  $\lambda f. v$  up to mutual may similarity.

The following corollary of the improvement theorem is closer to Sands' formulation (1998b).

**Corollary 8.4.6** 
$$\frac{(\lambda f. u) (\lambda x. \operatorname{fix}[\lambda f. u] x) \gtrsim (\lambda f. v) (\lambda x. \operatorname{fix}[\lambda f. u] x) : t_1 \rightharpoonup t_2}{\operatorname{fix}[\lambda f. u] \gtrsim \operatorname{fix}[\lambda f. v] : t_1 \rightharpoonup t_2}$$

**Proof** Since  $\operatorname{fix}[\lambda f. u] \approx \sqrt{(\lambda f. u)} (\lambda x. \operatorname{fix}[\lambda f. u] x) : t_1 \rightarrow t_2$ , the premise of the rule implies that  $\operatorname{fix}[\lambda f. u] \gtrsim \sqrt{(\lambda f. v)} (\lambda x. \operatorname{fix}[\lambda f. u] x) : t_1 \rightarrow t_2$  and the conclusion is immediate from Proposition 8.4.5.

Example 8.4.7 Let us consider the *shuffle* program from Example 8.4.3 again,

$$shuffle = \mathbf{Y} \lambda f. \lambda s. \mathbf{strm}(\mathbf{case} \ s \ \mathbf{of} \ \mathbf{strm}\langle x', s' \rangle.$$
  
 $\mathbf{amb}(\langle x', f \ s' \rangle,$   
 $\mathbf{case} \ f \ s' \ \mathbf{of} \ \mathbf{strm}\langle x'', s'' \rangle.$   
 $\langle x'', f \ (\mathbf{strm}\langle x', s'' \rangle) \rangle)$ 

It looks uneconomical to use two consecutive recursive calls in the second branch of the ambiguous choice. It would suffice to let the first of the two recursive calls move an arbitrary element of the stream s' to the front of the stream and let the second recursive call perform the recursive shuffling of the remainder of the stream. This optimisation is incorporated in the next program, *pshuffle*. The auxiliary function *promote* moves an arbitrary element of a stream to the front.

$$pshuffle = \mathbf{Y} \lambda f. \lambda s. \mathbf{strm}(\mathbf{case} \ s \ \mathbf{of} \ \mathbf{strm}\langle x', s' \rangle.$$
  

$$\mathbf{amb}(\langle x', f \ s' \rangle,$$
  

$$\mathbf{case} \ promote \ s' \ \mathbf{of} \ \mathbf{strm}\langle x'', s'' \rangle$$
  

$$\langle x'', f \ (\mathbf{strm}\langle x', s'' \rangle) \rangle)$$
  

$$promote = \mathbf{Y} \lambda f. \lambda s. \mathbf{strm}(\mathbf{case} \ s \ \mathbf{of} \ \mathbf{strm}\langle x', s' \rangle.$$
  

$$\mathbf{amb}(\langle x', s' \rangle,$$
  

$$\mathbf{case} \ f \ s' \ \mathbf{of} \ \mathbf{strm}\langle x'', s'' \rangle.$$
  

$$\langle x'', \mathbf{strm}\langle x', s'' \rangle))$$

Indeed, pshuffle is an improvement of shuffle in the improvement similarity ordering. This is proved by two simple applications of (the corollary of) the improvement theorem. First, we use (8.16) from Example 8.4.3 to deduce

$$shuffle \gtrsim promote : (t \text{ stream}) \rightarrow (t \text{ stream})$$

by Corollary 8.4.6. Given this, we deduce the result,

$$shuffle \gtrsim pshuffle : (t \text{ stream}) \rightarrow (t \text{ stream})$$

$$(8.20)$$

again by Corollary 8.4.6.

It is plausible that *shuffle* and *pshuffle* are must contextually equivalent. (8.20) implies that *shuffle* must contextually approximates *pshuffle*, but there does not seem to be any easy way of proving the reverse—*pshuffle* is not improvement similar to *shuffle* and we do not have other co-inductive methods for proving results about must contextual approximation between infinite data structures.  $\Box$ 

### 8.5 Future work

This chapter has presented promising but preliminary results about the theory of contextual equivalence for ambiguous choice. Much remains to be done. In order to assess the usefulness of the proof rules, they should be tested on more problems involving ambiguous choice. Since  $\beta_v$ -equivalence is valid, it should be possible to translate many problems about must contextual approximation and equivalence into a form where they can be solved up to improvement similarity, using the co-induction proof rules for improvement similarity.

The big open problem is whether convex bisimilarity (cf. §7.3) is a congruence in the presence of ambiguous choice. In that case convex bisimilarity would be included in must contextual equivalence, and we would get a good co-induction proof rule for establishing must contextual equivalence. Several people have attacked this problem, but, so far, it has eluded every attempt. A stronger result would be if refinement similarity is a pre-congruence and, hence, included in must contextual approximation. It would provide a co-induction proof rule for must contextual approximation. My preliminary investigations of the relationship between must contextual approximation and refinement similarity suggest the conjecture that must contextual approximation is included in refinement similarity. This does not directly lead to useful reasoning principles but it might assist the search for a solution to the open problem about the reverse inclusion.

An important property which we failed to establish for fair must contextual approximation is closure under open extension,  $(\Box_{\Box 0})^{\circ} \subseteq \Box_{\Box}$ . This would follow from the inclusions that we left as open problems in the preceding paragraph.

Improvement similarity is one out of several improvement relations that one could consider for our language. The ideas from our pre-congruence proof for improvement similarity seem to extend to some other choices. This topic should be worth pursuing further.

In a wider perspective, hopefully the understanding about fairness gained through the study of ambiguous choice can be employed in other settings with fairness constraints; for instance, the actor model of asynchronously communicating processes (Agha, Mason, Smith, and Talcott 1997), and the related communicative facet of action semantics (Mosses 1992).

### Appendix A

## Encoding ordinal-bounded fixed point operators

This appendix demonstrates ways of encoding the ordinal-bounded fixed point expressions from §6.7 in two different extensions of the language. The first extension is a parallel combinator, proposed by Boudol (1994) for  $\lambda$ -calculus. The other is references and control operators as found in Standard ML of New Jersey (Appel and MacQueen 1991).

The encodings are included as a curiosity and are presented without proofs. For readers who are familiar with Boudol's parallel operator or with control operators, the encodings may clarify the computational behaviour of the transfinite language constructs from  $\S6.7$ .

One would hope that the encodings could provide insights with regard to the open problem in §6.7: are the transfinite fixed point constructs a conservative extension of the theory of contextual approximation. The parallel combinator in itself is not a conservative extension and neither are references or control operators. It is not clear whether this has implications for the properties of the encoded constructs.

Let us first define a type ord for representing recursive ordinals:

ord 
$$\stackrel{\text{def}}{=} \mu \chi. \text{ unit } + \chi + (\text{nat} \rightarrow \chi)$$

Recall from §2.1.1 that  $\mathbf{0} \stackrel{\text{def}}{=} \mathbf{inj}_1 \langle \rangle$  and  $\mathbf{succ} \ u \stackrel{\text{def}}{=} \mathbf{inj}_2 u$ . Further, we define

$$\lim v \stackrel{\text{def}}{=} \inf_{3} v$$

Certain closed values of type ord represent ordinals: **0** represents 0; **succ** u represents  $\alpha + 1$  if u represents  $\alpha$ ; **lim** v represents  $\bigcup_{i < \omega} \alpha_i$ , if  $v \ulcorner i \urcorner \rightsquigarrow u_i$  and  $u_i$  represents  $\alpha_i$  for each  $i < \omega$ . Not all closed values of type ord encode any ordinal  $\alpha$ , e.g., (**lim**  $\lambda x$ .  $\Omega$ ) and the result of **Y** ( $\lambda f$ .  $\lambda x$ . **lim** f) do not. The ordinals encoded by values of type ord are the recursive ordinals (Church and Kleene 1937).

### Parallelism

Boudol (1994) introduces a parallel combinator,  $\|$ , into the  $\lambda$ -calculus. The meaning of  $a_1 \| a_2$  is that it splits the global computation in two parallel processes, and the global computation must terminate if either of the two processes must terminate. But it is difficult to say what

the outcome is. So let us just specify the must termination behaviour of the language when extended with the parallel combinator. Firstly, the must termination of the nondeterministic language from Chapter 5 can be specified by the following two rules.

$$\begin{array}{ll} \text{(Term}_{\Box} \text{ value)} & v \Downarrow_{\Box} \\ \text{(Term}_{\Box} \text{ redex)} & \frac{\forall b \in B. \ b \Downarrow_{\Box}}{a \Downarrow_{\Box}} & \text{if } a \rightarrowtail_{\Box} B \end{array}$$

That is, the must termination predicate is the least predicate closed under (Term<sub> $\Box$ </sub> value) and (Term<sub> $\Box$ </sub> trans).

We can specify the must termination behaviour of the parallel combinator by adding the rule:

(Term<sub>$$\square$$</sub> par)  $\frac{E\llbracket a_i \rrbracket \Downarrow_{\square}}{E\llbracket a_1 \parallel a_2 \rrbracket \Downarrow_{\square}}$  if  $i = 1, 2$ 

Now we are able to encode the transfinite fixed point expressions from §6.7. We define a function **Yord** which maps every ordinal encoding into a fixed point combinator whose depth of recursion is bounded by the encoded ordinal.

Now  $\mathbf{Y}^{(\alpha)}$  is encoded by  $\mathbf{Yord} u$  where u is some closed value u: ord that encodes the recursive ordinal  $\alpha$ .

### **References and control operators**

The code on pp. 115–116 runs under Standard ML of New Jersey<sup>1</sup> and uses references and control operators to implement ordinal-bounded fixed point combinators.

<sup>&</sup>lt;sup>1</sup>See Appel and MacQueen (1991) and URL http://cm.bell-labs.com/cm/cs/what/smlnj

```
(* ORDINAL-BOUNDED FIXED POINT OPERATORS IN SML/NJ
                                                                         *)
(* Time-stamp: <98/08/21 16:50:13 sbl21>
                                                                         *)
(* URL http://www.cl.cam.ac.uk/users/sbl21/thesis/fix.sml
                                                                         *)
                                                                         *)
(*
(* by S. B. Lassen
                                                                         *)
      University of Cambridge Computer Laboratory
                                                                         *)
(*
                Soeren.Lassen@cl.cam.ac.uk
      email:
                                                                         *)
(*
(*
      home page: http://www.cl.cam.ac.uk/users/sbl21
                                                                         *)
(* Uses SML/NJ's callcc and throw operations *)
open SMLofNJ.Cont;
(* ord is the datatype of recursive ordinals *)
type nat = int
datatype ord = Zero | Succ of ord | Lim of nat->ord
fun nat2ord n = if n<=0 then Zero else Succ (nat2ord (n-1))
val omega0 = Lim nat2ord
(* The following operations illustrate the workings of ordinals but
   are not used in the definitions below.
                                                                         *)
fun add a Zero
                   = a
  | add a (Succ b) = Succ (add a b)
  | add a (Lim 1) = Lim (fn n => add a (1 n))
fun mult a Zero
                    = Zero
  | mult a (Succ b) = add (mult a b) a
  | mult a (Lim l) = Lim (fn n => mult a (l n))
fun expon a Zero
                    = Succ Zero
  | expon a (Succ b) = mult (expon a b) a
  | expon a (Lim 1) = Lim (fn n => expon a (1 n))
(* A queue of "composed continuations" is needed
   for a breath-first traversal of an ordinal "tree".
   "Composed continations" are represented by unit->unit functions;
   they never return so the unit result type is chosen arbitrarily.
                                                                         *)
val queue = ref ([]:(unit->unit) list);
exception Bottom (* raised if taking first of the empty queue
                     (when recursion overruns a finite ordinal bound)
                                                                         *)
fun init q = q := []
fun insert q k = q:=(!q)@[k]
fun first q = case !q of [] => raise Bottom | k::ks \Rightarrow (q:=ks; k)
```

(\* Yord produces ordinal-bounded fixed point operators. Yord a f computes the a'th approximation to the least fixed point of the functional f. \*) exception DeadCode (\* never raised \*) (\* Yord : ord -> (('a -> 'b) -> 'a -> 'b) -> 'a -> 'b \*) fun Yord Zero f = (first queue (); (\* never returns \*) raise DeadCode) (\* unconstrained result type \*) | Yord (Succ a) f = f (fn x => Yord a f x) | Yord (Lim 1) f = callcc (fn k => let val a = Lim (fn n => 1 (n+1)) val k' = fn () => throw k (Yord a f) in insert queue k'; Yord (1 0) f end) (\* Examples \*) fun fac a = Yord a (fn fac  $\Rightarrow$  fn x  $\Rightarrow$  if x=0 then 1 else x\*(fac (x-1))) val example1 = (init queue; fac (nat2ord 5) 4); (\* result 24 \*) val example3 = (init queue; fac (nat2ord 5) 5); (\* raises Bottom \*)

### Bibliography

- Abadi, M. and L. Cardelli (1996). A Theory of Objects. Springer-Verlag.
- Abramsky, S. (1983). On semantic foundations for applicative multiprogramming. In J. Díaz (Ed.), Automata, Languages and Programming, 10th Colloquium, Barcelona, Volume 154 of Lecture Notes in Computer Science, pp. 1–14. Springer-Verlag.
- Abramsky, S. (1990). The lazy lambda calculus. In D. Turner (Ed.), Research Topics in Functional Programming, pp. 65–116. Addison-Wesley.
- Agha, G. A., I. A. Mason, S. F. Smith, and C. L. Talcott (1997). A foundation for actor computation. Journal of Functional Programming 7(1), 1–72.
- Appel, A. W. and D. B. MacQueen (1991). Standard ML of New Jersey. In J. Małuszyński and M. Wirsing (Eds.), Proc. 3rd Int. Symposium on Programming Language Implementation and Logic Programming, Passau, Germany, Volume 528 of Lecture Notes in Computer Science, pp. 1–13. Springer-Verlag.
- Apt, K. R. and G. D. Plotkin (1986). Countable nondeterminism and random assignment. J.ACM 33(4), 724–767.
- Astesiano, E. and G. Costa (1980). Nondeterminism and fully abstract models. *RAIRO* 14(4), 323–347.
- Astesiano, E. and G. Costa (1984). Distributive semantics for nondeterministic typed  $\lambda$ -calculi. Theoretical Computer Science 32, 121–156.
- Barendregt, H. P. (1984). The Lambda Calculus: Its Syntax and Semantics (revised ed.). Number 103 in Studies in Logic and the Foundations of Mathematics. Amsterdam: North-Holland.
- Bierman, G. M. (1998). A computational interpretation of the lambda-mu calculus. In L. Brim, J. Gruska, and J. Zlatuska (Eds.), Proc. 23rd Mathematical Foundations of Computer Science, Brno, Czech Republic, Volume 1450, pp. 336–345. Springer-Verlag.
- Bird, R. and O. de Moor (1997). *Algebra of Programming*. International Series in Computer Science. Prentice-Hall.
- Birkedal, L. and R. Harper (1997). Operational interpretations of recursive types in an operational setting (summary). In M. Abadi and T. Ito (Eds.), Symposium on Theoretical Aspects of Computer Science, Sendai, Japan, Volume 1281 of Lecture Notes in Computer Science. Springer-Verlag.
- Bloom, B. (1990). Can LCF be topped? Flat lattice models of typed  $\lambda$ -calculus. Information and Computation 87(1/2), 263–300.

- Boudol, G. (1980). Sémantique opérationelle et algébrique des programmes récursifs nondéterministes. Thèse d'etat, Université de Paris 7.
- Boudol, G. (1989). Towards a lambda-calculus for concurrent and communicating systems. In J. D. F. Orejas (Ed.), Proceedings of the International Joint Conference on Theory and Practice of Software Development : Vol. 1, Volume 351 of Lecture Notes in Computer Science, pp. 149–161. Springer-Verlag.
- Boudol, G. (1994). Lambda-calculi for (strict) parallel functions. Information and Computation 108(1), 51–127.
- Braüner, T. (1996, November). An axiomatic approach to adequacy. Technical Report DS-96-4, BRICS, Dept. of Computer Science, Univ. of Aarhus. Ph.D. thesis.
- Broy, M. (1986). A theory for nondeterminism, parallelism, communication, and concurrency. Theoretical Computer Science 45, 1–61.
- Broy, M. and G. Nelson (1994). Adding fair choice to Dijkstra's calculus. ACM Transactions on Programming Languages and Systems 16(3), 924–938.
- Church, A. and S. C. Kleene (1937). Formal definitions in the theory of ordinal numbers. Fundamenta Mathematica 28, 11–21.
- Crole, R. L. (1994). Categories for Types. Cambridge Mathematical Textbooks. Cambridge University Press.
- de'Liguoro, U. and A. Piperno (1995). Non deterministic extensions of untyped  $\lambda$ -calculus. Information and Computation 122(2), 149–177.
- DeNicola, R. and M. Hennessy (1984). Testing equivalences for processes. Theoretical Computer Science 34, 83–133.
- Dezani-Ciancaglini, M., U. de'Liguoro, and A. Piperno (1996). Finite models for conjunctive-disjunctive  $\lambda$ -calculi. Theoretical Computer Science 170(1–2), 83–128.
- Di Gianantonio, P., F. Honsell, and G. D. Plotkin (1995). Uncountable limits and the lambda calculus. *Nordic Journal of Computing* 2(2), 126–145. Extended version of a paper presented at CONCUR '94.
- Egidi, L., F. Honsell, and S. Ronchi della Rocca (1992). Operational, denotational and logical descriptions: a case study. *Fundamenta Informaticae* 16(2), 149–169.
- Felleisen, M. (1991). On the expressive power of programming languages. In Science of Computer Programming, Volume 17, pp. 35–75. Preliminary version in ESOP '90.
- Felleisen, M. and D. P. Friedman (1987). Control operators, the SECD-machine, and the λcalculus. In M. Wirsing (Ed.), Formal Description of Programming Concepts III, Proc. IFIP TC2 Working Conference, Gl. Avernæs, 1986. IFIP: North-Holland.
- Ferreira, W., M. Hennessy, and A. Jeffrey (1996). A theory of weak bisimulation for core CML. ACM SIGPLAN Notices 31(6), 201–212. Proc. ACM SIGPLAN International Conference on Functional Programming, Philadelphia.
- Fiore, M. P., A. Jung, E. Moggi, P. O'Hearn, J. Riecke, G. Rosolini, and I. Stark (1996). Domains and denotational semantics: History, accomplishments and open problems. Bulletin of the EATCS 59, 227–256.
- Gordon, A. D. (1994). Functional Programming and Input/Output. Cambridge University Press.

- Gordon, A. D. (1995a). Bisimilarity as a theory of functional programming. In Proc. 11th Conference of Mathematical Foundations of Programming Semantics, Volume 1 of Electronic Notes in Theoretical Computer Science. Elsevier.
- Gordon, A. D. (1995b, July). Bisimilarity as a theory of functional programming. Minicourse. BRICS Notes Series NS-95-3, BRICS, Dept. of Computer Science, Univ. of Aarhus. Errata at URL http://research.microsoft.com/~adg/Publications/BRICS-NS-95-3-errata.ps.gz.
- Gordon, A. D. (1995c). A tutorial on co-induction and functional programming. In Proceedings of the 1994 Glasgow Workshop on Functional Programming, Springer Workshops in Computing, pp. 78–95.
- Gordon, A. D. (1998). Operational equivalences for untyped and polymorphic object calculi. See Gordon and Pitts (1998).
- Gordon, A. D., P. Hankin, and S. B. Lassen (1997a). Compilation and equivalence of imperative objects. In S. Ramesh and G. Sivakumar (Eds.), Proc. 17th FST&TCS Conference, Kharagpur, India, December 1997, Volume 1346 of Lecture Notes in Computer Science, pp. 74–87. Springer-Verlag. Extended version appears in (Gordon, Hankin, and Lassen 1997b).
- Gordon, A. D., P. Hankin, and S. B. Lassen (1997b). Compilation and equivalence of imperative objects. Technical Report RS-97-19, BRICS, Dept. of Computer Science, Univ. of Aarhus. Extended version of paper in FST&TCS'97. Also appears as Technical Report 429, University of Cambridge Computer Laboratory.
- Gordon, A. D. and A. M. Pitts (Eds.) (1998). *Higher Order Operational Techniques in* Semantics. Publications of the Newton Institute. Cambridge University Press.
- Gordon, A. D. and G. D. Rees (1996). Bisimilarity for a first-order calculus of objects with subtyping. In *Proc. 23rd ACM Symposium on Principles of Programming Languages*.
- Gunter, C. A. and D. S. Scott (1990). Semantic domains. In J. van Leeuwen (Ed.), *Handbook* of Theoretical Computer Science. Elsevier.
- Hennessy, M. (1988). Algebraic Theory of Processes. MIT Press.
- Hennessy, M. C. B. and E. A. Ashcroft (1980). A mathematical semantics for a nondeterministic typed lambda-calculus. *Theoretical Computer Science* 11(3), 227–245.
- Howe, D. J. (1989). Equality in lazy computation systems. In Proc. 4th Annual IEEE Symposium on Logic in Computer Science.
- Howe, D. J. (1996). Proving congruence of bisimulation in functional programming languages. *Information and Computation* 124(2), 103–112.
- Jacobs, B. and J. Rutten (1997). A tutorial on (co)algebras and (co)induction. Bulletin of the EATCS 62, 222–259.
- Jagadeesan, R. and P. Panangaden (1990). A domain-theoretic model for a higher-order process calculus. In M. S. Paterson (Ed.), *Proceedings of the 17th International Col*loquium on Automata, Languages and Programming, Volume 443 of Lecture Notes in Computer Science, pp. 181–194. Springer-Verlag. Extended version as Technical Report 89-1058, Cornell University.

- Kennaway, R., J. W. Klop, R. Sleep, and F.-J. de Vries (1995). Transfinite reductions in orthogonal term rewriting systems. *Information and Computation* 119(1), 18–38.
- Klop, J. W. (1992). Term rewriting systems. In S. Abramsky et al. (Eds.), Handbook of Logic in Computer Science, Volume 2. Oxford University Press.
- Klop, J. W., V. van Oostrom, and F. van Raamsdonk (1993). Combinatory reduction systems: introduction and survey. *Theoretical Computer Science* 121, 279–308.
- Kumar, K. N. and P. K. Pandya (1993). Infinite parallelism without unbounded nondeterminism in CSP. Acta Informatica 31(5), 467–487.
- Lassen, S. B. (1994). Reasoning with actions. In U. H. Engberg, K. G. Larsen, and P. D. Mosses (Eds.), Proc. 6th Nordic Workshop on Programming Theory, Aarhus, October 1994, Number NS-94-6 in BRICS Notes Series, Dept. of Computer Science, Univ. of Aarhus, pp. 251–265. Superseded by (Lassen 1997).
- Lassen, S. B. (1995, May). Basic Action Theory. Technical Report RS-95-25, BRICS, Dept. of Computer Science, Univ. of Aarhus. Superseded by (Lassen 1997).
- Lassen, S. B. (1996, May). A context lemma for the imperative object calculus. Available from URL http://www.cl.cam.ac.uk/users/sbl21/docs/clioc.html.
- Lassen, S. B. (1997). Action semantics reasoning about functional programs. Mathematical Structures in Computer Science 7(5), 557–589. Special issue dedicated to the Workshop on Logic, Domains, and Programming Languages, Darmstadt, May 1995.
- Lassen, S. B. (1998). Relational reasoning about contexts. See Gordon and Pitts (1998), pp. 91–135.
- Lassen, S. B. and C. S. Pitcher (1998). Similarity and bisimilarity for countable nondeterminism and higher-order functions (extended abstract). In A. Gordon, A. Pitts, and C. Talcott (Eds.), Second Workshop on Higher-Order Operational Techniques in Semantics (HOOTS II), Stanford University, December 1997, Volume 10 of Electronic Notes in Theoretical Computer Science. Elsevier.
- Lehmann, D. J. (1976). *Categories for Fixpoint Semantics*. Ph. D. thesis, University of Warwick.
- Mason, I. A. (1996).Parametric computation. School of Mathematical and Sciences, Computing University of New England. Available from URL http://cs.une.edu.au/~iam/Data/publications.html.
- Mason, I. A., S. F. Smith, and C. L. Talcott (1996). From operational semantics to domain theory. *Information and Computation* 128(1), 26–47.
- Mason, I. A. and C. L. Talcott (1991). Equivalence in functional languages with effects. Journal of Functional Programming 1(3), 297–327.
- McCarthy, J. (1963). A basis for a mathematical theory of computation. In P. Braffort and D. Hirschberg (Eds.), *Computer Programming and Formal Systems*, pp. 33–70. Amsterdam: North-Holland.
- Meyer, A. R. (1988). Semantical paradigms: Notes for an invited lecture. In Proc. 3rd Annual IEEE Symposium on Logic in Computer Science, pp. 236–253. With two appendices by S. S. Cosmadakis.
- Milner, R. (1989). Communication and Concurrency. Prentice-Hall.

- Milner, R., M. Tofte, and R. Harper (1990). *The Definition of Standard ML*. Cambridge, Mass.: MIT Press.
- Mitchell, J. C. (1990). Type systems for programming languages. See van Leeuwen (1990), Chapter 8, pp. 365–458.
- Mitchell, J. C. (1993, October). On abstraction and the expressive power of programming languages. *Science of Computer Programming* 21(2), 141–163.
- Mitchell, J. C. (1996). Foundations for Programming Languages. Foundations of Computing. MIT Press.
- Moggi, E. (1989). Computational lambda-calculus and monads. In *Proc. 4th Annual IEEE Symposium on Logic in Computer Science*, pp. 14–23.
- Moggi, E. (1991). Notions of computation and monads. Information and Computation 93, 55–92.
- Moran, A. K. (1994, May). Natural semantics for non-determinism. Licentiate thesis, Department of Computing Science, Chalmers University of Technology and University of Gothenburg.
- Moran, A. K. (1998, September). Call-by-name, Call-by-need, and McCarthy's Amb. Ph. D. thesis, Department of Computing Science, Chalmers University of Technology and University of Gothenburg.
- Morris, J. H. (1968, December). Lambda-Calculus Models of Programming Languages. Ph. D. thesis, MIT.
- Mosses, P. D. (1992). *Action Semantics*. Number 26 in Cambridge Tracts in Theoretical Computer Science. Cambridge University Press.
- Mosses, P. D. (1996). Theory and practice of action semantics. In Proc. 21st Mathematical Foundations of Computer Science, Cracow, Poland, Volume 1113 of Lecture Notes in Computer Science, pp. 37–61. Springer-Verlag.
- Nordström, B., K. Petersson, and J. M. Smith (1990). Programming in Martin-Löf's Type Theory, Volume 7 of The International Series of Monographs in Computer Science. Clarendon Press, Oxford.
- O'Hearn, P. W. and J. G. Riecke (1995). Kripke logical relations and PCF. Information and Computation 120(1), 107–116.
- Ong, C.-H. L. (1992, August 7). Concurrent lambda calculus and a general pre-congruence theorem for applicative bisimulation (preliminary version). Unpublished.
- Ong, C.-H. L. (1993). Non-determinism in a functional setting. In Proc. 8th Annual IEEE Symposium on Logic in Computer Science, Montreal.
- Panangaden, P. and J. R. Russell (1989). A category-theoretic semantics for unbounded indeterminacy. In Proc. 5th Conference on Mathematical Foundations of Programming Semantics, New Orleans, Volume 442 of Lecture Notes in Computer Science, pp. 319– 332. Springer-Verlag.
- Park, D. M. (1981). Concurrency and automata on infinite sequences. In P. Deussen (Ed.), Conference on Theoretical Computer Science, Volume 104 of Lecture Notes in Computer Science, pp. 167–183. Springer-Verlag.

- Perez, R. P. (1991). An extensional partial combinatory algebra based on  $\lambda$ -terms. In A. Tarlecki (Ed.), *Proc. Mathematical Foundations of Computer Science*, Volume 520 of *Lecture Notes in Computer Science*, pp. 387–396. Springer-Verlag.
- Pfenning, F. and C. Elliott (1988). Higher-order abstract syntax. In *PLDI'88*, pp. 199–208. ACM.
- Phillips, I. C. C. (1992). Recursion theory. In S. Abramsky et al. (Eds.), Handbook of Logic in Computer Science, Volume 1, pp. 79–187. Oxford University Press.
- Pitts, A. M. (1994a, November). Inductive and co-inductive techniques in the semantics of functional programs. Course held at BRICS, Dept. of Computer Science, Univ. of Aarhus.
- Pitts, A. M. (1994b, December). Some notes on inductive and co-inductive techniques in the semantics of functional programs (draft version). BRICS Notes Series NS-94-5, BRICS, Dept. of Computer Science, Univ. of Aarhus.
- Pitts, A. M. (1995, March). An extension of Howe's construction to yield simulation-upto-context results. Unpublished Manuscript.
- Pitts, A. M. (1997a). Operationally-based theories of program equivalence. In P. Dybjer and A. M. Pitts (Eds.), Semantics and Logics of Computation. Cambridge University Press.
- Pitts, A. M. (1997b). Reasoning about local variables with operationally-based logical relations. In P. W. O'Hearn and R. D. Tennent (Eds.), *Algol-Like Languages*, Volume 2, Chapter 17, pp. 173–193. Birkhauser. Reprinted from *Proceedings Eleventh Annual IEEE Symposium on Logic in Computer Science*, 1996, pp 152–163.
- Pitts, A. M. (1998). Parametric polymorphism and operational equivalence (preliminary version). In A. Gordon, A. Pitts, and C. Talcott (Eds.), Second Workshop on Higher-Order Operational Techniques in Semantics (HOOTS II), Stanford University, December 1997, Volume 10 of Electronic Notes in Theoretical Computer Science. Elsevier.
- Plotkin, G. D. (1975). Call-by-name, call-by-value and the  $\lambda$ -calculus. Theoretical Computer Science 1, 125–159.
- Plotkin, G. D. (1977). LCF considered as a programming language. Theoretical Computer Science 5, 223–255.
- Plotkin, G. D. (1982). A powerdomain for countable non-determinism (extended abstract). In M. Nielsen and E. M. Schmidt (Eds.), Automata, Languages and Programming, 9th Int. Colloquium, Aarhus, Volume 140 of Lecture Notes in Computer Science, pp. 418– 428. Springer-Verlag.
- Plotkin, G. D. (1985, July). Denotational semantics with partial functions. Unpublished lecture notes, CSLI, Stanford University.
- Pratt, V. R. (1992). Origins of the calculus of binary relations. In Proc. 7th Annual IEEE Symposium on Logic in Computer Science, Santa Cruz, CA, pp. 248–254.
- Riecke, J. G. and A. Sandholm (1997). A relational account of call-by-value sequentiality. In Proc. 12th Annual IEEE Symposium on Logic in Computer Science, pp. 258–267.
- Ritter, E. and A. M. Pitts (1995). A fully abstract translation between a  $\lambda$ -calculus with reference types and Standard ML. In *Proc. 2nd International Conference on Typed*

Lambda Calculus and Applications, Edinburgh, Volume 902 of Lecture Notes in Computer Science. Springer-Verlag.

- Rogers, H. (1967). Theory of Recursive Functions and Effective Computability. New York: McGraw-Hill.
- Sabry, A. and P. Wadler (1996). A reflection on call-by-value. ACM SIGPLAN Notices 31(6), 13–24. Proc. ACM SIGPLAN International Conference on Functional Programming, Philadelphia. Extended version appears in ACM Transactions on Programming Languages and Systems 19(6):916–941, 1997.
- Sands, D. (1991). Operational theories of improvement in functional languages (extended abstract). In Proceedings of the Fourth Glasgow Workshop on Functional Programming, Workshops in Computing Series, pp. 298–311. Springer-Verlag.
- Sands, D. (1998a). Computing with contexts: A simple approach. In A. Gordon, A. Pitts, and C. Talcott (Eds.), Second Workshop on Higher-Order Operational Techniques in Semantics (HOOTS II), Stanford University, December 1997, Volume 10 of Electronic Notes in Theoretical Computer Science. Elsevier.
- Sands, D. (1998b). Improvement theory and its applications. See Gordon and Pitts (1998).
- Sangiorgi, D. (1994, August). On the bisimulation proof method. Technical Report LFCS-94-299, University of Edinburgh.
- Schwichtenberg, H. (1996). Finite notations for infinite terms. To appear: APAL, Recursion Theory '96 Volume.
- Schwichtenberg, H. and S. S. Wainer (1995). Ordinal bounds for programs. In P. Clote and J. Remmel (Eds.), *Feasible Mathematics II*, pp. 387–406. Boston: Birkhäuser.
- Sieber, K. (1992). Reasoning about sequential functions via logical relations. In M. P. Fourman, P. T. Johnstone, and A. M. Pitts (Eds.), Proc. LMS Symposium on Applications of Categories in Computer Science, Durham 1991, Volume 177 of LMS Lecture Note Series, pp. 258–269. Cambridge University Press.
- Sieber, K. (1993). Call-by-value and nondeterminism. In M. Bezem and J. F. Groote (Eds.), Proc. International Conference on Typed Lambda Calculus and Applications, Volume 664 of Lecture Notes in Computer Science, pp. 376–390. Springer-Verlag.
- Stoughton, A. (1988). Substitution revisited. *Theoretical Computer Science* 59, 317–325.
- Talcott, C. (1993). A theory of binding structures and applications to rewriting. Theoretical Computer Science 112, 99–143.
- Talcott, C. (1998). Reasoning about functions with effects. See Gordon and Pitts (1998), pp. 347–390.
- Tarski, A. (1941). On the calculus of relations. Journal of Symbolic Logic 6(3), 73–89.
- Ulidowski, I. (1992). Equivalences on observable processes. In Proc. 7th Annual IEEE Symposium on Logic in Computer Science, Santa Cruz, CA, pp. 148–159.
- van Leeuwen, J. (Ed.) (1990). *Handbook of Theoretical Computer Science*, Volume B: Formal Models and Semantics. MIT Press.
- Wadsworth, C. P. (1971, September). Semantics and pragmatics of the lambda calculus. Ph. D. thesis, Programming Research Group, Oxford University.

- Wand, M. and G. T. Sullivan (1997). Denotational semantics using an operationally-based term model. In Proc. 24th ACM Symposium on Principles of Programming Languages, pp. 386–399.
- Wansbrough, K. and J. Hamer (1997). A modular monadic action semantics. In *The Con*ference on Domain-Specific Languages, Proceedings, Santa Barbara, CA, pp. 157–170.
- Winskel, G. (1983, December). Non-deterministic recursive program schemes and powerdomains. Technical Report CMU-CS-83-169, Dept. of Computer Science, Carnegie-Mellon University.
- Winskel, G. (1993). The Formal Semantics of Programming Languages. Cambridge, Mass.: MIT Press.
- Wirsing, M. (1990). Algebraic specification. See van Leeuwen (1990), Chapter 13, pp. 675–788.

# Symbol Index

| ?  |
|--|
| $\emptyset$  |
| •  |
| $\asymp$   |
| $\asymp_{\Diamond} \dots \dots \dots \dots 69$   |
| $\asymp_{\Box} \dots \dots 76$   |
| $\asymp_{R} \dots \dots 99$  |
| $\preceq \dots \dots 39$   |
| $\preceq_{\diamond} \dots \dots \dots \dots 69$  |
| $\preceq_{\Box} \dots \dots 76$  |
| $\preceq_{R} \dots \dots 99$   |
| $\exists_v \dots \dots 86$   |
| $\sim_{\diamondsuit} \dots \dots 91$   |
| $\sim_{\Box} \dots \dots 91$   |
| 2  |
| $\gtrsim \dots \dots \dots \dots \dots 103$  |
| $\cong$  |
| $\cong_{\diamondsuit} \dots \dots$ |
| $\cong_{\Box} \dots 66$  |
| $\sqsubseteq \dots \dots 38$   |
| $\Box_{\diamond} \dots \dots$      |
| $\sqsubset_{\Box}^{\bullet}$   |
| $\lesssim$   |
| $\lesssim_{\diamond} \dots \dots \dots 89$   |
| $\lesssim_{\Box} \dots \dots 90$   |
| $\lesssim_{R} \dots \dots 92$  |
| $\sqrt{a} \dots \dots$             |
| $\varnothing \dots \dots 15$   |
| $(\cdot)^{\S}$ 105   |
| $(\cdot)^{\bullet}$  |
| $(\cdot)^* \dots 22$   |
| $(\cdot)^+$  |
| $(\cdot)^{C}$ 28   |
| $(\cdot)^{\mathrm{op}}$ 22   |
| $(\cdot)^{SC}$   |
| $(\cdot)^n \dots 22$   |
| $(\cdot)^{\circ}$  |

| $(\cdot)_{0}\ldots\ldots23$   |
|---|
| × 15  |
| $+ \dots \dots 15$  |
| <del>.</del>  |
| $\rightarrow$   |
| $\rightarrow$   |
| $\rightarrow_{\diamondsuit}$  |
| $\rightarrow_{\Box}$  |
| $\stackrel{\tau}{\rightarrow}_{\Box}$   |
| $\stackrel{}{\rightarrow}$ 103  |
| $\langle \cdot \rangle \dots 38$  |
| $\langle \cdot \rangle_1 \dots \dots$         |
| $\langle \cdot \rangle_{\diamond}$  |
| $\langle \cdot \rangle_{\Box}$  |
| $\langle \cdot \rangle_{\blacksquare}$  |
| $\langle \cdot \rangle_{R}^{-}$   |
| $\rightarrow$   |
| $ ightarrow_{\diamondsuit} \dots \dots$       |
| $\rightarrowtail_{\Box} \dots \dots$          |
| $\cdot \gg \cdot \dots \dots 25$  |
| $\cdot [\cdot] \dots \dots 24$  |
| $\hat{\cdot}$ 26  |
| $\cdot _{(\vec{t})t}$   |
| $\Downarrow_{\Box} \dots \dots 96$  |
| $\Downarrow^{\alpha}_{\Box} \dots \dots$      |
| ~~13  |
| $\rightsquigarrow_{\diamondsuit} \dots \dots$ |
| ~→□   |
| ~→∎73   |
| <b>0</b> 11   |
|   |
| <i>A</i> 19, 60   |
| $a \dots 9, 59, 79, 95$   |
| <i>a</i> 10   |
| <i>a v</i> 10   |
| $a:t\ldots\ldots 15$  |

| (a;b) 10   |
|--|
| $A[\![\vec{\phi}/\!\vec{\xi}]\!]$                                  |
| $a[\underline{u}/x]$   |
| $a[\vec{u}/\vec{x}]$   |
| <i>ADEQ</i> 30, 38   |
| $ADEQ_{\diamond} \dots \dots 65$                                   |
| $ADEQ_{\Box}$ 65   |
| $\alpha$ 33, 79  |
| $\mathbf{amb}(a_1, a_2) \dots 95$                                  |
| D  |
| <i>B</i>   |
| $b \dots 9, 59, 79, 95$  |
| $\beta$  |
| $\beta_v$ 41, 70, 76   |
| bool1/   |
| <i>C</i>   |
| C[a] 19  |
| $C[\vec{\phi}\vec{k}]$ 19  |
| case $a$ of  |
| $\langle \vec{x} \rangle. b$                                       |
| case $u$ of  |
| $0, a \parallel \mathbf{succ}(x), b \dots 11$                      |
| case $u$ of  |
| $\mathbf{ini}_1 x_1 \cdot a_1$                                     |
| $\ldots$ <b>[</b> ini <sub>n</sub> $x_n$ . $a_n$ 9                 |
| case $u$ of  |
| <b>nil</b> . $a \mid \mathbf{cons}\langle x, y \rangle$ . $b . 11$ |
| case $u$ of  |
| $\langle x_1,\ldots,x_n\rangle.a\ldots\ldots9$                     |
| $\chi$   |
| $\widehat{\operatorname{cons}}\langle u,v\rangle\dots\dots\dots11$ |
|  |
| <i>E</i> 18  |
| <i>E</i>   |
| $E[\![a]\!]$   |
| <b>eq</b> 68   |

| $\eta_v \ldots \ldots 53, 71$  |
|--|
| $Exp_0 \dots \dots \dots 10$   |
| $Exp_{\vec{x}}$ 10   |
| f 10   |
| falso 11   |
| fix 11 10  |
| $\mathbf{f}_{\mathbf{r}}^{(\alpha)}$   |
| $\prod_{n \in \mathbb{N}} u \dots \dots$ |
| TIX <sup>(1)</sup> 4(  |
| <i>g</i>   |
| $\Gamma$ 15  |
| $\gamma$   |
| $\Gamma \subseteq \Gamma' \dots \dots 15$  |
| $\Gamma \vdash a \ R \ a' : t \dots 22$  |
| $\Gamma \vdash a: t \dots 15$  |
| <i>h</i> 10  |
| <b>I</b>   |
| <i>Id</i> 22   |
| if a then $b_1$ else $b_2 \dots 11$  |
| <b>inj</b> <i>u</i> 11   |
| $inj_i u \dots $                   |
| <i>ι</i>   |
| λ 79   |
| $\lambda$ ini $r$ $a$ 11   |
| $\lambda r a \qquad 9$   |
| $\lambda \langle r_1, r_n \rangle a = 11$  |
| let $x = a$ in $b$   |
| list $17$  |
| 1150   |
| $\mu\chi.t_1+\ldots+t_n\ldots\ldots 15$  |
| nat17  |
| <b>nil</b> 11  |
| _  |

| $\omega_1^{CK}$                            |
|--|
| $a_1$ or $a_2$                             |
|  |
| $\phi$ 19                                  |
| $\phi:\theta\ldots\ldots\ldots19$          |
| <i>PREADEQ</i> 38                          |
| $PREADEQ_{\diamond} \dots \dots 65$        |
| $PREADEQ_{\Box} \dots \dots 65$            |
|  |
| $\boldsymbol{R}$                           |
| <i>R</i>                                   |
| $\mathbf{R}^{\S}$ 105                      |
| $R^{\bullet}$                              |
| $\mathcal{R}^* \dots \dots 22$             |
| $\mathcal{R}^+$                            |
| $\overline{R}$                             |
| <i>R</i> <sub>0</sub> 23                   |
| $R[S] \dots 24$                            |
| $\widehat{R}$                              |
| $R^{C}$                                    |
| $R_2^{(\gamma)}$                           |
| $R^{(i)}$                                  |
| $\mathcal{R}^n$                            |
| $R^{\circ}$                                |
| $\mathcal{R}^{\mathrm{op}}$                |
| B <sup>SC</sup>                            |
| R  |
| $\mathcal{R}S$ 22                          |
| <i>BEL</i> 22                              |
| Rel 22                                     |
| $REL(\alpha)$ 33                           |
| $\frac{Rel(\alpha)}{33}$                   |
| $\begin{array}{c} REL_{2} \\ \end{array} $ |
| 111110                                     |
| <b>S</b> 22                                |
| <i>S</i> 22                                |
| $S \gg R \dots 25$                         |
| succ <i>u</i>                              |

| T49, 93  |
|--|
| $(\vec{t})t$   |
| $t_1 \rightharpoonup t_2 \dots \dots \dots 15$   |
| $t_1 \times \ldots \times t_n \ldots \ldots 15$  |
| $t[t'/\chi]$   |
| $\theta$   |
| true 11  |
| $Type_0 \dots \dots 15$  |
| <i>U</i> 19.60   |
| U 44   |
| <i>u</i>   |
| $\langle u_1, \ldots, u_n \rangle \ldots \ldots 9$   |
| $\vec{u}$  |
| <i>u v</i>   |
| unit   |
|  |
| V 60   |
| <i>v</i>   |
| $Val_0$ 10<br>$Val_1$  |
| $Val_{\vec{x}} \dots $                           |
| V0Iu10   |
| $W_{2}^{(\alpha)}$   |
| $W^{(n)}$  |
| 10   |
| x10  |
| $x \dots \dots$                                  |
| (x)a19<br>$((\vec{x})a)$ 33  |
| $((x_{\iota})a_{\iota})_{\iota < \alpha} \dots $ |
| $\vec{x} \cdot \vec{t}$ 15   |
| ε  |
| ç  |
| <b>Y</b>   |
| <i>y</i> 10  |
| $\mathbf{Y}^{(n)}$   |
| ~ 10   |
| <i>~</i>   |

### **Recent BRICS Dissertation Series Publications**

- DS-98-2 Søren B. Lassen. *Relational Reasoning about Functions and Nondeterminism.* December 1998. PhD thesis. x+126 pp.
- DS-98-1 Ole I. Hougaard. *The CLP(OIH) Language*. February 1998. PhD thesis. xii+187 pp.
- DS-97-3 Thore Husfeldt. *Dynamic Computation*. December 1997. PhD thesis. 90 pp.
- DS-97-2 Peter Ørbæk. *Trust and Dependence Analysis*. July 1997. PhD thesis. x+175 pp.
- DS-97-1 Gerth Stølting Brodal. Worst Case Efficient Data Structures. January 1997. PhD thesis. x+121 pp.
- DS-96-4 Torben Braüner. An Axiomatic Approach to Adequacy. November 1996. Ph.D. thesis. 168 pp.
- DS-96-3 Lars Arge. *Efficient External-Memory Data Structures and Applications*. August 1996. Ph.D. thesis. xii+169 pp.
- DS-96-2 Allan Cheng. *Reasoning About Concurrent Computational Systems*. August 1996. Ph.D. thesis. xiv+229 pp.
- DS-96-1 Urban Engberg. Reasoning in the Temporal Logic of Actions The design and implementation of an interactive computer system. August 1996. Ph.D. thesis. xvi+222 pp.