BOSTON UNIVERSITY

GRADUATE SCHOOL OF ARTS AND SCIENCES

Dissertation

COMPUTATIONAL SOUNDNESS OF NON-CONFLUENT CALCULI WITH APPLICATIONS TO MODULES AND LINKING

by

ELENA L. MACHKASOVA

Engineer-Mathematician, Moscow Oil and Gas Institute, 1989

Submitted in partial fulfillment of the

requirements for the degree of

Doctor of Philosophy

2002

Approved by

First Reader

Assaf Kfoury, Ph.D.
Professor of Computer Science
Boston University

Second Reader

Franklyn Turbak, Ph.D.
Assistant Professor of Computer Science
Wellesley College

Third Reader

Walid Taha, Ph.D.
Research Scientist in Computer Science
Yale University

Acknowledgments

I would like to thank Franklyn Turbak for many hours of helpful discussion of this work, for the careful reading of numerous drafts of this document, for the tremendous amount of very valuable comments, suggestions and advice, and for all his help during many years of joint research.

Many thanks to my advisor Assaf Kfoury for helpful guidance and encouragement during the work on the thesis and throughout my studies at Boston University and for helpful comments on this work.

Many thanks also to Walid Taha for numerous detailed and insightful comments on several drafts of this work and for many valuable discussions.

Thanks are also due to Joe Wells for several helpful conversations about various aspects of the technique proposed in this work, and to other members of the Church project for interesting questions and important suggestions during several presentations of early versions of this work at the seminar series and for their friendly support.

Many thanks to my colleagues and students at Wellesley college for their support and understanding. The last, but by no means the least, I would like to thank my husband Joshua and my sons Yuri and Lenny for their help, patience, and understanding during this work, and Carole and Melvin Scott for being such wonderful grandparents to Yuri and Lenny and such great friends to me.

Without all this help and support this work would not have been possible.

COMPUTATIONAL SOUNDNESS OF NON-CONFLUENT CALCULI WITH APPLICATIONS TO MODULES AND LINKING

(Order No.)

ELENA MACHKASOVA

Boston University Graduate School of Arts and Sciences, 2002

Major Professor: Assaf Kfoury, Professor of Computer Science

ABSTRACT

A traditional formal framework for reasoning about program semantics is given by calculi, starting from the lambda-calculus developed by Alonzo Church. In this framework, we propose a technique for proving computational soundness of non-confluent calculi. Traditionally, there are two ways of reasoning about a program: a small step operational semantics that defines the meaning of a program and a calculus that describes program transformations. Computational soundness connects these two; it states that any transformation expressible in the calculus preserves meaning with respect to the small-step operational semantics.

The usual technique for proving computational soundness is based on two well-known properties of a calculus: confluence and standardization. However, this approach is not applicable to many calculi because of lack of confluence.

The core of the thesis is the presentation of a new technique for proving computational soundness which does not require confluence of the calculus. The technique is based on two properties, *lift* and *project*, introduced in Machkasova and Turbak "A Calculus for Link-time Compilation". The new approach does not require other properties commonly used in computational soundness proofs, such as left-linearity and finiteness of developments.

As an application of the new technique, we propose and study a calculus of records with mutually recursive components. Such records are used as a model for programs constructed of separate fragments, or modules. We use this model to show meaning preservation of program transformations, in particular of cross-module transformations. The calculus of records lacks confluence, left-linearity, and finiteness of developments. However, we are able to prove that it satisfies the lift and project properties, and therefore is computationally sound.

A linking calculus augments the calculus of records with several link-time operations on modules, such as combining two modules or renaming a component of a module. We show that it inherits computational soundness from the calculus of records.

Contents

Acknowledgments **iii**

Abstract **iv**

Contents **viii**

List of Figures **xiii**

1 Introduction **1**

1.1 Program Transformations 1

1.1.1 Examples of Program Transformations 3

1.1.2 Meaning Preservation, Computational Soundness 4

1.1.3 Non-confluent Calculi 4

1.1.4 Cross-Module Transformations 5

1.2 Plan of the Dissertation 7

2 Definitions and Overview of Calculi **9**

2.1 Notations; Main Calculus Relations 9

2.1.1 Mathematical Preliminaries 9

2.1.2 Calculus Relations 10

2.2 Definitions of Calculi 11

2.2.1 Call-by-value λ-calculus 11
2.2.2 Call-by-name λ-calculus 13
2.2.3 Calculus of Records 14

3 Computational Soundness **18**

3.1 Classification and Class Preservation 18
3.2 Computational and Observational Soundness 23
3.2.1 Computational Soundness Property 23
3.2.2 Compatible Closure, Observational Soundness 23
3.3 A Classical Technique for Proving Computational Soundness 25
3.3.1 Confluence 25
3.3.2 Standardization 26
3.3.3 Computational Soundness of a Confluent Calculus 27
3.4 Our Technique (High-level View) 29
3.4.1 Lift and Project Properties 29
3.4.2 Computational Soundness of a Non-confluent Calculus 33
3.4.3 Lift is Equivalent to Standardization 35
3.4.4 Weakening Project: Confluence w.r.t. Evaluation 36
3.5 Extension: Handling Evaluation Relations 39
3.5.1 Abstracting Over Order and Names of Components 39
3.5.2 Properties of an Evaluation Relation 42
3.5.3 Computational Soundness when Evaluation is not a Function . 44

4 Techniques for Proving Lift and Project **46**

4.1 Challenges in Proving Lift and Project 46
4.2 Marked Calculi 48
4.2.1 Redexes and Residuals 49

4.2.2 Reductions with Marked Redexes 52
4.2.3 Developments 55
4.3 The "Good Case" Scenario: λ-calculi 57
4.3.1 Parallel Moves Lemma and Confluence 58
4.3.2 Standardization of Developments 60
4.3.3 Lift and Project in the Call-by-value λ-calculus 62
4.4 Problems with the Calculus of Records 64
4.5 γ-Developments 68
4.6 Elementary Lift and Project Diagrams 71
4.7 Properties of $\Longrightarrow$ 74
4.8 Proof of Lift and Project 76
4.9 Properties Needed to Prove Lift and Project 81

5 Applications and Results **82**

5.1 Soundness of the Term Calculus 82
5.1.1 Notations and Definitions 83
5.1.2 Developments are Bound 85
5.1.3 Parallel Moves Lemma 88
5.1.4 Confluence of $\mathcal{T}$ 95
5.1.5 Class Preservation and Standardization in $\mathcal{T}$ 97
5.1.6 Computational Soundness of $\mathcal{T}$ 104
5.2 Soundness of the Calculus of Records 104
5.2.1 Definitions 105
5.2.2 γ-developments in Calculus of Records 110
5.2.3 Boundedness of γ-developments 118
5.2.4 Confluence and γ-confluence of $\Longrightarrow_{\mathcal{C}}$ 121

5.2.5 Elementary Lift and Project Diagrams in Calculus of Records 138
5.2.6 Weak Standardization of γ-developments in $\mathcal{C}$ 172
5.2.7 Class Preservation and Results 174

6 Overview of the Linking Calculus 175
6.1 Module and Linking Calculi 175
6.2 Syntax of the Linking Calculus 177
6.3 Linking Relations 180
6.4 Overview of α-renaming 183
6.5 Results 185
6.6 Example of Cross-Module Transformation 186

7 Related Work 188
7.1 Related Work on Computational Soundness 188
7.2 Related Proof Techniques 189
7.3 Module Calculi and Recursive Systems 192

8 Summary and Future Work 194
8.1 Summary 194
8.2 Future Work 195
8.2.1 Applications to Other Non-confluent Calculi 195
8.2.2 Integrating Program Analyses 196
8.2.3 Proving Other Cross-Module Transformations 196
8.2.4 Extending the Calculus 197
8.2.5 Work on Classification 198
8.2.6 Evaluation vs. Non-evaluation Steps 198

Bibliography 198

Curriculum Vitae **204**

List of Figures

2.1 Call-by-value λ-calculus 11

2.2 Call-by-name λ-calculus 13

2.3 Term calculus for the calculus of records 14

2.4 Calculus of records 15

3.1 Confluence and standardization. 26

3.2 Sketch of the traditional proof of computational soundness. 28

3.3 The lift and project properties. 29

3.4 Example of lift and project: one-step evaluation sequence. 30

3.5 Example of lift and project: two-step evaluation sequence. 31

3.6 Proof of soundness via lift and project. 33

3.7 Confluence w.r.t. evaluation. 36

3.8 Proof of lemma 3.4.10. 38

3.9 Divergence does not contradict existence of a normal form. 43

3.10 Additional cases when evaluation is not a function. 44

4.1 Example of the parallel moves property 58

4.2 Lift and project in marked call-by-value λ-calculus. 63

4.3 Elementary diagrams in call-by-value λ-calculus 64

4.4 Lift tiling diagram in call-by-value λ-calculus 65

4.5 Proof of lift in call-by-value λ-calculus 65
4.6 Parallel moves lemma fails 65
4.7 An attempt to adapt parallel moves lemma 66
4.8 Example of non-confluence in marked calculus of records. 67
4.9 Elementary project and lift diagrams 72
4.10 Inductive step of proofs of lemmas 4.6.3–4.6.4 73
4.11 γ-confluence of evaluation 75
4.12 Inductive steps of proof of lemma 4.7.2 75
4.13 Lift and project in the "marked" calculus 77
4.14 Inductive step of proof of theorem 4.8.4 78
4.15 Inductive step of proof of theorem 4.8.5 (2 cases). 79

6.1 The linking calculus 178
6.2 A sequence of calculus steps proving that lambda-splitting is meaning preserving in $\mathcal{L}_{GC}$. 187

8.1 Labels as "values": non-confluence. 197

Chapter 1

Introduction

1.1 Program Transformations

Since the early days of programming there have always been tradeoffs between clarity and efficiency of programs. The requirements of one are often practically the opposite of the requirements of the other! For instance, to make a program understandable for a human being, one tends to define many short functions rather than a few long ones. However, run-time overhead of a function call makes such a program inefficient. Similarly, a program written in a good programming style has a separate variable for each entity it uses, but efficiency requires that variables are reused. Another part of the equation is how general the program should be. Handling only specific cases leads to efficiency, but requires program rewriting if a slightly different case needs to be handled.

In short, tricks used to make a program more efficient often result in awkward code which is hard to read and practically impossible to modify. On the other hand, certain elegant programming solutions, such as using recursion, using a lot of short functions, using higher-order functions, and using intermediate data structures, such

as lists and trees, may result in an inefficient code.

It is true that choosing a good algorithm for large programs is an extremely important factor in writing efficient software. For instance, if one can replace an algorithm which is quadratic with respect to the size of the input by a linear one, the gain is clearly unmatched by small scale optimizations of code. But suppose an algorithm is chosen, and we need to write a program that implements it. Do we have to deal with tradeoffs between a good programming style and a good performance? Or can we have the best of both worlds: write clear readable programs and at the same time be sure that they are fast?

Actually, to a large extent we can! There are several factors in modern technology that allow programmers to focus on clarity and good style of code without worrying about small-scale inefficiencies that might result from it:

- The increase in speed of modern computers makes small-scale differences in running time of a program practically unnoticeable. However, in real-time and multi-threaded environment even minor inefficiencies may become significant.

- The second factor that bridges the two goals is the use of optimizing compilers. Such compilers take code which is readable for a human being, but not necessarily the most efficient, and perform program transformations to guarantee the efficiency of the program. For instance, functions may be inlined to decrease the function call overhead, references to variables whose values are constant may be replaced by the values themselves, a function general enough to handle many cases may be specialized based on its uses in the program, and complex intermediate data structures may be removed. The resulting executable recovers the efficiency of the program (and in many cases improves it) without sacrificing the clarity of the original program.

This work belongs to the realm of the second approach: it deals with program transformations, in particular with proving that transformations preserve the meaning of the program. Before we look more closely at the way of defining the meaning of a program, let us look at different kinds of program transformations.

1.1.1 Examples of Program Transformations

Traditionally, compiler writers distinguish between local and global transformations.

Local transformations are performed based on analysis of a small code fragment, such as a basic block. Numerous local transformations are performed by modern compilers. Some of the most common ones are constant propagation, constant folding, function inlining, and various optimizations related to loops, such as loop unrolling, etc. Many local transformations can be thought of as program evaluation steps performed at compile time.

Global transformations are based on the analysis of the entire program and often transform the entire program in non-trivial ways. They include, among others, closure conversion, function specialization, assignment conversion, uncurrying. Some of them require quite sophisticated program analyses, such as flow analysis.

The applications of our technique described in this work prove meaning-preservation of certain local transformations. However, it is also possible to handle a version of garbage collection in this framework, see [MT02]. Garbage collection is a border-line transformation, since it requires the analysis of the entire program (in our case, of the entire module), but it is a simple transformation which can be performed by a program rewrite step. More complex global transformations, such as closure conversion, which transforms higher-order functions into pairs of a first-order function and a data structure representing the environment, require proof methods

of a different nature. Reasoning about such transformations is beyond the scope of this work.

1.1.2 Meaning Preservation, Computational Soundness

Following a traditional approach of [Plo75, Bar84], we define the meaning of a program via small-step operational semantics given by the evaluation relation of the calculus. The terms are classified with respect to the evaluation relation: a term is called evaluatable if it can be evaluated further, otherwise it is an evaluation normal form. Evaluation normal forms are further classified based on their observable properties. The meaning of a term is defined via the notion of an outcome: an outcome of a term is the class of its evaluation normal form if it is eventually reached, otherwise we say that the term diverges.

In addition to the evaluation relation we define calculus rewrite relation (or just *calculus relation*), which describes program transformations. The relation subsumes the evaluation relation.

Computational soundness is a property that relates the calculus rewrite relation and the evaluation relation. The property states that a calculus rewrite step does not change the meaning (i.e. the outcome) of the term. If computational soundness holds, then any transformation expressible as a sequence of forward and backward calculus steps is meaning-preserving. Therefore proving computational soundness of a calculus immediately justifies a wide variety of local transformations.

1.1.3 Non-confluent Calculi

Traditionally, computational soundness is proven via two well-known calculus properties: confluence of the calculus relation and standardization.

However, many calculi which formalize interesting features of the language lack confluence. Records with mutually recursive components (such as the calculus introduced in section 2.2.3), calculi with letrec, as in [AK97], calculi which formalize references and state , and certain calculi with explicit substitution, such as [DL01] serve as examples. Since confluence is a traditional ingredient of proofs of computational soundness, the usual technique for computational soundness proofs is not applicable to calculi which lack confluence.

The main contribution of this work is the development of a new technique for proving computational soundness that does not require confluence. The new technique is based on two properties which will call *lift* and *project*. We introduce these properties and show that their combination implies computational soundness in chapter 3. Chapter 4 gives a general proof of lift and project.

Our main example of a non-confluent calculus is a calculus of recursively scoped records whose components are labeled terms of a call-by-value calculus. Performing a substitution in a record with two mutually recursive components provides an example of non-confluence (a similar example was given in [AK97] for a cyclic calculus).

We apply the new technique to the calculus of recursively scoped records in chapter 5 and show that the calculus enjoys the computational soundness property despite the lack of confluence.

1.1.4 Cross-Module Transformations

The calculus of recursively scoped records serves as a model for studying modules. We extend the calculus of records with a separate kind of labels – hidden labels – which represent private components of a module. We add the garbage collection operation

on modules to be able to remove private components which are not referenced in the module.

In order to model programs consisting of multiple modules we define a linking calculus on top of the module calculus with the following operations: linking two modules, renaming a component of a module, hiding a component, and naming module via link-level *let* to facilitate module reuse.

The goal of developing the linking calculus is to be able to represent certain kinds of cross-module transformation as calculus steps and to justify such transformations via computational soundness of the calculus. Without going into details of the definition of the linking calculus, we show an example of a cross-module transformation which is represented as a sequence of calculus steps in the linking calculus, and therefore is meaning-preserving.

The transformation that we consider is *cross-module lambda-splitting*, a transformation described by Blume and Appel in [BA97] and used in the SML/NJ compiler. Suppose a module exports a function named F which may be used in one or more other modules. We would like to inline F into a module that uses it, but it may be the case that the definition of F is too large for inlining to be efficient. (If we consider a calculus with side effects, then side effects may also prevent F from being inlined.). The proposed solution is to extract from F the expensive part of its definition, add it to the module as a separately named component F_{exp}, and fill the former location of the expensive part by a reference to the name F_{exp}. This transformation makes F cheap enough to inline into other modules.

As a concrete example of this technique, consider the following module expression written in the syntax of our calculus:

$$[F \mapsto \lambda x.\mathbb{C}\{\lambda y.M'\}] \oplus [X \mapsto \mathbb{A}\{F \mathbin{@} N\}].$$

Here, $\mathbb{C}$ and $\mathbb{A}$ are expression contexts – expressions with single holes that are filled using the squiggly bracket notation. Assume that the expensive part of the definition of F is $\lambda y.M'$, which is assumed to be a closed abstraction (i.e., it has no free variables). Also assume that the name F_{exp} does not appear free in $\mathbb{C}\{\lambda y.M'\}$ or in $\mathbb{A}\{F \mathbin{@} N\}$. Then we can extract the expensive part as a separate module component bound to F_{exp} and inline F in the second module to yield:

$$([F \mapsto \lambda x.\mathbb{C}\{F_{exp}\}, F_{exp} \mapsto \lambda y.M'] \oplus [X \mapsto \mathbb{A}\{\lambda x.\mathbb{C}\{F_{exp}\} \mathbin{@} N\}])\{\text{hide } F_{exp}\}.$$

The operator $L\{\text{hide } v\}$ makes a label v exported by expression L inaccessible to the "outside world". In the above example, it is used to hide the name F_{exp}, which should not be observable outside of the transformed expression. If it were, then the transformed expression would not be observationally equivalent to the original one, because a context that uses F_{exp} would distinguish the original and final expressions.

Cross-module lambda-splitting can be expressed as a sequence of calculus steps in the linking calculus. Since the linking calculus enjoys computational soundness property, the transformation is meaning preserving.

Computational soundness of the module calculus (i.e. the calculus of records with hidden labels and garbage collection) and of the linking calculus is proven in [MT02].

1.2 Plan of the Dissertation

The rest of this disseration is organized as follows:

- Chapter 2 introduces some notations used in the rest of the thesis and defines the calculi that we consider in this work: the call-by-value and the call-by-name

λ-calculi and the calculus of records.

- Chapter 3 gives an overview of the traditional technique for proving computational soundness and introduces the new properties, lift and project. We also discuss the case when the evaluation is not a function (as in the calculus of records), and show modifications of the proofs in this case.

- Chapter 4 gives a general proof of lift and project in two cases: the case when developments have such properties as boundedness, confluence, and standardization (the case of the call-by-value and the call-by-name λ-calculi), and the case when the developments do not have these properties (the case of the calculus of records). We introduce the notion of γ-developments to handle the latter case, and use it to prove lift and project.

- Chapter 5 gives the proofs of computational soundness of the term and the module calculi. We give the sketch of the proof of computational soundness of the linking calculus. The technical details of the proof are given in [MT02]

- Chapter 6 gives the overview of the linking calculus, demonstrates examples, and states the main results.

- Chapters 7 and 8 discuss the related work, summarize the work done in the dissertation and in the related technical report [MT02], and outline directions for future work.

Chapter 2

Definitions and Overview of Calculi

2.1 Notations; Main Calculus Relations

In this section we define notations and terminology used in the rest of this work. We also give some important examples of calculi to which we refer throughout this presentation. Definitions in this section are somewhat informal and refer to commonly known notions, such as relation, function, reflexivity, and others.

2.1.1 Mathematical Preliminaries

A *binary relation* R over sets S and T is a subset of $S \times T$, and the notation $x\ R\ y$ means $(x, y) \in R$. We say x is in the *domain* of R, written $x \in dom(R)$, if and only if there exists a y such that $x\ R\ y$.

We use $R^?$ to stand for the reflexive closure of a binary relation R, R^+ to stand for the transitive closure of R, R^* to stand for the reflexive, transitive closure of R, and $R^=$ to stand for the reflexive, symmetric, and transitive closure of R. We

sometimes use the abbreviation = for $R^{=}$. For a relation written as an arrow, $\rightarrow$, we sometimes use $\leftrightarrow$ as a synonym for $\rightarrow^{=}$.

Let X be a set, and let $\perp$ denote an element such that $\perp \notin X$. A sequence of elements of X is defined as follows:

Definition 2.1.1 (Sequences). Let $\mathbb{P}$ stand for the positive integers $\{1, 2, \ldots\}$. A *sequence* S over some set X is a function from $\mathbb{P}$ to $X \cup \{\perp\}$ such that if $S(i) \neq \perp$ and $j \leq i$ then $S(j) \neq \perp$.

Given a sequence S, let the length of S be 0 if $S(1) = \perp$, or the largest integer i such that $S(i) \neq \perp$, or ω if $S(i) \neq \perp$ for all $i \in \mathbb{N}$. Let $x_1, \ldots, x_n$ denote the sequence S such that $S(i) = x_i$ for $i \leq n$ and $S(i) = \perp_X$ otherwise. We use the symbol ϵ to denote a sequence of length 0, and use $S_1; S_2$ for concatenation of two sequences. By definition $\epsilon; S = S; \epsilon = S$. □

2.1.2 Calculus Relations

A small-step operational semantics of a calculus is defined via an evaluation step relation $\Longrightarrow$. In order to reason about program transformation, we extend $\Longrightarrow$ to a calculus relation $\longrightarrow$. Sometimes we refer to this relation as calculus rewrite relation. The evaluation relation is contained in the calculus relation ($\Longrightarrow \subset \longrightarrow$), and we define a non-evaluation relation as $\circ\!\!\longrightarrow = \longrightarrow \setminus \Longrightarrow$ (here $\setminus$ stands for set difference).

We use M, N, and L to range over terms of a calculus. **Term** denotes the set of all terms in the calculus. We define a set of terms $NF_{\Longrightarrow}$, called *evaluation normal forms*, as follows: $M \in NF_{\Longrightarrow}$ if and only if there is no term N such that $M \Longrightarrow N$. Similarly, the set $NF_{\longrightarrow}$ called *calculus normal forms* is defined the following way: $M \in NF_{\longrightarrow}$ if and only if there is no N such that $M \longrightarrow N$.

2.2 Definitions of Calculi

In this section we define three calculi that we will use throughout this presentation: a call-by-value λ-calculus with constants, a call-by-name λ-calculus with constants, and our main application: a calculus of records with recursive components.

2.2.1 Call-by-value λ-calculus

$$M, N, L \in \textbf{Term} \ ::= \ c \mid x \mid (\lambda x.M) \mid M_1 \ @ \ M_2 \mid M_1 \ op \ M_2$$

$$V \in \textbf{Value} \ ::= \ c \mid x \mid \lambda x.M$$

$$\mathbb{C} \in \textbf{Context} \ ::= \ \Box \mid (\lambda x.\mathbb{C}) \mid \mathbb{C} \ @ \ M \mid M \ @ \ \mathbb{C} \mid \mathbb{C} \ op \ M \mid M \ op \ \mathbb{C}$$

$$\mathbb{E} \in \textbf{EvalContext} \ ::= \ \Box \mid \mathbb{E} \ @ \ M \mid (\lambda x.M) \ @ \ \mathbb{E} \mid \mathbb{E} \ op \ M \mid c \ op \ \mathbb{E}$$

$$(\lambda x.M) \ @ \ V \quad \rightsquigarrow_v \quad M[x := V]$$

$$c_1 \ op \ c_2 \quad \rightsquigarrow_v \quad c, \text{ where } c = \delta(op, c_1, c_2) \quad \text{if } \delta(op, c_1, c_2) \text{ is defined,}$$

$$\mathbb{E}\{R\} \Rightarrow_v \mathbb{E}\{Q\}, \text{ where } R \rightsquigarrow_v Q,$$

$$\mathbb{C}\{R\} \rightarrow_v \mathbb{C}\{Q\}, \text{ where } R \rightsquigarrow_v Q.$$

Figure 2.1: Call-by-value λ-calculus

As an example, consider the call-by-value λ-calculus with constants. The calculus is defined on figure 2.1. The calculus is traditional (see [Plo75]). However, unlike [Plo75], we use evaluation contexts (a notion introduced in [FF86]) to define an evaluation step.

Definitions of terms and values are as usual. $\Rightarrow_v$ and $\rightarrow_v$ denote the evaluation step and the calculus reduction step, respectively, of this calculus. In order to define $\Rightarrow_v$ and $\rightarrow_v$, we define two kinds of contexts: a general context $\mathbb{C}$ and an evaluation context $\mathbb{E}$. $\mathbb{C}\{M\}$ denotes the result of filling the hole $\Box$ in a context $\mathbb{C}$ with a term M. The same notation is used for evaluation contexts $\mathbb{E}$.

We define a so-called notion of reduction $\rightsquigarrow_v$, which is used to define $\Rightarrow_v$ and $\rightarrow_v$. Using $\rightsquigarrow_v$ and the two kinds of contexts, we define $\Rightarrow_v$ and $\rightarrow_v$.

In the further discussion we omit the subscripts of the relations of a calculus when they are clear from context. For instance we write $\rightarrow$ instead of $\rightarrow_v$ whenever it is unambiguous which calculus the relation belongs to.

According to the rules in figure 2.1, the following are evaluation steps:

$$\begin{aligned}
(\lambda x.x) \mathbin{@} (\lambda y.y) &\Rightarrow \lambda y.y, \\
(\lambda x.x) \mathbin{@} (2+3) &\Rightarrow (\lambda x.x) \mathbin{@} 5, \\
((\lambda x.x) \mathbin{@} (\lambda y.y)) \mathbin{@} ((\lambda z.z) \mathbin{@} (\lambda w.w)) &\Rightarrow (\lambda y.y) \mathbin{@} ((\lambda z.z) \mathbin{@} (\lambda w.w)).
\end{aligned}$$

These are non-evaluation steps:

$$\begin{aligned}
(\lambda x.2+3) \mathbin{@} (\lambda y.y) &\circ\!\!\rightarrow (\lambda x.5) \mathbin{@} (\lambda y.y), \\
((\lambda x.x) \mathbin{@} (\lambda y.y)) \mathbin{@} ((\lambda z.z) \mathbin{@} (\lambda w.w)) &\circ\!\!\rightarrow ((\lambda x.x) \mathbin{@} (\lambda y.y)) \mathbin{@} (\lambda w.w).
\end{aligned}$$

Both evaluation and non-evaluation steps are calculus reduction steps, so each of the four reductions above is an instance of $\rightarrow$.

Both an evaluation step and a calculus step are defined by specifying a subterm that gets reduced (denoted by R in the rules) and the context which surrounds the subterm. The context is not changed by the reduction. The term that gets reduced is refered to as a *redex*. A precise definition of a redex is calculus-dependent. For instance, in the call-by-value calculus defined above, redexes are terms of the form $(\lambda x.M) \mathbin{@} V$ and $c_1\ op\ c_2$ if $\delta(op, c_1, c_2)$ is defined. Section 4.2.2 gives a more formal discussion of redexes.

$M, N, L \in \textbf{Term} \ ::= \ c \mid x \mid (\lambda x.M) \mid M_1 \ @ \ M_2 \mid M_1 \ op \ M_2$

$V \in \textbf{Value} \ ::= \ c \mid x \mid \lambda x.M$

$\mathbb{C} \in \textbf{Context} \ ::= \ \Box \mid (\lambda x.\mathbb{C}) \mid \mathbb{C} \ @ \ M \mid M \ @ \ \mathbb{C} \mid \mathbb{C} \ op \ M \mid M \ op \ \mathbb{C}$

$\mathbb{E} \in \textbf{EvalContext} \ ::= \ \Box \mid \mathbb{E} \ @ \ M \mid \mathbb{E} \ op \ M \mid c \ op \ \mathbb{E}$

$(\lambda x.M) \ @ \ N \quad \rightsquigarrow_n \quad M[x := N]$

$c_1 \ op \ c_2 \quad \rightsquigarrow_n \quad c$, where $c = \delta(op, c_1, c_2) \quad$ if $\delta(op, c_1, c_2)$ is defined,

$\mathbb{E}\{R\} \Longrightarrow_n \mathbb{E}\{Q\}$, where $R \rightsquigarrow_n Q$,

$\mathbb{C}\{R\} \longrightarrow_n \mathbb{C}\{Q\}$, where $R \rightsquigarrow_n Q$.

Figure 2.2: Call-by-name λ-calculus

2.2.2 Call-by-name λ-calculus

The call-by-name λ-calculus with constants is defined in figure 2.2. It is also a traditional calculus defined in [Plo75].

The terms, values, and general contexts $\mathbb{C}$ in this calculus are the same as in the call-by-value one. However, the notion of reduction and evaluation contexts reflect the call-by-name nature of the calculus. We use the subscript n to denote relations in the call-by-name calculus. As before, we omit the subscript whenever it is unambiguous.

Examples of evaluation steps in the call-by-name calculus:

$$\begin{aligned} (\lambda x.x) \ @ \ (\lambda y.y) \quad &\Longrightarrow \quad \lambda y.y, \\ (\lambda x.x) \ @ \ (2 + 3) \quad &\Longrightarrow \quad 2 + 3, \\ ((\lambda x.x) \ @ \ (\lambda y.y)) \ @ \ ((\lambda z.z) \ @ \ (\lambda w.w)) \quad &\Longrightarrow \quad (\lambda y.y) \ @ \ ((\lambda z.z) \ @ \ (\lambda w.w)). \end{aligned}$$

Examples of non-evaluation steps:

$$(\lambda x.x) \mathbin{@} (2+3) \quad \circ\!\!\longrightarrow \quad (\lambda x.x) \mathbin{@} 5,$$
$$((\lambda x.x) \mathbin{@} (\lambda y.y)) \mathbin{@} ((\lambda z.z) \mathbin{@} (\lambda w.w)) \quad \circ\!\!\longrightarrow \quad ((\lambda x.x) \mathbin{@} (\lambda y.y)) \mathbin{@} (\lambda w.w).$$

2.2.3 Calculus of Records

A calculus of records is defined over a calculus of terms (see figure 2.3): a call-by-value calculus with constants and labels. Labels in terms, denoted l, are used to refer to another component of the record; they do not affect the reductions in the term calculus itself. Note that labels are not considered values. We use subscript $\mathcal{T}$ for relations in the term calculus.

$M, N, L \in \textbf{Term} \ ::= \ c \mid x \mid l \mid (\lambda x.M) \mid M_1 \mathbin{@} M_2 \mid M_1 \ op \ M_2$

$V \in \textbf{Value} \ ::= \ c \mid x \mid \lambda x.M$

$\mathbb{C} \in \textbf{Context} \ ::= \ \Box \mid (\lambda x.\mathbb{C}) \mid \mathbb{C} \mathbin{@} M \mid M \mathbin{@} \mathbb{C} \mid \mathbb{C} \ op \ M \mid M \ op \ \mathbb{C}$

$\mathbb{E} \in \textbf{EvalContext} \ ::= \ \Box \mid \mathbb{E} \mathbin{@} M \mid (\lambda x.M) \mathbin{@} \mathbb{E} \mid \mathbb{E} \ op \ M \mid c \ op \ \mathbb{E}$

$(\lambda x.M) \mathbin{@} V \quad \rightsquigarrow_{\mathcal{T}} \quad M[x := V]$

$c_1 \ op \ c_2 \quad \rightsquigarrow_{\mathcal{T}} \quad c$, where $c = \delta(op, c_1, c_2)$ if $\delta(op, c_1, c_2)$ is defined,

$\mathbb{E}\{R\} \Rrightarrow_{\mathcal{T}} \mathbb{E}\{Q\}$, where $R \rightsquigarrow_{\mathcal{T}} Q$,

$\mathbb{C}\{R\} \rightarrow_{\mathcal{T}} \mathbb{C}\{Q\}$, where $R \rightsquigarrow_{\mathcal{T}} Q$.

Figure 2.3: Term calculus for the calculus of records

An example of evaluating a term with a label is the following:

$$(\lambda x.l) \mathbin{@} 2 \Rrightarrow l.$$

We define free variables of a term M as follows:

Definition 2.2.1 (Free variables in term calculus). The set of *free variables* of a term M, written FV(M), is defined as follows:

$$\begin{aligned} FV(x) &= \{x\}, \\ FV(c) &= \emptyset, \\ FV(l) &= \emptyset, \\ FV(\lambda x.M) &= FV(M) \setminus \{x\}, \\ FV(M_1 \mathbin{@} M_2) &= FV(M_1) \cup FV(M_2), \\ FV(M_1 \mathbin{op} M_2) &= FV(M_1) \cup FV(M_2). \end{aligned}$$

□

Note that labels are not variables, and therefore are not included in the set of free variables. Also note that λ does not bind labels, only variables.

$D \in \mathbf{Term} \ ::= \ [l_1 \mapsto M_1, \ldots, l_n \mapsto M_n]$ (abbreviated $[l_i \overset{n}{\underset{i=1}{\mapsto}} M_i]$),
provided $\bigcup_{i=1}^{i \leq n} FV(M_i) = \emptyset$ and $l_i = l_j$ implies $i = j$,

$\mathbb{D} \in \mathbf{Context} \ ::= \ [l \mapsto \mathbb{C}, l_1 \mapsto M_1, \ldots, l_n \mapsto M_n]$

$\mathbb{G} \in \mathbf{EvalContext} \ ::= \ [l \mapsto \mathbb{E}, l_1 \mapsto M_1, \ldots, l_n \mapsto M_n]$

Projection Notation:

$[l_i \overset{n}{\underset{i=1}{\mapsto}} M_i] \downarrow l_j = M_j$, if $1 \leq j \leq n$, and otherwise undefined.

$\mathbb{G}\{R\} \Rightarrow_{\mathcal{C}} \mathbb{G}\{Q\}$, where $R \rightsquigarrow_{\mathcal{T}} Q$.

$\mathbb{G}\{l\} \ \Rightarrow_{\mathcal{C}} \mathbb{G}\{V\}$, where $\mathbb{G}\{l\} \downarrow l = V$.

$\mathbb{D}\{R\} \rightarrow_{\mathcal{C}} \mathbb{D}\{Q\}$, where $R \rightsquigarrow_{\mathcal{T}} Q$.

$\mathbb{D}\{l\} \ \rightarrow_{\mathcal{C}} \mathbb{D}\{V\}$, where $\mathbb{D}\{l\} \downarrow l = V$.

Figure 2.4: Calculus of records

Having defined the term calculus, we now define the calculus of records. The

formal definition is given in figure 2.4. A record consists of mutually recursive components labeled by distinct labels. Each label is bound to a term in the term calculus. The components are unordered.

$\mathbb{D}$ and $\mathbb{G}$ are the context and the evaluation context for records, respectively. They are defined via the corresponding contexts of the term calculus: a module context is a record where a term in one of the components is replaced by a term context, $\mathbb{C}$ for a record context $\mathbb{D}$, and $\mathbb{E}$ for a record evaluation context $\mathbb{G}$. Module contexts serve the same purpose as in the call-by-value and call-by-name calculi: they facilitate definition of an evaluation step and a calculus reduction step. Note that these contexts are filled with terms of the term calculus (defined in figure 2.3).

The notations $\mathbb{D}\{M\}$ and $\mathbb{G}\{M\}$ imply that the result of filling the context with the term is a well-formed record, i.e. $\mathbb{D}\{M\} \in \mathbf{Term}$, where **Term** is as defined in figure 2.4.

Relations in the record calculus are annotated by a subscript $\mathcal{C}$ (an historical artifact from the original name, "core module calculus").

The following reduction sequence (with omitted subscripts) illustrates different kinds of reductions on records[1]:

$$
\begin{aligned}
& [A \mapsto 2+3, B \mapsto (\lambda x.x * (4 + A)) \mathbin{@} A] \\
\Longrightarrow \quad & [A \mapsto 5, B \mapsto (\lambda x.x * (4 + A)) \mathbin{@} A] \\
\circ\!\!\longrightarrow \quad & [A \mapsto 5, B \mapsto (\lambda x.x * (4 + 5)) \mathbin{@} A] \\
\circ\!\!\longrightarrow \quad & [A \mapsto 5, B \mapsto (\lambda x.x * 9) \mathbin{@} A] \\
\Longrightarrow \quad & [A \mapsto 5, B \mapsto (\lambda x.x * 9) \mathbin{@} 5] \\
\Longrightarrow \quad & [A \mapsto 5, B \mapsto 5 * 9] \\
\Longrightarrow \quad & [A \mapsto 5, B \mapsto 45]
\end{aligned}
$$

[1]We use capital letters for labels in examples.

Note that some abstractions may be substituted into themselves:

$$[F \mapsto \lambda x.F] \circ\!\!\longrightarrow [F \mapsto \lambda x.(\lambda x.F)] \circ\!\!\longrightarrow [F \mapsto \lambda x.(\lambda x.(\lambda x.(\lambda x.F)))]$$

This is a non-evaluation step, since F appears under a λ.

Chapter 3

Computational Soundness

3.1 Classification and Class Preservation

The notions of *classification* of terms and *class preservation* formalize important features of calculi which have been used in the literature for computational soundness proofs, but are rarely, if ever, made explicit. We find that formalizing the class preservation property clarifies computational soundness proofs. For example, the proofs in sections 3.3–3.4 illustrate the use of class preservation.

An evaluation relation of a calculus partitions all terms into two categories: those that can be evaluated and those that are normal forms with respect to evaluation. For instance, in a call-by-value λ-calculus, a term $(\lambda x.x)$ @ $(\lambda y.y)$ evaluates to $(\lambda y.y)$, whereas a term $\lambda z.((\lambda x.x)$ @ $(\lambda y.y))$ is in a normal form w.r.t. evaluation. Note that the latter term has a calculus redex, but not an evaluation redex, since the redex is under a λ. A *classification function* allows us to further characterize terms by their meaning in the calculus. In particular, it makes sense to classify evaluation normal forms, since the meaning of a term in our framework is given by a small-step operational semantics.

EXAMPLE 3.1.1. As an example, consider a classification of terms in the call-by-value λ-calculus with constants defined in section 2.2.1. A similar classification was used by Plotkin in [Plo75]. The classification distinguishes the following classes of terms:

1. *evaluatable* terms, i.e. those in the domain of the evaluation relation $\Rightarrow$,
2. λ-*abstractions* (all abstractions are joined in one class),
3. constants, where each constant in the calculus forms its own class, for instance 3 and 5 belong to different classes,
4. errors, which are terms that do not belong to any of the above classes. For instance, a term 3 @ 5 is an error, because it cannot be evaluated further, but is not a constant or a λ-abstraction. So is a term $(\lambda x.x) + 3$, where $+$ is an addition of two integers. All errors are joined in one class.

□

In the above classification we distinguish between constants and abstractions because these terms have different semantics in the language (for instance, an abstraction can be applied to an argument, but a constant cannot). We also distinguish between different constants, because the meaning of one constant is different from the meaning of another one. We cannot distinguish between two λ-abstractions, because the behavior of a λ-abstraction can be observed only by applying this abstraction to various values. Constants and λ-abstractions are referred to as *values*, i.e. "good" results of evaluation. The remaining class *errors* contains "bad" results of evaluation, i.e. terms that are not values, but cannot be evaluated further. Sometimes such terms are called *stuck*. However, we use a term "stuck" for another class of terms in our calculus , so we call such terms "errors".

The classification for the record calculus is formally defined later in section 5.2.7. Examples of classification from other calculi in the literature are discussed in chapter 7.

A classification of terms in a calculus is formalized via a *classification* function Cl which maps terms to a set of tokens, some of which are parameterized over a set of values. A set of tokens for this example may be defined as **evaluatable**, **abs**, **const**(n) for each integer number n, and **error**. We assume that such a set of tokens and a classification function are part of the definition of a calculus. The set of classification tokens of a calculus must have at least two tokens, one of which is **evaluatable**, and a classification function must satisfy the following condition:

Property 3.1.2. *Cl maps every evaluatable term (and no evaluation normal form) to* **evaluatable**. □

The way in which evaluation normal forms are divided into classes (in particular, which normal forms are considered "good" results of evaluation, and which ones are "bad", i.e. errors) depends on the intended meaning of terms.

Given a classification in a calculus, we expect that the non-evaluation relation of the calculus preserves the classification. We call this property *class preservation*:

Definition 3.1.3 (Class Preservation). A calculus has the *class preservation* property if $M \circ\!\!\rightarrow N$ implies $Cl(M) = Cl(N)$. □

Example 3.1.4. Given a call-by-value λ calculus with constants and a classification as in example 3.1.1, consider the following non-evaluation steps:

1. $\lambda x.2 + 3 \circ\!\!\rightarrow \lambda x.5$. The class of both terms is **abs**.

2. $(\lambda x.2 + 3)\ @\ 7 \circ\!\!\rightarrow (\lambda x.5)\ @\ 7$. Both terms are evaluatable.

3. $((\lambda x.x)\ @\ (\lambda y.y))\ @\ ((\lambda z.z)\ @\ 3) \circ\!\!\longrightarrow ((\lambda x.x)\ @\ (\lambda y.y))\ @\ 3$. Both terms are evaluatable, and in both terms the next thing to be evaluated is the operator.

4. $1\ @\ (4+3) \circ\!\!\longrightarrow 1\ @\ 7$. Assuming the evaluation relation for the call-by-value λ-calculus as defined in section 2.2.1, both terms are errors, because they cannot be evaluated further.

Non-evaluation steps in this calculus correspond to a reduction under a lambda or a reduction of the operand of an application before the operator. The above examples illustrate that such reductions do not change the next redex to be performed by evaluation and cannot create an evaluation redex if it does not exist in the original term. It is straightforward to prove the class preservation property for this calculus, but we are not going to do so here. □

The class preservation property has two important implications:

1. If a term is an evaluation normal form, then any sequence of reduction steps originating at the term must consist purely of non-evaluation steps and end in another normal form of the same class.

2. A non-evaluation reduction sequence cannot change an evaluatable term to an evaluation normal form, and vice versa.

These implications are formalized via the following lemmas:

Lemma 3.1.5. *If the calculus has class preservation, M is an evaluation normal form, and $M \longrightarrow^* N$, then each step in $M \longrightarrow^* N$ is a non-evaluation step and $Cl(M) = Cl(N)$.* □

Proof. By induction on the number of steps n in $M \longrightarrow^* N$. If $n = 0$, the result is trivially true. For $n > 0$, $M \longrightarrow M' \longrightarrow^* N$. By the induction hypothesis, each

step in $M' \longrightarrow^* N$ is a non-evaluation step and $Cl(M') = Cl(N)$. Since M is an evaluation normal form, the step $M \longrightarrow M'$ must be a non-evaluation step, and by class preservation, $Cl(M) = Cl(M')$. □

Lemma 3.1.6. *If the calculus has class preservation and $M \circ\!\!\longrightarrow^* N$, then M is an evaluation normal form if and only if N is.* □

Proof. It is easy to show that $Cl(M) = Cl(N)$ by induction on the number of steps $M \circ\!\!\longrightarrow^* N$. Suppose that M is not an evaluation normal form; then by property 3.1.2 and class preservation, **evaluatable** $= Cl(M) = Cl(N) =$ **evaluatable**, so N is not an evaluation normal form. A similar argument shows that if M is an evaluation normal form, then so is N. □

Classification of a term characterizes its current state with respect to evaluation. A related notion of an *outcome* characterizes the "ultimate fate" of a term from the point of view of evaluation: does the evaluation of the term converge, and, if yes, to which kind of a normal form? Outcome is defined as follows:

Definition 3.1.7 (Outcome). The total function *Outcome* maps a term M to the classification of its evaluation normal form $Cl(Eval(M))$ if it exists, and to the symbol $\perp$ otherwise. □

Intuitively, classification and outcome characterize the observable behavior of a term.

3.2 Computational and Observational Soundness

3.2.1 Computational Soundness Property

Definition 3.2.1 (Computational soundness). A calculus is *computationally sound* if $M \leftrightarrow N$ implies $Outcome(M) = Outcome(N)$. □

Computational soundness relates the calculus relation to evaluation: a calculus step preserves the meaning of a term as defined via small-step operational semantics, i.e. evaluation.

Note that outcome is defined as the class of the evaluation normal form if it exists. This means that the meaning is preserved up to the classification in the calculus. For instance, according to the classification presented above, any two λ-abstractions in the call-by-value calculus have the same meaning.

The next section introduces a more familiar notion of observational equivalence and observational soundness and shows that observational soundness is implied by computational soundness under certain natural conditions.

3.2.2 Compatible Closure, Observational Soundness

Observational soundness is a familiar property which is traditionally used to justify program transformations. It states that two terms related in the calculus have the same behavior in any one-hole context. Observational soundness is implied by computational soundness if the calculus relation is compatibly closed (see lemma 3.2.4).

Definition 3.2.2 (Observational Soundness). A calculus is *observationally sound* if $M \leftrightarrow N$ implies that, for any context $\mathbb{C}$, $Outcome(\mathbb{C}\{M\}) = Outcome(\mathbb{C}\{N\})$. □

Definition 3.2.3 (Compatible closure). If $\longrightarrow$ is a relation in a calculus, then the *compatible closure* of $\longrightarrow$, denoted $\longrightarrow_{\mathbb{C}}$, is a relation defined as follows: $M \longrightarrow_{\mathbb{C}} N$

if and only if there exist $\mathbb{C}$, M_1, and N_1 such that $M_1 \longrightarrow N_1$, $M = \mathbb{C}\{M_1\}$, and $N = \mathbb{C}\{N_1\}$. A relation $\longrightarrow$ is called *compatibly closed* if $\longrightarrow = \longrightarrow_{\mathbb{C}}$. □

Lemma 3.2.4. *If the calculus relation $\longrightarrow$ is compatibly closed, then computational soundness implies observational soundness.* □

Proof. Follows directly from definitions 3.2.1, 3.2.2, and 3.2.3. □

Placing terms into arbitrary contexts allows us to distinguish two λ-abstractions in a λ-calculus. As a simple example, consider $\lambda x.2$ and $\lambda x.5$ in the call-by-value calculus. The class of both terms is **abs**, but we can distinguish them by applying them to any value, say to a term 1. On the other hand, $\lambda x.x + x$ and $\lambda x.2 * x$ behave the same way in any context.

Definition 3.2.2 can be generalized to a two-level calculus in the case when terms of one calculus fill in contexts of the other. An example of such a relation between calculi is terms of the term calculus filling contexts $\mathbb{D}$ in the calculus of records. For instance, a context $\mathbb{D} = [A \mapsto \Box + 2]$ can be filled by terms of the term calculus, such as 3 or $\lambda x.x$ @ 5. This scenario can be formalized by extending definition 3.2.2 to define observational soundness of one calculus in another one (in this example, of the term calculus in the module calculus). See [MT02] for a formal definition and details.

Also note that, in the current presentation, there are no contexts in which records can be placed. In [MT02] we define one-hole contexts which are sets of record bindings. Such contexts can be filled with other records, and in this framework we can define compatible closure of the calculus relation of the calculus of records. In chapter 6, we introduce a linking calculus in which linking contexts can be filled with modules (i.e., records).

3.3 A Classical Technique for Proving Computational Soundness

In this section we review the traditional approach to proving computational soundness. This technique has been applied by Plotkin in [Plo75] to show the computational soundness of call-by-value and call-by-name λ-calculi. The traditional "recipe" for proving the computational soundness of a calculus has three "ingredients":

1. Confluence (see definition 3.3.2);
2. Standardization (see definition 3.3.4);
3. The class preservation property defined in definition 3.1.3.

Below, we define the first two properties and show the traditional proof of computational soundness.

3.3.1 Confluence

Confluence is a classical property that holds for many calculi, both traditional, such as call-by-value or call-by-name λ-calculus, and new ones, such as the call-by-need calculus introduced in [AFM$^+$95]. The lack of confluence of the calculus of recursively scoped records considered here motivated our development of a new technique for proving computation soundness (see section 3.4).

Definition 3.3.1 (Confluence of a Relation). A relation $\longrightarrow$ is *confluent* if $M_1 \longrightarrow^* M_2$ and $M_1 \longrightarrow^* M_3$ imply that there exists M_4 such that $M_2 \longrightarrow^* M_4$ and $M_3 \longrightarrow^* M_4$ (see figure 3.1). □

Definition 3.3.2 (Confluence of a Calculus). A calculus is *confluent* if its calculus relation $\longrightarrow$ is confluent. □

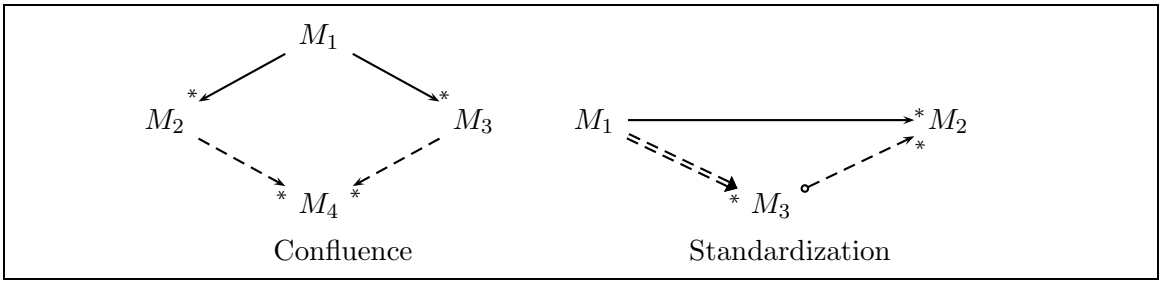


Figure 3.1: Confluence and standardization.

Our proofs of computational soundness use the following well-known property of a confluent relation:

Lemma 3.3.3. *If a relation $\longrightarrow$ is confluent and $M \leftrightarrow N$, then there exists a term L such that $M \longrightarrow^* L$ and $N \longrightarrow^* L$.* □

Proof. The proof is traditional; see the proof of theorem 3.1.12 in [Bar84]. □

3.3.2 Standardization

Standardization traditionally refers to a property of a calculus that requires every reduction sequence $M \longrightarrow^* N$ to be equivalent (in the calculus) to a sequence from M to N in which the reduction steps are performed in a "standard" order. The definition of "standard order" depends on a particular calculus. However, a standard sequence usually has the following two properties (as applicable to a particular calculus):

- if a term reaches a value by some reduction sequence, then it reaches a value by a standard sequence;

- for a calculus where the meaning of a term is defined via a small-step operational semantics, a standard reduction sequence first performs all the "standard" steps, i.e. those corresponding to the operational semantics, possibly followed by some "non-standard" steps, i.e. those that do not change the

observable behavior of a term (for instance, steps performed under a λ in a λ-calculus).

In a calculus where the calculus relation is partitioned into an evaluation relation and a non-evaluation relation, "standard" steps are evaluation steps, and "non-standard" steps are non-evaluation steps. In this presentation, we use the following definition of standardization, which captures both of the above properties:

Definition 3.3.4 (Standardization). A calculus has the *standardization property* if, for any sequence $M_1 \longrightarrow^* M_2$, there exists M_3 such that $M_1 \Longrightarrow^* M_3 \circ\!\!\longrightarrow^* M_2$ (see figure 3.1). □

Definition 3.3.5 (Standard Sequence). A sequence of the form $M_1 \Longrightarrow^* M_3 \circ\!\!\longrightarrow^* M_2$ is called *standard*. □

In section 7.2 we discuss definitions of the standardization property used in the literature in more detail and compare them to our definition.

3.3.3 Computational Soundness of a Confluent Calculus

Below, we present a traditional proof of computational soundness based on confluence and standardization.

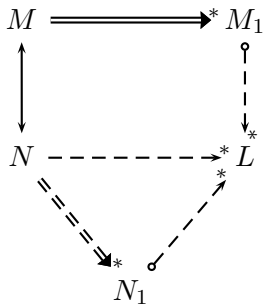
Theorem 3.3.6 (Computational Soundness via Confluence). *If $\longrightarrow$ is confluent and the calculus has the standardization and class preservation properties, then the calculus is computationally sound.* □

Proof. Assume that $M \leftrightarrow N$. We show that $\mathit{Outcome}(M) = \mathit{Outcome}(N)$ by considering the following two cases:

1. $Outcome(M) \neq \perp$: As shown in figure 3.2[1], $M \Longrightarrow^* M_1 = Eval(M)$. By confluence of $\longrightarrow$ and lemma 3.3.3, there exists a term L such that $M_1 \longrightarrow^* L$ and $N \longrightarrow^* L$. Since $M_1 \in NF_{\Longrightarrow}$, by class preservation and lemma 3.1.5, $M_1 \circ\!\!\longrightarrow^* L$. By standardization, $N \longrightarrow^* L$ implies that there exists a term N_1 such that $N \Longrightarrow^* N_1 \circ\!\!\longrightarrow^* L$. Since M_1, L, and N_1 are connected only by $\circ\!\!\longrightarrow$ steps, class preservation implies $Cl(M_1) = Cl(L) = Cl(N_1)$, and property 3.1.2 implies that $N_1 \in NF_{\Longrightarrow}$ since $M_1 \in NF_{\Longrightarrow}$. So $N_1 = Eval(N)$, and $Cl(N_1) = Cl(M_1)$.

2. $Outcome(M) = \perp$: If $Eval(N)$ exists, then by the above argument we could show that $Eval(M)$ also exists. So, by contradiction, $Outcome(N) = \perp$.

□



Let $M_1 = Eval(M)$. Then
$Cl(M_1) = Cl(L) = Cl(N_1)$, hence
$N_1 = Eval(N)$, hence
$Outcome(M) = Outcome(N)$

Figure 3.2: Sketch of the traditional proof of computational soundness.

[1]In diagrams here and below, arrows marked with an asterisk denote reflexive, transitive closures of the respective relations, and a line with arrows on both ends denotes the reflexive, symmetric, transitive closure of the respective relation. Solid lines denote given relations, while dashed lines denote relations whose existence is implied by the the proofs. Dotted lines denote relations that cannot be constructed.

3.4 Our Technique (High-level View)

Traditionally, proofs of computational soundness (such as the one in the previous section) requirethe confluence of $\longrightarrow$. But what if $\longrightarrow$ is not confluent? We propose a technique that allows proving computational soundness in some calculi that lack confluence. In chapter 5, we apply this technique to prove the computational soundness of a calculus of recursively scoped records.

The technique is based on the following observation. An inspection of the proof of theorem 3.3.6 reveals that the confluence of $\longrightarrow$ is stronger than what is actually required for the proof to go through. The fact that one side of the commutative square in figure 3.2 consists purely of evaluation steps rather than arbitrary calculus reduction steps suggests that a weaker form of "confluence" might suffice: one that only takes into account interactions between calculus steps and evaluation steps. Our novel technique for proving computational soundness replaces confluence of $\longrightarrow$ and standardization of the calculus by a pair of properties that we call *lift* and *project*, which constitute weaker requirements for a computational soundness proof.

3.4.1 Lift and Project Properties

The lift and project properties are defined below and depicted in figure 3.3.

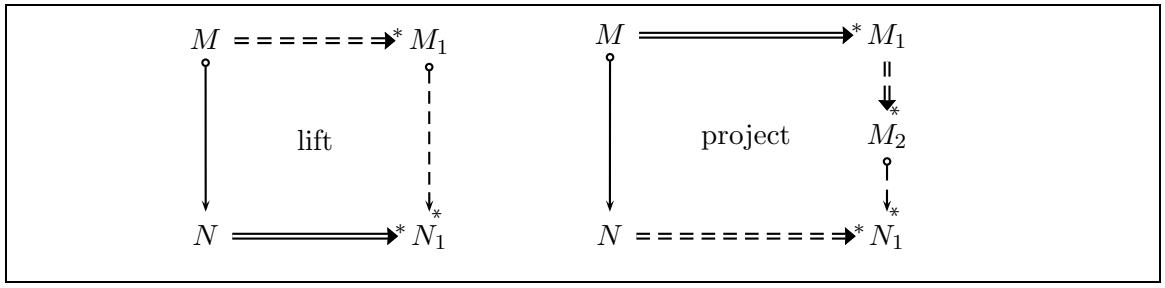


Figure 3.3: The lift and project properties.

Definition 3.4.1 (Lift). A calculus has the *lift property* if for any reduction sequence $M \circ\!\!\rightarrow N \Rrightarrow^* N_1$ there exists a sequence $M \Rrightarrow^* M_1 \circ\!\!\rightarrow^* N_1$. □

Definition 3.4.2 (Project). A calculus has the *project property* if $M \circ\!\!\rightarrow N$, $M \Rrightarrow^* M_1$ implies that there exist terms M_2 and N_1 such that $M_1 \Rrightarrow^* M_2$, $N \Rrightarrow^* N_1$, and $M_2 \circ\!\!\rightarrow^* N_1$. □

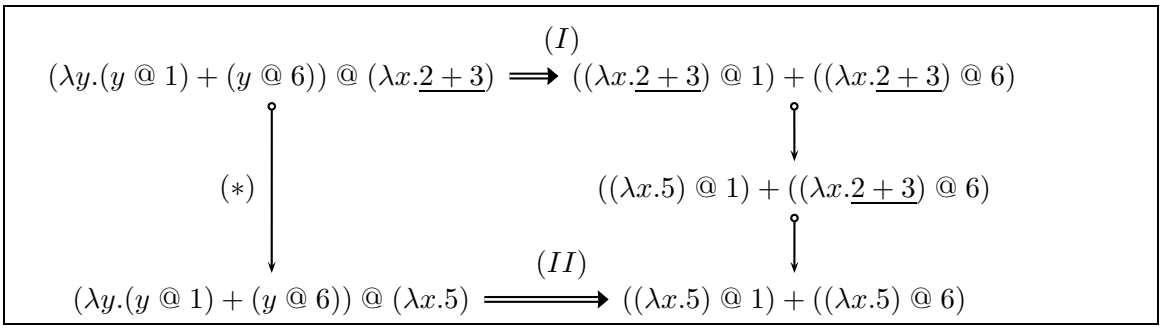


Figure 3.4: Example of lift and project: one-step evaluation sequence.

Example 3.4.3 (Lift and Project, one evaluation step). Consider a term $(\lambda y.(y \mathbin{@} 1) + (y \mathbin{@} 6)) \mathbin{@} (\lambda x.2 + 3)$ in a call-by-value λ-calculus with constants and a non-evaluation reduction of $2 + 3$ to 5, as shown on figure 3.4. We underline $2 + 3$ on the figures to make it easier to trace this redex in the diagrams. Figure 3.4 illustrates both lift and project properties:

- *Lift:* Suppose that we are given the non-evaluation step (*) and the evaluation step (II) in figure 3.4. Then the term $((\lambda x.\underline{2 + 3}) \mathbin{@} 1) + ((\lambda x.\underline{2 + 3}) \mathbin{@} 6)$ is the term whose existence is guaranteed by the lift property (denoted by M_1 in property 3.4.1 and on figure 3.3).

- *Project:* If the evaluation step annotated by (I) is given, together with the non-evaluation step (*), then the terms $((\lambda x.\underline{2 + 3}) \mathbin{@} 1) + ((\lambda x.\underline{2 + 3}) \mathbin{@} 6)$ and $((\lambda x.5) \mathbin{@} 1) + ((\lambda x.5) \mathbin{@} 6)$ serve as witnesses M_2 and N_1, respectively, for the

project property (see property 3.4.2 or figure 3.3 for notations). Note that the term $((\lambda x.\underline{2+3}) \mathbin{@} 1) + ((\lambda x.\underline{2+3}) \mathbin{@} 6)$ serves as both M_1 and M_2, in the notations of property 3.4.2.

□

It is easy to see in figure 3.4 that both non-evaluation steps in the right-hand side reduce underlined copies of the non-evaluation redex $2 + 3$, which was reduced in the left-hand side of the diagram. Incidentally, this is always the case in the lift and project properties: if we mark the original non-evaluation redex in the left-hand side of the diagram (i.e. the redex of the reduction $M \circ\!\!\rightarrow N$ in both diagrams in figure 3.3), then the vertical reduction in the right-hand side of the diagrams in figure 3.3 always reduces underlined copies (i.e. residuals) of that redex. We discuss this property of lift and project in chapter 4, where we also give a formal definition of a redex and a residual.

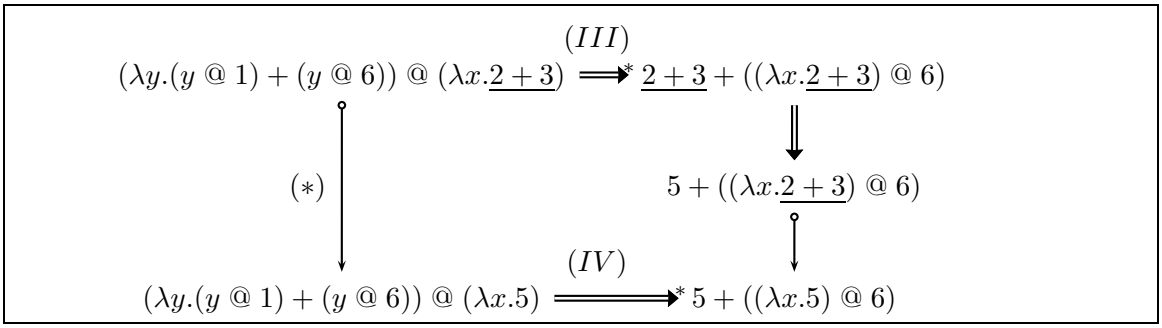


Figure 3.5: Example of lift and project: two-step evaluation sequence.

Example 3.4.4 (Lift and Project, two evaluation steps). This example shows the case when some of the vertical reduction steps in the right-hand side of the diagrams are evaluation steps. Consider the extension of the evaluation sequences (I) and (II) in figure 3.4 in example 3.4.3 by another step. The result is shown in figure 3.5. As in the previous example, we illustrate both lift and project:

- *Lift:* If we assume the existence of the reduction of $2 + 3$ to 5 in the original term (i.e. the non-evaluation reduction (*)) and of the evaluation sequence (IV), then the term $5 + ((\lambda x.\underline{2 + 3}) \mathbin{@} 6)$ serves as a witness M_1 for the lift property. Note that the evaluation sequence to M_1 from the original term $(\lambda y.(y \mathbin{@} 1) + (y \mathbin{@} 6)) \mathbin{@} (\lambda x.\underline{2 + 3})$ consists of 3 steps: two steps in the sequence (III) and the step $\underline{2 + 3} + ((\lambda x.\underline{2 + 3}) \mathbin{@} 6) \implies 5 + ((\lambda x.\underline{2 + 3}) \mathbin{@} 6)$ which reduces a marked copy of the redex $2 + 3$. This copy of the redex has become an evaluation redex.

- *Project:* Now suppose we are given the non-evaluation reduction (*) and the evaluation sequence (III). Then the terms $5+((\lambda x.\underline{2 + 3}) \mathbin{@} 6)$ and $5+((\lambda x.5) \mathbin{@} 6)$ serve as the witnesses M_2 and N_1 (respectively) for the project property. Note that in this case $M_1 \neq M_2$, because one of the marked copies of $2 + 3$ is now an evaluation redex.

□

One may notice that the lift and project properties are slightly asymmetric (see figure 3.3): the lift diagram is completed by two reduction sequences and requires just one term as a witness, but the project diagram requires three sequences and two terms as witnesses to complete. This is because a non-evaluation redex can have an evaluation redex as a residual (as has been illustrated by the example above). Since evaluation steps reducing the residuals of the original non-evaluation redex occur only in the evaluation sequence originating at the term M, but not in the evaluation sequence from N, where the non-evaluation redex has already been reduced (and therefore does not have residuals), the evaluation sequence from M may have more steps than the corresponding sequence from N (in the second part of the example above the former sequence has 3 steps, and the latter has 2). Such "extra"

evaluation steps are incorporated into the evaluation sequence constructed in lift, but since an evaluation sequence from the original term is *given* in project, it may be needed to extend this sequence by the reduction of "evaluation" residuals of the non-evaluation redex be be able to complete the diagram. This case is demonstrated by example 3.4.4.

3.4.2 Computational Soundness of a Non-confluent Calculus

The proof of computational soundness via lift and project has a different flavor than that used in theorem 3.3.6. The lift and project properties allow us to show directly that a non-evaluation step preserves the outcome of a term.

When evaluation is a function, as in the call-by-value and call-by-name λ-calculi, evaluation steps trivially preserve outcome of a term. In section 3.5 we extend our framework to calculi in which evaluation is not a function.

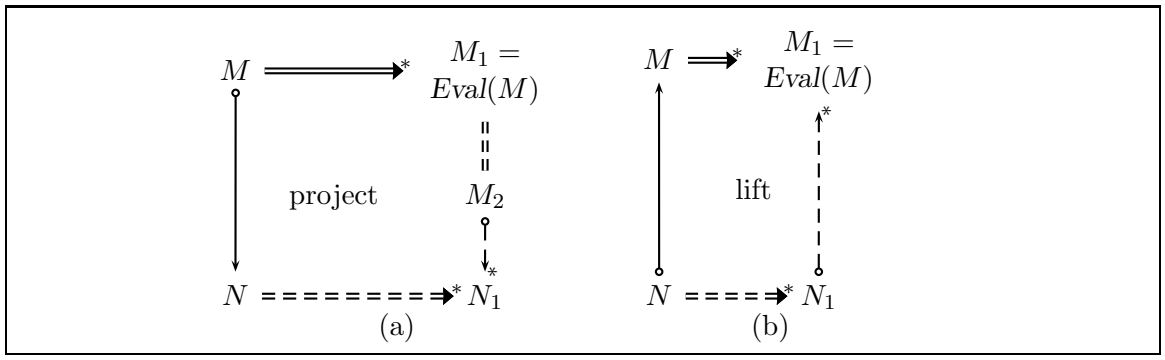


Figure 3.6: Proof of soundness via lift and project.

The following theorem embodies our new approach to proving computational soundness:

Theorem 3.4.5 (Computational Soundness via Lift and Project). *If the calculus has the lift, project, and class preservation properties, then it is computationally sound.* □

Proof. Assume that $M \leftrightarrow N$. We show that $\mathit{Outcome}(M) = \mathit{Outcome}(N)$ by the following two cases:

1. $\mathit{Outcome}(M) \neq \bot$: We show the result for the case where M and N are connected by a single non-evaluation step. Using the single-step result and the fact that evaluation is a function, and therefore an evaluation step preserves the meaning of a term, it is easy to prove the multi-step result via an induction on the length of the sequence $M \leftrightarrow N$.

   Let $M_1 = \mathit{Eval}(M)$. We show that $\mathit{Outcome}(N) = \mathit{Cl}(M_1) = \mathit{Outcome}(M)$ in both cases relating M and N by a non-evaluation step (see figure 3.6):

   (a) $M \circ\!\!\rightarrow N$. By the project property, $M \Longrightarrow^* M_1$ implies that there exist M_2, N_1 such that $M_1 \Longrightarrow^* M_2$, $N \Longrightarrow^* N_1$, and $M_2 \circ\!\!\rightarrow^* N_1$. But M_1 is an evaluation normal form, so $M_1 = M_2$. By the class preservation property, $\mathit{Cl}(M_1) = \mathit{Cl}(N_1)$, and property 3.1.2 implies that N_1 is an evaluation normal form. Hence, $N_1 = \mathit{Eval}(N)$, and $\mathit{Outcome}(N) = \mathit{Cl}(N_1) = \mathit{Cl}(M_1) = \mathit{Outcome}(M)$.

   (b) $N \circ\!\!\rightarrow M$. By the lift property, $M \Longrightarrow^* M_1$ implies that there exists N_1 such that $N \Longrightarrow^* N_1$ and $N_1 \circ\!\!\rightarrow^* M_1$. Class preservation implies $\mathit{Cl}(N_1) = \mathit{Cl}(M_1)$, and since M_1 is an evaluation normal form, property 3.1.2 implies that N_1 is an evaluation normal form, and, as above, $\mathit{Outcome}(N) = \mathit{Outcome}(M)$.

2. $\mathit{Outcome}(M) = \bot$: If $\mathit{Outcome}(N) \neq \bot$, then by the above argument we can show that $\mathit{Outcome}(M) = \mathit{Outcome}(N) \neq \bot$, and we get a contradiction. Therefore, if $\mathit{Outcome}(M) = \bot$, then $\mathit{Outcome}(N) = \bot$.

□

3.4.3 Lift is Equivalent to Standardization

It is easy to see that the lift property (3.4.1) is implied by standardization (see definition 3.3.4). As shown below, the converse is also true, so lift is in fact *equivalent* to standardization.

When then do we define a new property (lift) rather than using an existing one (standardization)? via lift and project. As we shall see later, proofs of lift and project use the same technique (reductions of marked terms and, based on such reductions, γ-developments); these proofs also share an important intermediate result: weak standardization of γ-developments. Since proving project is required in our framework, it is more convenient for us to formulate and prove lift property rather than the equivalent standardization property, so that we can use some of the results already established for proving project.

Lemma 3.4.6 (Lift Implies Standardization). *If a calculus has the lift property (i.e., for any reduction sequence $M \circ\!\!\rightarrow N \Rightarrow^* N_1$ there exists a sequence $M \Rightarrow^* M_1 \circ\!\!\rightarrow^* N_1$) then it has the standardization property (i.e., for any sequence $M_1 \rightarrow^* M_2$ there exists M_3 such that $M_1 \Rightarrow^* M_3 \circ\!\!\rightarrow^* M_2$).* □

Proof. A non-standard sequence S from M_1 to M_2 must have the form:

$$M_1 \quad \rightarrow^* \quad L_1 \quad \circ\!\!\rightarrow \quad L_2 \quad \Rightarrow^+ \quad L_3 \quad \circ\!\!\rightarrow^* \quad M_2.$$

By the lift property there exists $\tilde{L}_2$ such that $L_1 \Rightarrow^* \tilde{L}_2 \circ\!\!\rightarrow^* L_3$. Define $\Phi(S)$ as the sequence[2]:

$$M_1 \quad \rightarrow^* \quad L_1 \quad \Rightarrow^* \quad \tilde{L}_2 \quad \circ\!\!\rightarrow^* \quad L_3 \quad \circ\!\!\rightarrow^* \quad M_2.$$

[2]In general, there may be many sequences that satisfy the lift property, so $\Phi(S)$ may not be uniquely defined. However, we can always introduce an ordering on terms and a related ordering on reduction sequences such that $\Phi(S)$ can be uniquely defined as the sequence obtained by using the least or greatest sequence in this ordering satisfying the lift property.

Note that each application of Φ on a reduction sequence from M_1 to M_2 yields a new reduction sequence from M_1 to M_2 that has fewer non-evaluation steps to the left of the rightmost evaluation step in the sequence. Thus, iterating Φ starting with an arbitrary reduction sequence from M_1 to M_2 will eventually terminate with a reduction sequence from M_1 to M_2 in which there are no non-evaluation steps to the left of any evaluation step – i.e., when a standard sequence from M_1 to M_2 has been obtained. □

3.4.4 Weakening Project: Confluence w.r.t. Evaluation

Recall that our framework requires lift, project, and class preservation in order to prove computational soundness. It turns out that we can replace the project property by a slightly weaker property that we call *confluence w.r.t. evaluation* and still be able to prove computational soundness. Confluence w.r.t. evaluation is defined as follows and depicted in figure 3.7:

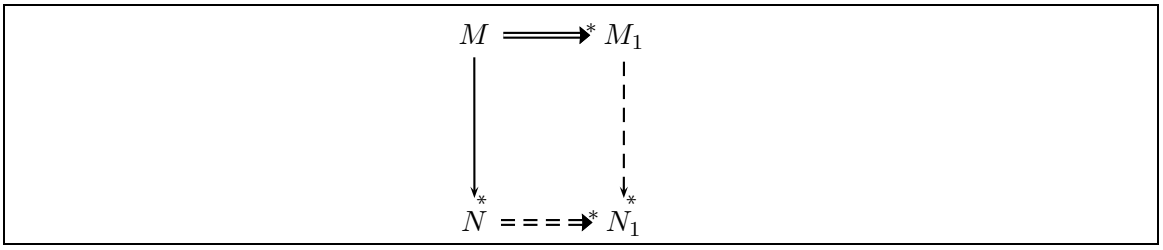


Figure 3.7: Confluence w.r.t. evaluation.

Definition 3.4.7 (Confluence w.r.t. evaluation). A calculus reduction relation $\longrightarrow$ is *confluent w.r.t.* an evaluation relation $\Longrightarrow$ iff $M \Longrightarrow^* M_1$ and $M \longrightarrow^* N$ implies the existence of a N_1 such that $N \Longrightarrow^* N_1$ and $M_1 \longrightarrow^* N_1$. □

We can prove computational soundness if we replace project by confluence with respect to evaluation in the preconditions:

Theorem 3.4.8 (Computational Soundness via Confluence w.r.t. Evaluation). *If $\longrightarrow$ is confluent w.r.t. evaluation, and the calculus enjoys the lift and class preservation properties, then the calculus is computationally sound.* □

Proof. The proof is similar to that for theorem 3.4.5. Case 2 and Case 1(b) are exactly the same. Case 1(a) follows from the fact that $\longrightarrow$ is confluent w.r.t. $\Rightarrow$. □

Project implies confluence w.r.t. evaluation, as shown in lemma 3.4.9. We conjecture that confluence w.r.t. evaluation is more general than project, i.e. there exist systems which have the former, but not the latter. However, we have not yet shown the existence of such a system. By lemma 3.4.9 below such systems do not have standardization, and therefore cannot be shown to be computationally sound via our new technique. We leave constructing such a system for a future work.

Lemma 3.4.9 (Project Implies Confluence w.r.t. Evaluation). *If a calculus has the project property, then $\longrightarrow$ is confluent w.r.t. $\Rightarrow$.* □

Proof. Suppose that $M \Rightarrow^* M_1$ and $M \longrightarrow^* N$. We wish to show that there is a N_1 such that $N \Rightarrow^* N_1$ and $M_1 \longrightarrow^* N_1$. We proceed by induction on the length n of the sequence $M \longrightarrow^* N$. If $n = 0$, then $N = M$ and $N_1 = M_1$. If $n > 0$, then $M \longrightarrow L \longrightarrow^* N$.

If $M \Rightarrow L$, then $L \Rightarrow^* M_1$, since $\Rightarrow$ is a function. Otherwise $M \circ\!\!\rightarrow L$, and the project property implies the existence of L_1 such that $L \Rightarrow^* L_1$ and $M_1 \longrightarrow^* L_1$. Applying the inductive hypothesis completes the proof. □

In a calculus that has standardization (or, equivalently, lift), the converse of lemma 3.4.9 also holds, i.e. confluence w.r.t. evaluation implies project:

Lemma 3.4.10. *If a calculus has confluence of $\Rightarrow$ w.r.t. evaluation and the standardization property (definition 3.3.4), then it has the project property.* □

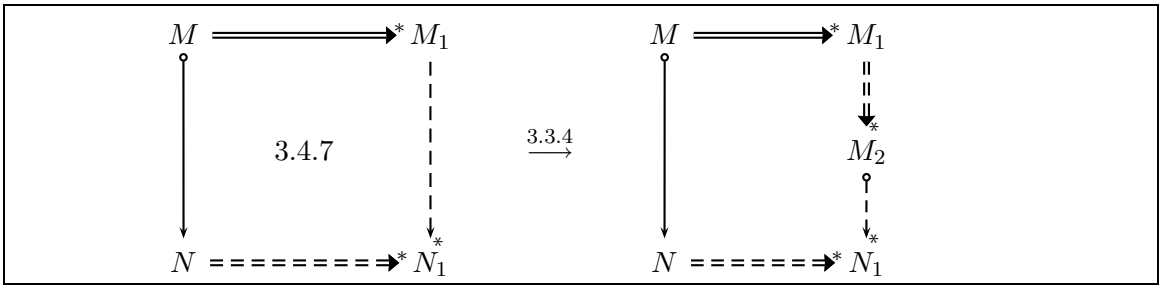


Figure 3.8: Proof of lemma 3.4.10.

Proof. The proof is shown on figure 3.8. Suppose $M \circ\!\!\rightarrow N$ and $M \Rightarrow^* M_1$. Then by confluence w.r.t. evaluation there exists N_1 such that $N \Rightarrow^* N_1$ and $M \rightarrow^* N_1$. By standardization the latter sequence implies that there exists M_2 such that $M_1 \Rightarrow^* M_2 \circ\!\!\rightarrow^* N_1$, which proves the project property. □

The above lemma is formulated for standardization, rather than for lift, because it is more convenient to use standardization in the proof.

Our framework proves computational soundness via lift and project. If a calculus does not have lift (and hence standardization), then our approach is not applicable. If lift holds, then, as we have shown, project is equivalent to confluence w.r.t. evaluation. Therefore, even though confluence w.r.t. evaluation is a more general property, there is no point of using it instead of project in our proofs, because in this framework the two properties are equivalent. For the rest of the presentation we will be using lift and project. While it is an interesting fact that project could be replaced by a slightly weaker property without affecting the computational soundness results, the two properties are equivalent in our framework because of the presence of standardization.

3.5 Extension: Handling Evaluation Relations

3.5.1 Abstracting Over Order and Names of Components

In typical calculi, evaluation is a function. I.e., for every term there is at most one term to which it evaluates. This is a natural way of defining evaluation for such calculi as λ-calculus, where there is an obvious ordering of redexes in a term, such as outside-in and left-to-right. However, if one considers a calculus of recursively scoped records, there is no clear order of evaluation of components. For instance, consider the following record:

$$[B \mapsto 5 + 6, A \mapsto \lambda x.x \ @ \ 3].$$

If evaluation is a function, then there is just one result of an evaluation step of this record. Is it $[B \mapsto 5 + 6, A \mapsto 3]$ or $[B \mapsto 11, A \mapsto \lambda x.x \ @ \ 3]$?

Of course, we could decide that we evaluate components in a record left-to-right (in which case we evaluate the component bound to B), but then we would not be able to identify such records up to reordering of components. If records represent modules (as it the case in our linking calculus), then we would like an operation of linking two modules together (i.e. combining their components) to be commutative. So a left-to-right order is not a good order of evaluation.

A better way of resolving this issue is to suppose that labels in modules are elements of some totally ordered set. For instance, we can assume that all labels are composed of letters and are ordered lexicographically (and in the example the component bound to A is evaluated, since A goes before B in alphabetical order). In this case the order of evaluation does not depend on the order of components, and we are able to define commutative linking of two modules.

However, this approach presents additional difficulties in defining operations

on modules at the linking level. One issue that we have to deal with is representing exported (visible) and private (hidden) components of modules. While exported components of modules have unique fixed names, private components represent the implementation part of a module not visible to the user of the module. The names of private components don't have to be unique or fixed, in fact they may be changed, added, or removed if the implementation of the module changes, and they may be generated automatically, in particular to resolve naming conflicts when linking two modules. We would like to identify modules up to consistent renaming of their private components (i.e. renaming a label binding the component and all references to this label throughout the module to the same new name), similarly to identifying modules up to the order of components.

There is still a way of defining at least a partial order on private components based on the exposed labels of the module. For instance, consider a module:

$$[A \mapsto 2 + h_2, B \mapsto \lambda x.x \ @ \ h_1, h_1 \mapsto 1 + 3, h_2 \mapsto 4 * 5]$$

Here A, B are exposed labels, and h_1, h_2 are private. Which of the two private components should be evaluated first? We cannot decide by their names for the reasons explained in the previous paragraph. However, since a visible label A uses h_2, and B uses h_1, we can use the ordering of the exposed labels to impose an order on the private labels. In this case h_2 has a priority over h_1. The same approach allows us to order "hidden" labels even if they are referenced in the same exposed component. For instance, in the module

$$[A \mapsto h_1 + h_2, h_1 \mapsto 1 + 3, h_2 \mapsto 4 * 5]$$

h_1 has a priority over h_2, since it appears first in the component bound to A (components are terms, and therefore all their subterms are ordered by the outside-in, left-to-right ordering).

But even this approach defines only a partial order on components. Consider a module:

$$[A \mapsto 2, h_1 \mapsto 3 + 4, h_2 \mapsto \lambda x.x \ @ \ 5].$$

Since the private components are not referenced in the exported part of the module, this approach does not allow us to specify the order in which they will be evaluated. Of course, one may argue that there is no need to order them, since they do not contribute to the exposed part of the module. But what if one of these components diverges, and the other gives an error? One has to be very careful in handling these issues, as well as other potential problems related to such operations as renaming of labels and hiding of components.

While we are not claiming that the suggested (or a similar) approach cannot handle these issues, we have just pointed out that a lot of care has to be taken if one wants to define an evaluation function for recursively scoped records and for modules with hidden components. At the same time, many of these problems can be avoided if we relax the restriction that evaluation has to be a function, i.e. allow more than one term to be a result of a one-step evaluation of a given term. This is similar to a calculus rewrite relation $\longrightarrow$, which also may rewrite the same term to different terms, depending on which redex gets reduced. If we adopt this approach, then for the module

$$[B \mapsto 5 + 6, A \mapsto \lambda x.x \ @ \ 3]$$

in the example above both $[B \mapsto 5 + 6, A \mapsto 3]$ and $[B \mapsto 11, A \mapsto \lambda x.x \ @ \ 3]$ are results of an evaluation step.

We adopt this approach for the calculus of recursively scoped records and for the linking calculus based on it. Evaluation is a function in the underlying call-by-value calculus which describes components of records.

3.5.2 Properties of an Evaluation Relation

Recall that the meaning of a term in our framework is given by small-step operational semantics defined via the evaluation relation. Therefore even if evaluation is not a function, it still has to be sufficiently "well-behaved" to define a unique meaning for each term in the calculus. It turns out that it is sufficient to require that the evaluation relation is confluent.

Lemma 3.5.1. *If $\Longrightarrow$ is confluent, then for every M there is a unique $Outcome(M)$ which satisfies definition 3.1.7.* □

Proof. Definition 3.1.7 defines outcome of a term M as $Cl(Eval(M))$ if it exists, and the symbol $\perp$ otherwise.

- Suppose M has an evaluation normal form, i.e. there exists a term N such that $M \Longrightarrow^* N$, and there is no N_1 such that $N \Longrightarrow N_1$. If M has another evaluation normal form N_1, then by confluence of $\Longrightarrow$ there must a term L such that $N \Longrightarrow^* L$ and $N_1 \Longrightarrow^* L$. But since both N and N_1 are evaluation normal forms, it must be the case that $N = L = N_1$. Therefore M has a unique evaluation normal form, and $Outcome(M) = Cl(Eval(M))$ is unique.

- Suppose M does not have an evaluation normal form, i.e. all evaluation sequences originating at M are infinite. Then by definition $Outcome(M) = \perp$, and again the outcome of M is unique.

□

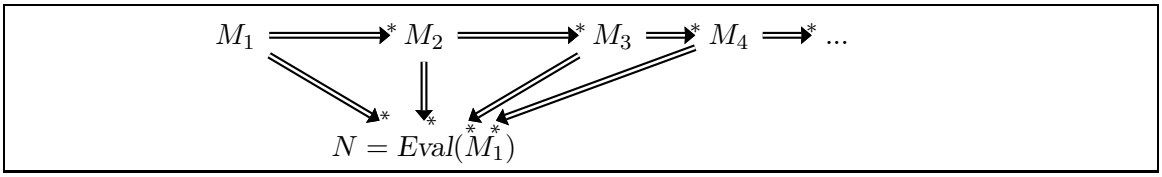


Figure 3.9: Divergence does not contradict existence of a normal form.

Note that the first case of the above proof states that M *has* an evaluation normal form. It does not require that all evaluation sequences originating from M converge. In fact, confluence of evaluation does not guarantee that if a term has an evaluation normal form, then all evaluation sequences originating from the term converge. See figure 3.9 for a case when a term has an infinite diverging sequence and a normal form, also see example below of a confluent relation that allows a term to have both a normal form and an infinite diverging sequence.

Example 3.5.2 (Evaluation Normal Form and Divergence). The calculus relation $\longrightarrow$ of the call-by-name λ-calculus is a relation that allows a term to have both a normal form and a diverging sequence. For instance, consider a term $M = (\lambda x.1)\ @\ \Omega$, where $\Omega = (\lambda y.y)\ @\ (\lambda y.y)$. M evaluates to its calculus normal form 1 in one step, but also has an infinite reduction path evaluating Ω. □

Note that even if evaluation behaves as described above, every term still has a unique outcome as defined in definition 3.1.7. However, in this case outcome does not completely characterize *observable behavior* of a term w.r.t. evaluation, because the behavior of the term allows two possibilities: evaluation to the normal form and divergence, but the outcome only takes into account the normal form.

Fortunately, in all the applications considered in this presentation evaluation relations are such that if a term diverges w.r.t. $\Longrightarrow$ on one path, then it diverges on all paths, so the technical issue discussed in this remark does not arise there. Therefore, in the rest of the presentation we make the following assumption:

Assumption 3.5.3. We assume confluence of $\Longrightarrow$ in all cases when $\Longrightarrow$ is not a function. In addition, we assume that $\Longrightarrow$ is such that if a term M has a normal form w.r.t. $\Longrightarrow$, then there is no infinite sequence of $\Longrightarrow$ steps originating from M. □

3.5.3 Computational Soundness when Evaluation is not a Function

In this section we extend the results of sections 3.3–3.4 for the case when evaluation is not a function. The first thing we observe is that assumption 3.5.3 guarantees that both an evaluation normal form of a term (if it exists) and an outcome of a term are uniquely defined. Hence all the definitions which use these notions are valid.

We show that the result of theorem 3.4.5 hold in the case when evaluation is not a function.

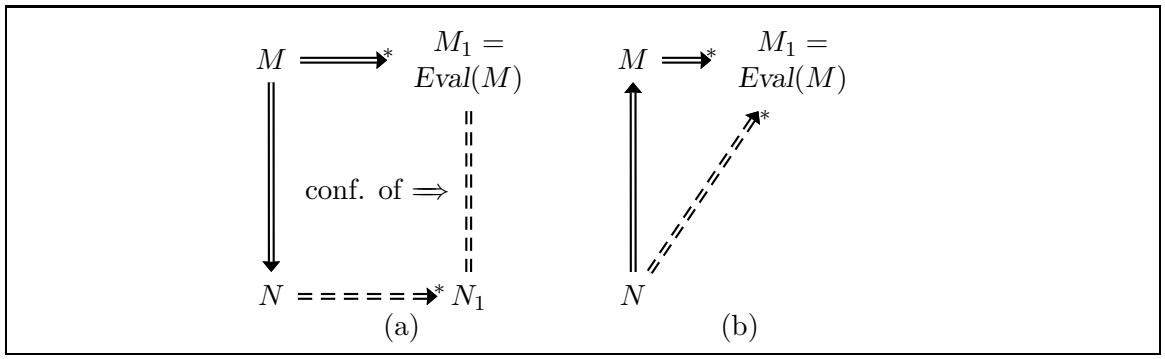


Figure 3.10: Additional cases when evaluation is not a function.

Theorem 3.5.4. *Consider a calculus whose evaluation relation $\Longrightarrow$ is not a function, but is confluent and satisfies assumption 3.5.3. If the calculus has the lift, project, and class preservation properties, then it is computationally sound.* □

Proof. Assume that $M \leftrightarrow N$. As in the proof of theorem 3.4.5, We show that $Outcome(M) = Outcome(N)$ by the following two cases:

1. $\mathit{Outcome}(M) \neq \perp$: We show the result for the case where M and N are connected by a single non-evaluation step. The rest of the proof follows by induction on the number of steps in $M \leftrightarrow N$.

   Let $M_1 = \mathit{Eval}(M)$. We have the following four cases: $M \circ\!\!\rightarrow N$, $N \circ\!\!\rightarrow M$, $M \Longrightarrow N$, and $N \Longrightarrow M$. The first two cases are exactly the same as in the proof of theorem 3.4.5. We show the result in the other two cases, they are illustrated in figure 3.10.

   (a) $M \Longrightarrow N$. Since $M \Longrightarrow^* M_1$, by confluence of $\Longrightarrow$ there exists N_1 such that $N \Longrightarrow^* N_1$ and $M_1 \Longrightarrow^* N_1$. But M_1 is an evaluation normal form, so $M_1 = N_1$, N_1 is an evaluation normal form, and $N_1 = \mathit{Eval}(N)$.

   (b) $N \Longrightarrow M$. Since $N \Longrightarrow M \Longrightarrow^* M_1$, we have $N \Longrightarrow^* M_1$, and therefore $M_1 = \mathit{Eval}(N)$ by the uniqueness of an evaluation normal form.

2. The case when $\mathit{Outcome}(M) = \perp$ is the same as in the proof of theorem 3.4.5.

□

Chapter 4

Techniques for Proving Lift and Project

4.1 Challenges in Proving Lift and Project

We have shown (see section 3.4.3, in particular lemma 3.4.6) that one of the properties that we need to prove, namely the lift property, is equivalent to standardization, and the other property, project, is a restricted version of confluence. One might expect that we would be able to adapt the methods commonly used for proving confluence and standardization for proofs of lift and project. There has been a variety of techniques proposed for proving confluence and standardization (together or separately), such as [Plo75, HL91, Tak95, GLM92, WM00]. See chapter 7 for a more detailed discussion.

However, while the methods proposed in these publications are quite general and cover a variety of different systems, each of these methods makes assumptions which do not hold in our main application.

- Huet and Levy in [HL91] give a framework for a first-order term rewriting

systems and they also require the system to be left-linear. A rewrite system is *left-linear* if each meta-variable appears once in the left-hand side of a rewrite rule.

- Plotkin's proof in [Plo75] is based on finiteness (or, more exactly, boundedness) and confluence of developments.

- Takahashi in [Tak95] defines parallel reductions based by induction on the term structure. However, this approach is implicitly based on finiteness and confluence of developments, since parallel reduction effectively performs a development of certain redexes, and finiteness and confluence of developments seem to be required for these reductions to be well-defined.

- Both [GLM92] and [WM00] give very general proofs of standardization, but in both cases the system is assumed to be left-linear.

Our new technique has been developed with the goal of of being able to handle the calculus of records (see section 2.2.3). As simple as it is, the calculus of records lacks many properties that one is accustomed to seeing in calculi. In addition to being non-confluent, it also has non-left-linear rules, and its developments, if defined straightforwardly, are not finite, non-confluent, and lack standardization (see section 4.4 for details).

As we have mentioned earlier, rather than restricting our calculus to satisfy these properties, we have undertaken a challenging task of proving computational soundness for the calculus as is, without excluding any of its capabilities. The proofs turned out to be somewhat long and tedious, but we hope that they pave a way for computational soundness proofs for other non-confluent calculi and similar systems.

In this chapter, we first give an overview of traditional techniques. We use

the call-by-value and the call-by-name λ-calculi as examples of calculi for which these traditional techniques are applicable and state the properties of developments required in these traditional cases. We demonstrate by examples that the calculus of recursively scoped records does not have several of these properties. We introduce the notion of γ-development (a generalization of a traditional notion of developments), and give a general proof of lift and project. Chapter 5 gives the details of the proof for the term calculus and the calculus of recursively scoped records, which follow the lines of the general proof given in this chapter.

4.2 Marked Calculi

We prove lift and project via marking a redex and tracing its residuals after a reduction. In order to formalize this approach, we need to consider a marked version of the calculus: a calculus in which some redexes in a term may be marked. We also need to define what a redex is and what are residuals of a redex with respect to a reduction step. These definitions are calculus-specific. We parameterize the definition of a redex over the calculus rules, and give an axiomatic definition of a residuals, i.e. we state the properties that residuals must have.

In section 4.3.1, we sketch proofs of confluence and standardization via marked reductions. We will take these proofs as an example of the technique of marked reductions. In section 4.4 we show why the approach taken in these proofs cannot be applied directly to the record calculus, and define a notion of γ-developments that allows us to prove lift and project via marked reductions.

4.2.1 Redexes and Residuals

The notion of a *redex* is commonly used and intuitively clear, but a formal definition of a redex presents some challenges. Consider the term $(\lambda x.2+3)\ @\ (\lambda y.2+3)$ in the λ-calculus. If we say that $2+3$ is a redex in this term, then we get the following ambiguity: both $(\lambda x.5)\ @\ (\lambda y.2+3)$ and $(\lambda x.2+3)\ @\ (\lambda y.5)$ are obtained by reducing a redex $2+3$ in the original term. In this presentation we resolve the ambiguity by specifying *subterm occurrences* of redexes in the term.

One may argue that specifying a pair of terms M and L such that $M \longrightarrow L$ uniquely defines the redex reduced in the reduction, so there is no need to annotate the reduction with the subterm occurrence of the redex. For instance, knowing the resulting term in a reduction of the term $(\lambda x.2+3)\ @\ (\lambda y.2+3)$ in the example above allows us to determine which of the two occurrences of $2+3$ has been reduced. However, consider the following term from the call-by-name λ-calculus: $(\lambda x.x)\ @\ ((\lambda y.y)\ @\ (\lambda z.z))$. Reducing the first application gives $(\lambda y.y)\ @\ (\lambda z.z)$, and reducing the second one results in $(\lambda x.x)\ @\ (\lambda z.z)$, but the latter two terms are the same up to α-renaming! Moreover, if we had chosen to use abbreviation I for $\lambda x.x$, the original term would have been written as $I\ @\ (I\ @\ I)$, and the results of *both* reductions as $I\ @\ I$ (see example 3.1.19 in [Bar84] due to Levy). We avoid dealing with these ambiguities by annotating reductions with a subterm occurrence of the redex to identify the reductions uniquely. Subterm occurrences are defined via one-hole contexts enclosing a subterm in a term:

Definition 4.2.1 (One-hole context). A *one-hole context* (also called a *context*) is the result of replacing one subtree of a syntactic tree of a term in a calculus by the symbol $\Box$, called *the context hole*. We use $\mathbb{A}, \mathbb{B}, \mathbb{C}$ to range over contexts in a calculus. We use $\mathbb{E}$ to range over evaluation contexts of a calculus. The notation $\mathbb{C}\{N\}$ stands

for the result of filling the context hole of $\mathbb{C}$ with a term N. See section 2.2 for examples of contexts and evaluation contexts. □

As an example, consider one-hole contexts in λ-calculus: a context $\Box$ @ 2 can be obtained by replacing the subtree $\lambda x.x$ by $\Box$ in the term $(\lambda x.x)$ @ 2. Note that $(\lambda x.x)$ @ $2 = (\Box$ @ $2)\{\lambda x.x\}$ in the above notations, i.e. the term is the result of filling the context $\Box$ @ 2 with the term $\lambda x.x$.

Definition 4.2.2 (Subterm occurrence). A *subterm occurrence* is a pair $(\mathbb{C}, N)$ of a one-hole context and a term. We say that two subterm occurrences are equal, and write $(\mathbb{C}_1, N_1) = (\mathbb{C}_2, N_2)$ if and only if $\mathbb{C}_1 = \mathbb{C}_2$ and $N_1 = N_2$. □

Subterm occurrences are used to identify a particular subtree of a syntactic tree of a term. For instance, a subterm occurrence $((\lambda x.\Box)$ @ $2, 2)$ identifies the first occurrence of the constant 2 in the term $(\lambda x.2)$ @ 2.

An alternative approach to identifying a particular subterm occurrence in a term is by giving the path to it from the root of the syntactic tree. Such a path is unique. This approach is used in [HL91]. The two approaches are equivalent. We find our approach to be easier to extend to multi-hole contexts which we use in computational soundness proofs for particular calculi.

Definition 4.2.2 makes it possible to distinguish the two redexes in the term $(\lambda x.2 + 3)$ @ $(\lambda y.2 + 3)$: the subterm occurrence of the first redex is written as $((\lambda x.\Box)$ @ $(\lambda y.2 + 3), 2 + 3)$, and the subterm occurrence of the second one is $((\lambda x.2 + 3)$ @ $(\lambda y.\Box), 2 + 3)$.

Using these notations, we formalize the notion of a redex. We assume that no two reduction rules have the same redex.

Definition 4.2.3 (Redex). Suppose the calculus reduction is defined by rules of the form $\mathbb{C}\{R\} \longrightarrow \mathbb{C}\{Q\}$, possibly with some restrictions on $\mathbb{C}$ and R. A subterm

occurrence $(\mathbb{A}, N)$ is called a *redex* if there is a reduction rule such that $\mathbb{A} = \mathbb{C}$, $N = R$, and $\mathbb{A}$ and N satisfy all the restrictions of the rule.

If $(\mathbb{C}, R)$ is a redex, then there exist term Q and $L = \mathbb{C}\{Q\}$ such that $M = \mathbb{C}\{R\} \longrightarrow \mathbb{C}\{Q\} = L$. In this case we write $M \xrightarrow{(\mathbb{C},R)} L$.

If the rule is an evaluation rule, then $(\mathbb{C}, R)$ is an *evaluation redex*, and we write $M \xRightarrow{(\mathbb{C},R)} L$, otherwise $(\mathbb{C}, R)$ is a *non-evaluation redex*, and we write $M \circ\!\!\xrightarrow{(\mathbb{C},R)} L$. We use $\mathbb{E}$ to denote the context of an evaluation redex, which we call an *evaluation context*. □

The following definition extends these notations to sequences of redexes:

Definition 4.2.4. Suppose we have a sequence of reduction steps $M \xrightarrow{(\mathbb{C}_1,R_1)} M_1 ... M_{n-1} \xrightarrow{(\mathbb{C}_n,R_n)} N$. Then we denote it by $M \xrightarrow{S}{}^{*} N$, where $S = (\mathbb{C}_1, R_1); ...; (\mathbb{C}_n, R_n)$ is a sequence of redexes. The notations $\overset{S}{\Longrightarrow}{}^{*}$ and $\circ\!\!\xrightarrow{S}{}^{*}$ are defined analogously. □

The definition and the assumption below abstract over details of particular calculi and specify general requirements that residuals have to satisfy. Each calculus provides calculus-specific details of the definition.

Definition 4.2.5 (Axiomatic definition of a residual). Let $(\mathbb{C}_1, R_1)$ and $(\mathbb{C}_2, R_2)$ be two redexes such that $\mathbb{C}_1\{R_1\} = \mathbb{C}_2\{R_2\} = M$, and let $M \xrightarrow{(\mathbb{C}_1,R_1)} N$. A *set of residuals of* $(\mathbb{C}_2, R_2)$ *w.r.t.* $(\mathbb{C}_1, R_1)$ denoted by $(\mathbb{C}_2, R_2)/(\mathbb{C}_1, R_1)$ is a set of redexes in N uniquely defined by the calculus. If $(\mathbb{C}_3, R_3) \in (\mathbb{C}_2, R_2)/(\mathbb{C}_1, R_1)$, then $(\mathbb{C}_3, R_3)$ is a *residual of* $(\mathbb{C}_2, R_2)$ *w.r.t.* $(\mathbb{C}_1, R_1)$. If a set $(\mathbb{C}_2, R_2)/(\mathbb{C}_1, R_1)$ consists of a single element $(\mathbb{C}_3, R_3)$, then sometimes we write $(\mathbb{C}_3, R_3) = (\mathbb{C}_2, R_2)/(\mathbb{C}_1, R_1)$, omitting the set notation. □

Assumption 4.2.6 (Unique ancestor). Let $(\mathbb{C}_1, R_1), (\mathbb{C}_2, R_2)$, and $(\mathbb{C}_3, R_3)$ be 3 redexes in M s.t. $(\mathbb{C}_2, R_2) \neq (\mathbb{C}_3, R_3)$, and suppose $M \xrightarrow{(\mathbb{C}_1,R_1)} N$. Then it must be the case that $(\mathbb{C}_2, R_2)/(\mathbb{C}_1, R_1) \cap (\mathbb{C}_3, R_3)/(\mathbb{C}_1, R_1) = \emptyset$. □

The assumption states that a residual originates from only one redex in the term being reduced. Note that in the assumption we do not require that $(\mathbb{C}_2, R_2)$ and $(\mathbb{C}_3, R_3)$ are different from $(\mathbb{C}_1, R_1)$. This is because in some calculi (in particular, in the calculus of records) $(\mathbb{C}_1, R_1)/(\mathbb{C}_1, R_1)$, i.e. a set of residuals of a redex w.r.t. itself, may be non-empty.

Definition 4.2.7. We extend definition 4.2.5 to sequences of reductions and to sets of redexes as follows:

1. Let $F = \{(\mathbb{C}_i, R_i) \mid 0 \leq i \leq n\}$ be a set of redexes in a term M, and let $M \xrightarrow{(\mathbb{C},R)} N$. Then *the set of residuals of F w.r.t.* $(\mathbb{C}, R)$ is denoted and defined as:
$$F/(\mathbb{C}, R) = \bigcup_{i=0}^{i \leq n} (\mathbb{C}_i, R_i)/(\mathbb{C}, R).$$

2. Let S be a reduction sequence $M \xrightarrow{S}{}^{*} N$ and $F = \{(\mathbb{C}_i, R_i) \mid 0 \leq i \leq n\}$ be a finite set of redexes in M. *The set of residuals of F w.r.t. a sequence S* is denoted and inductively defined as follows:

   - If $S = \epsilon$, then $F/S = F$;
   - If $S = (\mathbb{C}_1, R_1); S_1$, then $(\mathbb{C}, R)/S = ((\mathbb{C}, R)/(\mathbb{C}_1, R_1))/S_1$;
   - $F/S = \bigcup_{i=0}^{i \leq n} (\mathbb{C}_i, R_i)/S$.

   We use the notation $(\mathbb{C}, R)/S$ to denote $\{(\mathbb{C}, R)\}/S$.

□

4.2.2 Reductions with Marked Redexes

A common technique for reasoning about the interaction of two reduction steps originating from the same term is to mark one of the redexes and trace its residuals in the

term obtained by reducing the other redex. We use marked reductions for proving lift and project. See section 4.8 for details.

The technique for proving confluence and standardization via marking of redexes is described in detail in [Bar84]. The essence of the technique is that instead of the original calculus, we consider a calculus with terms in which some redexes may be marked. Then we show the desired properties for the calculus of marked terms and translate them back to the original calculus by erasing the marks.

In this presentation we assume that if M, N, L range over terms of a calculus, then M', N', L' range over terms of the marked version of the calculus, i.e. some redexes in these terms are marked. We show marked redexes by underlining. We assume that there is only one kind of a mark for redexes, i.e. every two marked terms are marked the same way. We define marked(M') to be the set of all marked redexes in M'.

Example 4.2.8 (Set of marked redexes in a term). Consider the following term in the call-by-value λ-calculus:

$$\begin{aligned} M' &= (\lambda x.\underline{((\lambda v.v) \mathbin{@} 1)}) \mathbin{@} (y \mathbin{@} \underline{((\lambda z.z) \mathbin{@} (\lambda w.w))}), \\ \text{marked}(M') &= \{((\lambda x.\Box) \mathbin{@} (y \mathbin{@} ((\lambda z.z) \mathbin{@} (\lambda w.w))), (\lambda v.v) \mathbin{@} 1), \\ &\quad ((\lambda x.((\lambda v.v) \mathbin{@} 1)) \mathbin{@} (y \mathbin{@} \Box), (\lambda z.z) \mathbin{@} (\lambda w.w))\}. \end{aligned}$$

Here $(\lambda x.\Box) \mathbin{@} (y \mathbin{@} ((\lambda z.z) \mathbin{@} (\lambda w.w)))$ is the context enclosing the first marked redex $(\lambda v.v) \mathbin{@} 1$, and $(\lambda x.((\lambda v.v) \mathbin{@} 1)) \mathbin{@} (y \mathbin{@} \Box)$ is the context enclosing the second marked redex $((\lambda z.z) \mathbin{@} (\lambda w.w))$. □

Note that only redexes, but not arbitrary applications or expressions, can be marked. For instance, the term $\underline{(\lambda x.x) \mathbin{@} ((\lambda y.y) \mathbin{@} (\lambda w.w))}$ is a valid term in a marked extension of the call-by-name λ-calculus, but not of the call-by-value λ-

calculus. This is because in the call-by-name calculus β-redexes are of the form $(\lambda x.M)\ @\ N$, so $(\lambda x.M)\ @\ N$ is a β-redex, but in the call-by-value calculus β-redexes are of the form $(\lambda x.M)\ @\ V$, i.e. the operand is a value, so the above term is not a β-redex.

We introduce the following "erasure" function: $|M'| = M$ for a marked term M' and an unmarked term M if M is obtained from M' by removing all the markings of all redexes. For instance,

$$|\underline{(\lambda y.((\lambda x.x)\ @\ 2))\ @\ (\lambda z.\underline{3+4})}| = (\lambda y.((\lambda x.x)\ @\ 2))\ @\ (\lambda z.3+4).$$

Given a definition of a reduction and an evaluation step in the original (i.e. unmarked) calculus, we define these notions in the marked version of the calculus as follows:

Definition 4.2.9 (Reductions on marked terms). $M' \xrightarrow{(\mathbb{C},R)} N'$ if and only if $|M'| \xrightarrow{(\mathbb{C},R)} |N'|$ and $\text{marked}(N') = \text{marked}(M')/(\mathbb{C}, R)$. We write $M' \xRightarrow{(\mathbb{C},R)} N'$ if and only if $M' \xrightarrow{(\mathbb{C},R)} N'$ and $|M'| \xRightarrow{(\mathbb{C},R)} |N'|$. Similarly, $M' \circ\!\!\xrightarrow{(\mathbb{C},R)} N'$ if and only if $M' \xrightarrow{(\mathbb{C},R)} N'$ and $|M'| \circ\!\!\xrightarrow{(\mathbb{C},R)} |N'|$. □

Note that a calculus of terms with marked redexes has somewhat different reduction rules from its unmarked version. For instance, the β-reduction rule for the marked extension of the call-by-value λ-calculus splits into two cases, one when the redex is marked, and the other one when it is not:

$$\begin{aligned} (\lambda x.M')\ @\ N' &\longrightarrow M'[x := N'] \\ \underline{(\lambda x.M')\ @\ N'} &\longrightarrow M'[x := N']. \end{aligned}$$

One has to be careful about how properties of the calculus of marked terms

are translated into those of the corresponding unmarked calculus. The translation from the marked calculus to the unmarked one is quite straightforward: $M' \longrightarrow N'$ in the marked calculus implies $|M'| \longrightarrow |N'|$.

Clearly, the inverse property does not hold: $((\lambda y.y)\ @\ 3)\ @\ 5 \longrightarrow (\lambda y.y)\ @\ 3$ in the unmarked calculus, but $((\lambda y.y)\ @\ 3)\ @\ 5 \not\longrightarrow \underline{(\lambda y.y)\ @\ 3}$ in the marked calculus. This is because the redex $(\lambda y.y)\ @\ 3$ is not marked in the first term of the reduction and is marked in the second, so the reduction does not satisfy definition 4.2.9.

We use the following property of reductions of marked terms:

Lemma 4.2.10. *If marked*$(M') = \emptyset$ *and* $M' \longrightarrow^* N'$*, then marked*$(N') = \emptyset$. □

Proof. This follows directly from definition 4.2.7. □

4.2.3 Developments

In this section, we define the notions of development and complete development and state useful properties of developments. Section 4.3 shows how these properties may be used in proving confluence and standardization of a calculus. Note that the development reduction is defined in a marked calculus, since it reduces only marked redexes.

Definition 4.2.11 (Development reduction). Let M', N' be two terms in a marked calculus. We say that M' reduces to N' by a *development reduction* (denoted $M' \underset{dev}{\overset{(\mathbb{C},R)}{\rightarrow}} N'$) if $M' \xrightarrow{(\mathbb{C},R)} N'$ and $(\mathbb{C}, R) \in \text{marked}(M')$. □

Definition 4.2.12 (Complete development). We say that a sequence $M' \underset{dev}{\rightarrow}^* N'$ is a *complete development*, denoted by $\underset{cd}{\rightarrow}^*$, if $\text{marked}(N') = \emptyset$. □

In a "good-case scenario", developments are bounded, i.e. for every term that has some marked redexes there exists a limit on the length of development sequences originating from the term.

Definition 4.2.13 (Maximal length of γ-development). Let MAX$[M']$ denote the number of steps in the longest reduction $M' \underset{dev}{\rightarrow}^* N'$ if such a reduction exists. MAX$[M'] = \perp$ otherwise. □

Property 4.2.14 (Boundedness of developments). *A calculus has the* bounded developments *property if for all M' there exists an integer number n such that $MAX[M'] = n$.* □

The boundedness of developments implies finiteness of developments:

Property 4.2.15 (Finiteness of developments). *A calculus has finiteness of developments if there is no infinite sequence $M'_1 \underset{dev}{\rightarrow} M'_2 \underset{dev}{\rightarrow} ....$* □

Note that finiteness of developments does not imply boundedness of developments: it is theoretically possible to have a term M' such that there is no infinite development reduction originating at M', but there are finite developments of arbitrary length. The property commonly known as "finiteness of developments" often, as in the case of [Bar84] or [AF97], is proven by constructing an integer-valued (or a multiset-valued, as in [AF97]) function on marked terms, called *measure*, and showing that every development reduction step reduces the measure. The measure of a term serves as a limit on the length of the reduction. Proofs based on "finiteness" of developments usually refer to the measure of a term as the limit of the length of the development reduction (see Corollary 11.2.22 in [Bar84]). While finiteness of developments may be sufficient for proofs of confluence of the calculus, boundedness is required for traditional proofs of standardization. See the proof of lemma 4.3.3 for details.

In addition to boundedness, we would like developments to have the following two properties:

Property 4.2.16 (Weak confluence of developments). *A calculus has a weak confluence of developments if* $\underset{dev}{\rightarrow}$ *is weakly confluent, i.e. if* $M'_1 \underset{dev}{\rightarrow} M'_2$, $M'_1 \underset{dev}{\rightarrow} M'_3$, *then there exists* M'_4 *such that* $M'_2 \underset{dev}{\rightarrow}^* M'_4$ *and* $M'_3 \underset{dev}{\rightarrow}^* M'_4$. □

Property 4.2.17 (Standardization of developments). *A calculus has a standardization of developments if* $M' \underset{dev}{\rightarrow}^* N'$ *implies that there exists* L' *such that* $M' \underset{dev}{\Rightarrow}^* L' \underset{dev}{\circ\!\!\rightarrow}^* N'$. □

Another useful property of developments gives the ability to replace a so-called non-standard pair of development steps (i.e. a non-evaluation step followed by an evaluation step) by a development sequence which begins with a standard step.

Property 4.2.18 (Replacing a non-standard development pair). *Given a sequence* $M'_1 \underset{dev}{\circ\!\!\rightarrow} M'_2 \underset{dev}{\Rightarrow} M'_3$, *there exists* M' *s.t.* $M'_1 \underset{dev}{\Rightarrow} M' \underset{dev}{\rightarrow}^* M'_3$. □

The next section details how these properties of developments are used in proving properties of the calculi. Section 4.4 discusses which of these properties do not hold in the calculus of records and motivates the need for definition of γ-developments.

4.3 The "Good Case" Scenario: λ-calculi

In this section we show how properties of developments introduced in the previous section allow us to prove confluence and standardization. The call-by-value and the call-by-name λ-calculi have all of the "good" properties of developments. Confluence and standardization of these calculi can be proven by the approach given in this section.

4.3.1 Parallel Moves Lemma and Confluence

Weak confluence of developments, combined with finiteness, implies that all complete developments of a term end at the same term. This property is known as *confluence of developments*.

A core property of calculi in the traditional approach is referred to as Parallel Moves Lemma (see [HL91]). Its proof, or, more precisely, the proof of its marked version (property 4.3.2), uses confluence of developments.

Property 4.3.1 (Parallel Moves Lemma). *A calculus satisfies parallel moves lemma if the following holds: if* $M_1 \xrightarrow{(\mathbb{C}_1,R_1)} M_2$ *and* $M_1 \xrightarrow{(\mathbb{C}_2,R_2)} M_3$*, then there exists* M_4 *such that* $M_2 \xrightarrow{(\mathbb{C}_2,R_2)/(\mathbb{C}_1,R_1)}{}^{*} M_4$ *and* $M_3 \xrightarrow{(\mathbb{C}_1,R_1)/(\mathbb{C}_2,R_2)}{}^{*} M_4$*.* □

Note that in the parallel moves lemma, the two reduction sequences terminating at M_4 (i.e., $M_2 \xrightarrow{(\mathbb{C}_2,R_2)/(\mathbb{C}_1,R_1)}{}^{*} M_4$ and $M_3 \xrightarrow{(\mathbb{C}_1,R_1)/(\mathbb{C}_2,R_2)}{}^{*} M_4$) reduce residuals of the two given redexes $(\mathbb{C}_1, R_1)$ and $(\mathbb{C}_2, R_2)$. In order to trace residuals of a redex, one has to mark the redex. Then the following property of the marked calculus implies the parallel moves lemma in the unmarked calculus. It is illustrated in figure 4.1.

$$\begin{array}{ccc} (\lambda x.x \mathbin{@} x) \mathbin{@} (\lambda y.\underline{2+3}) & \longrightarrow & (\lambda y.\underline{2+3}) \mathbin{@} (\lambda y.\underline{2+3}) \\ \Big\downarrow cd & & \Big\downarrow cd \\ (\lambda x.x \mathbin{@} x) \mathbin{@} (\lambda y.5) & \xrightarrow[*]{} & (\lambda y.5) \mathbin{@} (\lambda y.5) \end{array}$$

Figure 4.1: Example of the parallel moves property

Property 4.3.2 (Parallel Moves Lemma in a Marked Calculus). *A marked calculus satisfies parallel moves lemma if the following holds: for any* M_1', M_2'*, and* M_3' *such that* $M_1' \xrightarrow{(\mathbb{C}_1,R_1)} M_2'$ *and* $M_1' \xrightarrow{(\mathbb{C}_2,R_2)} M_3'$ *there exists* M_4' *such that* $M_2' \xrightarrow{(\mathbb{C}_2,R_2)/(\mathbb{C}_1,R_1)}{}^{*} M_4'$ *and* $M_3' \xrightarrow{(\mathbb{C}_1,R_1)/(\mathbb{C}_2,R_2)}{}^{*} M_4'$*.* □

Note that this property is stronger than property 4.3.1 since it requires that the appropriate marked reductions exist. Both the call-by-value and the call-by-name λ-calculi satisfy the marked parallel moves lemma.

If we additionally assume that $(\mathbb{C}, R)/(\mathbb{C}, R) = \emptyset$ for any redex $(\mathbb{C}, R)$, and if we mark one of the two redexes in M_1', i.e. $\text{marked}(M_1') = \{(\mathbb{C}_1, R_1)\}$, then $\text{marked}(M_2') = \emptyset$, which implies that $\text{marked}(M_4') = \emptyset$. Then $M_3' \xrightarrow{(\mathbb{C}_1,R_1)/(\mathbb{C}_2,R_2)}{}^{*} M_4'$ must be a complete development.

The property that $(\mathbb{C}, R)/(\mathbb{C}, R) = \emptyset$ for any $(\mathbb{C}, R)$ holds in the call-by-value and the call-by-name λ-calculi (as well as in other calculi that satisfy parallel moves lemma, such as the call-by-need calculus introduced in [AF97]), but not in the calculus of records. See section 4.4 for details.

It is a well-known result that the parallel moves lemma and the finiteness of developments (properties 4.3.2 and 4.2.15) together imply confluence of a calculus (see [Bar84] for details). Confluence of developments is also an important property which justifies parallel reductions ([Tak95]) in the λ-calculus. A parallel reduction reduces simultaneously a set of redexes in a term (i.e. performs a complete development of the set of redexes). To be able to use parallel reductions for proving confluence of the calculus, the result of simultaneous reduction of the set of redexes must not depend on the order in which the redexes are reduced.

In the section 4.4 we show that the parallel moves lemma does not hold in the calculus of records. However, we are able to achieve a restricted version of this property by introducing the notion of γ-developments.

4.3.2 Standardization of Developments

Standardization of developments is one of the key elements in proving the lift property. Since lift is equivalent to standardization (see section 3.4.3), standardization of developments can be used for proving standardization of the calculus.

It may seem that proving standardization of developments should be just as difficult as proving standardization of the calculus. The rest of the section shows how the boundedness of developments makes it much easier to prove standardization of developments than to prove standardization of the calculus. This way of proving standardization of developments works in the case when the calculus has the property 4.2.18. The property holds for both the call-by-value and the call-by-name λ-calculi. In section 4.4 we show that the calculus of records does not have this property.

Standardization of developments is implied by property 4.2.18 and the boundedness of developments.

Property 4.2.18 says that a so-called non-standard pair of development steps (i.e., a non-evaluation step followed by an evaluation step) can be replaced by a development sequence which begins with a standard step. The length of the rest of the sequence is unspecified, and the sequence may have other non-standard pairs. However, boundedness of developments allows us to use this property to transform any development sequence into a standard one, as shown in the lemma below:

Lemma 4.3.3. *If a calculus has properties 4.2.14 and 4.2.18, then it has standardization of developments (property 4.2.17).* □

Proof. Let $M' \underset{dev}{\rightarrow}{}^{*} M'_1$ be a development sequence. The following construction repeatedly applied to this sequence will in a finite number of iterations produce a standard sequence $M' \underset{dev}{\Rightarrow}{}^{*} M'_2 \underset{dev}{\circ\!\!\rightarrow}{}^{*} M'_1$:

- *Step 1.* Check if the given sequence $M' \underset{dev}{\rightarrow}^* M'_1$ is standard. If yes, then the construction is finished. Otherwise go to step 2.

- *Step 2.* Since the given sequence is not standard, it can be parsed as follows:

$$M' \underset{dev}{\Rightarrow}^* N'_0 \underset{dev}{\circ\!\!\rightarrow}^* N'_1 \underset{dev}{\circ\!\!\rightarrow} N'_2 \underset{dev}{\Rightarrow} N'_3 \underset{dev}{\rightarrow}^* M'_1.$$

  Then by property 4.2.18 there exists L'_2 such that $N'_1 \underset{dev}{\Rightarrow} L'_2 \underset{dev}{\rightarrow}^* N'_3$. Then we replace the initial reduction sequence by the sequence

$$M' \underset{dev}{\Rightarrow}^* N'_0 \underset{dev}{\circ\!\!\rightarrow}^* N'_1 \underset{dev}{\Rightarrow} L'_2 \underset{dev}{\rightarrow}^* N'_3 \underset{dev}{\rightarrow}^* M'_1.$$

  and repeat step 1.

To show that this construction terminates, we associate a pair of non-negative integers (n_1, n_2) to every sequence S s.t. $M' \underset{dev}{\overset{S}{\rightarrow}}^* M'_1$ in the following way: suppose S is not standard, then it can be parsed as above, i.e.

$$M' \underset{dev}{\overset{S_1}{\Rightarrow}}^* N'_0 \underset{dev}{\circ\overset{S_2}{\rightarrow}}^* N'_1 \underset{dev}{\circ\!\!\rightarrow} N'_2 \underset{dev}{\Rightarrow} N'_3 \underset{dev}{\rightarrow}^* M'_1.$$

Then n_1 is the number of evaluation steps immediately following M', i.e. the length of the sequence S_1, and n_2 is the number of non-evaluation steps immediately following S_1, i.e. $n_2 = m_2 + 1$, where m_2 is the length of the sequence S_2. If S is standard, then it can be parsed as

$$M' \underset{dev}{\overset{S_1}{\Rightarrow}}^* N'_0 \underset{dev}{\circ\overset{S_2}{\rightarrow}}^* M'_1,$$

and in this case n_1 is the length of S_1, and n_2 is the length of S_2.

We consider pairs (n_1, n_2) to be ordered lexicographically, i.e. $(n_1, n_2) < (n'_1, n'_2)$ if and only if $(n_1 < n'_1)$ or $(n_1 = n'_1$ and $n_2 > n'_2)$. Note the reversal of

the sign in the second component of a pair! It is easy to observe that if (n_1, n_2) corresponds to a sequence S before an iteration of the construction, and (n_1', n_2') corresponds to S' after the iteration, then $(n_1, n_2) > (n_1', n_2')$: either the subsequence $N_0' \underset{dev}{\overset{S_2}{\circ\!\!\rightarrow}}{}^{*} N_1'$ is non-empty, in which case n_1 does not change, and n_2 is decreases by 1, or S_2 is empty, and then the sequence $M' \underset{dev}{\overset{S_1}{\Longrightarrow}}{}^{*} N_0'$ is followed by at least one more evaluation step, so n_1 increases.

On the other hand, if (n_1, n_2) is associated with a sequence S s.t. $M' \underset{dev}{\overset{S}{\rightarrow}}{}^{*} M_1'$, then $(n_1, n_2) \leq (\mathrm{MAX}[M'], 0)$, since a development of M' can not have more than $\mathrm{MAX}[M']$ steps. Therefore the construction terminates. □

4.3.3 Lift and Project in the Call-by-value λ-calculus

The call-by-value λ-calculus has confluence and standardization, so lift and project are not needed for the computational soundness proof in this calculus. However, we will show how it would be possible to prove lift and project in this calculus. We will use these proof ideas as intuition for the proofs of lift and project in the calculus of records. In this section we focus on the proof of lift; the proof of project is analogous.

We formulate lift and project for the marked calculus, since we use the technique of marked redexes for proofs of these properties. Using the the notion of development (see definition 4.2.11), we formulate the marked version of lift and project as follows:

Property 4.3.4 (Lift with developments). *A calculus has the* lift property *if* $\mathit{marked}(M') = \{(\mathbb{C}, R)\}$ *and* $M' \underset{dev}{\overset{(\mathbb{C},R)}{\circ\!\!\rightarrow}} N' \Longrightarrow^{*} N_1'$ *imply that there exists a sequence* $M' \Longrightarrow^{*} M_1' \underset{cd}{\circ\!\!\longrightarrow}{}^{*} N_1'$. □

Property 4.3.5 (Project with developments). *A calculus has the* project property *if* $\mathit{marked}(M') = \{(\mathbb{C}, R)\}$, $M' \underset{dev}{\overset{(\mathbb{C},R)}{\circ\!\!\rightarrow}} N'$, *and* $M' \Longrightarrow^{*} M_1'$ *imply that there exist*

terms M_2', N_1' *such that* $M_1' \Rightarrow^* M_2'$, $N' \Rightarrow^* N_1'$, *and* $M_2' \underset{cd}{\circ\!\!\rightarrow}^* N_1'$. □

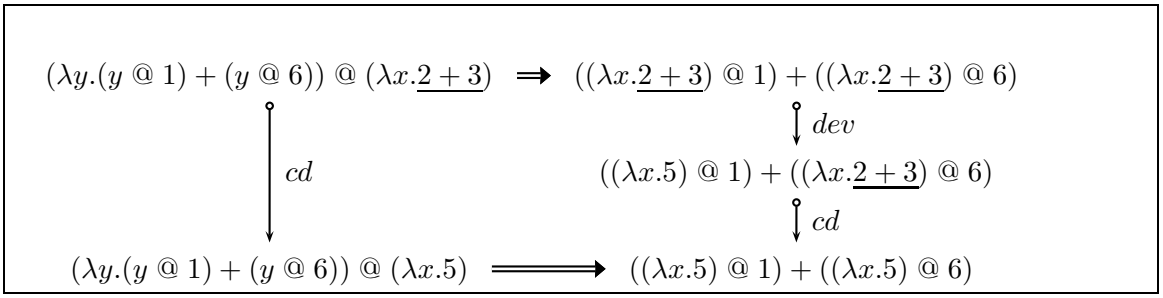


Figure 4.2: Lift and project in marked call-by-value λ-calculus.

Figure 4.2 illustrates both lift (property 3.4.1) and project (property 3.4.2) in the call-by-value λ-calculus. If the reductions $(\lambda y.(y \mathbin{@} 1) + (y \mathbin{@} 6)) \mathbin{@} (\lambda x.\underline{2+3}) \underset{dev}{\circ\!\!\rightarrow}$ $(\lambda y.(y \mathbin{@} 1) + (y \mathbin{@} 6)) \mathbin{@} (\lambda x.5)$ and $(\lambda y.(y \mathbin{@} 1) + (y \mathbin{@} 6)) \mathbin{@} (\lambda x.5) \Rightarrow ((\lambda x.5) \mathbin{@} 1) +$ $((\lambda x.5) \mathbin{@} 6)$ are given, then the figure illustrates lift. If the same non-evaluation reduction $(\lambda y.(y \mathbin{@} 1) + (y \mathbin{@} 6)) \mathbin{@} (\lambda x.\underline{2+3}) \underset{dev}{\circ\!\!\rightarrow} (\lambda y.(y \mathbin{@} 1) + (y \mathbin{@} 6)) \mathbin{@} (\lambda x.5)$ and the other evaluation step $(\lambda y.(y \mathbin{@} 1) + (y \mathbin{@} 6)) \mathbin{@} (\lambda x.\underline{2+3}) \Rightarrow ((\lambda x.\underline{2+3}) \mathbin{@} 1) +$ $((\lambda x.\underline{2+3}) \mathbin{@} 6)$ are given, then the figure illustrates project. Note that the arrows going down are developments since they reduce marked redexes. If the resulting term does not have any marked redexes, then these reductions are complete developments.

It is important to notice that there is just one redex marked in the term $(\lambda y.(y \mathbin{@} 1) + (y \mathbin{@} 6)) \mathbin{@} (\lambda x.\underline{2+3})$: the redex $\underline{2+3}$. The terms on the bottom row do not have any marked redexes, because the redex $\underline{2+3}$ has been reduced by the reductions represented as the arrows going down. The evaluation step on the top row duplicates the redex $\underline{2+3}$. Both copies of the marked redex are reduced by the reduction steps going down on the right-hand side of the diagram (the development steps). This example will serve as an intuition for the kinds of diagrams we want to construct for the calculus of records.

In order to prove lift and project, we need to show two properties:

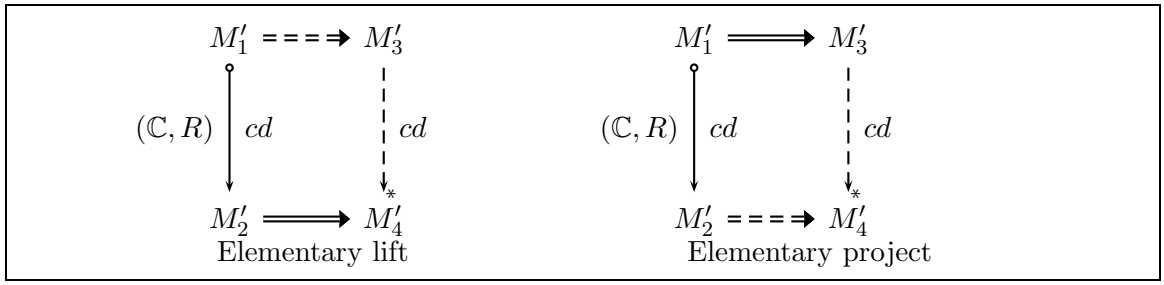


Figure 4.3: Elementary diagrams in call-by-value λ-calculus

Property 4.3.6 (Elementary lift diagram with developments). *Suppose that* $\{(\mathbb{C}, R)\} \in \mathit{marked}(M_1')$. *If* $M_1' \circ\xrightarrow[dev]{(\mathbb{C},R)} M_2' \Longrightarrow M_4'$, *then there exists* M_3' *s.t.* $M_1' \Longrightarrow M_3' \underset{dev}{\rightarrow^*} M_4'$ *(see figure 4.3).* □

Property 4.3.7 (Elementary project diagram with developments). *Suppose that* $\{(\mathbb{C}, R)\} \in \mathit{marked}(M_1')$. *If* $M_1' \circ\xrightarrow[dev]{(\mathbb{C},R)} M_2'$, *and* $M_1' \Longrightarrow M_3'$, *then there exists* M_4' *s.t.* $M_2' \Longrightarrow M_4'$, *and* $M_3' \underset{dev}{\rightarrow^*} M_4'$ *(see figure 4.3).* □

These properties are shown by cases. In particular, these are some of the cases of a proof of the parallel moves lemma.

Based on the elementary properties, we can construct and prove the diagram shown in figure 4.4 below. This diagram is obtained by tiling several copies of the diagram in property 4.3.6. The proof of lift (figure 4.5) uses the tiling property and standardization of developments (property 4.2.17).

4.4 Problems with the Calculus of Records

There are several reasons why the traditional approach does not apply to the calculus of records. First, the calculus does not satisfy parallel moves lemma (property 4.3.1). This is not surprising, since the parallel moves lemma implies confluence, but the calculus is not confluent.

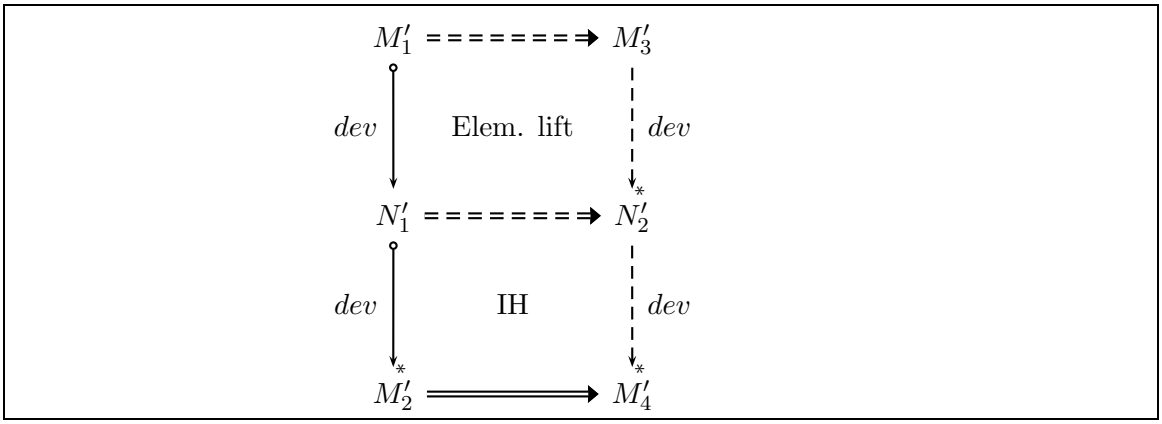


Figure 4.4: Lift tiling diagram in call-by-value λ-calculus

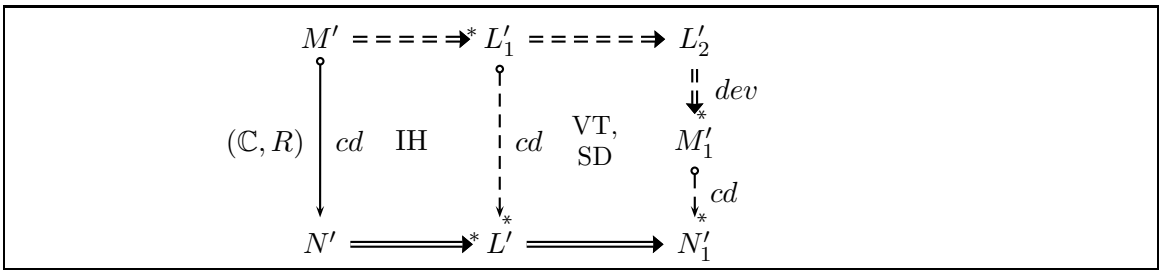


Figure 4.5: Proof of lift in call-by-value λ-calculus

Example 4.4.1 (Parallel moves lemma fails). The non-confluence example introduced in section 1.1.3 serves as a counter-example to parallel moves lemma. It is illustrated in figure 4.6.

$$[A \mapsto \lambda x.B, B \mapsto \lambda y.A] \quad \circ\!\!\rightarrow \quad [A \mapsto \lambda x.\lambda y.A, B \mapsto \lambda y.A]$$
$$[A \mapsto \lambda x.B, B \mapsto \lambda y.A] \quad \circ\!\!\rightarrow \quad [A \mapsto \lambda x.B, B \mapsto \lambda y.\lambda x.B]$$

□

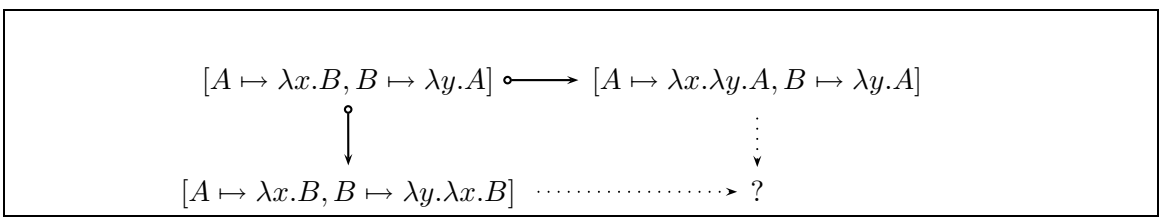


Figure 4.6: Parallel moves lemma fails

In the above example, both reduction steps are non-evaluation steps. But lift and project deal only with cases when at least one step originating from a term is an evaluation step, therefore in proofs of lift and project we do not need to consider cases when two non-evaluation steps originate from a term!

In the rest of the section we work with marked terms, since we are trying to adapt the approach of developments to the record calculus. We use the call-by-value λ-calculus as an example that gives us intuition. It turns out that the approach of developments cannot be adopted in the calculus of records in a straightforward manner. However, we can come up with a notion similar to developments which has the properties necessary to prove lift and project. Section 4.5 introduces this notion, which we call γ-development.

We show that we cannot apply the "good-case" approach to the record calculus in a straightforward way. Let us consider the case of parallel moves lemma when one of the reductions is an evaluation step. Recall that we require confluence of evaluation, so we can expect that a property similar to parallel moves lemma will hold if both reduction steps are evaluation steps. The crucial case is when one of the steps is an evaluation step, and the other is a non-evaluation step.

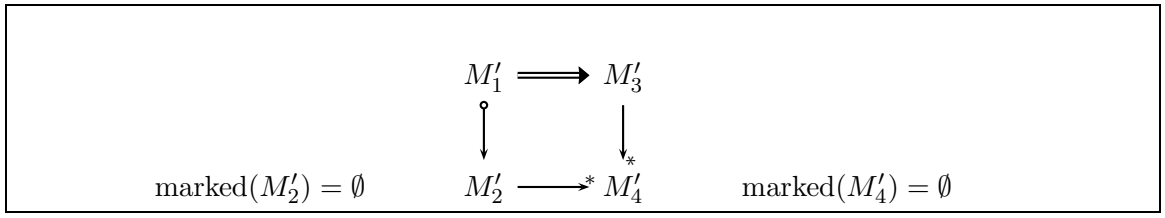


Figure 4.7: An attempt to adapt parallel moves lemma

We would like to get a property similar to property 4.3.2 in the calculus of records. In particular, we would like to mark a redex in the original term, and then get a diagram similar to the parallel moves lemma, as illustrated in figure 4.7.

However, we encounter some problems when we try to adapt this property in

a straightforward way. Consider

$$M_1' = [A \mapsto \lambda x.\underline{A}, B \mapsto A]$$

(in notations of figure 4.7). Here $\underline{A}$ denotes a substitution redex. It is a non-evaluation redex because it appears under a lambda. See definition of calculus of records in section 2.2.3 for details. Reducing the substitution redex in M_1' gives the result

$$M_2' = [A \mapsto \lambda x.\lambda x.\underline{A}, B \mapsto A]$$

But this contradicts our desired property that $\text{marked}(M_2') = \emptyset$! This example illustrates that developments in the calculus of records are in general not finite. Note that the example also shows that $(\mathbb{C}, R)/(\mathbb{C}, R)$ is in general non-empty.

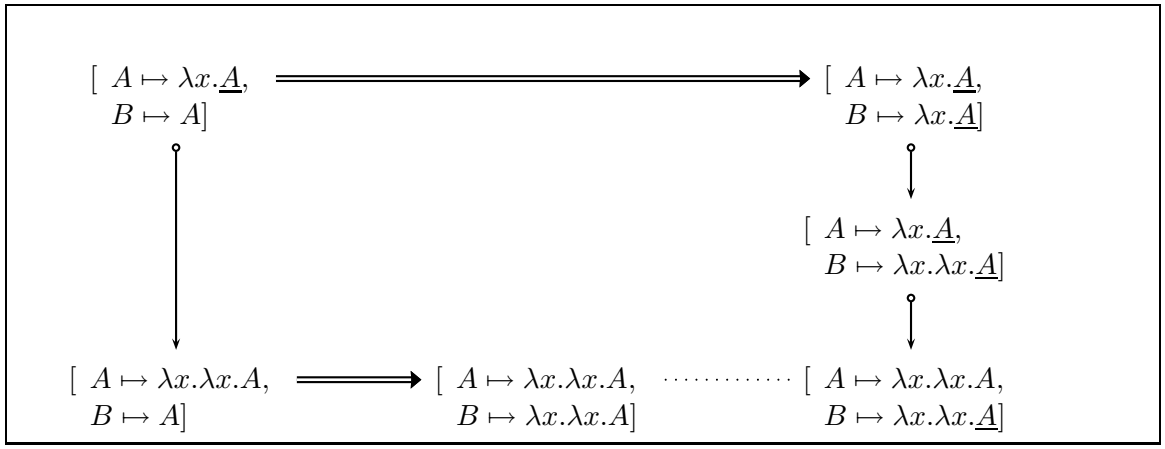


Figure 4.8: Example of non-confluence in marked calculus of records.

Our next attempt to get the desired property is not to mark the result of the substitution if the redex gets substituted into itself. But then we get another problem, as illustrated in figure 4.8. The two records $[A \mapsto \lambda x.\lambda x.A, B \mapsto \lambda x.\lambda x.A]$ and $[A \mapsto \lambda x.\lambda x.A, B \mapsto \lambda x.\lambda x.\underline{A}]$ are not equal in the marked calculus!

These examples show the need for an analog of the development reduction $\underset{dev}{\rightarrow}$

which will allow us to obtain commutative diagrams similar to the parallel moves property which will hold in the marked calculus. Such a reduction, which we call γ-development , is introduced in section 4.5.

4.5 γ-Developments

Let $M' \overset{(\mathbb{C}_1,R_1)}{\underset{dev}{\rightarrow}} N'$, and let $(\mathbb{C}_2, R_2) \in \text{marked}(M')$. All the residuals of $(\mathbb{C}_2, R_2)$ w.r.t. $(\mathbb{C}_1, R_1)$ are marked in N' (by definition 4.2.11 and definition 4.2.9). As we have seen in the previous section, developments in the calculus of records are in general not finite. However, it turns that if change the definition of developments slightly, we get a reduction which is finite for the sets of initially marked redexes which occur in lift and project. We call these reductions γ-*developments* .

A γ-development reduction is the same as a development reduction $\underset{dev}{\rightarrow}$, except for the fact that some redexes that would have been marked in the resulting term of a $\underset{dev}{\rightarrow}$ step are not marked in the result of a γ-development . While definition of γ-development reduction is calculus-specific, the axiomatic definition below states the requirements that the reduction must satisfy.

Definition 4.5.1 (Axiomatic definition of γ-development step). A γ-*development* is defined by a set $dom(\gamma)$ of marked terms (the domain of $\underset{\gamma}{\rightarrow}$) and by a relation $\underset{\gamma}{\rightarrow}$ on marked terms so that the following conditions are satisfied:

1. If $(\mathbb{C}, R)$ is a non-evaluation redex and $\text{marked}(M') = \{(\mathbb{C}, R)\}$, then $M' \in dom(\gamma)$;

2. If $|M'_1| = |M'_2|$, $\text{marked}(M'_2) \subset \text{marked}(M'_1)$, and $M'_1 \in dom(\gamma)$, then $M'_1 \underset{\gamma}{\rightarrow} M'_2$ (in this case we write $M'_1 \overset{\text{e}}{\underset{\gamma}{\rightarrow}} M'_2$);

3. If $M'_1 \underset{\gamma}{\rightarrow} M'_2$ and $\text{marked}(M'_2) \neq \emptyset$, then $M'_2 \in dom(\gamma)$;

4. $M'_1 \overset{(\mathbb{C},R)}{\underset{\gamma}{\rightarrow}} M'_2$ if and only if all of the following is true:

   (a) $|M'_1| \xrightarrow{(\mathbb{C},R)} |M'_2|$,

   (b) $(\mathbb{C}, R) \in \text{marked}(M'_1)$,

   (c) If M'_3 is such that $M'_1 \xrightarrow{(\mathbb{C},R)} M'_3$, then $\text{marked}(M'_2) \subseteq \text{marked}(M'_3)$,

   (d) $((\mathbb{C}, R)/(\mathbb{C}, R)) \cap \text{marked}(M'_2) = \emptyset$.

□

The above definition defines two kinds of γ-development steps: those that erase marks on one or more redexes without changing the term (defined in part 2), and those which reduce a marked redex, similar to regular developments $\underset{dev}{\rightarrow}$ (defined in part 4). We stress this distinction by the following notational convention:

Notation 4.5.2. $\overset{(\mathbb{C},R)}{\underset{\gamma}{\rightarrow}}$ denotes only γ-development steps that reduce a redex (as defined in definition 4.5.14). We call such reductions *non-erasing* γ-development steps.

The notation $\underset{\gamma}{\rightarrow}$ without an annotation may refer to either a $\overset{(\mathbb{C},R)}{\underset{\gamma}{\rightarrow}}$ step or a $\overset{\text{e}}{\underset{\gamma}{\rightarrow}}$ step.

Since S denotes a sequence of redexes, the notation $\overset{S}{\underset{\gamma}{\longrightarrow}}{}^*$ denotes a sequence of non-erasing γ-development steps.

We use the notation S^e for sequences over the set $\{(\mathbb{C}, R)\} \cup \text{e}$, i.e. sequences which consist of redexes and the symbol e. Then $\overset{S^e}{\underset{\gamma}{\longrightarrow}}{}^*$ denotes a sequence of γ-development steps, some of which are erasing steps, and some non-erasing, as specified in the sequence S^e. □

Definition 4.5.3 (Complete γ-developments). A γ-development sequence $M' \underset{\gamma}{\longrightarrow}{}^* N'$ is called a *complete γ-development* if $\text{marked}(N') = \emptyset$. We denote it by $M' \underset{c\gamma}{\longrightarrow}{}^* N'$. □

Definition 4.5.4 (Combined reduction). We say that M' reduces to N' by a *combined reduction*, denoted by $M' \underset{\cup}{\longrightarrow} N'$, if $M' \longrightarrow N'$ or $M' \underset{\gamma}{\rightarrow} N'$. □

Below we define some properties of γ-developments that are helpful for computational soundness proofs.

Definition 4.5.5 (Strong standardization of γ-developments). A calculus has a strong standardization of γ-developments if $M' \underset{c\gamma}{\longrightarrow}^* N'$ implies that there exists a term L' such that $M' \underset{\gamma}{\Rightarrow}^* L' \underset{c\gamma}{\circ\!\!\longrightarrow}^* N'$. □

Note that N' does not have any marked redexes, since the reduction $L' \underset{c\gamma}{\circ\!\!\longrightarrow}^* N'$ is a complete γ-development . Consequently, the reduction $M' \underset{\gamma}{\Rightarrow}^* L' \underset{c\gamma}{\circ\!\!\longrightarrow}^* N'$ is a complete γ-development .

The notion of lockstep equivalent evaluation sequence defined below is used in both lift and project proofs and captures the following intuition: suppose we mark a non-evaluation redex $(\mathbb{C}, R)$ in a term M'_1, and have reduced this redex by a γ-development step to get a term M'_2. Then M'_2 does not have any residuals of $(\mathbb{C}, R)$ by part 4(d) of definition 4.5.1, i.e. it does not have any marked redexes.

Now consider evaluation sequences originating from M'_1 and M'_2. Let $M'_2 \Rightarrow N'_2$, then there is "the same" evaluation redex in M'_1 (up to the subterm $(\mathbb{C}, R)$ reduced in M'_2 but not in M'_1), i.e. $M'_1 \Rightarrow L'_1$. The terms L'_1 and N'_2 differ only by the marked residuals of $(\mathbb{C}, R)$ which are present in L'_1, but not in N'_2. However, after the evaluation step some of these redexes may have become evaluation redexes. Therefore, to synchronize the two sequences for the next evaluation redex reduced in N'_2 one may have to reduce some evaluation residuals of $(\mathbb{C}, R)$ in L'_1. These steps extra steps will be γ-development steps, and account for the extra sequence $L'_2 \underset{\gamma}{\Rightarrow}^* N'_1$ in the definition below.

Definition 4.5.6 (Lockstep equivalent evaluation sequences). An evaluation sequence ${S^e}_1 : M'_1 \underset{\cup}{\Rightarrow}^* N'_1$ is *lockstep equivalent* to an evaluation sequence $S_2 : M'_2 \Rightarrow^* N'_2$ if $\text{marked}(M'_1) = \{(\mathbb{C}, R)\}$, $M'_1 \overset{(\mathbb{C},R)}{\underset{c\gamma}{\circ\!\!\rightarrow}} M'_2$, and one of the following is true:

1. ${S^e}_1 = \epsilon$ and $S_2 = \epsilon$, or
2. there exist L'_1, L'_2, L'_3 such that ${S^e}_1 : M'_1 \overset{{S^e}'_1}{\Longrightarrow}{}^* L'_1 \Rightarrow L'_2 \underset{\gamma}{\Rightarrow}^* N'_1$, $S_2 : M'_2 \overset{S'_2}{\Longrightarrow}{}^* L'_3 \Rightarrow N'_2$, $N'_1 \underset{c\gamma}{\rightarrow}^* N'_2$, and ${S^e}'_1$ is lockstep equivalent to S'_2.

□

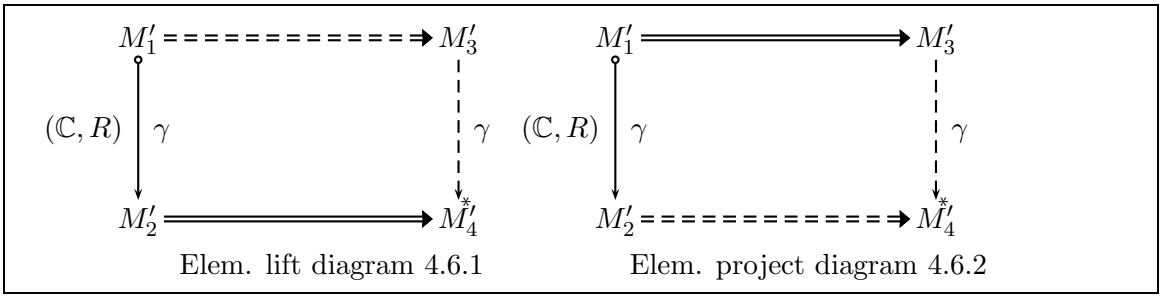
Definition 4.5.7 (Weak standardization of γ-developments). A calculus has a weak standardization of γ-developments if for every M'_1, M'_2, N'_1, N'_2 such that the two evaluation sequences ${S^e}_1 : M'_1 \underset{\cup}{\Rightarrow}^* N'_1$ and $S_2 : M'_2 \Rightarrow^* N'_2$ are lockstep equivalent, there exists a term L' such that $N'_1 \underset{\gamma}{\Rightarrow}^* L' \underset{c\gamma}{\circ\!\!\rightarrow}^* N'_2$. □

Lemma 4.5.8 (Strong standardization implies weak standardization). *Strong standardization of γ-developments (definition 4.5.5) implies weak standardization of γ-developments (definition 4.5.7).* □

Proof. Suppose strong standardization of γ-developments holds. If ${S^e}_1 : M'_1 \underset{\cup}{\Rightarrow}^* N'_1$ and $S_2 : M'_2 \Rightarrow^* N'_2$ are lockstep equivalent evaluation sequences, then by definition 4.5.6 $N'_1 \underset{c\gamma}{\rightarrow}^* N'_2$, so by strong standardization of γ-developments there exists L' such that $N'_1 \underset{\gamma}{\Rightarrow}^* L' \underset{c\gamma}{\circ\!\!\rightarrow}^* N'_2$. □

4.6 Elementary Lift and Project Diagrams

Both lift and project properties are proven via "tiling" diagrams. The following properties, which we call *elementary project and lift diagrams*, are the building blocks (or the "tiles") of these proofs.



Elem. lift diagram 4.6.1 Elem. project diagram 4.6.2

Figure 4.9: Elementary project and lift diagrams

Property 4.6.1 (Elementary lift diagram). *Suppose that* $\{(\mathbb{C}, R)\} \in \mathit{marked}(M_1')$. *If* $M_1' \overset{(\mathbb{C},R)}{\underset{\gamma}{\circ\!\!\longrightarrow}} M_2' \Longrightarrow M_4'$, *then there exists* M_3' *s.t.* $M_1' \Longrightarrow M_3' \underset{\gamma}{\longrightarrow}^* M_4'$ *(see figure 4.9).* □

Property 4.6.2 (Elementary project diagram). *Suppose that* $\{(\mathbb{C}, R)\} \in \mathit{marked}(M_1')$. *If* $M_1' \overset{(\mathbb{C},R)}{\underset{\gamma}{\circ\!\!\longrightarrow}} M_2'$, *and* $M_1' \Longrightarrow M_3'$, *then there exists* M_4' *s.t.* $M_2' \Longrightarrow M_4'$, *and* $M_3' \underset{\gamma}{\longrightarrow}^* M_4'$ *(see figure 4.9).* □

Elementary project diagram above is a restricted form of the parallel moves property: it can be obtained from the parallel moves property by restricting one of the given reductions to be an evaluation step, and the other to be a non-evaluation step. Recall that parallel moves property implies confluence, and therefore does not hold in the calculus of recursively scoped records.

The above diagrams can be viewed as preservation of an evaluation redex by a reduction of a non-evaluation one. To make these properties more intuitive, consider call-by-name and call-by-value λ-calculi. It is easy to see that in these calculi a reduction of a non-evaluation redex cannot create, duplicate, or remove an evaluation redex, thus both elementary diagrams hold. The following examples in the call-by-value calculus illustrate the properties. We give examples of all three possible positions of an evaluation redex:

1. *An outermost redex:* $(\lambda x.(\lambda y.y \mathbin{@} 2)) \mathbin{@} (\lambda z.3 + 4)$. Clearly reductions of non-

evaluation redexes in the operand or the operator cannot change the outermost evaluation redex.

2. *An evaluation redex in the operator:* $((\lambda x.x) \mathbin{@} (\lambda y.y)) \mathbin{@} ((\lambda z.z) \mathbin{@} 2)$. In this example the redex $(\lambda z.z) \mathbin{@} 2$ in the operand is a non-evaluation redex, and reducing this redex does not change the fact that the redex $(\lambda x.x) \mathbin{@} (\lambda y.y)$ in the operator is the evaluation redex. One can also see that if there were redexes inside the evaluation redex, reducing those redexes would not have changed the evaluation redex either.

3. *An evaluation redex in the operand:* $(\lambda x.2 + 3) \mathbin{@} ((\lambda y.(\lambda z.z \mathbin{@} 2)) \mathbin{@} 5)$. Again, reducing non-evaluation redexes does not change the evaluation redex.

Also, if a term does not have an evaluation redex, then the term is either a value or an error (s.a. $3 \mathbin{@} 4$), and clearly no reduction in such a term can create an evaluation redex.

Elementary lift and project diagrams hold in the calculus of recursively scoped records.

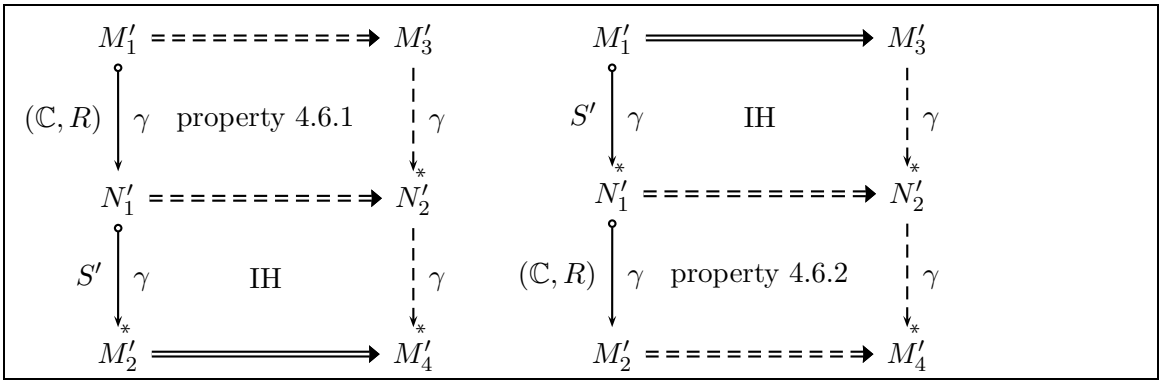


Figure 4.10: Inductive step of proofs of lemmas 4.6.3–4.6.4

Lemma 4.6.3. *Property 4.6.1 implies the following: if* $M_1' \overset{S}{\underset{\gamma}{\circ\!\!\rightarrow}}{}^{*} M_2'$ *and* $M_2' \Longrightarrow M_4'$, *then there exists* M_3' *s.t.* $M_1' \Longrightarrow M_3'$, *and* $M_3' \underset{\gamma}{\longrightarrow}{}^{*} M_4'$. □

Proof. The proof is by induction on n, where n is the number of steps in the non-evaluation sequence S.

The base case when $n = 0$ is trivial. The *base case* $n = 1$ is straightforward by property 4.6.1.

Induction step. The induction step is illustrated on figure 4.10, terms are denoted as on the figure. As the inductive hypothesis, suppose the claim holds for a sequence S' of length n, i.e. if $N_1' \underset{\gamma}{\circ\!\!\rightarrow}^{*} M_2'$ and $M_2' \Longrightarrow M_4'$, then there exists N_2' such that $N_1' \Longrightarrow N_2'$ and $N_2' \underset{\gamma}{\longrightarrow}^{*} M_4'$. Let $M_1' \underset{\gamma}{\circ\!\!\rightarrow}^{*} M_2'$, where $S = (\mathbb{C}, R); S'$, i.e. the length of S is $n + 1$. Then by property 4.6.1 there exists M_3' such that $M_1' \Longrightarrow M_3'$, $M_3' \underset{\gamma}{\longrightarrow}^{*} N_2'$.Then $M_3' \underset{\gamma}{\longrightarrow}^{*} N_2' \underset{\gamma}{\longrightarrow}^{*} M_4'$ is a γ-development . □

An analogous property based on project diagram also holds:

Lemma 4.6.4. *Property 4.6.2 implies the following: if* $M_1' \underset{\gamma}{\circ\!\!\rightarrow}^{*} M_2'$, *and* $M_1' \Longrightarrow M_3'$, *then there exists* M_4' *s.t.* $M_2' \Longrightarrow M_4'$, *and* $M_3' \underset{\gamma}{\longrightarrow}^{*} M_4'$. □

Proof. Similarly to the proof of lemma 4.6.3, the proof is by induction on the number of steps in S. The base case is by property 4.6.2. The induction step is illustrated on figure 4.10. □

4.7 Properties of $\Longrightarrow$

As we have mentioned earlier, the evaluation relation of the calculus of recursively scoped records is not a function. In section 3.5.2 we have motivated the requirement that $\Longrightarrow$ must be confluent (otherwise the outcome of a term would not be well-defined). It turns out that to be able to prove lift and project we need to impose stronger requirements on $\Longrightarrow$. Note that the reductions below are on marked terms.

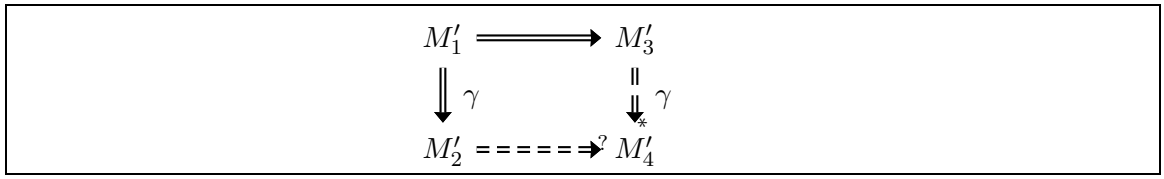
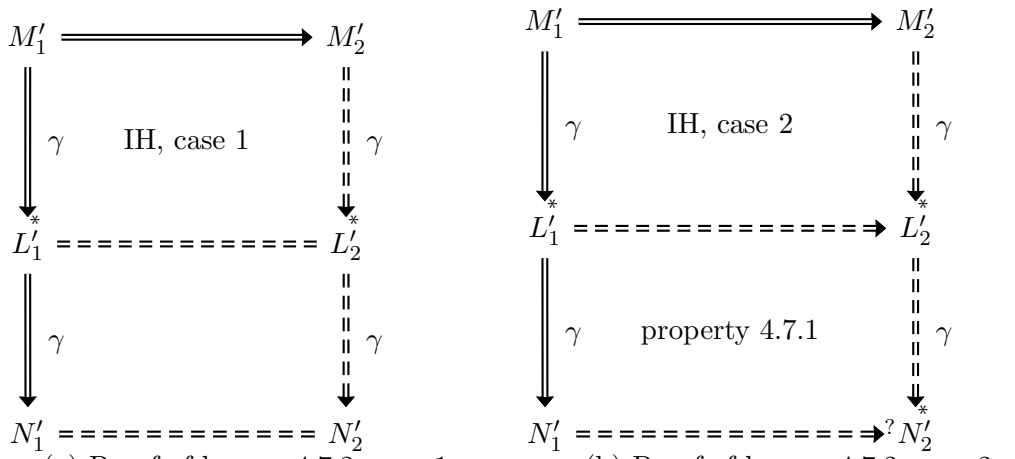


Figure 4.11: γ-confluence of evaluation

Property 4.7.1 (γ-confluence of $\Rightarrow$). *If $M_1' \underset{\gamma}{\Rightarrow} M_2'$ and $M_1' \Rightarrow M_3'$, then there exists M_4' such that $M_2' \Rightarrow^? M_4'$ and $M_3' \underset{\gamma}{\Rightarrow}^* M_4'$.* □

Recall that $\Rightarrow^?$ is the reflexive closure of $\Rightarrow$. The fact that the evaluation sequence from M_2' to M_4' has no more than one step is crucial for the proof of project (see section 4.8). The proofs use the following consequence of the above property:

Lemma 4.7.2 (Multi-step γ-confluence of $\Rightarrow$). *Property 4.7.1 implies the following: if $M_1' \underset{\gamma}{\Rightarrow}^* N_1'$ and $M_1' \Rightarrow M_2'$, then there exists N_2' such that $N_1' \Rightarrow^? N_2'$ and $M_2' \underset{\gamma}{\Rightarrow}^* N_2'$.* □



(a) Proof of lemma 4.7.2, case 1.

(b) Proof of lemma 4.7.2, case 2.

Figure 4.12: Inductive steps of proof of lemma 4.7.2

Proof. The proof is by induction on the number of steps in the sequence $M_1' \underset{\gamma}{\Rightarrow}^* N_1'$, denoted by n. The base case $n = 0$ is trivial.

The inductive step is illustrated on figure 4.12 Suppose the lemma holds for a sequence $M'_1 \underset{\gamma}{\Longrightarrow}^* L'_1$ of $n \geq 0$ steps, i.e. $M'_1 \Longrightarrow M'_2$, and we assume that there exists L'_2 such that $L'_1 \Longrightarrow^? L'_2$ and $M'_2 \underset{\gamma}{\Longrightarrow}^* L'_2$.

Let $L'_1 \underset{\gamma}{\Longrightarrow} N'_1$. We need to show that the lemma holds for the sequence $M'_1 \underset{\gamma}{\Longrightarrow}^* N'_1$ of $n+1$ steps. We have the following two cases:

Case 1: $L'_1 = L'_2$. Then we take $N'_2 = N'_1$. Then $M'_2 \underset{\gamma}{\Longrightarrow}^* N'_1$, and we are done (see figure 4.12(a)).

Case 2: $L'_1 \Longrightarrow L'_2$. Then by property 4.7.1 there exists N'_2 such that $L'_2 \underset{\gamma}{\Longrightarrow}^* N'_2$ and $N'_1 \Longrightarrow^? N'_2$. Then $M'_2 \underset{\gamma}{\Longrightarrow}^* N'_2$ which concludes the proof (see figure 4.12(b)). □

Note that the calculus of recursively scoped records has the following diamond property: If $M_1 \Longrightarrow M_2$ and $M_1 \Longrightarrow M_3$, then there exists M_4 such that $M_2 \Longrightarrow M_4$ and $M_2 \Longrightarrow M_4$. However, this diamond property does not imply the γ-confluence of $\Longrightarrow$, since the definition of a γ-development imposes additional restrictions on the evaluation relation, but also extends the evaluation relation with the "erasure" steps (see definition 4.5.1) which correspond to empty steps in the underlying unmarked calculus.

4.8 Proof of Lift and Project

Lift and project translate into the marked version of a calculus in the following way:

Property 4.8.1 (Lift in marked calculus). *If marked*$(M') = \{(\mathbb{C}, R)\}$, $M' \overset{(\mathbb{C},R)}{\underset{c\gamma}{\circ\!\!\longrightarrow}} N'$, *and* $N' \Longrightarrow^* N'_1$, *then there exists* M'_1 *such that* $M' \underset{\cup}{\Longrightarrow}^* M'_1$ *and* $M'_1 \underset{c\gamma}{\circ\!\!\longrightarrow}^* N'_1$ *(see figure 4.13).* □

Property 4.8.2 (Project in marked calculus). *If marked*$(M') = \{(\mathbb{C}, R)\}$, $M' \overset{(\mathbb{C},R)}{\underset{c\gamma}{\circ\!\!\longrightarrow}} N'$, *and* $M' \Longrightarrow^* M'_1$, *then there exist* M'_2 *and* N'_1 *such that* $N' \Longrightarrow^* N'_1$,

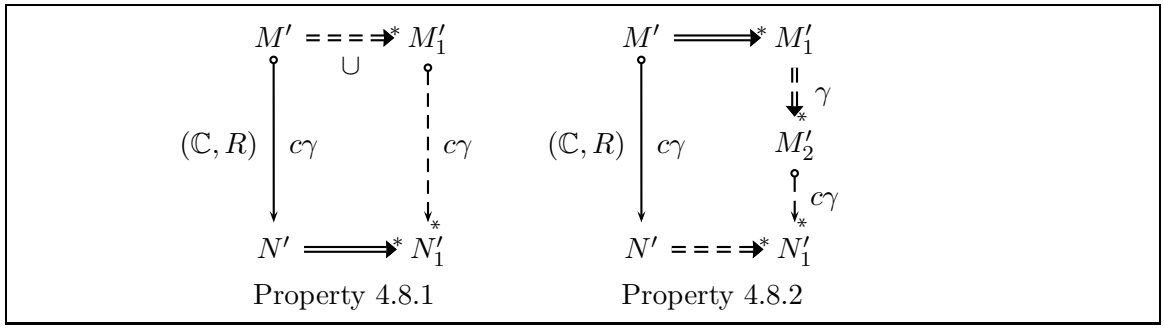


Figure 4.13: Lift and project in the "marked" calculus

$M_1' \underset{\gamma}{\Longrightarrow}^* M_2'$, *and* $M_2' \underset{c\gamma}{\circ\!\!\longrightarrow}^* N_1'$ *(see figure 4.13).* □

Lift and project for a marked calculus imply the respective properties for the associated unmarked calculus:

Lemma 4.8.3. *The lift property for a calculus of marked terms (property 4.8.1) implies lift property for the associated calculus of unmarked terms (property 3.4.1). Likewise, the project property for a calculus of marked terms (property 4.8.2) implies project property for the associated calculus of unmarked terms (property 3.4.2).* □

Proof. We show the proof for lift property, the proof for project is similar. Suppose $M \circ\!\!\longrightarrow N \Longrightarrow^* N_1$ in the unmarked calculus. Then there exists a redex $(\mathbb{C}, R)$ s.t. $M \overset{(\mathbb{C},R)}{\circ\!\!\longrightarrow} N$. Let M' be a marked term such that $|M'| = M$ and marked$(M') = \{(\mathbb{C}, R)\}$. Then $M' \in dom(\gamma)$ by definition 4.5.1, condition 1. By definition 4.5.1, condition 4(d) $M' \underset{\gamma}{\overset{(\mathbb{C},R)}{\circ\!\!\longrightarrow}} N'$ implies that marked$(N') = \emptyset$. By lemma 4.2.10 $N' \Longrightarrow^* N_1'$ implies that marked$(N_1') = \emptyset$. Then $|N'| = N$ and $|N_1'| = N_1$.

By property 3.4.1 there exists a marked term M_1' such that $M' \underset{\cup}{\Longrightarrow}^* M_1'$ and $M_1' \underset{c\gamma}{\circ\!\!\longrightarrow}^* N_1'$. Let $M_1 = |M_1'|$. Then by condition 4(a) of definition 4.5.1 and by definition of the combined reduction $\underset{\cup}{\longrightarrow}$ we get $M \Longrightarrow^* M_1 \circ\!\!\longrightarrow^* N_1$. □

Theorem 4.8.4 (Lift). *Suppose a calculus has the following properties:*

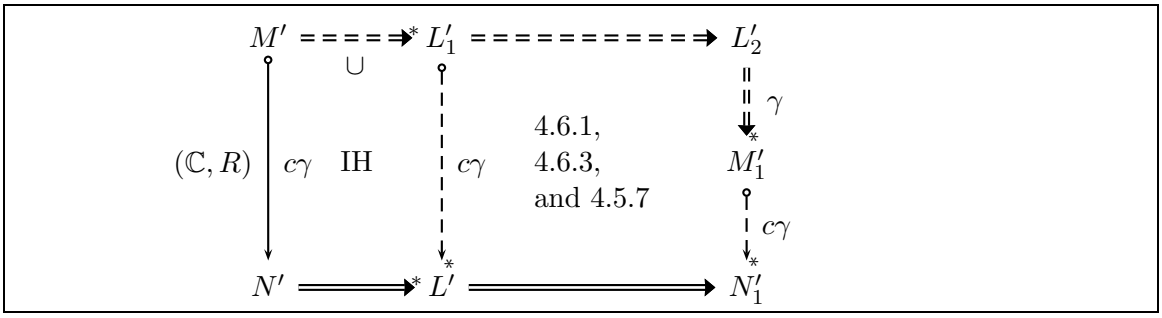


Figure 4.14: Inductive step of proof of theorem 4.8.4

- *Property 4.5.7 (weak standardization of developments),*
- *Property 4.6.1 (elementary lift property).*

Then the calculus has the lift property (property 4.8.1). □

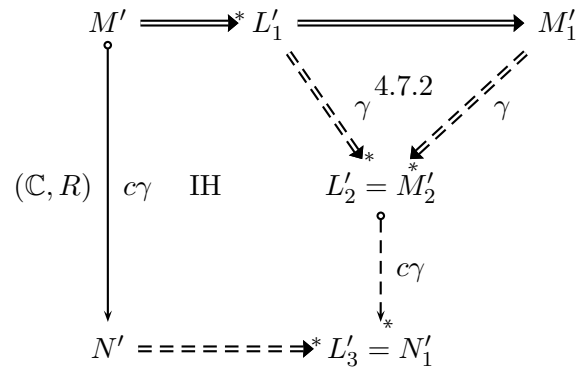
Proof. Assume that the calculus has properties 4.5.7 and 4.6.1. Let marked$(M') = \{(\mathbb{C}, R)\}$, $M' \underset{c\gamma}{\overset{(\mathbb{C},R)}{\circ\!\!\longrightarrow}} N'$, and $N' \Longrightarrow^* N'_1$.

We want to prove the following *stronger* claim: there exists M'_1 such that $M' \underset{\cup}{\Longrightarrow^*} M'_1$, $M'_1 \underset{c\gamma}{\circ\!\!\longrightarrow^*} N'_1$, and the sequences $M' \underset{\cup}{\Longrightarrow^*} M'_1$ and $N' \Longrightarrow^* N'_1$ are lockstep equivalent. The claim implies the statement of the theorem. The proof is by induction on the number of steps in the latter evaluation sequence. The proof is illustrated on figure 4.14.

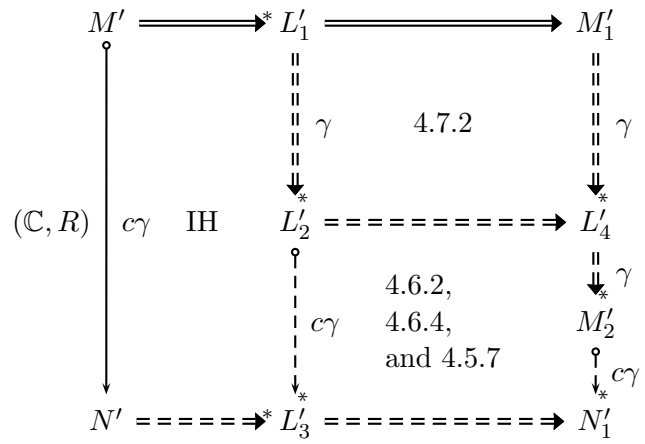
The base case $N' = N'_1$ is trivial. As an inductive hypothesis, suppose the claim holds for a sequence $N' \Longrightarrow^* L'$ of n steps, i.e. there exists L'_1 such that $M' \underset{\cup}{\Longrightarrow^*} L'_1$, $L'_1 \underset{c\gamma}{\circ\!\!\longrightarrow^*} L'$, and the sequences $M' \underset{\cup}{\Longrightarrow^*} L'_1$ and $N' \Longrightarrow^* L'$ are lockstep equivalent.

Let $L' \Longrightarrow N'_1$. The calculus has property 4.6.1, therefore by lemma 4.6.3 $L'_1 \underset{c\gamma}{\circ\!\!\longrightarrow^*} L'$ and $L' \Longrightarrow N'_1$ imply that there exists L'_2 such that $L'_1 \Longrightarrow L'_2$, $L'_2 \underset{\gamma}{\circ\!\!\longrightarrow^*} N'_1$. $M' \underset{c\gamma}{\overset{(\mathbb{C},R)}{\circ\!\!\longrightarrow}} N'$ implies that marked$(N') = \emptyset$. By lemma 4.2.10 $N' \Longrightarrow^* N'_1$ implies that marked$(N'_1) = \emptyset$, and therefore the sequence $L'_2 \underset{\gamma}{\circ\!\!\longrightarrow^*} N'_1$ is a complete development.

Therefore the sequences $M' \underset{\cup}{\Rightarrow}^* L'_1 \Rightarrow L'_2$ and $N' \Rightarrow^* N'_1$ are lockstep equivalent, and by property 4.5.7 there exists M'_1 such that $L'_1 \underset{\gamma}{\Rightarrow}^* M'_1 \underset{c\gamma}{\circ\!\!\rightarrow}^* N'_1$. Note that by definition 4.5.6 the sequences $M' \underset{\cup}{\Rightarrow}^* L'_1 \Rightarrow L'_2 \underset{\gamma}{\Rightarrow}^* M'_1$ and $N' \Rightarrow^* N'_1$ are lockstep equivalent. □



Case 1 of proof of theorem 4.8.5



Case 2 of proof of theorem 4.8.5

Figure 4.15: Inductive step of proof of theorem 4.8.5 (2 cases).

Theorem 4.8.5 (Project). *Suppose a calculus has the following properties:*

- *Property 4.5.7 (weak standardization of developments),*
- *Property 4.6.2 (elementary project property),*

- *Property 4.7.1 (γ-confluence of evaluation).*

Then the calculus has the project property (property 4.8.2). □

Proof. The proof is similar to that of theorem 4.8.4. It is illustrated on figure 4.15. Suppose the calculus has the three required properties, $\text{marked}(M') = \{(\mathbb{C}, R)\}$, $M' \underset{c\gamma}{\overset{(\mathbb{C},R)}{\circ\!\!\longrightarrow}} N'$, and $M' \Longrightarrow^* M'_1$.

We strengthen the claim similarly to the proof of theorem 4.8.4: we want to prove that there exist M'_2 and N'_1 such that $M'_1 \underset{\gamma}{\Longrightarrow}^* M'_2$, $M'_2 \underset{c\gamma}{\circ\!\!\longrightarrow}^* N'_1$, $N' \Longrightarrow^* N'_1$, and the sequences $M' \Longrightarrow^* M'_1 \underset{\gamma}{\Longrightarrow}^* M'_2$ and $N' \Longrightarrow^* N'_1$ are lockstep equivalent.

The proof is by induction on the number of steps in the sequence $M' \Longrightarrow^* M'_1$. The base case of zero steps is trivial. Suppose that the claim holds for a sequence $M' \Longrightarrow^* L'_1$, i.e. there exist L'_2, L'_3 such that $L'_1 \underset{\gamma}{\Longrightarrow}^* L'_2$, $L'_2 \underset{c\gamma}{\circ\!\!\longrightarrow}^* L'_3$, $N' \Longrightarrow^* L'_3$, and the sequences $M' \Longrightarrow^* L'_1 \underset{\gamma}{\Longrightarrow}^* L'_2$ and $N' \Longrightarrow^* L'_3$ are lockstep equivalent. Suppose $L'_1 \Longrightarrow M'_1$. We want to show that the claim holds for the sequence $M' \Longrightarrow^* M'_1$.

By property 4.7.1 and lemma 4.7.2 $L'_1 \Longrightarrow M'_1$ and $L'_1 \underset{\gamma}{\Longrightarrow}^* L'_2$ imply that there exists L'_4 such that $L'_2 \Longrightarrow^? L'_4$ and $M'_1 \underset{\gamma}{\Longrightarrow}^* L'_4$. We have the following two cases:

Case 1. $L'_2 = L'_4$. Then we take $M'_2 = L'_2$ and $N'_1 = L'_3$, and the claim holds. Note that the sequences $M' \Longrightarrow^* M'_1 \underset{\gamma}{\Longrightarrow}^* L'_2 = M'_2$ and $N' \Longrightarrow^* L'_3 = N'_1$ are lockstep equivalent (see definition 4.5.6).

Case 2. $L'_2 \Longrightarrow L'_4$. By lemma 4.6.4 property 4.6.2 implies that there exists N'_1 such that $L'_3 \Longrightarrow N'_1$ and $L'_4 \underset{\gamma}{\longrightarrow}^* N'_1$. By lemma 4.2.10 $\text{marked}(N'_1) = \emptyset$ since $\text{marked}(N') = \emptyset$, and therefore $L'_4 \underset{c\gamma}{\longrightarrow}^* N'_1$. The sequences $M' \Longrightarrow^* L'_1 \Longrightarrow M'_1 \underset{\gamma}{\Longrightarrow}^* L'_4$ and $N' \Longrightarrow^* L'_3 \Longrightarrow N'_1$ are lockstep equivalent by definition 4.5.6, and therefore there exists M'_2 such that $L'_4 \underset{\gamma}{\Longrightarrow}^* M'_2 \underset{c\gamma}{\circ\!\!\longrightarrow}^* N'_1$. The sequences $M' \Longrightarrow^* L'_1 \Longrightarrow M'_1 \underset{\gamma}{\Longrightarrow}^* L'_4 \underset{\gamma}{\Longrightarrow}^* M'_2$ and $N' \Longrightarrow^* L'_3 \Longrightarrow N'_1$ are lockstep equivalent, and the induction claim is proven. □

4.9 Properties Needed to Prove Lift and Project

Sections 4.5–4.8 outline a general proof of lift and project. This proof abstracts over calculus-specific details, giving axiomatic definitions of certain calculus entities and relations and by specifying properties of a calculus which imply lift and project.

In order to apply this framework, one needs to instantiate these definitions for a particular calculus and to show that the calculus satisfies the properties. Below is a complete list of such definitions and properties.

The two axiomatic definitions that we need to instantiate are:

1. definition of a residual (definition 4.2.5), we also need to show that residuals satisfy the unique ancestor assumption (assumption 4.2.6);

2. definition of a γ-development reduction (definition 4.5.1).

Given concrete definitions satisfying the axiomatic ones listed above, one needs to show the following properties in order to prove lift and project:

1. The elementary lift diagram property (property 4.6.1);

2. The elementary project diagram property (property 4.6.2);

3. The weak standardization of γ-developments (definition 4.5.7);

4. The γ-confluence of $\Longrightarrow$ property (property 4.7.1).

The details of the axiomatic definitions and the properties are given in sections 4.5–4.7. The proofs of lift and project are given in section 4.8.

Chapter 5

Applications and Results

5.1 Soundness of the Term Calculus

In this section we give the computational soundness proof for the the calculus defined in figure 2.3 in section 2.2.3. We use the symbol $\mathcal{T}$ to denote this calculus. We use $\mathcal{T}$ as a subscript for relations in the term calculus. $\mathbf{Term}_{\mathcal{T}}, \mathbf{Context}_{\mathcal{T}}$, and $\mathbf{EvalContext}_{\mathcal{T}}$ to denote sets of terms, one-hole contexts, and evaluation contexts of $\mathcal{T}$, respectively.

We give the calculus-specific definition of residual for $\mathcal{T}$ and show that it satisfies the axiomatic definition 4.2.5. We prove that the calculus is computationally sound by showing that it has confluence, class preservation, and standardization. The approach in this section largely follows the line of reasoning in [Bar84], in particular the proofs of boundedness of developments, confluence, and standardization. The approach has been slightly extended to cover the case of constants and operations on constants in the proof of boundedness of developments. A more significant change is that instead of head redexes we use evaluation redexes defined via an evaluation context.

5.1.1 Notations and Definitions

Before we define residuals in $\mathcal{T}$, let us introduce some important notations.

Definition 5.1.1 (Multi-hole contexts in $\mathcal{T}$). A multi-hole context in $\mathcal{T}$ is defined as follows:

$$\mathbb{A}, \mathbb{B}, \mathbb{C} \quad ::= \quad M \mid \Box \mid \mathbb{C} \mathbin{@} \mathbb{C} \mid \mathbb{C} \mathbin{op} \mathbb{C} \mid \lambda x.\mathbb{C}.$$

$H(\mathbb{C})$ denotes the number of holes in a context $\mathbb{C}$ (defined in the obvious way). $\mathbf{Context}^n_{\mathcal{T}}$ denotes the set $\{\mathbb{C} \mid H(\mathbb{C}) = n\}$. Note that $\mathbf{Context}^0_{\mathcal{T}} = \mathbf{Term}_{\mathcal{T}}$, $\mathbf{Context}^1_{\mathcal{T}} = \mathbf{Context}_{\mathcal{T}}$.

If $\mathbb{C} \in \mathbf{Context}^n_{\mathcal{T}}$, then $\mathbb{C}\{\mathbb{A}_1, \ldots, \mathbb{A}_n\}$ denotes the result of filling the holes of $\mathbb{C}$ left-to-right with contexts $\mathbb{A}_1, \ldots, \mathbb{A}_n$. □

By convention the notation $(\mathbb{C}, R)$ for a redex implies that $\mathbb{C} \in \mathbf{Context}^1_{\mathcal{T}}$.

Definition 5.1.2 (glb of contexts). The greatest lower bound of two contexts $\mathrm{glb}(\mathbb{C}_1, \mathbb{C}_2)$ is a context defined the following way:

$$\begin{array}{rcl}
\mathrm{glb}(M, M) & = & M, \\
\mathrm{glb}(\mathbb{C}, \Box) & = & \Box \quad \text{for any} \quad \mathbb{C}, \\
\mathrm{glb}(\Box, \mathbb{C}) & = & \Box \quad \text{for any} \quad \mathbb{C}, \\
\mathrm{glb}(\mathbb{C}_1 \mathbin{@} \mathbb{C}_2, \mathbb{C}'_1 \mathbin{@} \mathbb{C}'_2) & = & \mathrm{glb}(\mathbb{C}_1, \mathbb{C}'_1) \mathbin{@} \mathrm{glb}(\mathbb{C}_2, \mathbb{C}'_2), \\
\mathrm{glb}(\mathbb{C}_1 \mathbin{op} \mathbb{C}_2, \mathbb{C}'_1 \mathbin{op} \mathbb{C}'_2) & = & \mathrm{glb}(\mathbb{C}_1, \mathbb{C}'_1) \mathbin{op} \mathrm{glb}(\mathbb{C}_2, \mathbb{C}'_2), \\
\mathrm{glb}(\lambda x.\mathbb{C}_1, \lambda x.\mathbb{C}_2) & = & \lambda x.\, \mathrm{glb}(\mathbb{C}_1, \mathbb{C}_2), \\
\text{otherwise} & & \text{undefined}
\end{array}$$

The greatest lower bound of $n > 2$ contexts is defined as

$$\mathrm{glb}(\mathbb{C}_1, \mathbb{C}_2, \ldots, \mathbb{C}_n) \quad = \quad \mathrm{glb}(\mathrm{glb}(\mathbb{C}_1, \mathbb{C}_2), \mathbb{C}_3, \ldots, \mathbb{C}_n).$$

□

Now we give definition of a set of residuals of a redex in $\mathcal{T}$. It is easy to check that the set of residuals defined below satisfies the properties postulated in the axiomatic definition 4.2.5.

Definition 5.1.3. Let $(\mathbb{C}_1, R_1)$ be a redex in a term M, and suppose $M \xrightarrow{(\mathbb{C}_2,R_2)}_{\mathcal{T}} N$. A *set of residuals* of $(\mathbb{C}_1, R_1)$ w.r.t. the reduction $\xrightarrow{(\mathbb{C}_2,R_2)}_{\mathcal{T}}$ (denoted $(\mathbb{C}_1, R_1)/(\mathbb{C}_2, R_2)$) is defined as follows: let $\mathbb{A} = \text{glb}(\mathbb{C}_1, \mathbb{C}_2)$, and let $R_2 \rightsquigarrow Q_2$, then

1. if $\mathbb{C}_1 = \mathbb{C}_2$ (and therefore $R_1 = R_2$), then $(\mathbb{C}_1, R_1)/(\mathbb{C}_2, R_2) = \emptyset$,
2. if $\mathbb{A} \in \mathbf{Context}^2_{\mathcal{T}}$, then $(\mathbb{C}_1, R_1)/(\mathbb{C}_2, R_2) = \{(\mathbb{A}\{\Box, Q_2\}, R_1)\}$ (assuming without loss of generality that R_1 fills the first hole in $\mathbb{A}$).
3. if $M = \mathbb{A}\{\lambda x.\mathbb{B}^n\{x, \ldots, x\} \mathbin{@} V\}$, and $R_2 = \lambda x.\mathbb{B}^n\{x, \ldots, x\} \mathbin{@} V$, where $\mathbb{B}^n$ contains all occurrences of x in the operand, and $V = \mathbb{C}\{R_1\}$, then $(\mathbb{C}_1, R_1)/(\mathbb{C}_2, R_2) = \{(\mathbb{A}\{\mathbb{B}^n\{V, \ldots, V, \mathbb{C}_i, V, \ldots V\}\}, R_1) \mid 1 \leq i \leq n\}$, where $\mathbb{C}_i$ is the context $\mathbb{C}$ filling the i-th hole of $\mathbb{B}^n$.
4. if $M = \mathbb{A}\{\lambda x.\mathbb{B}\{R_1\} \mathbin{@} V\}$, where $R_2 = \lambda x.\mathbb{B}\{R_1\} \mathbin{@} V$, then $(\mathbb{C}_1, R_1)/(\mathbb{C}_2, R_2) = \{(\mathbb{A}\{\mathbb{B}\}, R_1[x := V])\}$.
5. if $M = \mathbb{A}\{R_1\}$, where $R_1 = \mathbb{B}\{R_2\}$, then $(\mathbb{C}_1, R_1)/(\mathbb{C}_2, R_2) = \{(\mathbb{A}, \mathbb{B}\{Q_2\})\}$.

□

A set of residuals of a *set of redexes* with respect to a reduction step and a set of residuals of a set of redexes with respect to *a reduction sequence* are defined in definition 4.2.7. Reductions on marked terms corresponding to terms of $\mathcal{T}$ is defined in definition 4.2.9, developments and complete developments are defined in 4.2.11 and 4.2.12, respectively.

Below we prove an important property of the term calculus: boundedness of developments. Even though the proof is traditional (see Chapter 11 of [Bar84]), we give it in some detail, since some notions defined for the proof will be used later in the proof of boundedness of γ-developments of the calculus of records. In particular the notion of a weighting introduced below will be used in both this and the next section.

We say that a term N has the *distinct variables property* if all bound variables in N are distinct and are different from all free variables of N. It is a fact that every term M can be α-renamed to a term N which has the distinct variables property. In [MT02] we give a formal framework with explicit and rigorous treatment of α-renaming. In this presentation we assume that all terms that we consider have the distinct variables property. All properties shown here hold up to α-renaming of term (see [MT02] for details).

5.1.2 Developments are Bound

Definition 5.1.4. A calculus of weighted terms $\mathcal{T}_w$ is defined as follows: let n range over positive integers, then

$$\begin{aligned}
x^n &\in \mathcal{T}_w, \\
c^n &\in \mathcal{T}_w, \\
l^n &\in \mathcal{T}_w \\
\lambda x.M &\in \mathcal{T}_w \quad \text{if } M \in \mathcal{T}_w \\
M \mathbin{@} N &\in \mathcal{T}_w \quad \text{if } M, N \in \mathcal{T}_w \\
M \mathbin{op} N &\in \mathcal{T}_w \quad \text{if } M, N \in \mathcal{T}_w
\end{aligned}$$

Here n is called a *weight* of the respective variable, constant, or label. Note that a variable immediately preceded by a lambda does not have a weight.

If $M \in \mathcal{T}_w$, the *measure* of M (denoted $|| M ||$) is by definition the sum of all weights (of variables, constants, and labels) occurring in M. Note that $|| M ||> 0$.

Assuming that all occurrences of variables (except for those immediately preceded by a lambda), constants, and labels in a term $M_1 \in \mathcal{T}_w$ are ordered by their position from left to right in M_1, we can consider M as a pair (M, I), where $M \in \mathcal{T}$ is a term obtained from M_1 by erasing all weights, and I is a list of weights of variables, labels, and constants in M_1 from left to right. I is called a *weighting* of M. Note that there is a one-to-one correspondence between terms $M_1 \in \mathcal{T}_w$ and pairs (M, I).

We write $I(x)$, where x is a particular occurrence of x in M, to denote the weight of the occurrence of x in the corresponding $M_1 \in \mathcal{T}_w$, and similarly for $I(c)$ and $I(l)$. □

We define reduction on weighted terms as follows:

Definition 5.1.5. Let $M, N \in \mathcal{T}_w$. Then $M \longrightarrow N$ if and only if $M = \mathbb{C}\{R\}$, $N = \mathbb{C}\{Q\}$, and one of the following takes place:

- $R = (\lambda x.\tilde{M})\ @\ V$, $Q = \tilde{M}[x := V]$, where $M[x := N]$ is defined as follows:

$$\begin{aligned} x^n[x := N] &= N, \\ y^n[x := N] &= y^n \quad \text{if } y \neq x, \\ c^n[x := N] &= c^n, \\ l^n[x := N] &= l^n, \end{aligned}$$

  and the rest of the rules as usual.

- $R = c_1^n\ op\ c_2^m$, $Q = c_3^{\max(n,m)}$, where $c_3 = \delta(c_1, c_2,\ op\ )$.

□

In the above definition $\mathbb{C}$ is a context over terms of $\mathcal{T}_w$. We omit a straightforward definition of such contexts.

Note that since terms of $\mathcal{T}_w$ are in one-to-one correspondence with pairs (M, I), definition 5.1.5 also defines a reduction on such pairs.

Definition 5.1.6. A weighting I is called a *decreasing weighting* of a marked term M' if for any $(\mathbb{C}, R) \in \text{marked}(M')$ such that $R = (\lambda x.N)\ @\ V$ for all occurrences of x in N it is the case that $||\ x\ ||>||\ V\ ||$. □

Lemma 5.1.7. *For any M' there exists a decreasing weighting.* □

Proof. We assign weights of non-negative integers in increasing order, starting at 0, to all occurrences of variables, constants, and labels in M' in the right-to-left order. Then the weight of the i-th occurrence is 2^i. The weighting is decreasing since $2^n > 2^{n-1} + \cdots + 2 + 1$. The weighting is decreasing for any redex in M', therefore it is decreasing for all redexes in marked(M'). □

We extend the reduction to terms with marked redexes and weights. Such terms are represented as pairs of a marked term M' and its weighting I.

Definition 5.1.8. We say that a pair (M'_1, I_1) reduces to a pair (M'_2, I_2) by a redex $(\mathbb{C}, R)$ (denoted $(M'_1, I_1) \xrightarrow{(\mathbb{C},R)} (M'_2, I_2)$) if both conditions below hold:

- $(M_1, I_1) \xrightarrow{(\mathbb{C},R)} (M_2, I_2)$, where $M_1 = |M'_1|$ and $M_2 = |M'_2|$ as defined in 5.1.5.
- $M'_1 \xrightarrow{(\mathbb{C},R)} M'_2$ by the reduction in the marked calculus.

□

Convention: the notation (M', I) assumes that I is a weighting of $M = |M'|$. Trivially generalizing definition of developments to pairs $(M'I)$, we say that a *development step* of (M', I) is a reduction step that reduces a redex in marked(M').

Note: in the lemma below we consider measure $|| \cdot ||$ on pairs (M, I), since this measure is defined on terms of $\mathcal{T}_w$ which are in one-to-one correspondence with such pairs.

Lemma 5.1.9. *If* $(M'_1, I_1) \xrightarrow{(\mathbb{C},R)} (M'_2, I_2)$*, and* I_1 *is a decreasing weighting of* M'_1*, then* I_2 *is a decreasing weighting of* M'_2*, and* $||\ (M_1, I_1)\ ||>||\ (M_2, I_2)\ ||$*, where* $M_1 = |M'_1|$ *and* $M_2 = |M'_2|$*.* □

Proof. The proof is a straightforward proof by cases, analogous to the proof in Chapter 11 of [Bar84]. □

Lemma 5.1.10. *Any development of a pair* M' *is has no more than* $||\ (M, I)\ ||$ *steps, where* $M = |M'|$*, and* I *is the decreasing weighting of* M' *as constructed in lemma 5.1.7.* □

Proof. By lemma 5.1.7 there exists a decreasing weighting of (M, F), let I be such a weighting. Consider a development of (M, F, I). By lemma 5.1.9 for every development step $(M, F, I) \underset{dev}{\rightarrow} (M', F', I')$, where I is decreasing, we have: I' is also decreasing, and $||\ (M, I)\ ||<||\ (M', I')\ ||$. Since $||\ (M, I)\ ||> 0$ for any well-defined pair (M, I), no developments of M' has more than $||\ (M, I)\ ||$ steps. □

5.1.3 Parallel Moves Lemma

In order to prove confluence of $\mathcal{T}$, we prove a parallel move lemma (lemma 5.1.16 below) for terms with sets of marked redexes. To do this, we need to show not only weak confluence for unmarked terms, but also that for any marked redex in the

original term the set of its residuals on both reduction paths given by the parallel move lemma is the same. Note that in general (i.e. for arbitrary reduction paths, not necessarily those given by the parallel move lemma) the former does not imply the latter: it is possible that a term M reduces to a term N by two different reduction paths, but a marked redex in M has different residual sets in M. For more on this issue see discussion of strongly equivalent reductions in [Bar84].

We prove parallel move lemma in two steps: first we show the property for one marked redex (lemma 5.1.15), and then generalize it to an arbitrary set of marked redexes.

Before we prove the lemmas, we need to define terminology for mutual positions of two subterms in a term.

Definition 5.1.11. 1. Two subterms occurrences $(\mathbb{C}_1, N_1)$ and $(\mathbb{C}_2, N_2)$ of a term M (see definition 4.2.2) are *independent* if there exists a two-hole context $\mathbb{A}$ s.t. $\mathbb{C}_1 = \mathbb{A}\{\Box, N_2\}$ and $\mathbb{C}_2 = \mathbb{A}\{N_1, \Box\}$ or, alternatively, $\mathbb{C}_1 = \mathbb{A}\{N_2, \Box\}$ and $\mathbb{C}_2 = \mathbb{A}\{\Box, N_1\}$. Note that this implies $M = \mathbb{A}\{N_1, N_2\}$ or $M = \mathbb{A}\{N_2, N_1\}$.

2. If $(\mathbb{C}_1, N_1)$ and $(\mathbb{C}_2, N_2)$ are subterms of the same term M and are not independent, then it must be the case that either N_2 is a subterm of N_1 (in which case we say that $(\mathbb{C}_1, N_1)$ *contains* $(\mathbb{C}_2, N_2)$) or vice versa (i.e. $(\mathbb{C}_2, N_2)$ *contains* $(\mathbb{C}_1, N_1)$). As in case 1, sometimes we will omit the contexts, e.g. say that N_1 contains N_2.

3. We say that two redexes $(\mathbb{C}_1, R_1)$ and $(\mathbb{C}_2, R_2)$ of a term M are *independent* (respectively $(\mathbb{C}_1, R_1)$ contains $(\mathbb{C}_2, R_2)$) if they are independent as subterms (respectively $(\mathbb{C}_1, R_1)$ contains $(\mathbb{C}_2, R_2)$ as subterms).

□

Now we state and prove some important properties of redexes and evaluation contexts which will be used in further proofs, both for the term and for the core module calculus.

Lemma 5.1.12. *If R is a redex and x is not bound in R, then $R[x := V]$ is a redex.*

If $\mathbb{E} \in \mathbf{EvalContext}_{\mathcal{T}}$ and x is not bound in $\mathbb{E}$ (i.e. there is no $\mathbb{A}$, $\mathbb{B}$ such that $\mathbb{E} = \mathbb{A}\{\lambda x.\mathbb{B}\}$) then $\mathbb{E}[x := V] \in \mathbf{EvalContext}_{\mathcal{T}}$ □

Proof. The two cases of a redex are $c_1\ op\ c_2$ and $\lambda y.M\ @\ V$, in both cases the claim clearly holds.

The proof for $\mathbb{E} \in \mathbf{EvalContext}_{\mathcal{T}}$ is straightforward by induction on the structure of an evaluation context. □

Lemma 5.1.13. *If $R = \mathbb{A}\{l\}$ is a redex, then $R_1 = \mathbb{A}\{V\}$ is also a redex.*

If $\mathbb{E} = \mathbb{A}\{\Box, l\} \in \mathbf{EvalContext}_{\mathcal{T}}$ (respectively $\mathbb{E} = \mathbb{A}\{l, \Box\} \in \mathbf{EvalContext}_{\mathcal{T}}$), then $\mathbb{E}' = \mathbb{A}\{\Box, V\} \in \mathbf{EvalContext}_{\mathcal{T}}$ (respectively $\mathbb{E}' = \mathbb{A}\{V, \Box\} \in \mathbf{EvalContext}_{\mathcal{T}}$).

□

Proof. Similar to the proof of lemma 5.1.12. Note that labels in a term are not bound, therefore the occurrence of l is free in a redex or in an evaluation context. □

Lemma 5.1.14. *If $\mathbb{E} = \mathbb{A}\{\Box, R\} \in \mathbf{EvalContext}_{\mathcal{T}}$ (or $\mathbb{E} = \mathbb{A}\{R, \Box\} \in \mathbf{EvalContext}_{\mathcal{T}}$) and $R \rightsquigarrow_{\mathcal{T}} Q$, then $\mathbb{E}' = \mathbb{A}\{\Box, Q\} \in \mathbf{EvalContext}_{\mathcal{T}}$ (respectively $\mathbb{E}' = \mathbb{A}\{Q, \Box\} \in \mathbf{EvalContext}_{\mathcal{T}}$).* □

Proof. By induction on the structure of an evaluation context. □

The following is a key lemma for many subsequent proofs. In addition to weak confluence of the calculus reduction $\longrightarrow_{\mathcal{T}}$, it shows that for any marked redex in M_1 its residuals are the same on both reduction paths. We also show confluence of a

complete development of $(\mathbb{C}_1, R_1)/(\mathbb{C}_2, R_2)$ for any two redexes. Note that we do not show and do not use a general confluence of complete developments, i.e. confluence of a complete development of an arbitrary F/S, even though such confluence holds in $\mathcal{T}$. This is because our goal is to show standardization of developments, and for this purpose it is sufficient to show that a non-evaluation step followed by an evaluation step in a development can be replaced by a development sequence which starts with an evaluation step (corollary 5.1.23). This property, in addition to boundedness of developments that we have already shown, implies standardization of developments.

Lemma 5.1.15. *Let marked*$(M'_1) = \{(\mathbb{A}, \tilde{R})\}$. *Suppose* $M'_1 \xrightarrow{(\mathbb{C}_1,R_1)} M'_2$, $M'_1 \xrightarrow{(\mathbb{C}_2,R_2)} M'_3$, *and* $(\mathbb{C}_1, R_1) \neq (\mathbb{C}_2, R_2)$. *Then there exists* M'_4 *s.t.* $M'_2 \xrightarrow{(\mathbb{C}_2,R_2)/(\mathbb{C}_1,R_1)}{}^{*} M'_4$, $M'_3 \xrightarrow{(\mathbb{C}_1,R_1)/(\mathbb{C}_2,R_2)}{}^{*} M'_4$.

Additionally, for every reduction sequence $M'_2 \xrightarrow{(\mathbb{C}_2,R_2)/(\mathbb{C}_1,R_1)}{}^{*} M''_4$ *the resulting term* $M''_4 = M'_4$, *and the same for the reduction sequences* $M'_3 \xrightarrow{(\mathbb{C}_1,R_1)/(\mathbb{C}_2,R_2)}{}^{*} M''_4$. □

Proof. The proof is by cases on mutual positions (and the kinds) of the three redexes in M'_1. We show enough cases to demonstrate the proof technique. The other cases are similar to the ones shown. We don't list cases symmetric to the ones given (i.e. those obtained by switching $(\mathbb{C}_1, R_1)$ and $(\mathbb{C}_2, R_2)$). We suppose that $R_1 \rightsquigarrow Q_1$, $R_2 \rightsquigarrow Q_2$, $\tilde{R} \rightsquigarrow \tilde{Q}$.

1. All three redexes are independent - trivial.

2. $(\mathbb{A}, \tilde{R}) = (\mathbb{C}_1, R_1)$, $(\mathbb{C}_2, R_2)$ is independent from $(\mathbb{C}_1, R_1)$ - trivial.

3. $(\mathbb{A}, \tilde{R}) = (\mathbb{C}_1, R_1)$, $(\mathbb{C}_2, R_2)$ is contained in $(\mathbb{C}_1, R_1)$.

   Since $(\mathbb{C}_1, R_1)$ contains $(\mathbb{C}_2, R_2)$, it must be the case that $R_1 = (\lambda x.N_1)\ @\ V_1$. We have two subcases:

- $N_1 = \mathbb{B}\{R_2, x, \ldots, x\}$, i.e. N_1 has several occurrences of x and an occurrence[1] of R_2. Then

$$(\lambda x.\mathbb{B}\{R_2, x, \ldots, x\}) \mathbin{@} V_1 \;\; \rightarrow_{\mathcal{T}} \;\; \mathbb{B}\{R_2, V_1, \ldots, V_1\} \;\; \rightarrow_{\mathcal{T}} \;\; \mathbb{B}\{Q_2, V_1, \ldots, V_1\},$$
$$(\lambda x.\mathbb{B}\{R_2, x, \ldots, x\}) \mathbin{@} V_1 \;\; \rightarrow_{\mathcal{T}} \;\; (\lambda x.\mathbb{B}\{Q_2, x, \ldots, x\}) \mathbin{@} V_1 \;\; \rightarrow_{\mathcal{T}} \;\; \mathbb{B}\{Q_2, V_1, \ldots, V_1\}.$$

- $R_1 = (\lambda x.\mathbb{B}\{x, \ldots, x\}) \mathbin{@} \lambda y.\tilde{\mathbb{B}}\{R_2\}$. Then

$$(\lambda x.\mathbb{B}\{x, \ldots, x\}) \mathbin{@} \lambda y.\tilde{\mathbb{B}}\{R_2\} \;\; \rightarrow_{\mathcal{T}} \;\; \mathbb{B}\{\lambda y.\tilde{\mathbb{B}}\{R_2\}, \ldots, \lambda y.\tilde{\mathbb{B}}\{R_2\}\} \;\; \rightarrow^*_{\mathcal{T}} \mathbb{B}\{\lambda y.\tilde{\mathbb{B}}\{Q_2\}, \ldots, \lambda y.\tilde{\mathbb{B}}\{Q_2\}\},$$
$$(\lambda x.\mathbb{B}\{x, \ldots, x\}) \mathbin{@} \lambda y.\tilde{\mathbb{B}}\{R_2\} \;\; \rightarrow_{\mathcal{T}} \;\; (\lambda x.\mathbb{B}\{x, \ldots, x\}) \mathbin{@} \lambda y.\tilde{\mathbb{B}}\{Q_2\} \;\; \rightarrow_{\mathcal{T}} \mathbb{B}\{\lambda y.\tilde{\mathbb{B}}\{Q_2\}, \ldots, \lambda y.\tilde{\mathbb{B}}\{Q_2\}\}.$$

In both cases since $(\mathbb{A}, \tilde{R}) = (\mathbb{C}_1, R_1)$, $(\mathbb{A}, \tilde{R})$ does not have a residual in the resulting term on both reduction paths. The second part of the lemma follows from the observation that the order in which copies of R_2 are reduced in the multi-step reduction does not matter.

4. $(\mathbb{A}, \tilde{R}) = (\mathbb{C}_1, R_1)$, $(\mathbb{C}_2, R_2)$ contains $(\mathbb{C}_1, R_1)$. By the result of the previous case if a redex contains another redex, then they can be reduced in any order. Since $(\mathbb{C}_2, R_2)$ in the previous case does not have a residual on any of the two reduction paths, we conclude that $(\mathbb{A}, \tilde{R})$ does not have a residual as well.

5. $(\mathbb{C}_1, R_1)$ contains $(\mathbb{C}_2, R_2)$, $(\mathbb{A}, \tilde{R})$ is independent from both $(\mathbb{C}_1, R_1)$ and $(\mathbb{C}_2, R_2)$.

   In this case $M_1 = \mathbb{B}\{R_1, \tilde{R}\}$. By case 3 we know that since $(\mathbb{C}_1, R_1)$ contains

[1] Here and below we show only one ordering of occurrences of subterms in a term, provided all other orderings are analogous.

$(\mathbb{C}_2, R_2)$, the two redexes can be performed in any order. $(\mathbb{A}, \tilde{R})$ is independent from these redexes, so clearly it has the same residual on both reduction paths. As in case 3, the order in which copies of R_2 are reduced in the multi-step reduction does not matter.

6. $(\mathbb{C}_1, R_1)$ contains $(\mathbb{C}_2, R_2)$, $(\mathbb{A}, \tilde{R})$ is contained in $(\mathbb{C}_1, R_1)$, $(\mathbb{A}, \tilde{R})$ is independent from $(\mathbb{C}_2, R_2)$.

   In this case $R_1 = (\lambda x.N_1) \mathbin{@} V_1$, and we have the following 4 subcases:

   - $R_1 = (\lambda x.\mathbb{B}\{R_2, \tilde{R}, x, \ldots, x\}) \mathbin{@} V_1$. In this case both reduction paths lead to a term $\mathbb{B}\{Q_2, \tilde{R}, V_1, \ldots, V_1\}$.
   - $R_1 = (\lambda x.\mathbb{B}\{R_2, x, \ldots, x\}) \mathbin{@} \lambda y.\tilde{\mathbb{B}}\{\tilde{R}\}$. Both reduction paths lead to a term $\mathbb{B}\{Q_2, \lambda y.\tilde{\mathbb{B}}\{\tilde{R}\}, \ldots, \lambda y.\tilde{\mathbb{B}}\{\tilde{R}\}\}$.
   - $R_1 = (\lambda x.\mathbb{B}\{\tilde{R}, x, \ldots, x\}) \mathbin{@} \lambda y.\tilde{\mathbb{B}}\{R_2\}$. The resulting term on both paths is $\mathbb{B}\{\tilde{R}, \lambda y.\tilde{\mathbb{B}}\{Q_2\}, \ldots, \lambda y.\tilde{\mathbb{B}}\{Q_2\}\}$.
   - $R_1 = (\lambda x.\mathbb{B}\{x, \ldots, x\}) \mathbin{@} \lambda y.\tilde{\mathbb{B}}\{R_2, \tilde{R}\}$. The resulting term on both paths is $\mathbb{B}\{\lambda y.\tilde{\mathbb{B}}\{Q_2, \tilde{R}\}, \ldots, \lambda y.\tilde{\mathbb{B}}\{Q_2, \tilde{R}\}\}$.

   In the second and the last cases the redex $(\mathbb{A}, \tilde{R})$ gets duplicated, but in all 4 cases the set of residuals of $(\mathbb{A}, \tilde{R})$ is the same on both reduction paths.

   In the first two cases both resulting reductions are one-step. In the other two cases we observe, as before, that the reduction of the multiple copies of R_2 can be performed in any order with the same resulting term.

7. $(\mathbb{C}_1, R_1)$ contains $(\mathbb{C}_2, R_2)$, $(\mathbb{A}, \tilde{R})$ contains $(\mathbb{C}_1, R_1)$.

   In this case $\tilde{R} = (\lambda x.\tilde{N}) \mathbin{@} \tilde{V}$. Let Q_1' be a term s.t. $M_1 = \mathbb{C}_1\{R_1\} \xrightarrow{(\mathbb{C}_1, R_1)} \mathbb{C}_1\{Q_1\} \xrightarrow{(\mathbb{C}_2, R_2)/(\mathbb{C}_1, R_1)}{}^{*} \mathbb{C}_1\{Q_1'\}$ By case 1 it is also the case that

$\mathbb{C}_1\{R_1\} \xrightarrow{(\mathbb{C}_2,R_2)} \mathbb{C}_1\{R_1'\} \xrightarrow{(\mathbb{C}_1,R_1)/(\mathbb{C}_2,R_2)}{}^{*} \mathbb{C}_1\{Q_1'\}$. We have two subcases:

- $\mathbb{A}\{\tilde{R}\} = \mathbb{A}\{(\lambda x.\mathbb{B}\{R_1\}) \mathbin{@} \tilde{V}\} \xrightarrow{(\mathbb{C}_1,R_1);(\mathbb{C}_2,R_2)/(\mathbb{C}_1,R_1)}{}^{*} \mathbb{A}\{(\lambda x.\mathbb{B}\{Q_1'\}) \mathbin{@} \tilde{V}\}$, so the residual of $(\mathbb{A}, \tilde{R})$ on both reduction paths is $(\mathbb{A}, (\lambda x.\mathbb{B}\{Q_1'\}) \mathbin{@} \tilde{V})$,
- $\mathbb{A}\{\tilde{R}\} = \mathbb{A}\{(\lambda x.\tilde{N}) \mathbin{@} \lambda y.\mathbb{B}\{R_1\}\} \xrightarrow{(\mathbb{C}_1,R_1);(\mathbb{C}_2,R_2)/(\mathbb{C}_1,R_1)}{}^{*} \mathbb{A}\{(\lambda x.\tilde{N}) \mathbin{@} \lambda y.\mathbb{B}\{Q_1'\}\}$, so the residual of $(\mathbb{A}, \tilde{R})$ on both reduction paths is $(\mathbb{A}, (\lambda x.\tilde{N}) \mathbin{@} \lambda y.\mathbb{B}\{Q_1'\})$.

As before, if the reduction of R_1 duplicates R_2, then the order in which copies of R_2 are reduced does not matter.

8. $(\mathbb{C}_1, R_1)$ contains $(\mathbb{C}_2, R_2)$, $(\mathbb{A}, \tilde{R})$ is contained in $(\mathbb{C}_2, R_2)$. Similar to the previous cases.

9. $(\mathbb{C}_1, R_1)$ contains $(\mathbb{C}_2, R_2)$, $(\mathbb{A}, \tilde{R})$ is contained in $(\mathbb{C}_1, R_1)$, $(\mathbb{A}, \tilde{R})$ contains $(\mathbb{C}_2, R_2)$. Similar to the previous cases.

10. $(\mathbb{C}_1, R_1)$ and $(\mathbb{C}_2, R_2)$ are independent, $(\mathbb{A}, \tilde{R})$ contains both $(\mathbb{C}_1, R_1)$ and $(\mathbb{C}_2, R_2)$. Similar to the previous cases.

11. $(\mathbb{C}_1, R_1)$ and $(\mathbb{C}_2, R_2)$ are independent, $(\mathbb{A}, \tilde{R})$ contains $(\mathbb{C}_1, R_1)$, but not $(\mathbb{C}_2, R_2)$. Similar to the previous cases.

12. $(\mathbb{C}_1, R_1)$ and $(\mathbb{C}_2, R_2)$ are independent, $(\mathbb{A}, \tilde{R})$ is contained in $(\mathbb{C}_1, R_1)$. Similar to the previous cases.

□

Lemma 5.1.16 (Parallel Moves Lemma). *Let* $M_1' \xrightarrow{(\mathbb{C}_1,R_1)} M_2'$, $M_1', \xrightarrow{(\mathbb{C}_2,R_2)} M_3'$. *Then there exists* M_4' *such that* $M_2' \xrightarrow{(\mathbb{C}_2,R_2)/(\mathbb{C}_1,R_1)}{}^{*} M_4'$, $M_3' \xrightarrow{(\mathbb{C}_1,R_1)/(\mathbb{C}_2,R_2)}{}^{*} M_4'$. □

Proof. By induction on the number of redexes in marked(M_1'). The base case (marked(M_1') has one redex) is by lemma 5.1.15.

Suppose M_1'' is such that $|M_1''| = |M_1'|$ and marked(M_1'') contains n redexes. Let $M_1'' \xrightarrow{(\mathbb{C}_1,R_1)} M_2''$ and $M_1'' \xrightarrow{(\mathbb{C}_2,R_2)} M_3''$. By the inductive hypothesis assume that there exists M_4'' such that $M_2'' \xrightarrow{(\mathbb{C}_2,R_2)/(\mathbb{C}_1,R_1)}{}^{*} M_4''$ and $M_3'' \xrightarrow{(\mathbb{C}_1,R_1)/(\mathbb{C}_2,R_2)}{}^{*} M_4''$.

Let M_1''' be such that $|M_1'''| = |M_1'|$ and marked$(M_1''') =$ marked$(M_1'') \cup \{(\mathbb{A}, \tilde{R})\}$, where $(\mathbb{A}, \tilde{R}) \notin$ marked(M_1'''). Let $\tilde{M}_1'$ be such that $|\tilde{M}_1'| = |M_1'|$ and marked$(\tilde{M}_1') = \{(\mathbb{A}, \tilde{R})\}$. Let $\tilde{M}_1' \xrightarrow{(\mathbb{C}_1,R_1)} \tilde{M}_2'$ and $\tilde{M}_1' \xrightarrow{(\mathbb{C}_2,R_2)} \tilde{M}_3'$. By lemma 5.1.15 there exists $\tilde{M}_4'$ such that $\tilde{M}_2' \xrightarrow{(\mathbb{C}_2,R_2)/(\mathbb{C}_1,R_1)}{}^{*} \tilde{M}_4'$ and $\tilde{M}_3' \xrightarrow{(\mathbb{C}_1,R_1)/(\mathbb{C}_2,R_2)}{}^{*} \tilde{M}_4'$.

By lemma 5.1.15 all sequences of the form $\xrightarrow{(\mathbb{C},R)/(\mathbb{C}',R')}{}^{*}$ end at the same term. Therefore we can assume that the sequences $\xrightarrow{(\mathbb{C}_2,R_2)/(\mathbb{C}_1,R_1)}{}^{*}$ and $\xrightarrow{(\mathbb{C}_1,R_1)/(\mathbb{C}_2,R_2)}{}^{*}$ constructed for the initial marked term M_1' in the inductive hypothesis. Then $\tilde{M}_4'$ is the same term as M_4' up to marked redexes, i.e. $|\tilde{M}_4'| = |M_4'|$. By the same argument $|M_4''| = |M_4'|$.

Finally, by definition of marked reductions 4.2.9 and by definition of residuals of sets of redexes 4.2.7 $M_1''' \xrightarrow{(\mathbb{C}_1,R_1)} M_2'''$, where marked$(M_2''') =$ marked$(M_2'') \cup$ marked$(\tilde{M}_2')$, and $M_1''' \xrightarrow{(\mathbb{C}_2,R_2)} M_3'''$, where marked$(M_3''') =$ marked$(M_3'') \cup$marked$(\tilde{M}_3')$ imply that $M_2''' \xrightarrow{(\mathbb{C}_2,R_2)/(\mathbb{C}_1,R_1)}{}^{*} M_4'''$, where marked$(M_4''') =$ marked$(M_4'') \cup$marked$(\tilde{M}_4')$, and $M_3''' \xrightarrow{(\mathbb{C}_1,R_1)/(\mathbb{C}_2,R_2)}{}^{*} M_4'''$. □

5.1.4 Confluence of $\mathcal{T}$

An important corollary of the finiteness of developments and the parallel move lemma is a so-called strip lemma which allows, in particular, to prove confluence of the calculus relation. Our proof of confluence of $\longrightarrow$ is similar to that in [Bar84] (Chapters 11 and 12).

Lemma 5.1.17 (Strip Lemma). *Let $M_1' \longrightarrow M_2'$, $M_1' \longrightarrow^* M_3'$. Then there exists M_4' such that $M_2' \longrightarrow^* M_4'$, $M_3' \longrightarrow^* M_4'$.* □

Proof. Let $(\mathbb{C}, R)$ be the redex reduced in the reduction $M'_1 \longrightarrow M'_2$, and let S be the reduction sequence $M'_1 \longrightarrow^* M'_3$.

Claim 1. Given $M'_1 \xrightarrow{(\mathbb{C},R)} M'_2$ and $M'_1 \xrightarrow{S}{}^* M'_3$, there exists M'_4 such that $M'_2 \xrightarrow{S'}{}^* M'_4$ for some sequence S', and $M'_3 \xrightarrow{(\mathbb{C},R)/S}{}^* M'_4$.

We prove the claim by induction on the number of steps in S. Base case follows immediately from lemma 5.1.16. Induction step: suppose Claim 1 holds for a sequence S of n steps. Let us consider a sequence $S; (\mathbb{A}, \tilde{R})$ of $n+1$ steps such that $M'_1 \xrightarrow{S}{}^* N'_1 \xrightarrow{(\mathbb{A},\tilde{R})} M'_3$. By the inductive hypothesis there exists N'_2 such that $M'_2 \xrightarrow{S'}{}^* N'_2$ and $M'_3 \xrightarrow{(\mathbb{C},R)/S}{}^* N'_2$.

Let $\hat{F} = (\mathbb{C}, R)/(S; (\mathbb{A}, \tilde{R}))$, and let $\tilde{F} = (\mathbb{A}, \tilde{R})/S''$, where S'' is the reduction sequence $N'_1 \xrightarrow{(\mathbb{C},R)/S}{}^* N'_2$. Let $(\mathbb{C}', R')$ be the first redex reduced in S'', i.e. $N'_1 \xrightarrow{(\mathbb{C}',R')} N'_3$. By lemma 5.1.16 there exists M' such that $M'_3 \xrightarrow{(\mathbb{C}',R')/(\mathbb{A},\tilde{R})}{}^* M'$ and $N'_1 \xrightarrow{(\mathbb{C}',R')} N'_3 \xrightarrow{(\mathbb{A},\tilde{R})/(\mathbb{C}',R')}{}^* M'$. Now we apply lemma 5.1.16 to the first redex in the sequence $\xrightarrow{(\mathbb{A},\tilde{R})/(\mathbb{C}',R')}{}^*$ and the first redex in $\xrightarrow{(\mathbb{C}',R')/(\mathbb{A},\tilde{R})}{}^*$, and so on. This process is guaranteed to terminate, because all redexes reduced in all the constructed sequences are residuals of redexes in the sets $\hat{F}$ or $\tilde{F}$. If we mark all redexes in these two sets, then the reduction sequences would be developments. Therefore by lemma 5.1.10 the length of such reductions is limited, so the construction will eventually terminate. □

Confluence of the calculus relation of the marked term calculus immediately follows from the strip lemma 5.1.17 (see [Bar84], Chapter 11 for the proof). The confluence of the unmarked calculus follows by erasing the marks.

Theorem 5.1.18 (Confluence of the Marked $\mathcal{T}$). *If $M'_1 \longrightarrow^*_{\mathcal{T}} M'_2$) and $M'_1 \longrightarrow^*_{\mathcal{T}} M'_3$, then there exists M'_4 such that $M'_2 \longrightarrow^*_{\mathcal{T}} M'_4$, $M'_3 \longrightarrow^*_{\mathcal{T}} M'_4$.* □

5.1.5 Class Preservation and Standardization in $\mathcal{T}$

In order to prove computational soundness it remains to show class preservation and standardization. Before we prove these two properties, we show some technical facts needed for the proofs.

Lemma 5.1.19. *Let* $M \circ\!\!\xrightarrow{(\mathbb{C},R_1)}_{\mathcal{T}} N$, *where* $N = R_2$ *is a redex in* $\mathcal{T}$. *Then* $N = (\lambda x.N_1)\ @\ V$ *and* $M = (\lambda x.\mathbb{A}\{R_1\})\ @\ V$ *or* $M = (\lambda x.N_1)\ @\ \lambda y.\mathbb{A}\{R_1\}$. □

Proof. Given $M \circ\!\!\xrightarrow{(\mathbb{C},R_1)}_{\mathcal{T}} N$, where $N = R_2$ is a redex, suppose $N = c_1\ op\ c_2$. By definition of a non-evaluation step $M = \mathbb{C}\{R_1\} \circ\!\!\xrightarrow{(\mathbb{C},R_1)}_{\mathcal{T}} \mathbb{C}\{Q_1\} = N$, where $\mathbb{C}$ is a non-evaluation context, i.e., in particular $\mathbb{C} \neq \Box$, $\mathbb{C} \neq c_1\ op\ \Box$, and $\mathbb{C} \neq \Box\ op\ c_2$. Therefore there are no possibilities for $\mathbb{C}$ in this case.

Now suppose $N = (\lambda x.N_1)\ @\ V$. Again we have $M = \mathbb{C}\{R_1\} \circ\!\!\xrightarrow{(\mathbb{C},R_1)}_{\mathcal{T}} \mathbb{C}\{Q_1\} = N$, where $\mathbb{C}$ is a non-evaluation context, i.e. $\mathbb{C} \neq \Box$, $\mathbb{C} \neq \Box\ @\ V$, and $\mathbb{C} \neq (\lambda x.N_1)\ @\ \Box$. The remaining cases are $\mathbb{C} = (\lambda x.\mathbb{A})\ @\ V$ and $\mathbb{C} = (\lambda x.N_1)\ @\ \lambda y.\mathbb{A}$, and the claim of the lemma is shown. □

Lemma 5.1.20. *Let* $M \circ\!\!\xrightarrow{(\mathbb{C},R_1)}_{\mathcal{T}} N$. *Then* $M = \mathbb{E}\{R\}$ *if and only if* $N = \mathbb{E}'\{R'\}$. *If* $M = \mathbb{E}\{R\}, N = \mathbb{E}'\{R'\}$, *then* $(\mathbb{E}', R') = (\mathbb{E}, R)/(\mathbb{C}, R_1)$ *(recall that the notation implies that* $(\mathbb{E}', R')$ *is the only residual of* $(\mathbb{E}, R)$*).* □

Proof. The proof is by induction on the structure of an evaluation context. We want to show that $M = \mathbb{E}\{R\}$ if and only if $N = \mathbb{E}'\{R'\}$ and $\mathbb{E}$ and $\mathbb{E}'$ are of the same shape (see 5 cases of definition of an evaluation context in $\mathcal{T}$ on figure 2.3).

Base case. Suppose $M = \mathbb{E}\{R\}$ and $\mathbb{E} = \Box$. Then $M = c_1\ op\ c_2$ or $M = \lambda x.M_1\ @\ V$. However, since $M \circ\!\!\longrightarrow_{\mathcal{T}} N$, it can not be the case that $M = c_1\ op\ c_2$. Therefore $M = (\lambda x.M_1)\ @\ V$. The redex R_1 reduced by the non-evaluation step may

occur either in M_1 or in V, i.e. there are the following two cases:

$$M = (\lambda x.\mathbb{A}\{R_1, x, \ldots, x\}) \ @ \ V \quad \circ\!\xrightarrow{(\mathbb{C},R_1)}_{\mathcal{T}} \quad (\lambda x.\mathbb{A}\{Q_1, x, \ldots, x\}) \ @ \ V = N$$
$$M = (\lambda x.\mathbb{A}\{x, \ldots, x\}) \ @ \ \lambda y.\mathbb{B}\{R_1\} \quad \circ\!\xrightarrow{(\mathbb{C},R_1)}_{\mathcal{T}} \quad (\lambda x.\mathbb{A}\{x, \ldots, x\}) \ @ \ \lambda y.\mathbb{B}\{Q_1\} = N$$

In both cases $N = \mathbb{E}'\{R_1'\}$, where $\mathbb{E}' = \Box$.

Now suppose that $N = \mathbb{E}'\{R_1'\}$, $\mathbb{E}' = \Box$. By lemma 5.1.19 $N = \lambda x.N_1 \ @ \ V$ and either $M = (\lambda x.\mathbb{A}\{R_1\}) \ @ \ V$ or $M = (\lambda x.N_1) \ @ \ \lambda y.\mathbb{A}\{R_1\}$. In both cases M is a redex, i.e. $M = \mathbb{E}\{R\}$, where $\mathbb{E} = \Box$.

Induction step. As an inductive hypothesis suppose that the claim holds for the evaluation subcontext of the context. We have 4 cases:

1. $M = \mathbb{E}\{R_2\}$, where $\mathbb{E} = \mathbb{E}_1 \ @ \ M_1$. Since $M \circ\!\xrightarrow{(\mathbb{C},R_1)}_{\mathcal{T}} N$, we have two possibilities: either $\mathbb{C} = \mathbb{C}_1 \ @ \ M_1$, in which case $\mathbb{C}_1$ is a non-evaluation context, or $\mathbb{C} = (\mathbb{E}_1\{R_2\}) \ @ \ \mathbb{C}_1$. In the former case $\mathbb{E}_1\{R_2\} = \mathbb{C}_1\{R_1\} \circ\!\xrightarrow{(\mathbb{C}_1,R_1)}_{\mathcal{T}} \mathbb{C}_1\{Q_1\}$, and by inductive hypothesis (since $\mathbb{E}_1$ is a subcontext of $\mathbb{E}$) we have $\mathbb{C}_1\{Q_1\} = \mathbb{E}_1'\{R_2'\}$, and hence $N = \mathbb{E}'\{R_2'\}$, where $\mathbb{E}' = \mathbb{E}_1' \ @ \ M_1$. In the latter case $(\mathbb{E}_1\{R_2\}) \ @ \ \mathbb{C}_1\{R_1\} \circ\!\xrightarrow{(\mathbb{C},R_1)}_{\mathcal{T}} (\mathbb{E}_1\{R_2\}) \ @ \ \mathbb{C}_1\{Q_1\}$, i.e. $N = \mathbb{E}'\{R_2\}$, where $\mathbb{E}' = \mathbb{E}_1 \ @ \ \mathbb{C}_1\{Q_1\}$.

   Now suppose that $N = \mathbb{E}'\{R_2'\}$, where $\mathbb{E}' = \mathbb{E}_1' \ @ \ N_1$. Since $M \circ\!\xrightarrow{(\mathbb{C},R_1)}_{\mathcal{T}} N$, we have two cases:

   - $M = (\mathbb{C}_1\{R_1\}) \ @ \ N_1$, where $\mathbb{C}_1$ is a non-evaluation context. Then $\mathbb{C}_1\{R_1\} \circ\!\xrightarrow{(\mathbb{C}_1,R_1)}_{\mathcal{T}} \mathbb{C}_1\{Q_1\} = \mathbb{E}_1'\{R_2'\}$, and by the inductive hypothesis $\mathbb{C}_1\{R_1\} = \mathbb{E}_1\{R_2\}$, so $M = (\mathbb{E}_1\{R_2\}) \ @ \ N_1$.
   - $M = (\mathbb{E}_1'\{R_2'\}) \ @ \ \mathbb{C}_1\{R_1\}$, where $\mathbb{C}_1$ may be an evaluation or a non-evaluation context. In this case $M = \mathbb{E}\{R_2'\}$, where $\mathbb{E} = \mathbb{E}_1' \ @ \ \mathbb{C}_1\{R_1\}$, i.e. $\mathbb{E}$ is of the same shape as $\mathbb{E}'$.

2. Suppose $M = (\lambda x.M_1)\ @\ \mathbb{E}\{R_2\}$. Then either $M = (\lambda x.\mathbb{C}_1\{R_1\})\ @\ \mathbb{E}\{R_2\}$ or $M = (\lambda x.M_1)\ @\ \mathbb{C}_1\{R_1\}$, in the latter case $\mathbb{C}_1$ is a non-evaluation context. Similarly to the first part of case 1 above, we get $N = (\lambda x.N_1)\ @\ \mathbb{E}\{R_2\}$ or $N = (\lambda x.M_1)\ @\ \mathbb{E}'_1\{R'_2\}$.

   Now suppose $N = (\lambda x.N_1)\ @\ \mathbb{E}'\{R'_2\}$. Then, as in the previous case, either $M = (\lambda x.N_1)\ @\ \mathbb{C}_1\{R_1\}$, where $\mathbb{C}_1$ is a non-evaluation context, or $M = \mathbb{C}_1\{R_1\}\ @\ \mathbb{E}'\{R'_2\}$. In the former case $M = (\lambda x.N_1)\ @\ \mathbb{E}\{R_2\}$ by inductive hypothesis, analogously to case 1. In the latter case note that $\mathbb{C}_1 \neq \Box$, since otherwise $\mathbb{C} = \Box\ @\ N_2$ is an evaluation context ($N_2 = \mathbb{E}'\{R'_2\}$). Therefore $M = (\lambda x.\mathbb{C}''_1\{R_1\})\ @\ \mathbb{E}'\{R'_2\}$ for some $\mathbb{C}''_1$, and the claim of the lemma holds.

3. The case of the evaluation context of the form $\mathbb{E}\ op\ M$ is analogous to case 1.

4. The case of the evaluation context of the form $c\ op\ \mathbb{E}$ is analogous to case 2.

□

Lemma 5.1.21 (Class Preservation). *If* $M \circ\!\xrightarrow{(\mathbb{C},R_1)}_{\mathcal{T}} N$, *then* $Cl_{\mathcal{T}}(M) = Cl_{\mathcal{T}}(N)$.

□

Proof. The cases of classes $\mathbf{const}(c), \mathbf{var}$, and $\mathbf{abs}$ are straightforward. Below we show the case $\mathbf{stuck}(l)$:

Suppose $Cl_{\mathcal{T}}(M) = \mathbf{stuck}(l)$. By induction on the structure of M we show that $Cl_{\mathcal{T}}(N) = \mathbf{stuck}(l)$.

Base case: the case when $M = l$ is impossible, since there is no N such that $M \circ\!\!\rightarrow_{\mathcal{T}} N$. Instead we use the following as base cases for M: $l\ @\ M_1$, $(\lambda x.M_1)\ @\ l$, $l\ op\ M_1$, $c\ op\ l$. If $M = l\ @\ M_1$, then $M \circ\!\!\rightarrow_{\mathcal{T}} N = l\ @\ N_1$, where $M_1 \circ\!\!\rightarrow_{\mathcal{T}} N_1$, i.e. $Cl_{\mathcal{T}}(N) = \mathbf{stuck}(l)$. Similarly, $N = (\lambda x.N_1)\ @\ l$, where $M_1 \rightarrow_{\mathcal{T}} N_1$ (i.e.

$\lambda x.M_1 \circ\!\!\rightarrow_{\mathcal{T}} \lambda x.N_1$) if $M = (\lambda x.M_1)$ @ l, $N = l\ op\ N_1$, where $M_1 \circ\!\!\rightarrow_{\mathcal{T}} N_1$ if $M = l\ op\ M_1$, and it may not be the case that $M \circ\!\!\rightarrow_{\mathcal{T}} N$ if $M = c\ op\ l$.

As the inductive hypothesis assume that if $M_1 = \mathbb{E}\{l\}$ is a subterm of M and $M_1 \circ\!\!\rightarrow_{\mathcal{T}} N_1$, then $N_1 = \mathbb{E}'\{l\}$. Suppose $M = (\mathbb{E}\{l\})$ @ $M' \circ\!\!\rightarrow_{\mathcal{T}} N$. If the reduction step is $(\mathbb{E}\{l\})$ @ $M' \circ\!\!\rightarrow_{\mathcal{T}} N'$ @ M', then by inductive hypothesis $N' = \mathbb{E}'\{l\}$, so $Cl_{\mathcal{T}}(N) = Cl_{\mathcal{T}}((\mathbb{E}'\{l\})$ @ $M') = \mathbf{stuck}(l)$. If the step is $(\mathbb{E}\{l\})$ @ $M' \circ\!\!\rightarrow_{\mathcal{T}}$ $(\mathbb{E}\{l\})$ @ N', then $Cl_{\mathcal{T}}(N) = \mathbf{stuck}(l)$. The cases $M = (\lambda x.M')$ @ $\mathbb{E}\{l\}$, $M = \mathbb{E}\{l\}\ op\ c$, and $M = c\ op\ \mathbb{E}\{l\}$ are analogous.

Now suppose $Cl_{\mathcal{T}}(N) = \mathbf{stuck}(l)$, and show that $Cl_{\mathcal{T}}(M) = \mathbf{stuck}(l)$.

Base case: as above, the case when $N = l$ is impossible, since there is no M such that $M \circ\!\!\rightarrow_{\mathcal{T}} l$. Therefore we consider the following base cases for N: $N = l$ @ M_1, $N = (\lambda x.M_1)$ @ l, $N = l\ op\ M_1$. The case $N = c\ op\ l$ is impossible. In all three base cases it easily follows that M has the same shape as N, i.e. $Cl_{\mathcal{T}}(M) = Cl_{\mathcal{T}}(N) = \mathbf{stuck}(l)$.

The induction step is similar to the one above. We assume that if $N_1 = \mathbb{E}\{l\}$ is a subterm of N and $M_1 \circ\!\!\rightarrow_{\mathcal{T}} N_1$, then $M_1 = \mathbb{E}'\{l\}$. By considering all cases of N we show that $Cl_{\mathcal{T}}(M) = \mathbf{stuck}(l)$ for every N.

It follows from lemma 5.1.20 that $Cl_{\mathcal{T}}(M) = \mathbf{evaluatable}$ if and only if $Cl_{\mathcal{T}}(N) = \mathbf{evaluatable}$. This concludes the proof since the remaining class **error** is defined as the class of terms that do not belong to any of the above classes. □

Our proof of standardization of developments follows the approach described in section 4.3, i.e. we show property 4.2.18 (replacing a non-standard pair in developments), and since we have shown the boundedness of developments (lemma 5.1.10), standardization of developments follows by lemma 4.3.3. The following lemma implies property 4.2.18 of $\mathcal{T}$.

Lemma 5.1.22. *If* $M_1 \circ\xrightarrow{(\mathbb{C},R_1)} M_2 \xRightarrow{(\mathbb{E}',R_2')} M_3$. *Then there exists* M_4 *such that* $M_1 \xRightarrow{(\mathbb{E},R_2)} M_4 \xrightarrow{(\mathbb{C},R_1)/(\mathbb{E},R_2)}{}^{*} M_3$, *where* $(\mathbb{E}', R_2') = (\mathbb{E}, R_2)/(\mathbb{C}, R_1)$. □

Proof. The proof is by induction on the structure of the evaluation context $\mathbb{E}'$, similarly to the proof of lemma 5.1.20 above.

Base case. Suppose $M_2 = \mathbb{E}'\{R_2'\}$ and $\mathbb{E}' = \Box$. By lemma 5.1.19 $M_2 = (\lambda x.N)\ @\ V$, and either $M_1 = (\lambda x.\mathbb{A}\{R_1, x, \ldots, x\})\ @\ V$ (where $\mathbb{A}$ is an $n+1$-hole context if x occurs in N n times) or $M_1 = (\lambda x.N)\ @\ \lambda y.\mathbb{B}\{R_1\}$. Suppose $R_1 \rightsquigarrow Q_1$. In the first case, we have:

$$\begin{array}{lllll}
(\lambda x.\mathbb{A}\{R_1, x, \ldots, x\})\ @\ V & \Rightarrow_{\mathcal{T}} & \mathbb{A}\{R_1, V, \ldots, V\} & \rightarrow_{\mathcal{T}} & \mathbb{A}\{Q_1, V, \ldots, V\}, \\
(\lambda x.\mathbb{A}\{R_1, x, \ldots, x\})\ @\ V & \circ\!\rightarrow_{\mathcal{T}} & (\lambda x.\mathbb{A}\{Q_1, x, \ldots, x\})\ @\ V & \Rightarrow_{\mathcal{T}} & \mathbb{A}\{Q_1, V, \ldots, V\}.
\end{array}$$

If we take $M_4 = \mathbb{A}\{R_1, V, \ldots, V\}$, then the claim of the lemma holds (note that the redex reduced in $\mathbb{A}\{R_1, V, \ldots, V\} \rightarrow_{\mathcal{T}} \mathbb{A}\{Q_1, V, \ldots, V\}$ is the residual of the non-evaluation redex w.r.t. the standard redex, i.e. the application). The second case is similar. Let $N = \mathbb{A}\{x, \ldots, x\}$, where $\mathbb{A}$ is an n-hole context. Then

$$\begin{array}{lllll}
(\lambda x.\mathbb{A}\{x, \ldots, x\})\ @\ \lambda y.\mathbb{B}\{R_1\} & \Rightarrow_{\mathcal{T}} & \mathbb{A}\{\lambda y.\mathbb{B}\{R_1\}, \ldots, \lambda y.\mathbb{B}\{R_1\}\} & \rightarrow_{\mathcal{T}}^{*} & \\
\mathbb{A}\{\lambda y.\mathbb{B}\{Q_1\}, \ldots, \lambda y.\mathbb{B}\{Q_1\}\}, & & & & \\
(\lambda x.\mathbb{A}\{x, \ldots, x\})\ @\ \lambda y.\mathbb{B}\{R_1\} & \circ\!\rightarrow_{\mathcal{T}} & (\lambda x.\mathbb{A}\{x, \ldots, x\})\ @\ \lambda y.\mathbb{B}\{Q_1\} & \Rightarrow_{\mathcal{T}} & \\
\mathbb{A}\{\lambda y.\mathbb{B}\{Q_1\}, \ldots, \lambda y.\mathbb{B}\{Q_1\}\}. & & & &
\end{array}$$

We take $M_4 = \mathbb{A}\{\lambda y.\mathbb{B}\{R_1\}, \ldots, \lambda y.\mathbb{B}\{R_1\}\}$ and observe that all redexes reduced in $\mathbb{A}\{\lambda y.\mathbb{B}\{R_1\}, \ldots, \lambda y.\mathbb{B}\{R_1\}\} \rightarrow_{\mathcal{T}}^{*} \mathbb{A}\{\lambda y.\mathbb{B}\{Q_1\}, \ldots, \lambda y.\mathbb{B}\{Q_1\}\}$ are residuals of the non-evaluation redex.

Induction Step. As in the proof of lemma 5.1.19, we have 4 cases. We only show one case, the rest is similar.

- Suppose $M_2 = (\mathbb{E}'_1\{R'_2\})$ @ $N \Longrightarrow_{\mathcal{T}} (\mathbb{E}'_1\{Q'_2\})$ @ $N = M_3$. Since $M_1 \circ\!\!\xrightarrow{(\mathbb{C},R_1)}_{\mathcal{T}} M_2$, we have one of the following: either $M_1 = (\mathbb{C}_1\{R_1\})$ @ N, or $M_1 = (\mathbb{E}'_1\{R'_2\})$ @ $\mathbb{C}_1\{R_1\}$.

  In the former case $\mathbb{C}_1\{R_1\} \circ\!\!\xrightarrow{(\mathbb{C}_1,R_1)}_{\mathcal{T}} \mathbb{E}'_1\{R'_2\}$, hence by lemma 5.1.19 $\mathbb{C}_1\{R_1\} = \mathbb{E}_1\{R_2\}$, and by the inductive hypothesis $(\mathbb{E}'_1, R'_2) = (\mathbb{E}_1, R_2)/(\mathbb{C}_1, R_1)$, and there exists M'_4 such that $\mathbb{E}_1\{R_2\} \xRightarrow{(\mathbb{E}_1,R_2)}_{\mathcal{T}} M'_4 \xrightarrow{(\mathbb{C},R_1)/(\mathbb{E}_1,R_2)}{}^{*}_{\mathcal{T}} \mathbb{E}'_1\{Q'_2\}$. Therefore if we take $M_4 = M'_4$ @ N, then the claim of the lemma holds.

  In the latter case $M_1 = (\mathbb{E}'_1\{R'_2\})$ @ $\mathbb{C}_1\{R_1\} \Longrightarrow_{\mathcal{T}} (\mathbb{E}'_1\{Q'_2\})$ @ $\mathbb{C}_1\{R_1\} \longrightarrow_{\mathcal{T}} (\mathbb{E}'_1\{Q'_2\})$ @ $\mathbb{C}_1\{Q_1\} = M_3$ (assuming $R_1 \rightsquigarrow Q_1$), and the claim holds.

Cases when $M_2 = (\lambda x.N)$ @ $\mathbb{E}'_1\{R'_2\}$, $M_2 = \mathbb{E}'_1\{R'_2\}$ *op* N, and $M_2 = c_1$ *op* $\mathbb{E}'_1\{R'_2\}$, are similar.

In all the cases we observe that $(\mathbb{E}', R'_2) = (\mathbb{E}, R_2)/(\mathbb{C}, R_1)$. □

In the case when both given redexes in M_1 are marked, lemma 5.1.22 implies the following:

Corollary 5.1.23. *If* $M'_1 \underset{dev}{\circ\!\!\longrightarrow} M'_2 \underset{dev}{\Longrightarrow} M'_3$, *then there exists* M'_4 *such that* $M'_1 \underset{dev}{\Longrightarrow} M'_4 \underset{dev}{\longrightarrow}{}^{*} M'_3$. □

Proof. Let $(\mathbb{C}, R_1)$ be the redex reduced in the step $M'_1 \underset{dev}{\circ\!\!\longrightarrow} M'_2$, and $(\mathbb{E}', R'_2)$ be the redex reduced in $M'_2 \underset{dev}{\Longrightarrow} M'_3$. Then by lemma 5.1.22 there exists a redex $(\mathbb{E}, R_2)$ of M'_1 such that for some M'_4 $M'_1 \xRightarrow{(\mathbb{E},R_2)} M'_4 \xrightarrow{(\mathbb{C},R_1)/(\mathbb{E},R_2)}{}^{*} M'_3$. Since both given steps are developments, i.e. they reduced marked redexes, it must be the case that $\{(\mathbb{C}, R_1), (\mathbb{E}, R_2)\} \subseteq \text{marked}(M'_1)$. Then $M'_1 \xRightarrow{(\mathbb{E},R_2)} M'_4$ is a development step, and the sequence $M'_4 \xrightarrow{(\mathbb{C},R_1)/(\mathbb{E},R_2)}{}^{*} M'_3$ is a development sequence (since all the redexes reduced in this sequence are marked).

If marked$(M_1') = \{(\mathbb{C}, R_1), (\mathbb{E}, R_2)\}$, then marked$(M_3') = \emptyset$ and we are done. Otherwise let $(\mathbb{A}, R) \notin \{(\mathbb{C}, R_1), (\mathbb{E}, R_2)\}$, $(\mathbb{A}, R) \in$ marked(M_1'). Let us consider a term M_1'' such that $|M_1'| = |M_1''|$ and marked$(M_1'') = \{(\mathbb{A}, R)\}$. By lemma 5.1.22 $M_1'' \underset{dev}{\circ\!\!\xrightarrow{(\mathbb{C},R_1)}} M_2'' \underset{dev}{\overset{(\mathbb{E}',R_2')}{\Longrightarrow}} M_3''$ implies that there exists M_4'' such that $M_1'' \overset{(\mathbb{E},R_2)}{\Longrightarrow} M_4''$ $\xrightarrow{(\mathbb{C},R_1)/(\mathbb{E},R_2)}{}^{*} M_3''$, where $(\mathbb{E}', R_2') = (\mathbb{E}, R_2)/(\mathbb{C}, R_1)$. Since $|M_1'| = |M_1''|$ and the reduction from M_1'' to M_4'' reduces the same redex as the reduction from M_1' to M_4', we have $|M_4'| = |M_4''|$.

By lemma 5.1.15 $M_1'' \underset{dev}{\circ\!\!\xrightarrow{(\mathbb{C},R_1)}} M_2''$ and $M_1'', \{(\mathbb{A}, R)\}) \overset{(\mathbb{E},R_2)}{\Longrightarrow} M_4''$ imply that there exists M_3''' such that $M_2'' \xrightarrow{(\mathbb{E},R_2)/(\mathbb{C},R_1)}{}^{*} M_3'''$ and $M_4'' \xrightarrow{(\mathbb{C},R_1)/(\mathbb{E},R_2)}{}^{*} M_3'''$. Since $(\mathbb{E}', R_2') = (\mathbb{E}, R_2)/(\mathbb{C}, R_1)$, we get $|M_3'''| = \overline{M_3'} = |M_3''|$. By lemma 5.1.15 all reductions $\xrightarrow{(\mathbb{C},R_1)/(\mathbb{E},R_2)}{}^{*}$ lead to the same result, therefore we may assume that the reductions $M_4' \xrightarrow{(\mathbb{C},R_1)/(\mathbb{E},R_2)}{}^{*} M_3'$ above and $M_4'' \xrightarrow{(\mathbb{C},R_1)/(\mathbb{E},R_2)}{}^{*} M_3'''$.

Now, when we have shown the claim for one redex, we can show the general claim by induction on the number of redexes in marked(M_1'), with the term M_1' such that marked$(M_1') = \{(\mathbb{C}, R_1), (\mathbb{E}, R_2)\})$ as the base case. The induction is similar to that in the proof of the parallel moves lemma 5.1.16 and uses definition 4.2.7 to join sets of residuals. □

Now we can show standardization of developments for $\mathcal{T}$:

Lemma 5.1.24 (Standardization of Developments in $\mathcal{T}$). *Given $M_1' \underset{dev}{\longrightarrow}{}^{*} M_2'$, there exists M' such that $M_1' \underset{dev}{\Longrightarrow}{}^{*} M' \underset{dev}{\circ\!\!\longrightarrow}{}^{*} M_2'$.* □

Proof. We have shown that the calculus $\mathcal{T}$ has boundedness of developments property 4.2.14 (lemma 5.1.10). Corollary 5.1.23 above states that $\mathcal{T}$ has property 4.2.18. Therefore by lemma 4.3.3 it has standardization of developments. □

Another property implied by lemma 5.1.22 is the elementary lift diagram (property 4.6.1).

Lemma 5.1.25 (Elementary Lift Diagram). *If* $M_1' \circ\!\xrightarrow[cd]{(\mathbb{C},R)} M_2' \Longrightarrow M_4'$, *then there exists* M_3' *such that* $M_1' \xRightarrow{(\mathbb{E},R')} M_3' \xrightarrow[cd]{(\mathbb{C},R)/(\mathbb{E},R')}{}^{*} M_4'$. □

Proof. Follows from lemma 5.1.22 in the case when only the redex $(\mathbb{C}, R)$ is marked. □

Finally, we get the following result:

Theorem 5.1.26 (Standardization). *If* $M' \rightarrow_{\mathcal{T}}^{*} N'$, *then there exists* M_1' *such that* $M' \Longrightarrow_{\mathcal{T}}^{*} M_1' \circ\!\!\rightarrow_{\mathcal{T}}^{*} N'$. □

Proof. $\mathcal{T}$ has the lift property 4.8.1 by theorem 4.8.4, lemma 5.1.25, and lemma 5.1.24. By results of section 3.4.3 it has standardization. □

5.1.6 Computational Soundness of $\mathcal{T}$

Theorem 5.1.27 (Computational Soundness of $\mathcal{T}$). *For any* $M, N \in \mathbf{Term}_{\mathcal{T}}$ *if* $M \leftrightarrow_{\mathcal{T}} N$, *then* $Outcome_{\mathcal{T}}(M) = Outcome_{\mathcal{T}}(N)$. □

Proof. The marked calculus associated with $\mathcal{T}$ has confluence (theorem 5.1.18), class preservation (lemma 5.1.21), and standardization (theorem 5.1.26). By lemma 4.8.3 standardization (i.e. lift) property holds for terms of $\mathcal{T}$ without marked redexes. Similarly confluence for marked terms implies confluence for terms with no marked redexes. Therefore by theorem 3.3.6 $\mathcal{T}$ is computationally sound. □

5.2 Soundness of the Calculus of Records

This section shows computational soundness of the record calculus defined in section 2.2.3. Unlike a more detailed presentation in [MT02], we do not distinguish between visible and hidden labels, and do not introduce garbage collection rule.

Module calculus (based on the calculus of records) identifies modules up to consistent renaming of hidden components. The approach used in this ection is applicable to modules with hidden labels, since all results shown on concrete records immediately extend to α-equivalence classes of modules. This is due to the fact that a reduction from one α-equivalence class to another exists if and only if such reduction exists for all representatives of the first α-equivalence class. See [MT02] for details.

We use $\mathcal{C}$ to denote the calculus of records, and $\mathbf{Term}_{\mathcal{C}}$, $\mathbf{Context}_{\mathcal{C}}$, and $\mathbf{EvalContext}_{\mathcal{C}}$ to denote the sets of records, one-hole record contexts, and record evaluation contexts, respectively.

5.2.1 Definitions

Sections 4.2 and 4.5 give axiomatic definitions of a set of residuals and of a γ-development step, respectively. In this section we fill in calculus-specific details of these definitions and show that they satisfy the requirements stated in the axiomatic definitions.

Similarly to definition of residuals in the term calculus (Definition 5.1.3) we define a set of residuals of a redex using multi-hole contexts and the greatest lower bound of contexts. To be able to define filling of a context in a record, we assume that labels in $\mathbf{Label}$ are ordered according to some total order $<$ (for instance, label names are ordered lexicographically). This allows us to specify the order in which holes of a context are filled. Note that in this section we are dealing with particular records, not with α-equivalence classes of records, and do not need to make a distinction between visible and hidden labels.

Definition 5.2.1. A multi-hole record context is defined as follows:

$$\mathbb{D} = [l_1 \mapsto \mathbb{C}_1, \ldots, l_n \mapsto \mathbb{C}_n],$$

where $\mathbb{C}_i$ are multi-hole term contexts.

Suppose that $l_1 < l_2 < \cdots < l_n$, and $H(\mathbb{C}_i) = m_i$ for $1 \leq i \leq n$. Then for a context $\mathbb{D}$ given above by definition

$$\mathbb{D}\{\mathbb{A}_{1,1}, \ldots, \mathbb{A}_{1,m_1}, \ldots, \mathbb{A}_{n,1}, \ldots, \mathbb{A}_{m_n}\} =$$
$$[l_1 \mapsto \mathbb{C}_1\{\mathbb{A}_{1,1}, \ldots, \mathbb{A}_{1,m_1}\}, \ldots, l_n \mapsto \mathbb{C}_n\{\mathbb{A}_{n,1}, \ldots, \mathbb{A}_{m_n}\}].$$

Recall that $\mathbb{A}_{i,j}$ are multi-hole term contexts. □

Definition 5.2.2. The greatest lower bound of two record contexts is defined as

$$\mathrm{glb}([l_1 \mapsto \mathbb{C}_1, \ldots, l_n \mapsto \mathbb{C}_n], [l_1 \mapsto \mathbb{C}'_1, \ldots, l_n \mapsto \mathbb{C}'_n]) =$$
$$[l_1 \mapsto \mathrm{glb}(\mathbb{C}_1, \mathbb{C}'_1), \ldots, l_n \mapsto \mathrm{glb}(\mathbb{C}_n, \mathbb{C}'_n)],$$

provided $\mathrm{glb}(\mathbb{C}_i, \mathbb{C}'_i)$ is defined for all i such that $1 \leq i \leq n$, otherwise it is undefined. Note that for glb to be defined, the two record contexts must have the same labels of all components.

As for the term calculus, for $n > 2$

$$\mathrm{glb}(\mathbb{D}_1, \mathbb{D}_2, \ldots, \mathbb{D}_n) = \mathrm{glb}(\mathrm{glb}(\mathbb{D}_1, \mathbb{D}_2), \mathbb{D}_3, \ldots, \mathbb{D}_n).$$

□

Definition 5.2.3 (Projection). If $D = [l_1 \mapsto M_1, \ldots, l_n \mapsto M_n]$, then M_i is called a *projection* of D on a label l_i, denoted by $D \downarrow l_i$.

If $\mathbb{D} = [l_1 \mapsto \mathbb{C}_1, \ldots, l_n \mapsto \mathbb{C}_n]$, then $\mathbb{C}_i$ is called a *projection* of $\mathbb{D}$ on a label l_i, denoted by $\mathbb{D} \downarrow l_i$. □

Definition 5.2.4. Let $(\mathbb{D}, R)$, where $\mathbb{D} = [l_i \overset{k}{\underset{i=1}{\mapsto}} M_i, l_k \mapsto \mathbb{C}, l_i \overset{n}{\underset{i=k+1}{\mapsto}} M_i]$, be a record redex. Then l_k is called *the binding label* of $(\mathbb{D}, R)$, and $(\mathbb{C}, R)$ is called *the term projection of the redex* $(\mathbb{D}, R)$ (denoted $(\mathbb{D}, R) \downarrow$). Note that $\mathbb{C} = \mathbb{D} \downarrow l_k$.

If $F = \{(\mathbb{D}_1, R_1), \ldots, (\mathbb{D}_n, R_n)\}$, then by definition

$$F \downarrow l = \{(\mathbb{D}_i, R_i) \mid (\mathbb{D}_i, R_i) \in F, \text{ and } l \text{ is the binding label of } (\mathbb{D}_i, R_i)\}.$$

If D' is a record in the marked version of $\mathcal{C}$, then $D' \downarrow l$ is a marked term M' such that $|D'| \downarrow l = |M'|$ and $\text{marked}(M') = \text{marked}(D') \downarrow l$. □

The following definition of independent subterms and redexes in the record calculus is analogous to definition 5.1.11 of independent subterms and redexes in the term calculus. Note that the "term" part of a record subterm is a term of the calculus $\mathcal{T}$, not $\mathcal{C}$.

Definition 5.2.5. 1. A $\mathcal{T}$-subterm of a record D is a pair $(\mathbb{D}, M)$, where $\mathbb{D} \in \mathbf{Context}_{\mathcal{C}}$, $M \in \mathbf{Term}_{\mathcal{T}}$, and $\mathbb{D}\{M\} = D$.

2. Two $\mathcal{T}$-subterms $(\mathbb{D}_1, M_1)$, $(\mathbb{D}_2, M_2)$ of a record D are called *independent* if either their binding labels are different, or $(\mathbb{D}_1 \downarrow l, M_1)$ and $(\mathbb{D}_2 \downarrow l, M_2)$ are independent, where l is the binding label of the two $\mathcal{T}$-subterms.

3. Two redexes $(\mathbb{D}_1, R_1)$ and $(\mathbb{D}_2, R_2)$ of a record D are independent if they are independent as $\mathcal{T}$-subterms.

□

Definition 5.2.6 (Self-referential redex). A redex $(\mathbb{D}, l)$ is called a *self-referential redex* if l is its binding label. In this case $\mathbb{D} \downarrow l$ is called the *self-referential context* of the redex. □

Definition 5.2.7. Let $D = \mathbb{D}_1\{R_1\}$, where $(\mathbb{D}_1, R_1)$ is a redex, and assume that $D \xrightarrow{(\mathbb{D}_2, R_2)} D'$. A *set of residuals* of $(\mathbb{D}_1, R_1)$ w.r.t. the reduction $\xrightarrow{(\mathbb{D}_2, R_2)}$ denoted $(\mathbb{D}_1, R_1)/(\mathbb{D}_2, R_2)$ is defined as follows: let $\mathbb{D}_3 = \text{glb}(\mathbb{D}_1, \mathbb{D}_2)$ and assume (without loss of generality) that $D = \mathbb{D}_3\{R_1, R_2\}$ in the case when $\mathbb{D}_3 \in \mathbf{Context}^2_{\mathcal{C}}$, i.e. that R_1, R_2 fill the two holes of $\mathbb{D}_3$ in this order, and $R_2 \rightsquigarrow Q_2$ if $R_2 \in \mathbf{TermRedex}$, then

1. If $(\mathbb{D}, l)$ is a self-referential redex and $\mathbb{D}\{l\} \downarrow l = \mathbb{A}\{l\}$, then $(\mathbb{D}, l)/(\mathbb{D}, l) = \{(\mathbb{D}\{\mathbb{A}\}, l)\}$. Otherwise $(\mathbb{D}, R)/(\mathbb{D}, R) = \emptyset$.

2. If $\mathbb{D}_3 \in \mathbf{Context}^2_{\mathcal{C}}$ and $R_2 \in \mathbf{TermRedex}$, then we define $(\mathbb{D}_1, R_1)/(\mathbb{D}_2, R_2) = \{(\mathbb{D}_3\{\Box, Q_2\}, R_1)\}$. The case when the reduction reduces a term redex, and the other redex (either a term or a substitution redex) is independent from the redex being reduced.

3. If $\mathbb{D}_3 \in \mathbf{Context}^2_{\mathcal{C}}$, $R_2 = l$, and $\mathbb{D}_2 \downarrow l = V \in \mathbf{Value}_{\mathcal{T}}$, then $(\mathbb{D}_1, R_1)/(\mathbb{D}_2, R_2) = \{(\mathbb{D}_3\{\Box, V\}, R_1), (\mathbb{D}_3\{R_1, \mathbb{A}\}, R_1)\}$ if $V = \mathbb{A}\{R_1\}$ for $\mathbb{A} = \mathbb{D}_1 \downarrow l$, otherwise $(\mathbb{D}_1, R_1)/(\mathbb{D}_2, R_2) = \{(\mathbb{D}_3\{\Box, V\}, R_1)\}$. I.e. if the reduction step reduces a substitution redex, then a given redex has two residuals if it has been duplicated by the substitution, and one otherwise.

4. In this case a given redex is contained in a term redex being reduced. Note that a term redex of the form $c_1\ op\ c_2$ can not contain another redex, so the term redex can only be an application. If $\mathbb{D}_3 \in \mathbf{Context}^1_{\mathcal{C}}$ and $R_2 = \mathbb{A}\{R_1\}$, then $R_2 \in \mathbf{TermRedex}$, and:

- If $R_2 = \lambda x.\mathbb{C}^n\{R_1, x, \ldots, x\}$ @ V, then by definition $(\mathbb{D}_1, R_1)/(\mathbb{D}_2, R_2) = \{(\mathbb{D}_2\{\mathbb{C}^n\{\Box, V, \ldots, V\}\}, R_1[x := V])\}$. This case defines the residual of a redex inside the abstraction when an application is reduced.
- If $R_2 = \lambda x.\mathbb{C}^n\{x, \ldots, x\}$ @ V, $V = \mathbb{A}\{R_1\}$, then $(\mathbb{D}_1, R_1)/(\mathbb{D}_2, R_2) = \{(\mathbb{D}_2\{\mathbb{B}^1_i\}, R_1) \mid 1 \leq i \leq n\}$, where $\mathbb{B}^1_i = \mathbb{C}^n\{V, \ldots, V, \Box_i, V \ldots, V\}$, so that $\Box$ fills the i-th hole of $\mathbb{C}^n$. In this case a given redex is contained in the operand part of an application.

5. If $\mathbb{D}_3 \in \mathbf{Context}^1_{\mathcal{C}}$ and $R_1 = \mathbb{A}\{R_2\}$, then $R_1 \in \mathbf{TermRedex}$, and we define $(\mathbb{D}_1, R_1)/(\mathbb{D}_2, R_2) = \{(\mathbb{D}_1, \mathbb{A}\{Q_2\})\}$ if $R_2 \in \mathbf{TermRedex}$, and $(\mathbb{D}_1, R_1)/(\mathbb{D}_2, R_2) = \{(\mathbb{D}_1, \mathbb{A}\{V\})\}$ if $R_2 = l$ and $\mathbb{D}_2 \downarrow l = V$. In this case R_2 is contained in R_1, so R_1 has one residual.

□

Note that the definition exhausts all possible cases of the kinds and mutual positions of $(\mathbb{D}_1, R_1)$ and $(\mathbb{D}_2, R_2)$.

Lemma 5.2.8. *A set of residuals of a redex $(\mathbb{D}_1, R_1)$ w.r.t. a redex $(\mathbb{D}_2, R_2)$ defined in 5.2.7 satisfies the axiomatic definition 4.2.5.* □

Proof. We need to check, firstly, that every element of a set $(\mathbb{D}_1, R_1)/(\mathbb{D}_2, R_2)$ defined in 5.2.7 is a record redex, and, secondly, that the set it satisfies the requirements of axiomatic definition 4.2.5. This can be easily shown by case analysis. □

As for the term calculus, we generalize the notion of a set of residuals to a set of redexes (rather than a single redex) and to a sequence of reduction steps. The precise definitions and notations are given in section 4.2.

5.2.2 γ-developments in Calculus of Records

According to the framework described in section 4.5, we define γ-developments for $\mathcal{C}$. γ-developments are bounded, and enjoy the weak standardization property (definition 4.5.7). As shown in section 4.8, this is sufficient for the proofs of lift and project to go through.

In order to define a γ-development step, we first define a non-restricted γ-development step, and then restrict it to particular kinds of records with marked redex, namely, those that originate from a record with a single marked non-evaluation redex. In this case the γ-development reduction has the desired properties, s.a. boundedness and weak standardization.

We use D' to range over records in the marked version of the calculus $\mathcal{C}$. We define the reductions on marked records, erasure of the marks, set marked(D'), developments, and complete developments in the usual way (see section 4.2).

Definition 5.2.9. • Let $F_1 = \text{marked}(D'_1)$, $F_2 = \text{marked}(D'_2)$. A *non-restricted γ-development step* of D'_1 is a reduction $D'_1 \underset{n-\gamma}{\overset{(\mathbb{D},R)}{\rightarrow}} D'_2$ such that $|D'_1| \xrightarrow{(\mathbb{D},R)} |D'_2|$, $(\mathbb{D}, R) \in F_1$, and

- if R is a term redex, then $F_2 = F_1/(\mathbb{D}, R)$;
- if $R = l$ is a substitution redex, then $F_2 = F_1/(\mathbb{D}, R) - \bigcup_{(\tilde{\mathbb{D}},l)\in\tilde{F}_l} \{(\mathbb{D}\{\tilde{\mathbb{D}} \downarrow l\}, l)\}$, where $\tilde{F}_l \subset F_1$ is the set of marked self-referential redexes of D'_1 whose binding label is l;
- or $D'_1 \underset{n-\gamma}{\overset{\overline{(\mathbb{D},R)}}{\rightarrow}} D'_2$, where $|D'_1| = |D'_2|$, $(\mathbb{D}, R) \in F_1$, $F_2 = F_1 - \{(\mathbb{D}, R)\}$, and $(\mathbb{D}, R)$ is not self-referential. In this case we say that the redex $(\mathbb{D}, R)$ gets *erased*.

Let $\underset{n-\gamma}{\overset{(\mathbb{G},R)}{\Longrightarrow}}$ denote a step $\underset{n-\gamma}{\overset{(\mathbb{G},R)}{\rightarrow}}$, where $(\mathbb{G}, R)$ is an evaluation redex. If $D'_1 \underset{n-\gamma}{\overset{\overline{(\mathbb{D},R)}}{\rightarrow}}$

D'_2, then $D'_1 \overset{\overline{(\mathbb{D},R)}}{\underset{n-\gamma}{\Longrightarrow}} D'_2$, regardless of whether the redex $(\mathbb{D}, R)$ is an evaluation redex, i.e. a step erasing a redex is always considered an evaluation step. If $D'_1 \underset{n-\gamma}{\overset{(\mathbb{D},R)}{\rightarrow}} D'_2$, but not $D'_1 \overset{(\mathbb{D},R)}{\underset{n-\gamma}{\Longrightarrow}} D'_2$, then $D'_1 \overset{(\mathbb{D},R)}{\underset{n-\gamma}{\circ\!\!\rightarrow}} D'_2$.

- A domain of γ-development $\mathrm{dom}(\gamma)$ is defined as follows: $D' \in \mathrm{dom}(\gamma)$ if
  - either there exists a record D'_0 and a non-evaluation redex $(\mathbb{D}, R) \in \mathrm{marked}(D'_0)$ such that $D'_0 \underset{\cup'}{\Longrightarrow}^* D''$, where $\underset{\cup'}{\Longrightarrow}$ is either a $\Longrightarrow$ step or a $\underset{n-\gamma}{\Longrightarrow}$ step, $|D'| = |D''|$, and $\mathrm{marked}(D') \subset \mathrm{marked}(D'')$,
  - or there exists $D'' \in \mathrm{dom}(\gamma)$ such that $D'' \underset{n-\gamma}{\rightarrow} D'$.
- If $D' \in \mathrm{dom}(\gamma)$ and $D' \underset{n-\gamma}{\rightarrow} D''$, then we say that this reduction is a γ-*development step*, and write $D' \underset{\gamma}{\rightarrow} D''$ (and respectively $\underset{n-\gamma}{\Longrightarrow}$ and $\underset{n-\gamma}{\circ\!\!\rightarrow}$). We also use the notation $\overset{\mathrm{e}}{\underset{\gamma}{\Longrightarrow}}$ for a single erasing γ-development step and $\overset{\mathrm{e}}{\underset{\gamma}{\Longrightarrow}}^*$ for a sequence of such steps.

□

Note that for a term redex or for a non-self-referential redex a γ-development step $\overset{(\mathbb{D},R)}{\underset{\gamma}{\rightarrow}}$ is just a regular development step in the marked calculus of records. This fact is used in further proofs, so we formulate it as a lemma.

Lemma 5.2.10. *If $(\mathbb{D}, R)$ is not a self-referential substitution redex, then $D'_1 \overset{(\mathbb{D},R)}{\underset{\gamma}{\rightarrow}} D'_2$ implies that marked(D'_2) = marked$(D'_1)/(\mathbb{D}, R)$, i.e. $D'_1 \overset{(\mathbb{D},R)}{\underset{dev}{\rightarrow}} D'_2$.* □

Proof. By definition 5.2.9 if R is a term redex, then $\mathrm{marked}(D'_2) = \mathrm{marked}(D'_1)/(\mathbb{D}, R)$. If $R = l$ is a substitution redex and it is not self-referential, then the binding label of $(\mathbb{D}, R)$ is not l, so $\widetilde{\mathrm{marked}}(D'_1)_l = \emptyset$, and therefore $\mathrm{marked}(D'_2) = \mathrm{marked}(D'_1)/(\mathbb{D}, R)$.

□

The difference between a marked reduction step and a γ-development step is as follows: if $(\mathbb{D}, l)$ is a self-referential redex substituted for a marked label l, then, even though by the extended calculus reduction a redex duplication takes place, one of the residuals is not included in the set of residuals by the $\underset{\gamma}{\rightarrow}$ step. For instance:

$$[l \mapsto \lambda x.\underline{l}, l' \mapsto \underline{l}] \underset{\gamma}{\Longrightarrow} [l \mapsto \lambda x.\underline{l}, l' \mapsto \lambda x.l].$$

The result of the substitution of $\lambda x.\underline{l}$ for the marked label in the second component is $\lambda x.l$ (i.e. the label is not marked). The self-referential redex has a single residual – itself. In contrast, the extended calculus reduction preserves the marking of the label:

$$[l \mapsto \lambda x.\underline{l}, l' \mapsto \underline{l}] \Longrightarrow [l \mapsto \lambda x.\underline{l}, l' \mapsto \lambda x.\underline{l}].$$

In this case the self-referential redex has two residuals.

Lemma 5.2.11. *A reduction $D'_1 \underset{\gamma}{\rightarrow} D'_2$ defined in 5.2.9 satisfies the requirements of the axiomatic definition 4.5.1.* □

Proof. By definition 5.2.9 for all non-evaluation redexes $(\mathbb{D}, R)$ if $\text{marked}(D'_1) = \{(\mathbb{D}, R)\})$, then $D'_1 \in dom(\gamma)$. This satisfies part 1 of definition 4.5.1.

Part 2 of definition 4.5.1 is satisfied by definition of erasing γ-development step $D'_1 \underset{\gamma}{\overset{\overline{(\mathbb{D},R)}}{\Longrightarrow}} D'_2$.

Part 3 holds due to the definition of $dom(\gamma)$ in 5.2.9.

Part 4 of definition 4.5.1 consists of 4 subparts:

1. If $D'_1 \underset{\gamma}{\overset{(\mathbb{D},R)}{\rightarrow}} D'_2$, then $|D'_1| \xrightarrow{(\mathbb{D},R)} |D'_2|$: explicitly stated in definition 5.2.9.
2. If $D'_1 \underset{\gamma}{\overset{(\mathbb{D},R)}{\rightarrow}} D'_2$, then $(\mathbb{D}, R) \in \text{marked}(D'_1)$: also explicitly stated in definition 5.2.9.

3. If $D'_1 \xrightarrow{(\mathbb{D},R)} D'_3$, then $\mathrm{marked}(D'_2) \subseteq \mathrm{marked}(D'_3)$: the set $\mathrm{marked}(D'_2)$ is $\mathrm{marked}(D'_1)/(\mathbb{D},R)$ for a term redex and $\mathrm{marked}(D'_1)/(\mathbb{D},R)$ with (possibly) some redexes excluded for a substitution redex.

4. $(\mathbb{D},R)/(\mathbb{D},R) \cap \mathrm{marked}(D'_2) = \emptyset$: if $(\mathbb{D},R)$ is not a self-referential redex, then $(\mathbb{D},R)/(\mathbb{D},R) = \emptyset$. If $(\mathbb{D},l)$ is a self-referential redex, then the reduction is

$$[l \mapsto \lambda x.\mathbb{C}\{\underline{l}\},\dots] \underset{\gamma}{\circ\!\!\longrightarrow} [l \mapsto \lambda x.\mathbb{C}\{\lambda x.\mathbb{C}\{l\}\},\dots],$$

so $(\mathbb{D},l)/(\mathbb{D},l)$ is not included into the set of marked redexes in the resulting record.

□

Note that if $D'_1 \in \mathrm{dom}(\gamma)$ and $D'_1 \underset{n-\gamma}{\overset{(\mathbb{D},R)}{\rightarrow}} D'_2$, then $D'_1 \underset{\gamma}{\overset{(\mathbb{D},R)}{\rightarrow}} D'_2$, and therefore if $D'_0 \underset{\cup'}{\Rightarrow}{}^* D'_2$ and $\mathrm{marked}(D'_0) = \{(\mathbb{D},R)\}$, then $D'_0 \underset{\cup}{\Rightarrow}{}^* D'_2$ (see 4.5.4 for definition of $\underset{\cup}{\Rightarrow}$). In the rest of this work we use the notation $\underset{\cup}{\Rightarrow}{}^*$ in this situation.

It easily follows from definition 5.2.9 that $D' \in \mathrm{dom}(\gamma)$ if and only if there exist a record D'_0 such that $\mathrm{marked}(D'_0) = \{(\mathbb{D},R)\}$ and $D'_0 \underset{\cup}{\Rightarrow}{}^* D'_1 \underset{\gamma}{\rightarrow}{}^* D'$.

Definition 5.2.12. If $\mathrm{marked}(D'_0) = \{(\mathbb{D},R)\}$ and $D'_0 \underset{\cup}{\Rightarrow}{}^* D'_1 \underset{\gamma}{\rightarrow}{}^* D'$, then $(\mathbb{D},R)$ is called a *starting redex* of D'. □

A starting redex of D' is not uniquely defined, but this ambiguity does not affect our proofs.

Let us consider three kinds of record redexes: a term redex, a self-referential substitution redex (see definition 5.2.6), and a substitution redex which is not self-referential.

Lemma 5.2.13. *Let $(\mathbb{D},R)$ be a starting redex of D' and let $F = marked(D')$.*

- *If* $(\mathbb{D}, R)$ *is a term redex, then every* $(\mathbb{D}', R') \in F$ *is a term redex,*
- *If* $(\mathbb{D}, R) = (\mathbb{D}, l)$ *is a non-self-referential substitution redex, then every redex in* F *is a non-self-referential substitution redex of the form* $(\mathbb{D}', l)$*.*
- *If* $(\mathbb{D}, R) = (\mathbb{D}, l)$ *is a self-referential substitution redex, then there is no more than one self-referential redex* $(\tilde{\mathbb{D}}, l) \in F$*, and all other redexes in* F *(if any) are of the form* $(\mathbb{D}', l)$*.*

□

Proof. Firstly we note that a step $\xrightarrow[\gamma]{\overline{(\mathbb{D},R)}}$ does not produce new marked redexes, only removes already existing ones. Therefore we do not need to consider this step in the rest of the proof.

The first claim of the lemma easily follows from the observation that if $(\mathbb{D}, R)$ is a term redex, then for every redex $(\mathbb{D}_1, R_1)$ if $(\mathbb{D}', R') \in (\mathbb{D}, R)/(\mathbb{D}_1, R_1)$, then $(\mathbb{D}', R')$ is a term redex (see definition 5.2.7). Therefore all residuals of a term redex w.r.t. any reduction sequence are term redexes.

Similarly, it is clear that if a redex $(\mathbb{D}, l)$ is a substitution redex, and $(\mathbb{D}', R') \in (\mathbb{D}, l)/(\mathbb{D}_1, R_1)$, then $R' = l$, i.e. all residuals of a substitution redex with a label l are substitution redexes with the same label l. However, this is not enough to prove the second and the third claims of the lemma, since, in general, a residual of a self-referential redex may be not self-referential, and vice versa.

Let $D'_0 \underset{\cup}{\Rightarrow}^* D'_1 \underset{\gamma}{\rightarrow}^* D'_2$, where $\mathrm{marked}(D'_0) = \{(\mathbb{D}, l)\}$. Let $F_1 = \mathrm{marked}(D'_1)$ and $F_2 = \mathrm{marked}(D'_2)$. Suppose that $(\mathbb{D}, l)$ is a non-self-referential substitution redex. We show that F_2 has only non-self-referential redexes by induction on the total number of steps in the above sequence. The base case of 0 steps satisfies the claim, since the pair D'_0 has only non-self-referential marked substitution redexes.

Part 1. Suppose $D'_0 \underset{\cup}{\Longrightarrow}^* D'_1 \xrightarrow[\cup]{(\mathbb{G},R)} D'_2$, and F_1 (in the notations above) has only non-self-referential redexes. We want to show that F_2 also has only such redexes. Note that $D_1 \downarrow l = V \in \mathbf{Value}_{\mathcal{T}}$ by definition of a substitution redex. By the inductive hypothesis there is no $(\mathbb{D}_1, l) \in F_1$ such that the binding label of $(\mathbb{D}_1, l)$ is l, in other words V does not contain marked occurrences of l. By class preservation property of $\mathcal{T}$ (lemma 5.1.21) $V \neq \mathbb{E}\{R\}$ for any term redex R and $V \neq \mathbb{E}\{l'\}$ for any l'. Therefore the evaluation step occurs in a component of D_1 other than the one bound to l. Let $\tilde{l}$ denote the label of this component.

There are several possibilities for a reduction step $\xrightarrow[\cup]{(\mathbb{G},R)}$: it may be a $\Longrightarrow$, which either reduces an unmarked redex (which may be a term or a substitution redex), or a marked redex (i.e. a redex $(\mathbb{D}', l) \in F_1$). Alternatively, $\xrightarrow[\cup]{(\mathbb{G},R)}$ may be a $\underset{\gamma}{\Longrightarrow}$ step. However, in all these cases we observe the following: by considering the cases in definition of a residual (definition 5.2.7) we can see that the only way a self-referential marked redex may appear in the component bound to l in D'_2 is if a marked occurrence of l is copied to this component by a substitution. However, since the reduction step happens in a different component of D_1 (a component with a label $\tilde{l}$), a self-referential redex may not be created.

Part 2. Similarly, suppose that $D'_0 \underset{\cup}{\Longrightarrow}^* D' \underset{\gamma}{\longrightarrow}^* D'_1 \overset{(\mathbb{D}_1,l)}{\underset{\gamma}{\longrightarrow}} D'_2$, where F_1 contains only non-self-referential redexes (note that the step $\overset{(\mathbb{D}_1,l)}{\underset{\gamma}{\longrightarrow}}$ is a part of a development sequence, therefore it reduces a substitution redex with a label l). By the inductive hypothesis F_1 does not contain self-referential redexes, so the binding label of $(\mathbb{D}_1, l)$ is not l. Therefore reducing this redex can not create a marked occurrence of l in the component bound to l, so no self-referential redexes occur in F_2. This concludes the proof of the second claim of the lemma.

Now suppose $D'_0 \underset{\cup}{\Longrightarrow}^* D'_1, \underset{\gamma}{\longrightarrow}^* D'_2$, $(\mathbb{D}, l)$ is a self-referential redex, i.e. $D_0 \downarrow$

$l = \mathbb{C}\{l\}$, where $\mathbb{C}$ is a non-evaluation context, and $\mathbb{D} \downarrow l = \mathbb{C}$. We prove this claim by induction, similarly to the previous claim. Base case: D'_0 satisfies the claim, since $\{(\mathbb{D}, l)\}$ contains just one self-referential redex.

Part 1. Suppose that $D'_0 \underset{\cup}{\Longrightarrow}^* D'_1, \underset{\cup}{\xRightarrow{(\mathbb{G},R)}} D'_2$, and F_1 contains no more than one self-referential redex. We want to show that F_2 contains no more than one self-referential redex. If F_1 does not contain a self-referential redex, then by the same argument as in Part 1 of the proof of the second claim we show that F_2 does not have a self-referential redex. Now suppose that F_1 has a self-referential redex. Since the component bound to l is a value, the evaluation step occurs in a different component (let $\tilde{l}$ denote its label). There are three cases of evaluation step $\Longrightarrow$ reducing an unmarked redex. We show marked redexes by underlining. Note that $\mathbb{C}$ is a non-evaluation context.

Term redex:

$$[l \mapsto \mathbb{C}\{\underline{l}\}, \tilde{l} \mapsto \mathbb{E}\{R\}, \dots] \quad \Longrightarrow \quad [l \mapsto \mathbb{C}\{\underline{l}\}, \tilde{l} \mapsto \mathbb{E}\{Q\}, \dots],$$

Substitution redexes:

$$[l \mapsto \mathbb{C}\{\underline{l}\}, \tilde{l} \mapsto \mathbb{E}\{l_1\}, l_1 \mapsto V', \dots] \quad \Longrightarrow \quad [l \mapsto \mathbb{C}\{\underline{l}\}, \tilde{l} \mapsto \mathbb{E}\{V'\}, l_1 \mapsto V', \dots],$$

$$[l \mapsto \mathbb{C}\{\underline{l}\}, \tilde{l} \mapsto \mathbb{E}\{l\}, \dots] \quad \Longrightarrow \quad [l \mapsto \mathbb{C}\{\underline{l}\}, \tilde{l} \mapsto \mathbb{E}\{\mathbb{C}\{\underline{l}\}\}, \dots].$$

No new self-referential redexes has been created as the result of the evaluation step. Note that in the last case, even though the self-referential redex has been duplicated, the second copy is not self-referential, since it does not occur in the component bound to l.

If the $\underset{\cup}{\xRightarrow{(\mathbb{G},R)}}$ step is a $\Longrightarrow$ step reducing a marked redex, we have the only possibility (note that a marked redex must be a substitution redex with the label l,

and may not occur in a component bound to l since $D_1 \downarrow l \in \mathbf{Value}_{\mathcal{T}}$):

$$[l \mapsto \mathbb{C}\{\underline{l}\}, \tilde{l} \mapsto \mathbb{E}\{\underline{l}\}, \ldots] \implies [l \mapsto \mathbb{C}\{\underline{l}\}, \tilde{l} \mapsto \mathbb{E}\{\mathbb{C}\{\underline{l}\}\}].$$

Again, no other self-referential redexes has been created in the component bound to l.

The case when the $\xrightarrow[\cup]{(\mathbb{G},R)}$ step is a $\underset{\gamma}{\Longrightarrow}$ step is very similar to the previous one, with the difference that the occurrence of l in the component bound to $\tilde{l}$ is not marked in the resulting record:

$$[l \mapsto \mathbb{C}\{\underline{l}\}, \tilde{l} \mapsto \mathbb{E}\{\underline{l}\}, \ldots] \underset{\gamma}{\Longrightarrow} [l \mapsto \mathbb{C}\{\underline{l}\}, \tilde{l} \mapsto \mathbb{E}\{\mathbb{C}\{l\}\}].$$

The difference, however, does not affect the fact that the resulting record still has just one self-referential redex.

Part 2. Suppose that $D'_0 \underset{\cup}{\Longrightarrow}{}^{*} D' \underset{\gamma}{\longrightarrow}{}^{*} D'_1 \overset{(\mathbb{D}_1,l)}{\underset{\gamma}{\longrightarrow}} D'_2$, where F_1 contains no more than one self-referential redex (recall that $F_1 = \text{marked}(D'_1)$, $F_2 = \text{marked}(D'_2)$). If F_1 does not contain any such redexes, then by the same argument as in part 2 of the proof of the second claim we show that F_2 does not contain self-referential redexes as well. Suppose F_1 has one self-referential redex. We have one of the following two cases:

$$\begin{array}{lll} \text{Case 1:} & [l \mapsto \mathbb{C}\{\underline{l}\}, \tilde{l} \mapsto \mathbb{A}\{\underline{l}\}, \ldots] & \underset{\gamma}{\rightarrow} \quad [l \mapsto \mathbb{C}\{\underline{l}\}, \tilde{l} \mapsto \mathbb{A}\{\mathbb{C}\{l\}\}, \ldots], \\ \text{Case 2:} & [l \mapsto \mathbb{C}\{\underline{l}\}, \ldots] & \underset{\gamma}{\circ\!\!\rightarrow} \quad [l \mapsto \mathbb{C}\{\mathbb{C}\{l\}\}, \ldots]. \end{array}$$

In the first case a marked occurrence of l is reduced in a component other than the one bound to l. Then the original self-referential redex is still marked, and therefore F_2 still has one self-referential redex. Note that in the first case $\mathbb{A}$ may or

may not be an evaluation context, so the reduction may or may not be an evaluation step. In the second case the self-referential redex is reduced (the reduction step must be a non-evaluation step, since l occurs in a non-evaluation context). Since the step is a γ-development step, l is not marked after the substitution. By the inductive hypothesis D_1' has just one self-referential redex, so there are no other labelled occurrences of l in $\mathbb{C}\{\underline{l}\}$. Therefore there are no self-referential redexes in F_2. Combining the two cases, we conclude that F_2 has one or no self-referential redexes. This proves the third claim of the lemma. □

Lemma 5.2.10 suggests that in the case of a term redex or a non-self-referential substitution redex the proofs of properties implying lift and project may be simplified, since γ-developments of these redexes are just extended calculus reductions. Lemma 5.2.13 guarantees that these two cases are indeed separate from the case of a self-referential redex (and from each other), i.e. by starting from a single non-evaluation redex, which is either a term redex, or a non-self-referential substitution redex, in a lift or project diagram one can get only developments of redexes of the same kind. This allows us to consider the first two (simpler) cases of redexes separately from the third (i.e. self-referential redex) in the proofs below.

5.2.3 Boundedness of γ-developments

In order to prove lift and project (4.8.1 and 4.8.2 respectively) for the record calculus $\mathcal{C}$, we need to show properties required by theorems 4.8.4 and 4.8.5. We also prove boundedness of γ-developments and the diamond property of the evaluation relation $\Longrightarrow$, which implies its confluence. Recall that confluence of $\Longrightarrow$ is used in the proof of computational soundness from lift and project (see theorem 3.4.5).

Lemma 5.2.14 (Boundedness of γ-developments). *Let $(\mathbb{D}, R)$ be a starting redex of D'. There exists a development $D' \underset{\gamma}{\longrightarrow}^{*} D''$ of the maximal length.* □

Proof. Let $F = \text{marked}(D')$. We have two cases:

Case 1. $(\mathbb{D}, R)$ is a term redex. By lemma 5.2.13 all redexes in F are term redexes.

Let $D' = [l_i \overset{m}{\underset{i=1}{\mapsto}} M_i]$, and suppose that for every record component M_i we have defined a weighting I_i according to definition 5.1.4. Then we define a *weighting* of the record to be a tuple $I = (I_1, \ldots, I_m)$. We extend a reduction of marked records to a reduction of pairs $D', I)$ analogously to definition 5.1.8.

We say that a weighting $I = (I_1, \ldots, I_m)$ of a record D' is *decreasing* if $I_1, \ldots, I_m$ are decreasing on $D' \downarrow l_1, \ldots D' \downarrow l_m$ respectively. Let $|| \ (D, I) \ || = \sum_{i=1}^{m} || \ (D \downarrow l_i, I_i) \ ||$.

Suppose $(D'_1, I_1) \overset{(\tilde{\mathbb{D}}, \tilde{R})}{\underset{dev}{\rightarrow}} (D'_2, I_2)$, and I_1 is decreasing. We want to show that:

1. I_2 is decreasing on D'_2,
2. $|| \ (D'_1, I_1) \ || > || \ (D'_2, I'_2) \ ||$.

Let l_k be the binding label of $(\tilde{\mathbb{D}}, \tilde{R})$, and let $(\tilde{\mathbb{C}}, \tilde{R}) = (\tilde{\mathbb{D}}, \tilde{R}) \downarrow l_k$. Then

$$D'_1 \downarrow l_k \overset{(\tilde{\mathbb{C}}, \tilde{R})}{\underset{dev}{\rightarrow}} D'_2 \downarrow l_k.$$

Taking into account the weighting of D'_1, we can consider the term reduction on pairs (recall definition 5.1.8):

$$(D'_1 \downarrow l_k, I_{1k}) \overset{(\tilde{\mathbb{C}}, \tilde{R})}{\underset{dev}{\rightarrow}} (D'_2 \downarrow l_k, I_{2k}),$$

where I_{1k} is the k-th component of the record weighting for I_1, and I_{2k} is the k-th

component of the record weighting for I_2.

Note that I_{1k} is decreasing on $D \downarrow l_k$ by definition of a decreasing record weighting. Then by lemma 5.1.9 I_{2k} is decreasing and $||\ (D'_1 \downarrow l_k, I_k)\ ||>||\ (D'_2 \downarrow l_k, I'_k)\ ||$. Since the other components of the record are not changed by the reduction, the new record weighting $I_2 = (I_{11}, \ldots, I_{1k-1}, I_{2k}, I_{1k+1}, \ldots, I_{1m})$ is decreasing, and $||\ (D'_1, I_1)\ ||>||\ (D'_2, I_2)\ ||$.

We also note the following important facts:

1. $||\ (D, I)\ ||> 0$ for any D, I (since the measure of every record component is greater than 0),

2. for every D' there exists a decreasing weighting I (such a weighting can be obtained as a combination of decreasing weightings of all components, which exist by lemma 5.1.7)

Combining all the properties of decreasing record weightings that we have shown, we can apply the same argument as in lemma 5.1.10 to show that all record developments are bounded when $(\mathbb{D}, R)$ is a term redex.

Note that the erasing γ-development step $\xRightarrow[\gamma]{\overline{(\mathbb{D},R)}}$ also reduces the measure, since it "erases" the marking of a redex.

Case 2. $(\mathbb{D}, R) = (\mathbb{D}, l)$ *is a substitution redex.* By lemma 5.2.13 all redexes in F are substitution redexes with the label l, and at most one of them is a self-referential redex.

We define a measure[2] $||\ D'\ ||$ to be the number of elements in marked(D'). Note that since every $(\mathbb{D}, R)$ is a record redex in D', the measure $||\ D'\ ||$ is finite for every D' (every record has a finite number of redexes). Clearly $||\ D'\ ||\geq 0$ for any D'.

[2]This measure is independent from the measure defined in case 1 of the proof.

By definition 5.2.7 of a residual and 5.2.9 of γ-development every substitution step reduces the number of marked redexes by one. Similarly to the case of the term redex, an erasing step $\xrightarrow[\gamma]{\overline{(\mathbb{D},l)}}$ also reduces the number of marked redexes, and therefore the measure, by one. Therefore the γ-developments are bounded in $\mathcal{C}$. □

5.2.4 Confluence and γ-confluence of $\Longrightarrow_{\mathcal{C}}$

Lemma 5.2.15. *The evaluation relation of the calculus $\mathcal{C}$ satisfies the following property: if $D_1 \xrightarrow{(\mathbb{G}_1,R_1)} D_2$ and $D_1 \xrightarrow{(\mathbb{G}_2,R_2)} D_3$, then there exists D_4 such that $D_2 \xrightarrow{(\mathbb{G}_2,R_2)/(\mathbb{G}_1,R_1)} D_4$ and $D_3 \xrightarrow{(\mathbb{G}_1,R_1)/(\mathbb{G}_2,R_2)} D_4$, provided $(\mathbb{G}_1,R_1) \neq (\mathbb{G}_2,R_2)$.* □

Proof. The proof is by cases on pairs of redexes $(\mathbb{G}_1,R_1)$ and $(\mathbb{G}_2,R_2)$. Note that the two redexes occur in two different components of D_1, since they are both evaluation redexes, and therefore are independent. □

Note that, as the lemma suggests, each of the sets $(\mathbb{G}_1,R_1)/(\mathbb{G}_2,R_2)$ and $(\mathbb{G}_2,R_2)/(\mathbb{G}_1,R_1)$ consists of just a single redex.

Lemma 5.2.15 implies two important properties of $\Longrightarrow_{\mathcal{C}}$ stated in lemma 5.2.16 and in lemma 5.2.17 below.

Lemma 5.2.16 (Confluence of $\Longrightarrow_{\mathcal{C}}$). *The evaluation relation $\Longrightarrow_{\mathcal{C}}$ of the calculus $\mathcal{C}$ is confluent.* □

Proof. By lemma 5.2.15. □

Lemma 5.2.17. *If $D \Longrightarrow_{\mathcal{C}}^{*} D' = Eval(D)$, then there is no infinite sequence $D \Longrightarrow_{\mathcal{C}} D_1 \Longrightarrow_{\mathcal{C}} D_2 \ldots$.* □

Proof. By lemma 5.2.15 if $D \overset{S_1}{\Longrightarrow}_{\mathcal{C}}^{*} D'$ and $D \overset{S_2}{\Longrightarrow}_{\mathcal{C}}^{*} D'$, then the two sequences S_1 and S_2 have the same number of steps. By confluence (lemma 5.2.16) for every D_i

such that $D \Longrightarrow_{\mathcal{C}}^{*} D_i$ there exists a sequence $D_i \Longrightarrow_{\mathcal{C}}^{*} D'$, therefore there is no infinite evaluation sequence starting from D. □

The following results deal with the marked version of $\mathcal{C}$.

Lemma 5.2.18. *The evaluation relation of the calculus $\mathcal{C}$ has the following property: if marked$(D'_1) = \{(\mathbb{G}, R)\}$, $D'_1 \xrightarrow[\gamma]{(\mathbb{G},R)} D'_2$, $D'_1 \xrightarrow{(\mathbb{G}',R')} D'_3$, and $(\mathbb{G}, R) \neq (\mathbb{G}', R')$, then marked$(D'_2) = \emptyset$ and there exists D'_4 such that $D'_2 \xrightarrow{(\mathbb{G}',R')/(\mathbb{G},R)} D'_4$, $D'_3 \xrightarrow[\gamma]{(\mathbb{G},R)/(\mathbb{G}',R')} D'_4$, and marked$(D'_4) = \emptyset$.* □

Proof. The two given reductions imply that $|D'_1| \xrightarrow{(\mathbb{G},R)} |D'_2|$ by definition 4.5.1, and $D_1 \xrightarrow{(\mathbb{G}',R')} D_3$ by definition of a marked reduction. marked$(D'_2) = \emptyset$ by definition of γ-development. By lemma 5.2.15 there exists D_4 such that $|D'_2| \xrightarrow{(\mathbb{G}',R')/(\mathbb{G},R)} D_4$ and $|D'_3| \xrightarrow{(\mathbb{G},R)/(\mathbb{G}',R')} D_4$. Note that $F = (\mathbb{G}, R)/(\mathbb{G}', R')$ contains just one redex (by 5.2.15).

The redex $(\mathbb{G}, R)$ can not be a self-referential redex, since the step $D'_1 \xrightarrow[\gamma]{(\mathbb{G},R)} D'_2$ is an evaluation step, but a self-referential redex is a non-evaluation redex. Therefore (lemma 5.2.13) the redex $(\mathbb{G}'', R'') = (\mathbb{G}, R)/(\mathbb{G}', R')$ also can not be self-referential. Since no self-referential redexes are marked, by definition of a marked calculus reduction the reduction step $D_3 \xrightarrow{(\mathbb{G}'',R'')} D_4$ implies that $D'_3, \xrightarrow{(\mathbb{G}'',R'')} D'_4$, and marked$(D'_3) = \{(\mathbb{G}'', R'')\}$, marked$(D'_4) = \emptyset$. This, and the fact that by definition 5.2.9 $D'_3 \in \mathrm{dom}(\gamma)$, imply by lemma 5.2.10 that the step $D'_3 \xrightarrow{(\mathbb{G}'',R'')} D'_4$, is a $\underset{\gamma}{\Longrightarrow}$ step. □

The condition $(\mathbb{G}, R) \neq (\mathbb{G}', R')$ is necessary. Otherwise the two resulting records are the same:

Lemma 5.2.19. *If $D'_1 \xrightarrow[\gamma]{(\mathbb{G},R)} D'_2$, marked$(D'_1) = \{(\mathbb{G}, R)\}$, and $D'_1 \xrightarrow{(\mathbb{G},R)} D'_3$, then $D'_2 = D'_3$ and marked$(D'_2) = \emptyset$.* □

Proof. The only marked redex in this case is not a self-referential one. If $(\mathbb{D}, R)$ is not self-referential, then $(\mathbb{D}, R)/(\mathbb{D}, R) = \emptyset$. □

Lemmas 5.2.18 and 5.2.19 prove γ-confluence of evaluation for a single marked redex. Our goal is to generalize this proof arbitrary marked records $D'_1 \in dom(\gamma)$, and also to arbitrary γ-development steps (i.e. not only those reducing a redex, but also erasing steps). Below we consider two cases. If the set $F_1 = \text{marked}(D'_1)$ does not contain a self-referential redex, it is enough to show the property for one marked redex other than the redex $(\mathbb{G}, R)$ reduced in the given γ-development step. Since in this case a γ-development step is the same as the corresponding extended reduction step, we can use lemma **??** to combine the results for each marked redex in F_1. The second case is when F_1 contains a self-referential redex. Since in this case an γ-development step is different from an extended reduction step and its result depends on the marking of the self-referential redex, we need to consider two marked redexes (the evaluation redex and the self-referential one). We also need to take into account erasing steps $\overset{e}{\Longrightarrow}$. The proofs are given in the following lemmas.

Lemma 5.2.20. *Let $F_1 = marked(D'_1) = \{(\mathbb{G}, R), (\mathbb{D}, \tilde{R})\}$, where $(\mathbb{D}, \tilde{R})$ is not a self-referential redex, and suppose $D'_1 \xrightarrow[\gamma]{(\mathbb{G},R)} D'_2$ $D'_1 \xrightarrow{(\mathbb{G}',R')} D'_3$, where $(\mathbb{G}, R) \neq (\mathbb{G}', R')$. Then there exists D'_4 such that $D'_2 \xrightarrow{(\mathbb{G}',R')/(\mathbb{G},R)} D'_4$ and $D'_3 \xrightarrow[\gamma]{(\mathbb{G},R)/(\mathbb{G}',R')} D'_4$.* □

Proof. By lemma 5.2.13 the redexes $(\mathbb{G}, R)$ and $(\mathbb{D}, \tilde{R})$ are of the same kind. Note also that $(\mathbb{G}, R)$ and $(\mathbb{G}', R')$ can not be in the same record component, since they are both evaluation redexes. Let l, l' denote the labels of the components where $(\mathbb{G}, R)$ and $(\mathbb{G}', R')$ appear, respectively. We also use notation $\tilde{l}$ for the label where $(\mathbb{D}, \tilde{R})$ appears. It may be the case that $\tilde{l}$ is the same as l or l'.

The proof is by cases of the kinds of the three redexes. Note that $(\mathbb{G}, R)$ and $(\mathbb{D}, \tilde{R})$ are of the same kind by lemma 5.2.13.

- $(\mathbb{G}, R)$, $(\mathbb{D}, \tilde{R})$, *and* $(\mathbb{G}', R')$ *are all term redexes.* If $\tilde{l} \neq l$, $\tilde{l} \neq l'$, then all three redexes occur in different components of the record, so the claim obviously holds. Suppose $\tilde{l} = l'$, i.e. R' and $\tilde{R}$ occur in the same component. If these redexes are independent (i.e. one does not contain the other), then the claim of the lemma clearly holds. It may not be the case that $\tilde{R}$ contains R', since $(\mathbb{G}', R')$ is an evaluation redex, and therefore cannot be contained in another redex. If R' contains $\tilde{R}$, then $R' = (\lambda x.\mathbb{B}\{\tilde{R}\})\ @\ V$ or $R' = (\lambda x.M)\ @\ \lambda y.\mathbb{B}\{\tilde{R}\}$. In the first case, assuming $\mathbb{B}' = \mathbb{B}[x := V]$ and $\tilde{R}' = \tilde{R}'[x := V]$, we have:

$$
\begin{array}{ll}
[l \mapsto \mathbb{G}\{R\}, l' \mapsto \mathbb{G}'\{(\lambda x.\mathbb{B}\{\tilde{R}\})\ @\ V\}, \ldots] & \underset{\gamma}{\Longrightarrow} \\
[l \mapsto \mathbb{G}\{Q\}, l' \mapsto \mathbb{G}'\{(\lambda x.\mathbb{B}\{\tilde{R}\})\ @\ V\}, \ldots] & \Longrightarrow \\
[l \mapsto \mathbb{G}\{Q\}, l' \mapsto \mathbb{G}'\{\mathbb{B}'\{\tilde{R}'\}\}, \ldots], & \\
[l \mapsto \mathbb{G}\{R\}, l' \mapsto \mathbb{G}'\{(\lambda x.\mathbb{B}\{\tilde{R}\})\ @\ V\}, \ldots] & \Longrightarrow \\
[l \mapsto \mathbb{G}\{R\}, l' \mapsto \mathbb{G}'\{\mathbb{B}'\{\tilde{R}'\}\}, \ldots] & \underset{\gamma}{\Longrightarrow} \\
[l \mapsto \mathbb{G}\{Q\}, l' \mapsto \mathbb{G}'\{\mathbb{B}'\{\tilde{R}'\}\}, \ldots]. &
\end{array}
$$

  The other case is similar, with the only difference that the redex $\tilde{R}$ gets duplicated rather than changed. The case when $\tilde{l} = l$ is similar to the case when $\tilde{l} = l'$, since the marked redexes are term redexes, and therefore γ-developments are just extended reductions.

- $(\mathbb{G}, R)$ *and* $(\mathbb{D}, \tilde{R})$ *are term redexes,* $(\mathbb{G}', R')$ *is a substitution redex.* In this case R' is an unmarked label (since F_1 contains term redexes, it can not contain a label). Let $R' = l_1$. The label l_1 is bound to a value, so $l_1 \neq l$, $l_1 \neq l'$. If $l_1 \neq \tilde{l}$, then the three redexes are independent, and the claim of the lemma clearly holds. If $l_1 = \tilde{l}$, then reducing the redex $(\mathbb{G}', l_1)$ duplicates $\tilde{R}$. However, it is clear that $(\mathbb{D}, \tilde{R})$ has the same two residuals on both reduction paths.

- $(\mathbb{G}, R)$ *and* $(\mathbb{D}, \tilde{R})$ *are substitution redexes,* $(\mathbb{G}', R')$ *is a term redex.* Then $R = \tilde{R} = l_1$. Since F_1 does not contain self-referential redexes, $l_1 \neq l$, $l_1 \neq \tilde{l}$. Since l_1 is bound to a value, $l_1 \neq l'$.

  If $\tilde{l} \neq l$, $\tilde{l} \neq l'$, then all 4 labels are distinct, and the claim of the lemma clearly holds. If $\tilde{l} = l'$, i.e. of l_1 of the marked redex $(\mathbb{D}, l_1)$ occurs in the same component as R', and the record can be of one of the following forms (as before, we show marked labels by underlining):

$$
\begin{array}{ll}
[l \mapsto \mathbb{E}\{\underline{l_1}\}, l' \mapsto \mathbb{A}\{R', \underline{l_1}\}, l_1 \mapsto V, \ldots], & \\
[l \mapsto \mathbb{E}\{\underline{l_1}\}, l' \mapsto \mathbb{E}'\{(\lambda x.\mathbb{B}\{\underline{l_1}\}) \mathbin{@} V\}, l_1 \mapsto V, \ldots], & \text{or} \\
[l \mapsto \mathbb{E}\{\underline{l_1}\}, l' \mapsto \mathbb{E}'\{(\lambda x.M) \mathbin{@} \lambda y.\mathbb{B}\{\underline{l_1}\}\}, l_1 \mapsto V, \ldots], &
\end{array}
$$

  where in the first case $\mathbb{A}\{\Box, \underline{l_1}\}$ is an evaluation context[3]. It is easy to see that the claim holds in all three cases (in the last case the redex $(\mathbb{D}, l_1)$ gets duplicated, but its residuals are the same on both reduction paths). The remaining case is when $\tilde{l} = l$, then the record is of the form

$$[l \mapsto \mathbb{A}\{\underline{l_1}, \underline{l_1}\}, l' \mapsto \mathbb{E}\{R'\}, l_1 \mapsto V, \ldots].$$

  Again, clearly both reduction paths lead to the same record with the same residual of $(\mathbb{D}, l_1)$.

- $(\mathbb{G}, R)$, $(\mathbb{D}, \tilde{R})$, *and* $(\mathbb{G}', R')$ *are all substitution redexes.* Let $R = l_1$ (hence $\tilde{R} = l_1$) and $R' = l_2$. There are 3 possibilities:

  1. $l_1 \neq l_2$,

[3]Here and below we specify only one order in which two (or more) subterms occur in a context if all the cases are analogous. For instance, here we do not consider the case when the evaluation context is $\mathbb{A}\{\underline{l_1}, R'\}$, because it is completely analogous to the case we have considered

2. $l_1 = l_2$, $(\mathbb{G}', l_1) \notin F_1$, i.e. it is not marked,

3. $l_1 = l_2$, $(\mathbb{G}', l_1) \in F_1$, i.e. it is marked.

Let us consider all three possibilities:

1. If $l_1 \neq l_2$, then the 4 labels l, l', l_1, l_2 are all pairwise distinct (in addition to the given conditions, we notice that l_1, l_2 are bound to values, and l, l' contain evaluation redexes, so none of the first two labels equals to any of the other two). We have the following 4 possibilities for $\tilde{l}$:

   If $\tilde{l} = l$, then the record is of the form

   $$[l_1 \mapsto V_1, l_2 \mapsto V_2, l \mapsto \mathbb{A}\{\underline{l_1}, \underline{l_1}\}, l' \mapsto \mathbb{E}\{l_2\}, \dots],$$

   where, as above, $\mathbb{A}\{\Box, \underline{l_1}\}$ is an evaluation context One can check that reducing $(\mathbb{G}, l_1)$ and $(\mathbb{G}', l_2)$ in any order leads to a record with the same residual of $(\mathbb{D}, l_1)$.

   If $\tilde{l} = l'$, then the case is similar to the case when $\tilde{l} = l$.

   If $\tilde{l} = l_2$, then the marked redex $(\mathbb{D}, l_1)$ gets duplicated:

   $$\begin{aligned}
   &[l_1 \mapsto V_1, l_2 \mapsto \lambda x.\mathbb{C}\{\underline{l_1}\}, l \mapsto \mathbb{E}\{\underline{l_1}\}, l' \mapsto \mathbb{E}'\{l_2\}, \dots] && \underset{\gamma}{\Longrightarrow} \\
   &[l_1 \mapsto V_1, l_2 \mapsto \lambda x.\mathbb{C}\{\underline{l_1}\}, l \mapsto \mathbb{E}\{V_1\}, l' \mapsto \mathbb{E}'\{l_2\}, \dots] && \Longrightarrow \\
   &[l_1 \mapsto V_1, l_2 \mapsto \lambda x.\mathbb{C}\{\underline{l_1}\}, l \mapsto \mathbb{E}\{V_1\}, l' \mapsto \mathbb{E}'\{\lambda x.\mathbb{C}\{\underline{l_1}\}\}, \dots], && \\
   &[l_1 \mapsto V_1, l_2 \mapsto \lambda x.\mathbb{C}\{\underline{l_1}\}, l \mapsto \mathbb{E}\{\underline{l_1}\}, l' \mapsto \mathbb{E}'\{l_2\}, \dots] && \Longrightarrow \\
   &[l_1 \mapsto V_1, l_2 \mapsto \lambda x.\mathbb{C}\{\underline{l_1}\}, l \mapsto \mathbb{E}\{\underline{l_1}\}, l' \mapsto \mathbb{E}'\{\lambda x.\mathbb{C}\{\underline{l_1}\}\}, \dots] && \underset{\gamma}{\Longrightarrow} \\
   &[l_1 \mapsto V_1, l_2 \mapsto \lambda x.\mathbb{C}\{\underline{l_1}\}, l \mapsto \mathbb{E}\{V_1\}, l' \mapsto \mathbb{E}'\{\lambda x.\mathbb{C}\{\underline{l_1}\}\}, \dots]. &&
   \end{aligned}$$

   The remaining possibility is that $\tilde{l} \notin \{l, l', l_2\}$. Then all five labels are

distinct, and the claim of the lemma clearly holds. Note that the case $\tilde{l} = l_1$ is impossible because $(\mathbb{D}, l_1) \in F_1$, but F_1 does not contain self-referential redexes.

2. If $l_1 = l_2$, and $(\mathbb{G}', l_1)$ is not marked, then we have one of the following choices:

$$\begin{aligned} \tilde{l} \notin \{l, l'\}: &\quad [l_1 \mapsto V_1, l \mapsto \mathbb{E}\{\underline{l_1}\}, l' \mapsto \mathbb{E}'\{l_1\}, \tilde{l} \mapsto \mathbb{C}\{\underline{l_1}\}, \ldots], \\ \tilde{l} = l: &\quad [l_1 \mapsto V_1, l \mapsto \mathbb{A}\{\underline{l_1}, \underline{l_1}\}, l' \mapsto \mathbb{E}'\{l_1\}, \ldots], \\ \tilde{l} = l': &\quad [l_1 \mapsto V_1, l \mapsto \mathbb{E}\{\underline{l_1}\}, l' \mapsto \mathbb{A}\{l_1, \underline{l_1}\}, \ldots]. \end{aligned}$$

   In all three cases $(\mathbb{D}, l_1)$ has the same residual on both reduction paths.

3. If $l_1 = l_2$, and $(\mathbb{G}', l_1)$ is marked, then the case is completely analogous to case 2. Even though the occurrence of l_1 in the component bound to l' is marked, this does not affect the non-development reduction, because this reduction does not remove the mark of a label regardless of whether the target of the substitution is marked or unmarked.

□

The remaining case is when the γ-development step erases a redex.

Lemma 5.2.21. *Suppose $F_1 = \mathit{marked}(D_1')$ does not contain any self-referential redexes. If $D_1' \xrightarrow[\gamma]{\overline{(\mathbb{D},R)}} D_2'$, $D_1' \xrightarrow{(\mathbb{G},\tilde{R})} D_3'$, then there exists D_4' such that $D_2' \xrightarrow{(\mathbb{G},\tilde{R})} D_4'$ and $D_3' \overset{e}{\underset{\gamma}{\Longrightarrow}}{}^{*} D_4'$.* □

Proof. By definition of an erasing γ-development step $(\mathbb{D}, R)$ is marked. Therefore if $(\mathbb{G}, \tilde{R})$ is marked, by lemma 5.2.13 it must be of the same kind as $(\mathbb{D}, R)$. We also observe that the residuals of both redexes do not depend on what other redexes are marked in the records. We have the following cases:

1. $(\mathbb{D}, R)$ is a term redex. Then:

   - If $(\mathbb{G}, \tilde{R})$ is an unmarked term redex, then either the two redexes are independent, in which case the claim clearly holds, or $(\mathbb{D}, R)$ is contained in $(\mathbb{G}, \tilde{R})$ (note that $(\mathbb{G}, \tilde{R})$ is an evaluation redex, and therefore can not be contained in $(\mathbb{D}, R)$). In the latter case suppose $(\mathbb{D}, R)$ is contained in the operand of $(\mathbb{G}, \tilde{R})$:

$$
\begin{aligned}
&[l \mapsto \mathbb{E}\{(\lambda x.\mathbb{A}\{x, \ldots, x\}) \mathbin{@} \lambda y.\mathbb{B}\{\underline{R}\}\}, \ldots] && \Longrightarrow \\
&[l \mapsto \mathbb{E}\{\mathbb{A}\{\lambda y.\mathbb{B}\{\underline{R}\}, \ldots, \lambda y.\mathbb{B}\{\underline{R}\}\}\}, \ldots] && \overset{\mathrm{e}}{\underset{\gamma}{\Longrightarrow}}{}^{*} \\
&[l \mapsto \mathbb{E}\{\mathbb{A}\{\lambda y.\mathbb{B}\{R\}, \ldots, \lambda y.\mathbb{B}\{R\}\}\}, \ldots], && \\
&[l \mapsto \mathbb{E}\{(\lambda x.\mathbb{A}\{x, \ldots, x\}) \mathbin{@} \lambda y.\mathbb{B}\{\underline{R}\}\}, \ldots] && \overset{\mathrm{e}}{\underset{\gamma}{\Longrightarrow}} \\
&[l \mapsto \mathbb{E}\{(\lambda x.\mathbb{A}\{x, \ldots, x\}) \mathbin{@} \lambda y.\mathbb{B}\{R\}\}, \ldots] && \Longrightarrow \\
&[l \mapsto \mathbb{E}\{\mathbb{A}\{\lambda y.\mathbb{B}\{R\}, \ldots, \lambda y.\mathbb{B}\{R\}\}\}, \ldots]. &&
\end{aligned}
$$

     The case when $(\mathbb{D}, R)$ is contained in the operator of $(\mathbb{G}, \tilde{R})$ is similar, but no redex duplication happens in this case.

   - If $(\mathbb{G}, \tilde{R})$ is a marked term redex, then the case is analogous to the previous one, but additionally we need to consider the following case: if $(\mathbb{G}, \tilde{R}) = (\mathbb{D}, R)$, then $D'_3 = D'_4$, since the residual of $(\mathbb{G}, \tilde{R})$ is unmarked in D_3.

   - If $(\mathbb{G}, \tilde{R}) = (\mathbb{G}, l)$ is a substitution redex, then it is unmarked. Since $(\mathbb{G}, l)$ is an evaluation redex, the label l can not be contained in $(\mathbb{D}, R)$. Suppose

$(\mathbb{D}, R)$ is contained in the value bound to l:

$$
\begin{array}{ll}
[l \mapsto \lambda x.\mathbb{C}\{\underline{R}\}, l' \mapsto \mathbb{E}\{l\}, \ldots] & \Longrightarrow \\
[l \mapsto \lambda x.\mathbb{C}\{\underline{R}\}, l' \mapsto \mathbb{E}\{\lambda x.\mathbb{C}\{\underline{R}\}\}, \ldots] & \overset{\mathrm{e}}{\underset{\gamma}{\Longrightarrow}}{}^{*} \\
[l \mapsto \lambda x.\mathbb{C}\{R\}, l' \mapsto \mathbb{E}\{\lambda x.\mathbb{C}\{R\}\}, \ldots], & \\
[l \mapsto \lambda x.\mathbb{C}\{\underline{R}\}, l' \mapsto \mathbb{E}\{l\}, \ldots] & \overset{\mathrm{e}}{\underset{\gamma}{\Longrightarrow}} \\
[l \mapsto \lambda x.\mathbb{C}\{R\}, l' \mapsto \mathbb{E}\{l\}, \ldots] & \Longrightarrow \\
[l \mapsto \lambda x.\mathbb{C}\{R\}, l' \mapsto \mathbb{E}\{\lambda x.\mathbb{C}\{R\}\}, \ldots]. &
\end{array}
$$

In the remaining case $(\mathbb{D}, R)$ is independent from $(\mathbb{G}, l)$, and the claim of the lemma clearly holds.

2. $(\mathbb{D}, R) = (\mathbb{D}, l)$ is a substitution. We have the subcases:

   - $(\mathbb{G}, \tilde{R})$ is a term redex. It can not be contained in the value bound to l.

     If $(\mathbb{G}, \tilde{R})$ contains the marked occurrence of l of $(\mathbb{D}, l)$, then this occurrence may be duplicated, in which case the resulting erasing γ-development remove all residuals of the marked l from the set of marked redexes. The case similar to the cases above where duplication of the marked redex occurs.

     Otherwise $(\mathbb{D}, l)$ is independent from $(\mathbb{G}, \tilde{R})$, no duplication occurs, and the claim clearly holds.

   - $(\mathbb{G}, \tilde{R}) = (\mathbb{G}, l')$, $l' \neq l$. If $(\mathbb{D}, l)$ is not contained in the value bound to l', then the claim clearly holds. Otherwise the marked occurrence of l gets duplicated by the evaluation redex, and both marked copies of l need to be "erased".

   - $(\mathbb{G}, \tilde{R}) = (\mathbb{G}, l)$. Note that the value bound to l does not contain a marked occurrence of l, since by the condition of the lemma there is no

self-referential redex among marked redexes. In this case either $(\mathbb{G}, l)$ is independent of $(\mathbb{D}, l)$ and the claim clearly holds, or $(\mathbb{G}, l) = (\mathbb{D}, l)$:

$$
\begin{aligned}
&[l \mapsto V, l' \mapsto \mathbb{E}\{\underline{l}\}, \ldots] \quad \Longrightarrow \quad [l \mapsto V, l' \mapsto \mathbb{E}\{V\}, \ldots], \\
&[l \mapsto V, l' \mapsto \mathbb{E}\{\underline{l}\}, \ldots] \quad \underset{\gamma}{\overset{\mathrm{e}}{\Longrightarrow}} \quad [l \mapsto V, l' \mapsto \mathbb{E}\{l\}, \ldots] \quad \Longrightarrow \\
&[l \mapsto V, l' \mapsto \mathbb{E}\{V\}, \ldots].
\end{aligned}
$$

□

Lemma 5.2.22. *Suppose $F_1 = marked(D'_1)$ does not contain any self-referential redexes. If $D'_1 \underset{\gamma}{\overset{(\mathbb{G},R)}{\Longrightarrow}} D'_2$ $D'_1 \overset{(\mathbb{G}',R')}{\Longrightarrow} D'_3$, then:*

- *If $(\mathbb{G}, R) \neq (\mathbb{G}', R')$, then there exists D'_4 such that $D'_2 \overset{(\mathbb{G}',R')/(\mathbb{G},R)}{\Longrightarrow} D'_4$ and $D'_3 \underset{\gamma}{\overset{(\mathbb{G},R)/(\mathbb{G}',R')}{\Longrightarrow}} D'_4$;*
- *If $(\mathbb{G}, R) = (\mathbb{G}', R')$, then $D'_2 = D'_3$.*

□

Proof. The first part by lemmas 5.2.18 and 5.2.20 and by definition of marked reductions. The second part by lemma 5.2.19 and by definition of marked reductions. □

The case when F_1 has a self-referential redex is slightly more complicated: we cannot use definition of marked reductions, as we did for the case above when F_1 does not have self-referential redexes, because in the case of a self-referential redex γ-development reduction is different from the marked reduction:

$$
\begin{aligned}
&[l_1 \mapsto \lambda x.\underline{l_1}, l_2 \mapsto \underline{l_1}] \quad \Longrightarrow \quad [l_1 \mapsto \lambda x.\underline{l_1}, l_2 \mapsto \lambda x.\underline{l_1}], \quad \text{but} \\
&[l_1 \mapsto \lambda x.\underline{l_1}, l_2 \mapsto \underline{l_1}] \quad \underset{\gamma}{\Longrightarrow} \quad [l_1 \mapsto \lambda x.\underline{l_1}, l_2 \mapsto \lambda x.l_1].
\end{aligned}
$$

The label l_1 in the second component of the result is marked in the first case, but not in the second. However, it turns out that for both reductions a residual set of a redex does not depend on the marking of any redexes in the record other than the self-referential one, as shown in the following two lemmas. Note that the second lemma considers an arbitrary γ-development step, not just an evaluation step, and thus will be used for the proofs of elementary lift and project diagrams later in this section.

Lemma 5.2.23. *Suppose* $D'_1 \xrightarrow{(\mathbb{G},R)} D'_2$, $(\mathbb{D}, l) \in \mathit{marked}(D'_1)$ *is a self-referential redex, and* $D'_1 \in \mathit{dom}(\gamma)$. *Let* $(\mathbb{D}', l)$ *be a non-self-referential redex in* D'_1 *such that* $(\mathbb{D}', l) \notin \mathit{marked}(D'_1)$.

Let D''_1 *be such that* $|D'_1| = |D''_1|$ *and* $\mathit{marked}(D''_1) = \mathit{marked}(D'_1) \cup \{(\mathbb{D}', l)\}$, *and let* $D''_1 \xrightarrow{(\mathbb{G},R)} D''_2$. *Then* $\mathit{marked}(D''_2) = \mathit{marked}(D'_2) \cup ((\mathbb{D}', l)/(\mathbb{G}, R))$. □

Proof. This follows directly from definition of marked reduction. Note that the lemma holds for both marked and unmarked $(\mathbb{G}, R)$. □

Lemma 5.2.24. *Let* $\mathit{marked}(D'_1) = \{(\mathbb{D}, l), (\mathbb{D}', l)\}$, $D'_1 \overset{(\mathbb{D},l)}{\underset{\gamma}{\rightarrow}} D'_2$. *Let* $\mathit{marked}(D''_1) = \{(\mathbb{D}, l), (\mathbb{D}'', l)\}$, $|D''_1| = |D'_1|$, *and* $D''_1 \overset{(\mathbb{D},l)}{\underset{\gamma}{\rightarrow}} D''_2$. *Suppose that no more than one of the three redexes is self-referential, and that* $D'''_1 \in \mathit{dom}(\gamma)$, *where* $\mathit{marked}(D'''_1) = \{(\mathbb{D}, l), (\mathbb{D}', l), (\mathbb{D}'', l)\}$, $|D'''_1| = |D'_1|$. *Then* $D'''_1 \overset{(\mathbb{D},l)}{\underset{\gamma}{\rightarrow}} D'''_2$, *where* $\mathit{marked}(D'''_2) = \mathit{marked}(D'_2) \cup \mathit{marked}(D''_2)$. □

Proof. Since at most one of the three redexes may be a self-referential redex, we consider the following 3 cases:

- None of the three redexes is self-referential. Let $\tilde{\mathbb{D}}$ be a 4-hole context such

that the first hole contains the value of the component bound to l, then

$$\begin{aligned} \text{Given} \quad & \tilde{\mathbb{D}}\{V, \underline{l}, \underline{l}, l\} \overset{(\mathbb{D},l)}{\underset{\gamma}{\rightarrow}} \tilde{\mathbb{D}}\{V, V, \underline{l}, l\} \\ \text{and} \quad & \tilde{\mathbb{D}}\{V, \underline{l}, l, \underline{l}\} \overset{(\mathbb{D},l)}{\underset{\gamma}{\rightarrow}} \tilde{\mathbb{D}}\{V, V, l, \underline{l}\}, \\ \text{we have} \quad & \tilde{\mathbb{D}}\{V, \underline{l}, \underline{l}, \underline{l}\} \overset{(\mathbb{D},l)}{\underset{\gamma}{\rightarrow}} \tilde{\mathbb{D}}\{V, V, \underline{l}, \underline{l}\}. \end{aligned}$$

Here the first reduction corresponds is the reduction when only $(\mathbb{D}', l)$, but not $(\mathbb{D}'', l)$, is marked, and the second reduction is the case when $(\mathbb{D}'', l)$, but not $(\mathbb{D}', l)$, is marked.

- $(\mathbb{D}', l)$ is self-referential (note that this is completely analogous to the case when $(\mathbb{D}'', l)$ is self-referential, so we show only one of these cases). In this case no other marked redex occurs in the component bound to l. The redexes $(\mathbb{D}, l)$ and $(\mathbb{D}'', l)$ may occur in the same or in different components. Let us show the case when they occur in the same component, the other case is analogous:

  Given

  $$[l \mapsto \lambda x.\mathbb{C}\{\underline{l}\}, l' \mapsto \mathbb{A}\{\underline{l}, l\}, \ldots] \overset{(\mathbb{D},l)}{\underset{\gamma}{\rightarrow}} [l \mapsto \lambda x.\mathbb{C}\{\underline{l}\}, l' \mapsto \mathbb{A}\{\lambda x.\mathbb{C}\{l\}, l\}, \ldots]$$

  and

  $$[l \mapsto \lambda x.\mathbb{C}\{l\}, l' \mapsto \mathbb{A}\{\underline{l}, \underline{l}\}, \ldots] \overset{(\mathbb{D},l)}{\underset{\gamma}{\rightarrow}} [l \mapsto \lambda x.\mathbb{C}\{l\}, l' \mapsto \mathbb{A}\{\lambda x.\mathbb{C}\{l\}, \underline{l}\}, \ldots],$$

  we have

  $$[l \mapsto \lambda x.\mathbb{C}\{\underline{l}\}, l' \mapsto \mathbb{A}\{\underline{l}, \underline{l}\}, \ldots] \overset{(\mathbb{D},l)}{\underset{\gamma}{\rightarrow}} [l \mapsto \lambda x.\mathbb{C}\{\underline{l}\}, l' \mapsto \mathbb{A}\{\lambda x.\mathbb{C}\{l\}, \underline{l}\}, \ldots].$$

- $(\mathbb{D}, l)$ is self-referential (in this case the γ-development step is a non-evaluation step). As in the previous case, the other two redexes may occur in the same component or in two different ones. As before, we show only the former case,

the latter one is analogous.

Given

$$[l \mapsto \lambda x.\mathbb{C}\{\underline{l}\}, l' \mapsto \mathbb{A}\{\underline{l}, l\}, \ldots] \overset{(\mathbb{D},l)}{\underset{\gamma}{\rightarrow}} [l \mapsto \lambda x.\mathbb{C}\{\lambda x.\mathbb{C}\{l\}\}, l' \mapsto \mathbb{A}\{\underline{l}, l\}, \ldots]$$

and

$$[l \mapsto \lambda x.\mathbb{C}\{\underline{l}\}, l' \mapsto \mathbb{A}\{l, \underline{l}\}, \ldots] \overset{(\mathbb{D},l)}{\underset{\gamma}{\rightarrow}} [l \mapsto \lambda x.\mathbb{C}\{\lambda x.\mathbb{C}\{l\}\}, l' \mapsto \mathbb{A}\{l, \underline{l}\}, \ldots],$$

we have

$$[l \mapsto \lambda x.\mathbb{C}\{\underline{l}\}, l' \mapsto \mathbb{A}\{\underline{l}, \underline{l}\}, \ldots] \overset{(\mathbb{D},l)}{\underset{\gamma}{\rightarrow}} [l \mapsto \lambda x.\mathbb{C}\{\lambda x.\mathbb{C}\{l\}\}, l' \mapsto \mathbb{A}\{\underline{l}, \underline{l}\}, \ldots].$$

□

We can generalize the previous lemma to the case of arbitrary sets of marked redexes in a record:

Corollary 5.2.25. *Let* $F_1 = \mathit{marked}(D'_1)$ *and let* D''_1 *be such that* $|D'_1| = |D''_1|$, $\mathit{marked}(D''_1) = \{(\mathbb{D}, l), (\mathbb{D}', l)\}$ *and* $(\mathbb{D}'l) \notin F_1$. *If* $D'_1 \overset{(\mathbb{D},l)}{\underset{\gamma}{\rightarrow}} D'_2$ *and* $D''_1 \overset{(\mathbb{D},l)}{\underset{\gamma}{\rightarrow}} D''_2$ *and* $D'''_1 \in \mathit{dom}(\gamma)$ *(where* $|D'''_1| = |D'_1|$ *and* $\mathit{marked}(D'''_1) = F_1 \cup \{(\mathbb{D}', l)\}$*), then* $D'''_1 \overset{(\mathbb{D},l)}{\underset{\gamma}{\rightarrow}} D'''_2$, $|D'''_2| = |D'_2|$, *and* $\mathit{marked}(D'''_2) = \mathit{marked}(D'_2) \cup \mathit{marked}(D''_2)$. □

Proof. If $F_1 = \emptyset$, the claim trivially holds. If F_1 consists of one redex, the claim holds by lemma 5.2.24.

Suppose F_1 consists of n redexes. By the condition $D'''_1 \in \mathit{dom}(\gamma)$ at most one of the marked redexes is self-referential. By lemma 5.2.24 the set of residuals of the marked redex $(\mathbb{D}', l)$ with respect to the redex $(\mathbb{D}, l)$ does not depend on whether any other redexes in the record are marked. Similarly the set of residuals of any redex in F_1 does not depend on whether $(\mathbb{D}', l)$ is marked. Therefore the resulting set of marked redexes is the union $\mathrm{marked}(D'_2) \cup \mathrm{marked}(D''_2)$, where $\mathrm{marked}(D'_2) = F_1/(\mathbb{D}, l)$, and $\mathrm{marked}(D''_2) = (\mathbb{D}', l)/(\mathbb{D}, l)$. □

The following two lemmas deal with the cases when the self-referential redex is the only marked one (in addition to the redex being reduced) and when there is one more marked redex. Lemmas 5.2.28 and 5.2.29 generalize these two cases to an arbitrary set of marked redexes containing a self-referential redex.

Lemma 5.2.26. *Let* $marked(D'_1) = \{(\mathbb{D}, l_1), (\mathbb{G}, l_1)\}$. *If* $D'_1 \underset{\gamma}{\overset{(\mathbb{G},l_1)}{\Longrightarrow}} D'_2$, $D'_1 \overset{(\mathbb{G}',R')}{\Longrightarrow} D'_3$, *where* $(\mathbb{D}, l_1)$ *is a self-referential redex and* $(\mathbb{G}, l_1) \neq (\mathbb{G}', R')$, *then there exists* D'_4 *such that* $D'_2 \overset{(\mathbb{G}',R')/(\mathbb{G},l_1)}{\Longrightarrow} D'_4$ *and* $D'_3 \underset{\gamma}{\overset{(\mathbb{G},l_1)/(\mathbb{G}',R')}{\Longrightarrow}} D'_4$. □

Proof. The proof is by cases on $(\mathbb{G}', R')$. Note that both $(\mathbb{G}', R')$ and $(\mathbb{G}, l_1)$ are evaluation redexes, and therefore they occur in two different components, and none of these components is bound to l_1.

- $(\mathbb{G}', R')$ is a term redex:

$$D_1 \quad = \quad [l_1 \mapsto \lambda x.\mathbb{C}\{\underline{l_1}\}, l \mapsto \mathbb{E}\{\underline{l_1}\}, l' \mapsto \mathbb{E}'\{R'\}, \ldots]$$

  The two redexes reduced are independent, and on both reduction paths we arrive at the record:

$$D_4 \quad = \quad [l_1 \mapsto \lambda x.\mathbb{C}\{\underline{l_1}\}, l \mapsto \mathbb{E}\{\lambda x.\mathbb{C}\{l_1\}\}, l' \mapsto \mathbb{E}'\{Q'\}, \ldots]$$

  So the self-referential redex has only one residual - itself.

- $(\mathbb{G}', R')$ is a substitution redex, $R' = l_2 \neq l_1$. Analogous to the previous case.

- $(\mathbb{G}', R')$ is a substitution redex, $R' = l_1$. Note that l_1 in $(\mathbb{G}', R')$ is not marked, since, by the conditions of the lemma, only the other two redexes are marked.

Then

$$\begin{aligned} D_1 &= [l_1 \mapsto \lambda x.\mathbb{C}\{\underline{l_1}\}, l \mapsto \mathbb{E}\{\underline{l_1}\}, l' \mapsto \mathbb{E}'\{l_1\}, \ldots], \\ D_4 &= [l_1 \mapsto \lambda x.\mathbb{C}\{\underline{l_1}\}, l \mapsto \mathbb{E}\{\lambda x.\mathbb{C}\{l_1\}\}, l' \mapsto \mathbb{E}'\{\lambda x.\mathbb{C}\{\underline{l_1}\}\}, \ldots]. \end{aligned}$$

□

The case when the two evaluation steps reduce the same redex needs to be treated separately, since the label on one of the redexes gets preserved, and the resulting γ-development step erases this redex:

Lemma 5.2.27. *Suppose* $marked(D_1') = \{(\mathbb{D}, l), (\mathbb{G}, l)\}$, $D_1' \xrightarrow[\gamma]{(\mathbb{G},l)} D_2'$ *and* $D_1' \xrightarrow{(\mathbb{G},l)} D_3'$, *where* $(\mathbb{D}, l)$ *is self-referential. Then* $D_3' \xrightarrow[\gamma]{\overline{(\mathbb{G},l)}} D_2'$. □

Proof. The following reduction sequences prove the claim. Here $\xRightarrow[\gamma]{\mathrm{e}}$ denotes the erasing γ-development step.

$$\begin{aligned} &[l \mapsto \lambda x.\mathbb{C}\{\underline{l}\}, l' \mapsto \mathbb{E}\{\underline{l}\}, \ldots] && \xRightarrow[\gamma]{} \\ &[l \mapsto \lambda x.\mathbb{C}\{\underline{l}\}, l' \mapsto \mathbb{E}\{\lambda x.\mathbb{C}\{l\}\}, \ldots], \\ &[l \mapsto \lambda x.\mathbb{C}\{\underline{l}\}, l' \mapsto \mathbb{E}\{\underline{l}\}, \ldots] && \xRightarrow{} \\ &[l \mapsto \lambda x.\mathbb{C}\{\underline{l}\}, l' \mapsto \mathbb{E}\{\lambda x.\mathbb{C}\{\underline{l}\}\}, \ldots] && \xRightarrow[\gamma]{\mathrm{e}} \\ &[l \mapsto \lambda x.\mathbb{C}\{\underline{l}\}, l' \mapsto \mathbb{E}\{\lambda x.\mathbb{C}\{l\}\}, \ldots]. \end{aligned}$$

□

Lemma 5.2.28. *Suppose* $F_1 = marked(D_1')$ *has a self-referential redex. If* $D_1' \xrightarrow[\gamma]{(\mathbb{G},l)} D_2'$, $D_1' \xrightarrow{(\mathbb{G}',R')} D_3'$, *then there exists* D_4' *such that* $D_2' \xrightarrow{(\mathbb{G}',R')/(\mathbb{G},l)} D_4'$, *and* $D_3' \xrightarrow[\gamma]{(\mathbb{G},l)/(\mathbb{G}',R')} D_4'$. □

Proof. The proof is by induction on the number of redexes in F_1. If F_1 consists of

just the self-referential redex in addition to the redex $(\mathbb{G}, l)$, then by lemma 5.2.26 the claim of the lemma holds.

Suppose the lemma holds for an n-redex subset of F_1, let F_1' denote this subset. Let $(\mathbb{D}', l) \in F_1$, $(\mathbb{D}', l) \notin F_1'$. Let $(\mathbb{D}, l)$ be the self-referential redex of F_1'. Note that $(\mathbb{D}', l)$ is not self-referential, since $D_1' \in dom(\gamma)$, and therefore by lemma 5.2.13 F_1 has no more than one self-referential redex.

Let D_1'' be such that $|D_1''| = |D_1'|$ and $\text{marked}(D_1'') = F_1'$. By the inductive hypothesis $D_1'' \xrightarrow[\gamma]{(\mathbb{G},l)} D_2''$, $D_1'' \xrightarrow{(\mathbb{G}',R')} D_3''$ implies that there exists D_4'' such that $D_2'' \xrightarrow{(\mathbb{G}',R')/(\mathbb{G},l)} D_4''$ and $D_3'' \xrightarrow[\gamma]{(\mathbb{G},l)/(\mathbb{G}',R')} D_4''$.

Let D_1''' be such that $|D_1'''| = |D_1'|$ and $\text{marked}(D_1''') = \{(\mathbb{D}, l), (\mathbb{G}, l)\})$. By lemma 5.2.20 $D_1''' \xrightarrow[\gamma]{(\mathbb{G},l)} D_2'''$ and $D_1''' \xrightarrow{(\mathbb{G}',R')} D_3'''$ implies that there exists D_4''' such that $D_2''' \xrightarrow{(\mathbb{G}',R')/(\mathbb{G},l)} D_4'''$, $D_3''' \xrightarrow[\gamma]{(\mathbb{G},l)/(\mathbb{G}',R')} D_4'''$.

We combine the two diagrams by lemma 5.2.23 for the $\Longrightarrow$ steps and by lemma 5.2.24 for the $\underset{\gamma}{\Longrightarrow}$ steps: let $\tilde{D'}_1'$ be such that $\text{marked}(\tilde{D'}_1') = F_1' \cup \{(\mathbb{D}, l)\})$. Then $\tilde{D}_1 \xrightarrow[\gamma]{(\mathbb{G},l)} \tilde{D}_2$, where $\text{marked}(\tilde{D}_2) = \text{marked}(D_2'') \cup \text{marked}(D_2''')$ and $\tilde{D}_1 \xrightarrow{(\mathbb{G}',R')} \tilde{D}_3$, where $\text{marked}(\tilde{D}_3) = \text{marked}(D_3'') \cup \text{marked}(D_3''')$ imply that $\tilde{D}_2 \xrightarrow{(\mathbb{G}',R')/(\mathbb{G},l)} \tilde{D}_4$, $\tilde{D}_3 \xrightarrow[\gamma]{(\mathbb{G},l)/(\mathbb{G}',R')} \tilde{D}_4$. This shows that the claim of the lemma holds for a set of $n+1$ redexes of D_1'. □

Lemma 5.2.29. *Suppose marked$(D_1') = F_1$ has a self-referential redex. If $D_1' \xrightarrow[\gamma]{\overline{(\mathbb{D},l)}} D_2'$, and $D_1' \xrightarrow{(\mathbb{G},\tilde{R})} D_3'$, then there exists D_4' such that $D_2' \xrightarrow{(\mathbb{G},\tilde{R})} D_4'$, and $D_3' \xrightarrow[\gamma]{e}{}^{*} D_4'$.* □

Proof. We observe that the only redex in F_1 that affects the reductions is the self-referential redex. The marking of all other redexes (i.e. those not equal to $(\mathbb{D}, l)$ and $(\mathbb{G}, \tilde{R})$) is preserved by the reductions, so we are not considering any other marked redexes.

The subcases are as follows:

1. If $(\mathbb{G}, \tilde{R})$ is a term redex, then the marked occurrence of l corresponding to $(\mathbb{D}, l)$ is either independent of $(\mathbb{G}, \tilde{R})$, or is contained in $(\mathbb{G}, \tilde{R})$. In the former case the claim of the lemma clearly holds. In the latter case the marked occurrence of l may be duplicated by the reduction of $(\mathbb{G}, \tilde{R})$, in which case all the copies of l need to be "erased" by the $\overset{\mathrm{e}}{\underset{\gamma}{\Longrightarrow}}^{*}$ sequence.

2. If $(\mathbb{G}, \tilde{R}) = (\mathbb{G}, l')$ is a substitution redex and $l' \neq l$, then the only dependency between the two redexes is when $(\mathbb{D}, l)$ occurs in the value bound to l'. In this case:

$$
\begin{aligned}
&[l \mapsto \lambda x.\mathbb{C}\{\underline{l}\}, l' \mapsto \lambda y.\mathbb{B}\{\underline{l}\}, l'' \mapsto \mathbb{E}\{l'\}, \ldots] && \Longrightarrow \\
&[l \mapsto \lambda x.\mathbb{C}\{\underline{l}\}, l' \mapsto \lambda y.\mathbb{B}\{\underline{l}\}, l'' \mapsto \mathbb{E}\{\lambda y.\mathbb{B}\{\underline{l}\}\}, \ldots] && \overset{\mathrm{e}}{\underset{\gamma}{\Longrightarrow}}^{*} \\
&[l \mapsto \lambda x.\mathbb{C}\{\underline{l}\}, l' \mapsto \lambda y.\mathbb{B}\{l\}, l'' \mapsto \mathbb{E}\{\lambda y.\mathbb{B}\{l\}\}, \ldots], && \\
&[l \mapsto \lambda x.\mathbb{C}\{\underline{l}\}, l' \mapsto \lambda y.\mathbb{B}\{\underline{l}\}, l'' \mapsto \mathbb{E}\{l'\}, \ldots] && \overset{\mathrm{e}}{\underset{\gamma}{\Longrightarrow}} \\
&[l \mapsto \lambda x.\mathbb{C}\{\underline{l}\}, l' \mapsto \lambda y.\mathbb{B}\{l\}, l'' \mapsto \mathbb{E}\{l'\}, \ldots] && \Longrightarrow \\
&[l \mapsto \lambda x.\mathbb{C}\{\underline{l}\}, l' \mapsto \lambda y.\mathbb{B}\{l\}, l'' \mapsto \mathbb{E}\{\lambda y.\mathbb{B}\{l\}\}, \ldots]. &&
\end{aligned}
$$

   The other cases are trivial.

3. If $(\mathbb{G}, \tilde{R}) = (\mathbb{G}, l)$, then we have the following possibilities:

   - $(\mathbb{G}, l)$ is not marked. In this case the two redexes are independent (since $(\mathbb{D}, l)$ is not self-referential, and therefore can not occur in the component bound to l), and the claim of the lemma holds.
   - $(\mathbb{G}, l)$ is marked, $(\mathbb{G}, l) \neq (\mathbb{D}, l)$. The two redexes also must be independent, and the case is similar to previous one.
   - $(\mathbb{G}, l) = (\mathbb{D}, l)$. This case is considered in lemma 5.2.27.

□

Lemma 5.2.30 (γ-confluence of Evaluation of $\mathcal{C}$). *The calculus $\mathcal{C}$ has γ-confluence of evaluation (property 4.7.1), i.e. if $D'_1 \underset{\gamma}{\Longrightarrow} D'_2$, $D'_1 \Longrightarrow D'_3$, then there exists D'_4 such that $D'_2 \Longrightarrow D'_4$ and $D'_3 \underset{\gamma}{\Longrightarrow} D'_4$.* □

Proof. If marked(D'_1) does not contain a self-referential redex, then the lemma follows by lemma 5.2.22, otherwise by lemmas 5.2.28 and 5.2.29. □

5.2.5 Elementary Lift and Project Diagrams in Calculus of Records

Properties 4.6.1 and 4.6.2 are another two properties required for the proof of lift and project. To prove these properties, we use an approach similar to that of the proof of γ-confluence of evaluation above.

The following property of terms is used in the further proofs.

Lemma 5.2.31. *If $M = \mathbb{E}\{l\}$, then there is no such context $\mathbb{C}$ and redex R that $M = \mathbb{C}\{R\}$ and $R = \mathbb{A}\{l\}$, i.e. such that R contains the occurrence of l.* □

Proof. Suppose such $\mathbb{C}$ and R exist. If $M = \mathbb{E}\{l\}$ and l is contained in a redex R, then $M = \mathbb{C}\{\mathbb{A}\{l\}\}$, and $\mathbb{E} = \mathbb{C}\{\mathbb{A}\}$. Therefore $\mathbb{C} = \mathbb{E}' \in \mathbf{EvalContext}_{\mathcal{T}}$, and $M = \mathbb{E}'\{R\}$, which contradicts the class preservation lemma 5.1.21. □

The following 4 lemmas show all cases for the proof of property 4.6.2 (elementary project diagram).

Lemma 5.2.32. *Suppose marked(D'_1) $= \{(\mathbb{D}, R), (\mathbb{D}', R')\}$, $D'_1 \underset{\gamma}{\circ\!\!\xrightarrow{(\mathbb{D},R)}} D'_2$, and $D'_1 \xrightarrow{(\mathbb{G},\tilde{R})} D'_3$, where $(\mathbb{D}, R)$ and $(\mathbb{D}', R')$ are not self-referential. Then there exists D'_4 such that $D'_2 \xrightarrow{(\mathbb{G},\tilde{R})/(\mathbb{D},R)} D'_4$ and $D'_3 \underset{\gamma}{\overset{(\mathbb{D},R)/(\mathbb{G},\tilde{R})}{\longrightarrow^*}} D'_4$. Moreover, if the sequence*

$\overset{(\mathbb{D},R)/(\mathbb{G},\tilde{R})}{\underset{\gamma}{\longrightarrow}}{}^{*}$ *consists of more than one step, then for any such reduction sequence* $D_3' \overset{(\mathbb{D},R)/(\mathbb{G},\tilde{R})}{\underset{\gamma}{\longrightarrow}}{}^{*} D_4'$. □

Proof. Let l, l', $\tilde{l}$ be the labels of the components containing $(\mathbb{D}, R), (\mathbb{D}', R')$, and $(\mathbb{G}, \tilde{R})$ respectively. We have the following cases:

1. All three redexes are term redexes. When all three redexes are independent, the claim obviously holds. It also clearly holds in the case when $(\mathbb{D}, R)$ and $(\mathbb{G}, \tilde{R})$ are in different terms, even if $(\mathbb{D}', R')$ is in the same term with one of the former redexes. If $(\mathbb{G}, \tilde{R})$ contains $(\mathbb{D}, R)$, then we apply lemma 5.1.15 to the term where both of these redexes occur. This is the only case when $(\mathbb{D}, R)$ may be duplicated, i.e. the sequence $\overset{(\mathbb{D},R)/(\mathbb{G},\tilde{R})}{\underset{\gamma}{\longrightarrow}}{}^{*}$ may consist of more than one step. By lemma 5.1.15 the order of steps in this sequence does not matter.

   There are no other cases, since $(\mathbb{D}, R)$ can not contain $(\mathbb{G}, \tilde{R})$ because the latter is an evaluation redex.

2. The marked redexes are term redexes, and the evaluation redex is a substitution redex. Let $\tilde{R} = l_1$. Note that $l_1 \neq \tilde{l}$, since $(\mathbb{G}, l_1)$ is an evaluation redex, and therefore can not be self-referential. We have the following subcases:

   (a) Among the 4 labels l, l', l_1, and $\tilde{l}$, no two are equal. In this case the claim of the lemma clearly holds.

   (b) $l_1 \notin \{l, l'\}$. It may be the case that $l = \tilde{l} \neq l'$, $l' = \tilde{l} \neq l$, or $l = l' = \tilde{l}$. Assume the third possibility. By lemma 5.2.31 the redexes R and R' can

not contain the redex occurrence of l_1. We have the following options:

I.	$[\tilde{l} \mapsto \mathbb{A}\{l_1, R, R'\}, l_1 \mapsto V, \dots]$	R, R' independent
II.	$[\tilde{l} \mapsto \mathbb{A}\{l_1, \mathbb{B}\{R'\}\}, l_1 \mapsto V, \dots]$	$R = \mathbb{B}\{R'\}$
III.	$[\tilde{l} \mapsto \mathbb{A}\{l_1, \mathbb{B}\{R\}\}, l_1 \mapsto V, \dots]$	$R' = \mathbb{B}\{R\}$

By lemma 5.1.14 in all three cases the context containing l is still an evaluation context after the reduction of R. The substitution does not affect the reduction of R, and from the properties of the term calculus it follows that in all three cases the residual(s) of R' will be the same regardless of which of the redexes has been reduced first: l_1 or R. No duplication happens in this case, so the resulting γ-development sequence consists of a single step.

The other two cases ($l = \tilde{l} \neq l'$ and $l' = \tilde{l} \neq l$) are analogous to the case we have considered, but simpler.

(c) $\tilde{l} \notin \{l, l'\}$. As in the previous case, let us consider the subcase $l = l' = l_1$ in detail, the other two subcases ($l = l_1 \neq l'$ and $l' = l_1 \neq l$) are similar, but easier.

If $l = l' = l_1$, then both R and R' occur in the value being substituted by the evaluation redex. Since this case is more complex than the previous one, we show the actual reductions rather than just the initial records. As above, we have the following possibilities for mutual positions of R and R':

I. R and R' are independent:

$$
\begin{aligned}
&[\tilde{l} \mapsto \mathbb{E}\{l_1\}, l_1 \mapsto \lambda x.\mathbb{C}\{R, R'\}, \ldots] && \overset{\circ}{\underset{\gamma}{\longrightarrow}} \\
&[\tilde{l} \mapsto \mathbb{E}\{l_1\}, l_1 \mapsto \lambda x.\mathbb{C}\{Q, R'\}, \ldots] && \Longrightarrow \\
&[\tilde{l} \mapsto \mathbb{E}\{\lambda x.\mathbb{C}\{Q, R'\}\}, l_1 \mapsto \lambda x.\mathbb{C}\{Q, R'\}, \ldots], && \\
&[\tilde{l} \mapsto \mathbb{E}\{l_1\}, l_1 \mapsto \lambda x.\mathbb{C}\{R, R'\}, \ldots] && \Longrightarrow \\
&[\tilde{l} \mapsto \mathbb{E}\{\lambda x.\mathbb{C}\{R, R'\}\}, l_1 \mapsto \lambda x.\mathbb{C}\{R, R'\}, \ldots] && \underset{\gamma}{\longrightarrow}^* \\
&[\tilde{l} \mapsto \mathbb{E}\{\lambda x.\mathbb{C}\{Q, R'\}\}, l_1 \mapsto \lambda x.\mathbb{C}\{Q, R'\}, \ldots]. &&
\end{aligned}
$$

The order of reduction of the copies of R in the sequence $\underset{\gamma}{\longrightarrow}^*$ does not matter. II. R contains R'. Suppose R' is contained in the operand of R. Let N denote $\lambda z.\mathbb{B}\{R'\}$, L denote $\mathbb{A}\{y, \ldots, y\}$. We have the following:

$$
\begin{aligned}
&[\tilde{l} \mapsto \mathbb{E}\{l_1\}, l_1 \mapsto \lambda x.\mathbb{C}\{\lambda y.L \mathbin{@} N\}, \ldots] && \overset{\circ}{\underset{\gamma}{\longrightarrow}} \\
&[\tilde{l} \mapsto \mathbb{E}\{l_1\}, l_1 \mapsto \lambda x.\mathbb{C}\{\mathbb{A}\{N, \ldots, N\}\}, \ldots] && \Longrightarrow \\
&[\tilde{l} \mapsto \mathbb{E}\{\lambda x.\mathbb{C}\{\mathbb{A}\{N, \ldots, N\}\}\}, l_1 \mapsto \lambda x.\mathbb{C}\{\mathbb{A}\{N, \ldots, N\}\}, \ldots], && \\
&[\tilde{l} \mapsto \mathbb{E}\{l_1\}, l_1 \mapsto \lambda x.\mathbb{C}\{\lambda y.L \mathbin{@} N\}, \ldots] && \Longrightarrow \\
&[\tilde{l} \mapsto \mathbb{E}\{\lambda x.\mathbb{C}\{\lambda y.L \mathbin{@} N\}\}, l_1 \mapsto \lambda x.\mathbb{C}\{\lambda y.L \mathbin{@} N\}, \ldots] && \underset{\gamma}{\longrightarrow}^* \\
&[\tilde{l} \mapsto \mathbb{E}\{\lambda x.\mathbb{C}\{\mathbb{A}\{N, \ldots, N\}\}\}, l_1 \mapsto \lambda x.\mathbb{C}\{\mathbb{A}\{N, \ldots, N\}\}, \ldots]. &&
\end{aligned}
$$

In this case R' gets duplicated by both redexes, but its residuals are the same on both reduction paths. The order of reduction of the two copies of the redex $\lambda y.\mathbb{A}\{y, \ldots, y\} \mathbin{@} \lambda z.\mathbb{B}\{R'\}$ does not matter.

If R' is contained in the operator of R, then it gets changed, rather than duplicated, by reduction of R. The redex R gets duplicated by the substitution. It is clear that R' has the same residuals on both reduction paths

and that the order of reduction of the two copies of R does not matter.

III. R' contains R. Let $R' = \mathbb{A}\{R\}$. It does not matter if R is contained in the operand or in the operator of R', because R' is not reduced in these reductions. We have:

$$\begin{aligned}
&[\tilde{l} \mapsto \mathbb{E}\{l_1\}, l_1 \mapsto \lambda y.\mathbb{C}\{\mathbb{A}\{R\}\}, \ldots] && \underset{\gamma}{\circ\!\!\longrightarrow} \\
&[\tilde{l} \mapsto \mathbb{E}\{l_1\}, l_1 \mapsto \lambda y.\mathbb{C}\{\mathbb{A}\{Q\}\}, \ldots] && \Longrightarrow \\
&[\tilde{l} \mapsto \mathbb{E}\{\lambda y.\mathbb{C}\{\mathbb{A}\{Q\}\}\}, l_1 \mapsto \lambda y.\mathbb{C}\{\mathbb{A}\{Q\}\}, \ldots]. && \\
&[\tilde{l} \mapsto \mathbb{E}\{l_1\}, l_1 \mapsto \lambda y.\mathbb{C}\{\mathbb{A}\{R\}\}, \ldots] && \Longrightarrow \\
&[\tilde{l} \mapsto \mathbb{E}\{\lambda y.\mathbb{C}\{\mathbb{A}\{R\}\}\}, l_1 \mapsto \lambda y.\mathbb{C}\{\mathbb{A}\{R\}\}, \ldots] && \underset{\gamma}{\longrightarrow}{}^{*} \\
&[\tilde{l} \mapsto \mathbb{E}\{\lambda y.\mathbb{C}\{\mathbb{A}\{Q\}\}\}, l_1 \mapsto \lambda y.\mathbb{C}\{\mathbb{A}\{Q\}\}, \ldots] &&
\end{aligned}$$

The redex R' has two residuals of the form $\mathbb{A}\{Q\}$ on both reduction paths. The copies of the redex R can be reduced in any order.

(d) $l = \tilde{l}$, $l' = l_1$. As in case (b), by lemma 5.2.31 the redex occurrence of l_1 in the component bound to $\tilde{l}$ is independent from the redex R. The initial record is

$$[\tilde{l} \mapsto \mathbb{A}\{l_1, R\}, l_1 \mapsto \lambda x.\mathbb{C}\{R'\}, \ldots].$$

Here R' gets duplicated by the substitution redex, but clearly has the same residuals on both reduction paths. By lemma 5.1.14 $\mathbb{A}\{l_1, Q\} \in \mathbf{EvalContext}_{\mathcal{T}}$.

(e) $l = l_1$, $l' = \tilde{l}$. The initial record is

$$[\tilde{l} \mapsto \mathbb{A}\{l_1, R'\}, l_1 \mapsto \lambda x.\mathbb{C}\{R\}, \ldots].$$

Here R, not R', gets duplicated by the substitution. The two residuals of

R can be reduced in any order. R' has only one residual in this case.

3. The marked redexes are substitution redexes, and the evaluation redex is a term redex. Let $R = l_1$, then $R' = l_1$ by lemma 5.2.13. By the condition of this lemma the marked redexes are not self-referential, so $l_1 \neq l$, $l_1 \neq l'$. Also $l_1 \neq \tilde{l}$, since l_1 is bound to a value, and $\tilde{l}$ is bound to an evaluatable term. As in the previous case, we have several subcases of mutual positions of the labels. Let us consider the case when $l = l' = \tilde{l}$, i.e. all three redexes occur in the same term. There may be several possibilities:

   - $\tilde{R}$ and the two occurrences of l_1 in the component bound to $\tilde{l}$ are independent, i.e. the initial record is:

     $$[\tilde{l} \mapsto \mathbb{A}\{\tilde{R}, \underline{l_1}, \underline{l_1}\}, l_1 \mapsto V, \dots]$$

     Without loss of generality, suppose that the first occurrence of $\underline{l_1}$ is the redex R. By lemma 5.1.13 $\mathbb{A}\{\Box, \underline{l_1}, \underline{l_1}\} \in \mathbf{EvalContext}_{\mathcal{T}}$ implies that $\mathbb{A}\{\Box, V, \underline{l_1}\} \in \mathbf{EvalContext}_{\mathcal{T}}$, therefore the reduction of $(\mathbb{G}, \tilde{R})/(\mathbb{D}, l_1)$ is indeed an evaluation step. It is clear that both reduction paths result in the same record with the same (single) residual of R'.

- Both R and R' occur in the operand part of $\tilde{R}$:

$$
\begin{array}{ll}
[\tilde{l} \mapsto \mathbb{E}\{(\lambda x.\mathbb{A}\{x,\ldots,x\}) \mathbin{@} \lambda y.\mathbb{B}\{\underline{l_1},\underline{l_1}\}\}, l_1 \mapsto V, \ldots] & \underset{\gamma}{\circ\!\!\longrightarrow} \\
[\tilde{l} \mapsto \mathbb{E}\{(\lambda x.\mathbb{A}\{x,\ldots,x\}) \mathbin{@} \lambda y.\mathbb{B}\{V,\underline{l_1}\}\}, l_1 \mapsto V, \ldots] & \Longrightarrow \\
[\tilde{l} \mapsto \mathbb{E}\{\mathbb{A}\{\lambda y.\mathbb{B}\{V,\underline{l_1}\},\ldots,\lambda y.\mathbb{B}\{V,\underline{l_1}\}\}\}, l_1 \mapsto V, \ldots], & \\
[\tilde{l} \mapsto \mathbb{E}\{(\lambda x.\mathbb{A}\{x,\ldots,x\}) \mathbin{@} \lambda y.\mathbb{B}\{\underline{l_1},\underline{l_1}\}\}, l_1 \mapsto V, \ldots] & \Longrightarrow \\
[\tilde{l} \mapsto \mathbb{E}\{\mathbb{A}\{\lambda y.\mathbb{B}\{\underline{l_1},\underline{l_1}\},\ldots,\lambda y.\mathbb{B}\{\underline{l_1},\underline{l_1}\}\}\}, l_1 \mapsto V, \ldots] & \underset{\gamma}{\longrightarrow}^{*} \\
[\tilde{l} \mapsto \mathbb{E}\{\mathbb{A}\{\lambda y.\mathbb{B}\{V,\underline{l_1}\},\ldots,\lambda y.\mathbb{B}\{V,\underline{l_1}\}\}\}, l_1 \mapsto V, \ldots]. &
\end{array}
$$

  The copies of $R = \underline{l}$ can be reduced in any order in the sequence $\underset{\gamma}{\longrightarrow}^{*}$.

- The other cases (when both R and R' occur in the operator part of $\tilde{R}$, or when one of them occurs in the operator and the other in the operand) are similar.

The other cases ($l = l' \neq \tilde{l}$, $l = \tilde{l} \neq l'$, $l' = \tilde{l} \neq l$, and all the three labels distinct) are similar to the one we have considered, but easier.

4. All three redexes are substitution redexes, where $\tilde{R} = l_2 \neq l_1$. In this case $\tilde{l} \neq l_2$, since $\tilde{R}$ is an evaluation redex. As in case 3, $l_1 \notin \{l, l', \tilde{l}\}$. We have the following cases:

   (a) All 5 labels are different. In this case the lemma trivially holds.

   (b) $l = l' = \tilde{l}$. The initial record is

$$[\tilde{l} \mapsto \mathbb{A}\{l_2, \underline{l_1}, \underline{l_1}\}, l_1 \mapsto V_1, l_2 \mapsto V_2, \ldots].$$

   By lemma 5.1.13 $\mathbb{A}\{\Box, V_1, \underline{l_1}\} \in \mathbf{EvalContext}_{\mathcal{T}}$, so the residual of the evaluation redex is indeed an evaluation redex. In this case no redexes get

duplicated, and the claim clearly holds.

(c) $l = l' = l_2$. Assuming (without loss of generality) that the first marked occurrence of l_1 corresponds to the redex R, the initial and the final records in this case are:

$$[\tilde{l} \mapsto \mathbb{E}\{l_2\}, l_1 \mapsto V_1, l_2 \mapsto \lambda x.\mathbb{C}\{\underline{l_1}, \underline{l_1}\}, \ldots],$$
$$[\tilde{l} \mapsto \mathbb{E}\{\lambda x.\mathbb{C}\{V_1, \underline{l_1}\}\}, l_1 \mapsto V_1, l_2 \mapsto \lambda x.\mathbb{C}\{V_1, \underline{l_1}\}, \ldots].$$

The two copies of R can be reduced in any order.

(d) $l = l'$, $l \neq \tilde{l}$, $l \neq l_2$. The claim clearly holds.

(e) $l = \tilde{l}$, $l' = l_2$. The initial and the final records are:

$$[\tilde{l} \mapsto \mathbb{A}\{l_2, \underline{l_1}\}, l_1 \mapsto V_1, l_2 \mapsto \lambda x.\mathbb{B}\{\underline{l_1}\}, \ldots],$$
$$[\tilde{l} \mapsto \mathbb{A}\{\lambda x.\mathbb{B}\{\underline{l_1}\}, V_1\}, l_1 \mapsto V_1, l_2 \mapsto \lambda x.\mathbb{B}\{\underline{l_1}\}, \ldots].$$

Again we use lemma 5.1.13 to show that the residual of the evaluation redex is an evaluation redex. Redex R' gets duplicated.

(f) $l = \tilde{l}$, $l' \neq l_2$, $l' \neq \tilde{l}$. Similar to the previous case.

(g) $l = l_2$, $l' = \tilde{l}$. The initial and the final records are:

$$[\tilde{l} \mapsto \mathbb{A}\{l_2, \underline{l_1}\}, l_1 \mapsto V_1, l_2 \mapsto \lambda x.\mathbb{B}\{\underline{l_1}\}, \ldots],$$
$$[\tilde{l} \mapsto \mathbb{A}\{\lambda x.\mathbb{B}\{V_1\}, \underline{l_1}\}, l_1 \mapsto V_1, l_2 \mapsto \lambda x.\mathbb{B}\{V_1\}, \ldots].$$

Here the redex R gets duplicated. Its two residuals can be reduced in any order. The redex R' has just a single residual on both reduction paths.

(h) $l = l_2$, $l' \neq l_2$, $l' \neq \tilde{l}$. Similar to the previous case.

5. All three redexes are substitution redexes, $\tilde{R} = l_1$. Note that $\tilde{R}$ is not marked

since by the condition of the lemma the only two marked redexes are R and R'. Similarly to the previous case, $l_1 \notin \{l, l', \tilde{l}\}$. Since none of the redexes R and R' occur in the component bound to l_1 (i.e. in the value being substituted), no redex duplication is possible, so the claim of the lemma clearly holds in this case.

□

Lemma 5.2.33. *Suppose marked*$(D'_1) = \{(\mathbb{D}, l), (\mathbb{D}', l)\}$. *Let* $D'_1 \overset{(\mathbb{D},l)}{\underset{\gamma}{\circ\!\!\longrightarrow}} D'_2$ *and* $D'_1 \overset{(\mathbb{G},\tilde{R})}{\Longrightarrow} D'_3$, *where* $(\mathbb{D}', l)$ *is self-referential. Then there exists* D'_4 *such that* $D'_2 \overset{(\mathbb{G},\tilde{R})/(\mathbb{D},l)}{\Longrightarrow} D'_4$ *and* $D'_3 \overset{(\mathbb{D},l)/(\mathbb{G},\tilde{R})}{\underset{\gamma}{\longrightarrow}{}^*} D'_4$. *If the sequence* $\overset{(\mathbb{D},l)/(\mathbb{G},\tilde{R})}{\underset{\gamma}{\longrightarrow}{}^*}$ *consists of more than one step, then for any order of steps in the reduction sequence we have* $D'_3 \overset{(\mathbb{D},l)/(\mathbb{G},\tilde{R})}{\underset{\gamma}{\longrightarrow}{}^*} D'_4$. □

Proof. Let $l, l_1, \tilde{l}$ be the labels of components where $(\mathbb{D}', l)$, $(\mathbb{D}, l)$, and $(\mathbb{G}, \tilde{R})$ occur, respectively. Note that $l \neq \tilde{l}$ by class preservation, since $(\mathbb{D}', l)$ is a self-referential redex, and therefore the component bound to l is a value. Also $l_1 \neq l$ by lemma 5.2.13, since $(M_1, F_1) \in dom(\gamma)$, and therefore F_1 contains at most one self-referential redex. We have the following cases:

1. $\tilde{R}$ is a term redex, suppose $\tilde{R} \rightsquigarrow_{\mathcal{T}} \tilde{Q}$. If $l_1 \neq \tilde{l}$, we have the following initial and final records:

$$[l \mapsto \lambda x.\mathbb{C}\{\underline{l}\}, l_1 \mapsto \mathbb{A}\{\underline{l}\}, \tilde{l} \mapsto \mathbb{E}\{\tilde{R}\}, \dots],$$
$$[l \mapsto \lambda x.\mathbb{C}\{\underline{l}\}, l_1 \mapsto \mathbb{A}\{\lambda x.\mathbb{C}\{l\}\}, \tilde{l} \mapsto \mathbb{E}\{\tilde{Q}\}, \dots].$$

   The self-referential redex has just one marked residual – itself.

   If $l_1 = \tilde{l}$, i.e. $\tilde{R}$ and the non-self-referential marked occurrence of l are in the same component, we have one of the following:

- $\tilde{R}$ and the marked occurrence of l are independent. This case is similar to the case when $l_1 \neq \tilde{l}$. By lemma 5.1.13 the residual of the evaluation redex is itself evaluation redex.

- The marked l occurs in the operator of $\tilde{R}$ ($\tilde{R}$ is an application since it contains l). Without loss of generality assume that $\underline{l}$ occurs in the first hole of the multi-hole context $\mathbb{A}$ below. The initial and the final records are:

$$[l \mapsto \lambda x.\mathbb{C}\{\underline{l}\}, \tilde{l} \mapsto \mathbb{E}\{(\lambda y.\mathbb{A}\{\underline{l}, y, \ldots, y\}) \mathbin{@} V\}, \ldots],$$
$$[l \mapsto \lambda x.\mathbb{C}\{\underline{l}\}, \tilde{l} \mapsto \mathbb{E}\{\mathbb{A}\{\lambda x.\mathbb{C}\{l\}, V, \ldots, V\}\}, \ldots].$$

  As before, the self-referential redex has a single marked residual – itself.

- The marked l occurs in the operand of $\tilde{R}$. The initial and the final records are:

$$[l \mapsto \lambda x.\mathbb{C}\{\underline{l}\}, \tilde{l} \mapsto \mathbb{E}\{(\lambda y.\mathbb{A}\{y, \ldots, y\}) \mathbin{@} \lambda z.\mathbb{B}\{\underline{l}\}\}, \ldots],$$
$$[l \mapsto \lambda x.\mathbb{C}\{\underline{l}\}, \tilde{l} \mapsto \mathbb{E}\{\mathbb{A}\{\lambda z.\mathbb{B}\{\lambda x.\mathbb{C}\{l\}\}, \ldots, \lambda z.\mathbb{B}\{\lambda x.\mathbb{C}\{l\}\}\}\}, \ldots].$$

  Here the non-self-referential marked occurrence of l gets duplicated by the term redex. The resulting record is the same, regardless of which redex (the marked substitution or the application) is performed first and regardless of the order in which the residuals of $(\mathbb{D}, l)$ w.r.t. $(\mathbb{G}, \tilde{R})$ are reduced. The self-referential redex does not have any marked residuals besides itself.

2. $\tilde{R} = l_2 \neq l$ is a substitution redex. Note that $\tilde{l} \neq l_2$, since $\tilde{R}$ is an evaluation redex. We have the following possibilities:

   - $l_1 \neq l_2$, $l_1 \neq \tilde{l}$. It is easy to check that the claim of the lemma holds in this case.

- $l_1 = l_2$. The reductions on both paths lead to the same record with the same residuals of the self-referential redex. The two residuals of $(\mathbb{D}, l)$ in the γ-development reduction can be reduced in any order.

$$
\begin{array}{ll}
[l \mapsto \lambda x.\mathbb{C}\{\underline{l}\}, \tilde{l} \mapsto \mathbb{E}\{l_1\}, l_1 \mapsto \lambda y.\mathbb{A}\{\underline{l}\}, \ldots] & \underset{\gamma}{\circ\!\!\rightarrow} \\
[l \mapsto \lambda x.\mathbb{C}\{\underline{l}\}, \tilde{l} \mapsto \mathbb{E}\{l_1\}, l_1 \mapsto \lambda y.\mathbb{A}\{\lambda x.\mathbb{C}\{l\}\}, \ldots] & \Rightarrow \\
[l \mapsto \lambda x.\mathbb{C}\{\underline{l}\}, \tilde{l} \mapsto \mathbb{E}\{\lambda y.\mathbb{A}\{\lambda x.\mathbb{C}\{l\}\}\}, l_1 \mapsto \lambda y.\mathbb{A}\{\lambda x.\mathbb{C}\{l\}\}, \ldots], & \\
[l \mapsto \lambda x.\mathbb{C}\{\underline{l}\}, \tilde{l} \mapsto \mathbb{E}\{l_1\}, l_1 \mapsto \lambda y.\mathbb{A}\{\underline{l}\}, \ldots] & \Rightarrow \\
[l \mapsto \lambda x.\mathbb{C}\{\underline{l}\}, \tilde{l} \mapsto \mathbb{E}\{\lambda y.\mathbb{A}\{\underline{l}\}\}, l_1 \mapsto \lambda y.\mathbb{A}\{\underline{l}\}, \ldots] & \underset{\gamma}{\longrightarrow}^{*} \\
[l \mapsto \lambda x.\mathbb{C}\{\underline{l}\}, \tilde{l} \mapsto \mathbb{E}\{\lambda y.\mathbb{A}\{\lambda x.\mathbb{C}\{l\}\}\}, l_1 \mapsto \lambda y.\mathbb{A}\{\lambda x.\mathbb{C}\{l\}\}, \ldots]. &
\end{array}
$$

- $l_1 = \tilde{l}$. In this case the initial and the final records are as shown below. By lemma 5.1.13 $\mathbb{A}\{\Box, \lambda x.\mathbb{C}\{l\}\} \in \mathbf{EvalContext}_{\mathcal{T}}$.

$$
\begin{array}{l}
[l \mapsto \lambda x.\mathbb{C}\{\underline{l}\}, \tilde{l} \mapsto \mathbb{A}\{l_2, \underline{l}\}, l_2 \mapsto V, \ldots], \\
[l \mapsto \lambda x.\mathbb{C}\{\underline{l}\}, \tilde{l} \mapsto \mathbb{A}\{V, \lambda x.\mathbb{C}\{l\}\}, l_2 \mapsto V, \ldots].
\end{array}
$$

3. $\tilde{R} = l$ is a substitution redex. This occurrence of l is not marked, since by the condition of the lemma only the other two redexes are marked.

   If $l_1 = \tilde{l}$, the initial and the final records are as follows:

$$
\begin{array}{l}
[l \mapsto \lambda x.\mathbb{C}\{\underline{l}\}, \tilde{l} \mapsto \mathbb{A}\{l, \underline{l}\}, \ldots], \\
[l \mapsto \lambda x.\mathbb{C}\{\underline{l}\}, \tilde{l} \mapsto \mathbb{A}\{\lambda x.\mathbb{C}\{\underline{l}\}, \lambda x.\mathbb{C}\{l\}\}, \ldots].
\end{array}
$$

   The residual of the self-referential redex obtained by the $\Rightarrow$ step is marked, since the marking is removed only by a development step that substitutes into a marked occurrence of the same label. Thus in this case the self-referential

redex has two marked residuals in the resulting record. It is easy to see that these residuals are the same on both reduction paths. As in the previous case, we have used lemma 5.1.13 to show that the residual of the evaluation redex is evaluation redex.

The case when $l_1 \neq \tilde{l}$ is similar to case when $l_1 = \tilde{l}$. In this case the self-referential redex also has two marked residuals.

□

The next case is when a self-referential redex gets reduced by the $\underset{\gamma}{\circ\!\!\longrightarrow}$ step:

Lemma 5.2.34. *Suppose* $marked(D'_1) = \{(\mathbb{D}, l), (\mathbb{D}', l)\}$. *Let* $D'_1 \overset{(\mathbb{D},l)}{\underset{\gamma}{\circ\!\!\longrightarrow}} D'_2$ *and* $D'_1, \overset{(\mathbb{G},\tilde{R})}{\Longrightarrow} D'_3$, *where* $(\mathbb{D}, l)$ *is self-referential. Then there exists* D'_4 *such that* $D'_2 \overset{(\mathbb{G},\tilde{R})/(\mathbb{D},l)}{\Longrightarrow} D'_4$ *and* $D'_3 \overset{(\mathbb{D},l)/(\mathbb{G},\tilde{R})}{\underset{\gamma}{\longrightarrow}{}^*} D'_4$. *The* γ*-development sequence* $\overset{(\mathbb{D},l)/(\mathbb{G},\tilde{R})}{\underset{\gamma}{\longrightarrow}{}^*}$ *either consists of one step, or of two steps, where the self-referential redex is reduced the last.* □

Proof. Let l, l', and $\tilde{l}$ denotes labels of the components where $(\mathbb{D}, l)$, $(\mathbb{D}', l)$, and $(\mathbb{G}, \tilde{R})$ occur, respectively. Similarly to the previous lemma 5.2.33, $l \neq l'$, $l \neq \tilde{l}$. As in the proof of the previous lemma, we have 3 cases:

1. $\tilde{R}$ is a term redex. Let $\tilde{R} \rightsquigarrow_{\mathcal{T}} \tilde{Q}$.

   Suppose $l' = \tilde{l}$. We have the following possibilities:

   - $\tilde{R}$ and the marked occurrence of l are independent in the component bound to $\tilde{l}$. The initial and the final records are:

$$[l \mapsto \lambda x.\mathbb{C}\{\underline{l}\}, \tilde{l} \mapsto \mathbb{A}\{\tilde{R}, \underline{l}\}, \ldots],$$
$$[l \mapsto \lambda x.\mathbb{C}\{\lambda x.\mathbb{C}\{l\}\}, \tilde{l} \mapsto \mathbb{A}\{\tilde{Q}, \underline{l}\}, \ldots].$$

Note that since the self-referential redex is reduced by a γ-development step, it does not have a residual, i.e. the occurrence of l after the substitution is unmarked.

- The marked l occurs in the operator of $\tilde{R}$. Below are the initial and the final records:

$$[l \mapsto \lambda x.\mathbb{C}\{\underline{l}\}, \tilde{l} \mapsto \mathbb{E}\{(\lambda y.\mathbb{A}\{\underline{l}, y, \ldots, y\}) \mathbin{@} V\}, \ldots],$$
$$[l \mapsto \lambda x.\mathbb{C}\{\lambda x.\mathbb{C}\{l\}\}, \tilde{l} \mapsto \mathbb{E}\{\mathbb{A}\{\underline{l}, V, \ldots, V\}\}, \ldots].$$

- The marked l occurs in the operand of $\tilde{R}$. In this case the marked non-self-referential redex gets duplicated. As in the previous cases, we show the initial and the final records:

$$[l \mapsto \lambda x.\mathbb{C}\{\underline{l}\}, \tilde{l} \mapsto \mathbb{E}\{(\lambda y.\mathbb{A}\{y, \ldots, y\}) \mathbin{@} \lambda z.\mathbb{B}\{\underline{l}\}\}, \ldots],$$
$$[l \mapsto \lambda x.\mathbb{C}\{\lambda x.\mathbb{C}\{l\}\}, \tilde{l} \mapsto \mathbb{E}\{\mathbb{A}\{\lambda z.\mathbb{B}\{\underline{l}\}, \ldots, \lambda z.\mathbb{B}\{\underline{l}\}\}\}, \ldots].$$

The case when $l' \neq \tilde{l}$ is analogous to the first of the subcases above.

Note that in all cases the resulting γ-development reduction is one-step.

2. $\tilde{R} = l_1 \neq l$ is a substitution redex. In this case $l_1 \neq \tilde{l}$, since an evaluation redex can not be self-referential. We have the following subcases:

   - $l_1 = l'$. The marked non-self-referential redex gets duplicated. The initial and the final records are:

$$[l \mapsto \lambda x.\mathbb{C}\{\underline{l}\}, \tilde{l} \mapsto \mathbb{E}\{l_1\}, l_1 \mapsto \lambda y.\mathbb{A}\{\underline{l}\}, \ldots],$$
$$[l \mapsto \lambda x.\mathbb{C}\{\lambda x.\mathbb{C}\{l\}\}, \tilde{l} \mapsto \mathbb{E}\{\lambda y.\mathbb{A}\{\underline{l}\}\}, l_1 \mapsto \lambda y.\mathbb{A}\{\underline{l}\}, \ldots].$$

- $l' = \tilde{l}$. The initial record in this case is

$$[l \mapsto \lambda x.\mathbb{C}\{\underline{l}\}, \tilde{l} \mapsto \mathbb{A}\{l_1, \underline{l}\}, l_1 \mapsto V, \ldots],$$

  and the claim clearly holds.

- $l' \neq l_1$, $l' \neq \tilde{l}$. All three redexes are independent, and the claim clearly holds.

In all three subcases the resulting γ-development sequence is one-step.

3. $\tilde{R} = l$ is a substitution redex. Note that this occurrence of l is unmarked, since by the condition of the lemma only the other two redexes are marked. In this case the self-referential redex gets duplicated.

   Suppose $l' \neq \tilde{l}$. The order in which its two residuals are reduced is important. It is easy to see that reducing them in the other order leads to a different record, i.e. to a record that does not satisfy the lemma. In this case we show both reduction sequences step-by-step:

$$\begin{aligned}
&[l \mapsto \lambda x.\mathbb{C}\{\underline{l}\}, \tilde{l} \mapsto \mathbb{E}\{l\}, l' \mapsto \mathbb{A}\{\underline{l}\}, \ldots] && \overset{\circ}{\underset{\gamma}{\longrightarrow}} \\
&[l \mapsto \lambda x.\mathbb{C}\{\lambda x.\mathbb{C}\{l\}\}, \tilde{l} \mapsto \mathbb{E}\{l\}, l' \mapsto \mathbb{A}\{\underline{l}\}, \ldots] && \Longrightarrow \\
&[l \mapsto \lambda x.\mathbb{C}\{\lambda x.\mathbb{C}\{l\}\}, \tilde{l} \mapsto \mathbb{E}\{\lambda x.\mathbb{C}\{\lambda x.\mathbb{C}\{l\}\}\}, l' \mapsto \mathbb{A}\{\underline{l}\}, \ldots]. && \\
&[l \mapsto \lambda x.\mathbb{C}\{\underline{l}\}, \tilde{l} \mapsto \mathbb{E}\{l\}, l' \mapsto \mathbb{A}\{\underline{l}\}, \ldots] && \Longrightarrow \\
&[l \mapsto \lambda x.\mathbb{C}\{\underline{l}\}, \tilde{l} \mapsto \mathbb{E}\{\lambda x.\mathbb{C}\{\underline{l}\}\}, l' \mapsto \mathbb{A}\{\underline{l}\}, \ldots] && \overset{\circ}{\underset{\gamma}{\longrightarrow}} \\
&[l \mapsto \lambda x.\mathbb{C}\{\underline{l}\}, \tilde{l} \mapsto \mathbb{E}\{\lambda x.\mathbb{C}\{\lambda x.\mathbb{C}\{l\}\}\}, l' \mapsto \mathbb{A}\{\underline{l}\}, \ldots] && \overset{\circ}{\underset{\gamma}{\longrightarrow}} \\
&[l \mapsto \lambda x.\mathbb{C}\{\lambda x.\mathbb{C}\{l\}\}, \tilde{l} \mapsto \mathbb{E}\{\lambda x.\mathbb{C}\{\lambda x.\mathbb{C}\{l\}\}\}, l' \mapsto \mathbb{A}\{\underline{l}\}, \ldots]. &&
\end{aligned}$$

   The case when $l' = \tilde{l}$, i.e. when the marked non-self-referential redex occurs

in the component bound to $\tilde{l}$, is completely analogous to the case when $l' \neq \tilde{l}$. The resulting $\underset{\gamma}{\longrightarrow}^*$ also consists of two steps such that the second one reduces the self-referential redex. The initial record in this case is:

$$[l \mapsto \lambda x.\mathbb{C}\{\underline{l}\}, \tilde{l} \mapsto \mathbb{A}\{l, \underline{l}\}, \ldots].$$

□

Lemma 5.2.35. *Suppose marked*$(D'_1) = \{(\mathbb{D}, R), (\mathbb{D}', R'), (\mathbb{G}, \tilde{R})\}$. *Let* $D'_1 \overset{(\mathbb{D},R)}{\underset{\gamma}{\circ\!\!\longrightarrow}} D'_2$ *and* $D'_1 \overset{(\mathbb{G},\tilde{R})}{\Longrightarrow} D'_3$. *Then there exists* D'_4 *such that* $D'_2 \overset{(\mathbb{G},\tilde{R})/(\mathbb{D},R)}{\Longrightarrow} D'_4$ *and* $D'_3 \overset{(\mathbb{D},R)/(\mathbb{G},\tilde{R})}{\underset{\gamma}{\longrightarrow}^*} D'_4$. *If* $(\mathbb{D}, R)$ *is a self-referential substitution redex, then the sequence* $\overset{(\mathbb{D},R)/(\mathbb{G},\tilde{R})}{\underset{\gamma}{\longrightarrow}^*}$ *consists of two steps such that the last step reduces the self-referential redex. Otherwise the* γ*-development sequence is either one-step, or the order of reduction in it does not matter.* □

Proof. By lemma 5.2.13 we have two cases:

1. All three marked redexes are term redexes. The reduction of a term redex does not depend on the marking of any other redex in the record. It is also the case that for a term redex $(\mathbb{D}, R)/(\mathbb{D}, R) = \emptyset$. Therefore the fact that $(\mathbb{G}, \tilde{R})$ is marked does not affect the reduction, and the case is analogous to case 1 of lemma 5.2.32. If the resulting γ-development sequence is multi-step, then the order of reduction of the redexes does not matter.

2. $R = R' = \tilde{R} = l$. Let l_1, l', and $\tilde{l}$ be the labels of components where the redexes R, R', and $\tilde{R}$ occur, respectively. $l \neq \tilde{l}$, since $(\mathbb{G}, \tilde{R})$ is an evaluation redex. We have the following subcases:

(a) $l_1 \neq l$, $l' \neq l$, i.e. none of the redexes is self-referential. Suppose that $l' = l_1 = \tilde{l}$, i.e. all the redexes occur in the same component. Assuming, without loss of generality, that context $\mathbb{A}$ contains them in the order $\tilde{R}, R, R'$, we have the following initial and final records:

$$[\tilde{l} \mapsto \mathbb{A}\{\underline{l}, \underline{l}, \underline{l}\}, l \mapsto V, \dots],$$
$$[\tilde{l} \mapsto \mathbb{A}\{V, V, \underline{l}\}, l \mapsto V, \dots].$$

We use lemma 5.1.13 to show that after the $\underset{\gamma}{\circ\!\!\longrightarrow}$ step the residual of $(\mathbb{G}, \tilde{R})$ is an evaluation redex. The cases when the three redexes occur in different components are similar.

(b) $l' = l$, i.e. the redex $(\mathbb{D}', R')$ is self-referential. Assuming that the other two redexes occur in the same component, we have the following reductions:

$$\begin{aligned}
&[l \mapsto \lambda x.\mathbb{C}\{\underline{l}\}, \tilde{l} \mapsto \mathbb{A}\{\underline{l}, \underline{l}\}, \dots] && \underset{\gamma}{\circ\!\!\longrightarrow} \\
&[l \mapsto \lambda x.\mathbb{C}\{\underline{l}\}, \tilde{l} \mapsto \mathbb{A}\{\underline{l}, \lambda x.\mathbb{C}\{l\}\}, \dots] && \Longrightarrow \\
&[l \mapsto \lambda x.\mathbb{C}\{\underline{l}\}, \tilde{l} \mapsto \mathbb{A}\{\lambda x.\mathbb{C}\{\underline{l}\}, \lambda x.\mathbb{C}\{l\}\}, \dots], && \\
&[l \mapsto \lambda x.\mathbb{C}\{\underline{l}\}, \tilde{l} \mapsto \mathbb{A}\{\underline{l}, \underline{l}\}, \dots] && \Longrightarrow \\
&[l \mapsto \lambda x.\mathbb{C}\{\underline{l}\}, \tilde{l} \mapsto \mathbb{A}\{\lambda x.\mathbb{C}\{\underline{l}\}, \underline{l}\}, \dots] && \underset{\gamma}{\longrightarrow} \\
&[l \mapsto \lambda x.\mathbb{C}\{\underline{l}\}, \tilde{l} \mapsto \mathbb{A}\{\lambda x.\mathbb{C}\{\underline{l}\}, \lambda x.\mathbb{C}\{l\}\}, \dots]. &&
\end{aligned}$$

Here the self-referential redex has two marked residuals. The case when the two non-self-referential redexes occur in different components is analogous.

(c) $l_1 = l$, i.e. $(\mathbb{D}, R)$ is self-referential. Again we show the case when the other two redexes occur in the same component. The following reductions

prove the claim of the lemma.

$$
\begin{aligned}
&[l \mapsto \lambda x.\mathbb{C}\{\underline{l}\}, \tilde{l} \mapsto \mathbb{A}\{\underline{l}, \underline{l}\}, \ldots] && \overset{\circ}{\underset{\gamma}{\longrightarrow}} \\
&[l \mapsto \lambda x.\mathbb{C}\{\lambda x.\mathbb{C}\{l\}\}, \tilde{l} \mapsto \mathbb{A}\{\underline{l}, \underline{l}\}, \ldots] && \Longrightarrow \\
&[l \mapsto \lambda x.\mathbb{C}\{\lambda x.\mathbb{C}\{l\}\}, \tilde{l} \mapsto \mathbb{A}\{\lambda x.\mathbb{C}\{\lambda x.\mathbb{C}\{l\}\}, \underline{l}\}, \ldots], \\
&[l \mapsto \lambda x.\mathbb{C}\{\underline{l}\}, \tilde{l} \mapsto \mathbb{A}\{\underline{l}, \underline{l}\}, \ldots] && \Longrightarrow \\
&[l \mapsto \lambda x.\mathbb{C}\{\underline{l}\}, \tilde{l} \mapsto \mathbb{A}\{\lambda x.\mathbb{C}\{\underline{l}\}, \underline{l}\}, \ldots] && \overset{\circ}{\underset{\gamma}{\longrightarrow}} \\
&[l \mapsto \lambda x.\mathbb{C}\{\underline{l}\}, \tilde{l} \mapsto \mathbb{A}\{\lambda x.\mathbb{C}\{\lambda x.\mathbb{C}\{l\}\}, \underline{l}\}, \ldots] && \overset{\circ}{\underset{\gamma}{\longrightarrow}} \\
&[l \mapsto \lambda x.\mathbb{C}\{\lambda x.\mathbb{C}\{l\}\}, \tilde{l} \mapsto \mathbb{A}\{\lambda x.\mathbb{C}\{\lambda x.\mathbb{C}\{l\}\}, \underline{l}\}, \ldots]. &&
\end{aligned}
$$

Here, as in case 3 of lemma 5.2.34, the order in which the two resulting γ-development steps are performed matters: switching the order will result in a reduction that does not satisfy the lemma.

Note that in cases (a) and (b) the γ-development reduction is one-step, and in (c) the self-referential redex is reduced last.

□

Lemma 5.2.36 (Elementary Project Diagram for $\mathcal{C}$). *If* $D_1' \overset{\circ}{\underset{\gamma}{\xrightarrow{(\mathbb{D},R)}}} D_2'$ *and* $D_1' \xrightarrow{(\mathbb{G},\tilde{R})} D_3'$*, then there exists* D_4' *such that* $D_2' \xrightarrow{(\mathbb{G},\tilde{R})/(\mathbb{D},R)} D_4'$ *and* $D_3' \overset{(\mathbb{D},R)/(\mathbb{G},\tilde{R})}{\underset{\gamma}{\longrightarrow}{}^*} D_4'$. □

Proof. If marked(D_1') does not have a self-referential redex, then the lemma follows from lemma 5.2.32 if $\tilde{R}$ is not marked and from the cases 1 or 2(a) of lemma 5.2.35 if $\tilde{R}$ is marked by induction on the number of redexes in marked(D_1'). The proof is analogous to the proof of lemma 5.2.22.

Suppose marked(D_1') has a self-referential redex, but $(\mathbb{D}, R)$ is not self-referential. Let $(\mathbb{D}', R')$ be the self-referential redex in marked(D_1'). By lemma 5.2.33 or part

2(b) of lemma 5.2.35 the claim of the lemma holds for D_1'' such that $|D_1''| = |D_1'|$ and marked$(D_1'') = \{(\mathbb{D}, R), (\mathbb{D}', R')\})$ or marked$(D_1'') = \{(\mathbb{D}, R), (\mathbb{D}', R'), (\mathbb{G}, \tilde{R})\})$ when $(\mathbb{G}, \tilde{R})$ is marked. The rest of the redexes in marked(D_1') are not self-referential, since $D_1' \in dom(\gamma)$. Let $(\mathbb{D}'', R'') \neq (\mathbb{D}, R)$ be another redex in marked(D_1'). Note that it is a substitution redex by lemma 5.2.13. Then by lemma 5.2.32 or by case 2(a) of lemma 5.2.35 the claim of the lemma holds for D_1'' (in both cases). Then by corollary 5.2.25 for the $\underset{\gamma}{\rightarrow}$ steps and by lemma 5.2.23 for the $\Longrightarrow$ steps we can combine the diagrams for individual redexes in marked(D_1') into a diagram for D_1'. We use the fact that, since a self-referential redex is not reduced in any of the reductions, the redexes in the sequence $\overset{(\mathbb{D},R)/(\mathbb{G},\tilde{R})}{\underset{\gamma}{\longrightarrow}{}^*}$ can be reduced in any order, so we can assume that for all individual diagrams they are reduced in the same fixed order.

Suppose now that $(\mathbb{D}, R)$ is a self-referential redex. Then by lemma 5.2.34 or by case 2(c) of lemma 5.2.35 the lemma holds for D_1'' (again, for both cases). Then, as in the previous case, we combine the individual diagrams into the diagram for D_1'. Note that in this case the order of reduction of the sequence $\overset{(\mathbb{D},R)/(\mathbb{G},\tilde{R})}{\underset{\gamma}{\longrightarrow}{}^*}$ is fixed (the self-referential redex in this sequence is reduced last). □

Now we show the dual property 4.6.2 (elementary project diagram).

Before we can prove it, let us show some properties of substitution which we are going to use in the proofs. The essence of these properties is that a non-evaluation substitution step can not create an evaluation redex (either a term, or a substitution redex) that has not existed in the original term. The property is the converse of the one stated in lemma 5.1.13: the lemma below explicitly restricts the substitution to be a non-evaluation step, however one may notice that the only substitution possible into a term of the form $\mathbb{E}\{R\}$ containing a label is a non-evaluation substitution, since such a label may appear only in a non-evaluation context.

Lemma 5.2.37. *1. If* $\mathbb{C} \notin \mathbf{EvalContext}_{\mathcal{T}}$ *and* $\mathbb{C}\{V\}$ *is a term redex, then* $\mathbb{C}\{l\}$ *is a term redex.*

2. If $\mathbb{A}\{V, \Box\} \in \mathbf{EvalContext}_{\mathcal{T}}$ *and* $\mathbb{A}\{\Box, R\} \notin \mathbf{EvalContext}_{\mathcal{T}}$ *(or, respectively,* $\mathbb{A}\{\Box, V\} \in \mathbf{EvalContext}_{\mathcal{T}}$ *and* $\mathbb{A}\{R, \Box\} \notin \mathbf{EvalContext}_{\mathcal{T}}$*), then* $\mathbb{A}\{l, \Box\} \in \mathbf{EvalContext}_{\mathcal{T}}$ *(respectively,* $\mathbb{A}\{\Box, l\} \in \mathbf{EvalContext}_{\mathcal{T}}$*).*

3. If $\mathbb{A}\{V, \Box\} \in \mathbf{EvalContext}_{\mathcal{T}}$ *and* $\mathbb{A}\{\Box, l_1\} \notin \mathbf{EvalContext}_{\mathcal{T}}$ *(or, respectively,* $\mathbb{A}\{\Box, V\} \in \mathbf{EvalContext}_{\mathcal{T}}$ *and* $\mathbb{A}\{l_1, \Box\} \notin \mathbf{EvalContext}_{\mathcal{T}}$*), then* $\mathbb{A}\{l, \Box\} \in \mathbf{EvalContext}_{\mathcal{T}}$ *(respectively,* $\mathbb{A}\{\Box, l\} \in \mathbf{EvalContext}_{\mathcal{T}}$*), where* l_1 *may or may not be the same as* l*.*

□

Proof. 1. A term redex R is either c_1 *op* c_2, where both constants occur in evaluation context, so $R \neq \mathbb{C}\{V\}$, where $\mathbb{C} \notin \mathbf{EvalContext}_{\mathcal{T}}$, or $\lambda x.M \ @ \ V$, where the only non-evaluation contexts occur inside V or inside M, so the claim clearly holds.

2. By induction on the structure of an evaluation context.

3. Same as the previous case.

□

Similarly to the proof of the property 4.6.1 (lemma 5.2.36 above), we first show the property for 4 different cases of kinds, positions, and markings of the redexes. The following 4 lemmas correspond to these cases.

Remark 5.2.38. In the proofs of the auxiliary lemmas for elementary project diagrams (lemmas 5.2.32–5.2.35) we have used the notion of independent redexes (definition 5.2.5), in particular some of the cases considered in the proofs where the

cases when the evaluation redex $(\mathbb{G}, \tilde{R})$ was independent from the non-evaluation redex $(\mathbb{D}, R)$ (see lemma 5.2.36). This definition is not applicable in the case of lemma 5.2.43 (elementary lift diagram), since the non-evaluation redex occurs in the initial record D'_1, but the evaluation redex occurs in the record D'_2 obtained from D'_1 by reducing the non-evaluation redex. We extend the notion of independent redexes to this case as follows: we say that $(\mathbb{G}, \tilde{R})$ and $(\mathbb{D}, R)$ are *independent* if $(\mathbb{G}, \tilde{R})$ is independent from $(\mathbb{D}, Q)$ in D'_2 (as a subterm, see definition 5.2.5), where $R \rightsquigarrow Q$ if R is a term redex, and $Q = V$ if $R = l$ is a substitution redex. The notion is well-defined, since $(\mathbb{G}, \tilde{R})$ is an evaluation redex, and therefore can not be contained in the value being substituted.

We also say that $(\mathbb{G}, \tilde{R})$ is independent from $(\mathbb{D}', R')$ if it is independent from its residual(s) in D_2. □

Lemma 5.2.39. *Suppose* $(\mathbb{D}, R)$ *and* $(\mathbb{D}', R')$ *are not self-referential, marked*$(D'_1) = \{(\mathbb{D}, R), (\mathbb{D}', R')\}$, *and* $D'_1 \underset{\gamma}{\overset{(\mathbb{D},R)}{\circ\!\!\longrightarrow}} D'_2 \overset{(\mathbb{G},\tilde{R})}{\Longrightarrow} D'_4$, *then there exists* D'_3 *such that* $D'_1, \overset{(\mathbb{G}',\tilde{R}')}{\Longrightarrow} D'_3 \overset{(\mathbb{D},R)/(\mathbb{G}',\tilde{R}')}{\underset{\gamma}{\longrightarrow}{}^*} D'_4$ *such that* $(\mathbb{G}, \tilde{R}) = (\mathbb{G}', \tilde{R}')/(\mathbb{D}, R)$. *If the sequence* $\overset{(\mathbb{D},R)/(\mathbb{G}',\tilde{R}')}{\underset{\gamma}{\longrightarrow}{}^*}$ *consists of more than one step, then for any such reduction* D'_3, $\overset{(\mathbb{D},R)/(\mathbb{G}',\tilde{R}')}{\underset{\gamma}{\longrightarrow}{}^*}$ D'_4. □

Proof. Since $D'_1 \in dom(\gamma)$, the two marked redexes are of the same kind, and if they are substitution redexes, then $R = R' = l_1$. Let $l, l', \tilde{l}$ be the labels of the components containing $(\mathbb{D}, R)$, $(\mathbb{D}', R')$, and $(\mathbb{G}, \tilde{R})$, respectively. We have the following cases:

1. All three redexes are term redexes. If all three redexes occur in different terms, the claim of the lemma clearly holds. If the three redexes all occur in the same term, then the claim holds by lemma 5.1.25. If $l = \tilde{l} \neq l'$, then the claim holds by lemma 5.1.22 (clearly $(\mathbb{D}', R')$, which occurs not in the same component

where the other two redexes occur, has a single residual on both reduction paths). If $l = l' \neq \tilde{l}$ or $\tilde{l} = l' \neq l$, then $(\mathbb{D}, R)$ and $(\mathbb{G}, \tilde{R})$ are independent (see remark 5.2.38), and clearly the claim of the lemma holds.

2. The marked redexes are term redexes, and the evaluation redex is a substitution redex. Let $\tilde{R} = l_1$. Since $(\mathbb{G}, l_1)$ is an evaluation redex, $l_1 \neq \tilde{l}$. We have the following subcases:

   (a) Among the 4 labels l, l', l_1, and $\tilde{l}$, no two are equal. Then the claim of the lemma clearly holds.

   (b) $l_1 \notin \{l, l'\}$. Then

   $$D_2' = [\tilde{l} \mapsto \mathbb{E}\{l_1\}, l_1 \mapsto V, \ldots].$$

   By lemma 5.1.21

   $$D_1' = [\tilde{l} \mapsto \mathbb{E}'\{l_1\}, l_1 \mapsto V, \ldots].$$

   Suppose $l = l' = \tilde{l}$, i.e. both marked redexes occur in the same component as $(\mathbb{G}, l_1)$ and the marked redexes are independent from each other. By lemma 5.2.31 the substitution occurrence of l_1 is not contained in R or R', i.e. $\mathbb{E}'\{l_1\} = \mathbb{A}\{l_1, R, R'\}$. Recall that in the proofs in this section we write $\mathbb{A}\{l_1, R, R'\}$ to mean that the context $\mathbb{A}$ contains the three terms in some, not necessarily the specified, order. Then the following reductions prove the claim of the lemma:

   $$[\tilde{l} \mapsto \mathbb{A}\{l_1, R, R'\}, l_1 \mapsto V, \ldots] \implies [\tilde{l} \mapsto \mathbb{A}\{V, R, R'\}, l_1 \mapsto V, \ldots] \underset{\gamma}{\rightarrow}$$
   $$[\tilde{l} \mapsto \mathbb{A}\{V, Q, R'\}, l_1 \mapsto V, \ldots],$$
   $$[\tilde{l} \mapsto \mathbb{A}\{l_1, R, R'\}, l_1 \mapsto V, \ldots] \underset{\gamma}{\circ\!\!\rightarrow} [\tilde{l} \mapsto \mathbb{A}\{l_1, Q, R'\}, l_1 \mapsto V, \ldots] \implies$$
   $$[\tilde{l} \mapsto \mathbb{A}\{V, Q, R'\}, l_1 \mapsto V, \ldots].$$

The cases when $\tilde{l} = l = l'$ and R contains R' or R' contains R are similar. The cases when $l = \tilde{l}, l' \neq \tilde{l}$, or $l' = \tilde{l}, l \neq \tilde{l}$, or $l = l' \neq \tilde{l}$ are also similar, but simpler.

(c) $\tilde{l} \notin \{l, l'\}$. Note that the case when $l = l' \neq l_1$ has been considered above. The remaining subcases are $l = l' = l_1$, $l = l_1 \neq l'$, and $l' = l_1 \neq l$. As above, we consider the case $l = l' = l_1$ in detail, the other two cases are similar, but simpler. We have the following possibilities:

I. R, R' are independent in D_1. In this case:

$$D'_2 = [\tilde{l} \mapsto \mathbb{E}\{l_1\}, l_1 \mapsto \lambda y.\mathbb{A}\{Q, R'\},$$

where $R \rightsquigarrow_{\mathcal{T}} Q$. Then

$$D'_1 = [\tilde{l} \mapsto \mathbb{E}\{l_1\}, l_1 \mapsto \lambda y.\mathbb{A}\{R, R'\},$$

and the reduction in case 2(c)I of the proof of lemma 5.2.32 proves the claim. Note that the component bound to l_1 in D'_1 is a value, therefore the substitution is possible.

II. R contains R'. Then

$$D_2 = [\tilde{l} \mapsto \mathbb{E}\{l_1\}, l_1 \mapsto \lambda y.\mathbb{A}\{Q\},$$

where Q contains R', possibly several copies if R' is contained in the operand of R, and therefore is duplicated. Again the reduction in the respective case (case 2(c)II) of lemma 5.2.32 proves the claim.

III. R' contains R. The case is similar to the previous case and to case

2(c)III of lemma 5.2.32.

(d) $l = \tilde{l}$, $l' = l_1$. As in case (a) of this proof, the substitution redex l_1 in D_2 is independent from the result of reduction of R by lemma 5.2.31. Therefore

$$D_1' = [\tilde{l} \mapsto \mathbb{A}\{l_1, R\}, l_1 \mapsto \lambda x.\mathbb{B}\{R'\}],$$

and the claim follows by the reduction.

(e) $l' = \tilde{l}$, $l = l_1$. Then

$$D_1' = [\tilde{l} \mapsto \mathbb{A}\{l_1, R'\}, l_1 \mapsto \lambda x.\mathbb{B}\{R\}].$$

Note that since l_1 is bound to a value containing the result of reduction of a non-evaluation redex, this component is a λ-abstraction in D_2', and therefore in D_1'.

3. The marked redexes are substitution redexes, and the evaluation redex is a term redex. Let $R = R' = l_1$ (since the initial record is in $dom(\gamma)$, it must be the case that R and R' are the same label). By the condition of the lemma none of the redexes is self-referential, therefore $l \neq l_1$, $l' \neq l_1$. It also must be the case that $l_1 \neq \tilde{l}$, since l_1 is bound to a value in D_1', and therefore in D_2', and $\tilde{l}$ is bound to an evaluatable term in D_2'. As in the previous case, let us assume that $l = l' = \tilde{l}$. There are the following possibilities:

- The evaluation redex is independent from both substitution redexes. Then

$$D_2' = [\tilde{l} \mapsto \mathbb{A}\{\tilde{R}, V, \underline{l_1}\}, l_1 \mapsto V, \dots].$$

By lemma 5.2.37 $\mathbb{A}\{\Box, \underline{l_1}, \underline{l_1}\} \in \mathbf{EvalContext}_{\mathcal{T}}$, and it is easy to check

that the claim of the lemma holds. No duplication of redexes occurs in this case.

- Both the marked occurrence of l_1 of the redex R' and the value V which is the result of the non-evaluation substitution redex occur in the operand of $\tilde{R}$. Then

$$D_2' = [\tilde{l} \mapsto \mathbb{E}\{(\lambda x.\mathbb{A}\{x, \ldots, x\}) \mathbin{@} \lambda y.\mathbb{B}\{V, \underline{l_1}\}\}, l_1 \mapsto V, \ldots].$$

  The occurrence of V here is under a λ, i.e. in a non-evaluation context, and therefore by $(\lambda x.\mathbb{A}\{x, \ldots, x\}) \mathbin{@} \lambda y.\mathbb{B}\{\underline{l_1}, \underline{l_1}\}$ is a redex by part 1 of lemma 5.2.37. The rest of the claim follows by the reduction in the proof of the analogous case of lemma 5.2.32.

- The remaining cases (both R and R' occur in the operator part of $\tilde{R}$, or when one of the two marked redexes occurs in the operand, and the other in the operator) are similar.

The other cases ($l = l' \neq \tilde{l}$, $l = \tilde{l} \neq l'$, $l' = \tilde{l} \neq l$, and the case when all the three labels are distinct) are similar.

4. All three redexes are substitution redexes, where $\tilde{R} = l_2 \neq l_1$. In this case $\tilde{l} \neq l_2$, since $\tilde{R}$ is an evaluation redex. As in case 3, $l_1 \notin \{l, l', \tilde{l}\}$. As in the proof of lemma 5.2.32, we have the following subcases:

   (a) All 5 labels are distinct, then the lemma trivially holds.

   (b) $l = l' = \tilde{l}$. Then

$$D_2' = [\tilde{l} \mapsto \mathbb{A}\{l_2, V_1, \underline{l_1}\}, l_1 \mapsto V_1, l_2 \mapsto V_2, \ldots].$$

The label l_2 occurs independently from V_1 in the component bound to $\tilde{l}$, since l_2 must occur in an evaluation context, and therefore can not occur inside a value[4]. By lemma 5.2.37 $\mathbb{A}\{l_2, \underline{l_1}, \underline{l_1}\} \in \mathbf{EvalContext}_{\mathcal{T}}$, and the claim of the lemma holds.

(c) $l = l' = l_2$. Then

$$D_2' = [\tilde{l} \mapsto \mathbb{E}\{l_2\}, l_2 \mapsto \lambda x.\mathbb{C}\{V_1, \underline{l_1}\}, l_1 \mapsto V_1, \ldots].$$

In this case

$$D_1' = [\tilde{l} \mapsto \mathbb{E}\{l_2\}, l_2 \mapsto \lambda x.\mathbb{C}\{\underline{l_1}, \underline{l_1}\}, l_1 \mapsto V_1, \ldots],$$

and the claim clearly holds.

(d) $l = l'$, $l \neq \tilde{l}$, $l \neq l_2$. The two marked redexes are independent from both the evaluation redex and the value being substituted into it. The claim of the lemma clearly holds.

(e) $l = \tilde{l}$, $l' = l_2$. Then

$$D_2' = [\tilde{l} \mapsto \mathbb{A}\{l_2, V_1\}, l_2 \mapsto \lambda x.\mathbb{B}\{\underline{l_1}\}, l_1 \mapsto V_1, \ldots].$$

Similarly to the case (b), l_2 is independent from V_1 in the component bound to $\tilde{l}$. By lemma 5.2.37 $\mathbb{A}\{\Box, \underline{l_1}\} \in \mathbf{EvalContext}_{\mathcal{T}}$, and again it is easy to check that the claim holds.

(f) $l = \tilde{l}$, $l' \neq l_2$, $l' \neq \tilde{l}$. Similar to the previous case.

[4]Note that even if labels were values, V_1 still could not have been equal to l_2, since the substitution $D_1' \circ\!\!\rightarrow D_2'$ is a non-evaluation step, but l_2 must occur in an evaluation context in D_2'.

(g) $l = l_2$, $l' = \tilde{l}$. In this case

$$D'_2 = [\tilde{l} \mapsto \mathbb{A}\{l_2, \underline{l_1}\}, l_2 \mapsto \lambda x.\mathbb{B}\{V_1\}, l_1 \mapsto V_1, \ldots].$$

Again, the claim of the lemma clearly holds.

(h) $l = l_2$, $l' \neq l_2$, $l' \neq \tilde{l}$. Similar to the previous case.

5. All the three redexes are substitution redexes, $R = R' = \tilde{R} = l_1$. $\tilde{R}$ can not be marked by the condition of the lemma (only the other two redexes are marked). Similarly to the previous case, $l_1 \notin \{l, l', \tilde{l}\}$. Therefore no redex duplication is possible, and the claim of the lemma holds in this case.

□

Lemma 5.2.40. *Let $(\mathbb{D}', R')$ be a self-referential redex. Suppose marked$(D'_1) = \{(\mathbb{D}, R), (\mathbb{D}', R')\}$. If $D'_1 \overset{(\mathbb{D},R)}{\underset{\gamma}{\circ\!\!\longrightarrow}} D'_2 \overset{(\mathbb{G},\tilde{R})}{\Longrightarrow} D'_4$, then there exists D'_3 such that $D'_1 \overset{(\mathbb{G}',\tilde{R}')}{\Longrightarrow} D'_3 \overset{(\mathbb{D},R)/(\mathbb{G}',\tilde{R}')}{\underset{\gamma}{\longrightarrow}{}^*} D'_4$ such that $(\mathbb{G}, \tilde{R}) = (\mathbb{G}', \tilde{R}')/(\mathbb{D}, R)$. If the sequence $\overset{(\mathbb{D},R)/(\mathbb{G}',\tilde{R}')}{\underset{\gamma}{\longrightarrow}{}^*}$ consists of more than one step, then for any such reduction $D'_3, \overset{(\mathbb{D},R)/(\mathbb{G}',\tilde{R}')}{\underset{\gamma}{\longrightarrow}{}^*} D'_4$.* □

Proof. Let $l, l_1, \tilde{l}$ be the labels of the components where $(\mathbb{D}', l)$, $(\mathbb{D}, l)$, and $(\mathbb{G}', \tilde{R}')$ occur, respectively. Note that $\tilde{l} \neq l$, since $(\mathbb{G}', \tilde{R}')$ is an evaluation redex, and $l_1 \neq l$, since there may be at most one marked self-referential redex. We have the following subcases:

1. $\tilde{R}$ is a term redex. If $l_1 \neq \tilde{l}$, then the evaluation redex is independent from both substitution redexes, and the claim of the lemma clearly holds. If $l_1 = \tilde{l}$, then we have one of the following:

   - $\tilde{R}$ is independent from the marked occurrence of l. The case is similar to

the case when $l_1 \neq \tilde{l}$. By lemma 5.2.37 $\tilde{R}$ occurs in an evaluation context in D'_1.

- The result of the substitution of l occurs in $\tilde{R}$.

  The case when $\tilde{R}$ is of the form $c_1\ op\ c_2$ is impossible, since then either c_1 or c_2 must be the result of the substitution, but the component bound to l is a λ-abstraction, not a constant, since it contains an occurrence of l.

  If $\tilde{R}$ is an application and the result of the substitution occurs in its operator, then we have

$$D'_2 = [l \mapsto \lambda x.\mathbb{C}\{\underline{l}\}, \tilde{l} \mapsto \mathbb{E}\{(\lambda y.\mathbb{A}\{\lambda x.\mathbb{C}\{l\}, y, \ldots, y\}) \ @\ V\}, \ldots],$$
$$D'_1 = [l \mapsto \lambda x.\mathbb{C}\{\underline{l}\}, \tilde{l} \mapsto \mathbb{E}\{(\lambda y.\mathbb{A}\{\underline{l}, y, \ldots, y\}) \ @\ V\}, \ldots].$$

  Note that in l occurs unmarked in the component bound to $\tilde{l}$ in D_2, since the non-evaluation step is a γ-development . The claim easily follows, similarly to the respective case of lemma 5.2.33.

  If $\tilde{R}$ is an application and the result of the substitution occurs in its operand, then

$$D'_2 = [l \mapsto \lambda x.\mathbb{C}\{\underline{l}\}, \tilde{l} \mapsto \mathbb{E}\{(\lambda y.\mathbb{A}\{y, \ldots, y\}) \ @\ \lambda z.\mathbb{B}\{\lambda x.\mathbb{C}\{l\}\}\}, \ldots],$$
$$D'_1 = [l \mapsto \lambda x.\mathbb{C}\{\underline{l}\}, \tilde{l} \mapsto \mathbb{E}\{(\lambda y.\mathbb{A}\{y, \ldots, y\}) \ @\ \lambda z.\mathbb{B}\{\underline{l}\}\}, \ldots].$$

  The claim easily follows.

2. $\tilde{R} = l_2 \neq l$ is a substitution redex. $\tilde{R}$ is an evaluation redex, therefore $\tilde{l} \neq l_2$. We have the following possibilities:

   - $l_1 \neq l_2, l_1 \neq \tilde{l}$. In this case the three redexes occur in different components, and the claim of the lemma clearly holds.

- $l_1 = l_2$, i.e. the non-evaluation redex occurs in the value being substituted by the evaluation step. In this case

$$D'_2 = [l \mapsto \lambda x.\mathbb{C}\{\underline{l}\}, \tilde{l} \mapsto \mathbb{E}\{l_1\}, l_1 \mapsto \mathbb{A}\{\lambda x.\mathbb{C}\{l\}\}, \dots].$$

  Then $\mathbb{A} \neq \Box$, since the γ-development step is a non-evaluation step. Note that l_1 must be bound to a value for $\tilde{R}$ to be a redex in D_2, so $\mathbb{A} = \lambda y.\mathbb{B}$, and

$$D'_1 = [l \mapsto \lambda x.\mathbb{C}\{\underline{l}\}, \tilde{l} \mapsto \mathbb{E}\{l_1\}, l_1 \mapsto \lambda y.\mathbb{B}\{\underline{l}\}, \dots].$$

  The rest of the claim follows from the reduction in the respective case of lemma 5.2.33.

- $l_1 = \tilde{l}$. Then

$$D'_2 = [l \mapsto \lambda x.\mathbb{C}\{\underline{l}\}, \tilde{l} \mapsto \mathbb{A}\{l_2, \lambda x.\mathbb{C}\{l\}\}, \dots].$$

  Similarly to the case 4(b) of lemma 5.2.39, the label l_2 occurs independently from $\lambda x.\mathbb{C}\{l\}$ in the component bound to $\tilde{l}$, since l_2 must occur in an evaluation context, and therefore can not occur under a λ. By lemma 5.2.37 $\mathbb{A}\{\Box, \underline{l}\} \in \mathbf{EvalContext}_{\mathcal{T}}$. The rest of the claim easily follows.

3. $\tilde{R} = l$ is a substitution redex. This occurrence of l is not marked, since by the condition of the lemma only the other two redexes are marked.

   If $l_1 = \tilde{l}$, then

$$D'_2 = [l \mapsto \lambda x.\mathbb{C}\{\underline{l}\}, \tilde{l} \mapsto \mathbb{A}\{l, \lambda x.\mathbb{C}\{l\}\}, \dots].$$

Similarly to the previous case, l is independent from $\lambda x.\mathbb{C}\{\underline{l}\}$ in the component bound to $\tilde{l}$. The rest of the proof is similar to the respective case of lemma 5.2.33.

The case when $l_1 \neq \tilde{l}$ is similar.

□

Lemma 5.2.41. *Let $(\mathbb{D}, R)$ be a self-referential redex. Suppose that marked$(D'_1) = \{(\mathbb{D}, R), (\mathbb{D}', R')\}$. If $D'_1 \overset{(\mathbb{D},R)}{\underset{\gamma}{\circ\!\!\longrightarrow}} D'_2 \overset{(\mathbb{G},\tilde{R})}{\Longrightarrow} D'_4$, then there exists D'_3 such that $D'_1 \overset{(\mathbb{G}',\tilde{R}')}{\Longrightarrow} D'_3 \overset{(\mathbb{D},R)/(\mathbb{G}',\tilde{R}')}{\underset{\gamma}{\longrightarrow}{}^*} D'_4$ such that $(\mathbb{G}, \tilde{R}) = (\mathbb{G}', \tilde{R}')/(\mathbb{D}, R)$. The sequence $\overset{(\mathbb{D},R)/(\mathbb{G}',\tilde{R}')}{\underset{\gamma}{\longrightarrow}{}^*}$ consists either of a single step, or of two steps, where the self-referential residual of $(\mathbb{D}, R)$ is reduced second.* □

Proof. Let l, l', and $\tilde{l}$ denotes labels of the components where $(\mathbb{D}, l)$, $(\mathbb{D}', l)$, and $(\mathbb{G}, \tilde{R})$ occur, respectively. Similarly to the previous lemma 5.2.40, $l \neq l'$, $l \neq \tilde{l}$. We have 3 cases:

1. $\tilde{R}$ is a term redex. Suppose $l' = \tilde{l}$. We have the following possibilities:

   (a) $\tilde{R}$ and the marked occurrence of l are independent in the component bound to $\tilde{l}$. Then

   $$D'_2 = [l \mapsto \lambda x.\mathbb{C}\{\lambda x.\mathbb{C}\{l\}\}, \tilde{l} \mapsto \mathbb{A}\{\tilde{R}, \underline{l}\}, \dots].$$

   Then

   $$D'_1 = [l \mapsto \lambda x.\mathbb{C}\{\underline{l}\}, \tilde{l} \mapsto \mathbb{A}\{\tilde{R}, \underline{l}\}, \dots],$$

   and the rest of the proof is the same as in the respective case of lemma 5.2.34.

   (b) The marked l occurs in the operator of $\tilde{R}$. The case is similar to the previous one.

(c) The marked l occurs in the operand of $\tilde{R}$. The case is similar to the previous one.

If $l' \neq \tilde{l}$, the case is also similar to the previous ones.

2. $\tilde{R} = l_1 \neq l$ is a substitution redex. If $l' = l_1$, then

$$D'_2 = [l \mapsto \lambda x.\mathbb{C}\{\lambda x.\mathbb{C}\{l\}\}, \tilde{l} \mapsto \mathbb{E}\{l_1\}, l_1 \mapsto \lambda y.\mathbb{A}\{\underline{l}\}, \dots]$$
$$D'_1 = [l \mapsto \lambda x.\mathbb{C}\{\underline{l}\}, \tilde{l} \mapsto \mathbb{E}\{l_1\}, l_1 \mapsto \lambda y.\mathbb{A}\{\underline{l}\}, \dots].$$

The claim easily follows.

If $l' = \tilde{l}$, then

$$D'_2 = [l \mapsto \lambda x.\mathbb{C}\{\lambda x.\mathbb{C}\{l\}\}, \tilde{l} \mapsto \mathbb{A}\{l_1, \underline{l}\}, l_1 \mapsto V, \dots].$$

The case is similar to the case when $l' = l_1$, and so is the case when $l' \notin \{l_1, \tilde{l}\}$.

3. $\tilde{R} = l$. By the condition of the lemma this occurrence of l is unmarked, since only the other two redexes are marked.

Suppose $l' \neq \tilde{l}$. Then

$$D'_2 = [l \mapsto \lambda x.\mathbb{C}\{\lambda x.\mathbb{C}\{l\}\}, \tilde{l} \mapsto \mathbb{E}\{l\}, l' \mapsto \mathbb{A}\{\underline{l}\}, \dots],$$

and the reduction given in the case 3 of lemma 5.2.34 proves the claim. Note that the self-referential redex gets duplicated by the evaluation step $D'_1 \Longrightarrow D'_3$, and the reduction $D'_3 \underset{\gamma}{\longrightarrow}^* D'_4$ first reduces the copy of the redex in the component bound to $\tilde{l}$, and then the original self-referential redex (in the component bound to l).

The case when $l' = \tilde{l}$ is analogous.

□

Lemma 5.2.42. *Let* $\{(\mathbb{D}, R), (\mathbb{D}', R')\} \cap \tilde{F} = \emptyset$, *marked*$(D_1') = \{(\mathbb{D}, R), (\mathbb{D}', R')\} \cup \tilde{F}$, *and marked*$(D_2') = F \cup F' \cup \{(\mathbb{G}, \tilde{R})\}$. *Suppose* $D_1' \circ\!\xrightarrow[\gamma]{(\mathbb{D},R)} D_2' \xRightarrow{(\mathbb{G},\tilde{R})} D_4'$, *where* F, F' *are the sets of marked residuals of* $(\mathbb{D}, R), (\mathbb{D}', R')$, *respectively. Then* $\tilde{F} = \{(\mathbb{G}', \tilde{R}')\}$ *such that* $(\mathbb{G}, \tilde{R}) = (\mathbb{G}', \tilde{R}')/(\mathbb{D}, R)$, *and there exists* D_3 *such that* $D_1'' \xRightarrow{(\mathbb{G}',\tilde{R}')} D_3' \overset{(\mathbb{D},R)/(\mathbb{G}',\tilde{R}')}{\underset{\gamma}{\longrightarrow}{}^*} D_4'$, *where marked*$(D_1'') = \{(\mathbb{D}, R), (\mathbb{D}', R')\}$, $|D_1''| = |D_1'|$

If $(\mathbb{D}, R)$ *is a self-referential redex, then its self-referential residual is reduced last in the two-step sequence* $\overset{(\mathbb{D},R)/(\mathbb{G}',\tilde{R}')}{\underset{\gamma}{\longrightarrow}{}^*}$. *Otherwise the order of reductions in the sequence* $\overset{(\mathbb{D},R)/(\mathbb{G}',\tilde{R}')}{\underset{\gamma}{\longrightarrow}{}^*}$ *does not matter.* □

Proof. A marked redex can not be a residual of a non-marked one, therefore $\tilde{F} \neq \emptyset$. By lemma 5.2.13 we have the following two cases:

1. All the marked redexes are term redexes. Let $l_1, l', \tilde{l}$ be the labels of the components where the redexes $(\mathbb{D}, R), (\mathbb{D}', R')$, and $(\mathbb{G}, \tilde{R})$ occur, respectively.

   If $\tilde{l} \neq l_1$, then the reduction of the non-evaluation redex $(\mathbb{D}, R)$ does not affect the evaluation redex $(\mathbb{G}, \tilde{R})$. The component bound to $\tilde{l}$ does not change, so $\tilde{F} = \{(\mathbb{G}', \tilde{R})\}$, and $(\mathbb{G}, \tilde{R}) = (\mathbb{G}', \tilde{R})/(\mathbb{D}, R)$. The evaluation redex is marked in D_1 since it is marked in D_2. The rest of the claim follows similarly to case 1 of the proof of lemma 5.2.35.

   If $\tilde{l} = l_1$, then the claim follows from lemma 5.1.25. In this case when $l' \neq \tilde{l}$ $(\mathbb{D}', R')$ has one residual (itself), and when we apply lemma 5.1.25, we don't consider $(\mathbb{D}', R')$ as a marked redex, since it occurs in a different component. If $l' = \tilde{l}$, then we consider all three marked redexes in the application of lemma 5.1.25.

2. All the marked redexes are substitution redexes with the same label l. Let $l_1, l', \tilde{l}$ be as in case 1. Note that $l \neq \tilde{l}$, since $(\mathbb{G}, \tilde{R})$ is an evaluation redex. We have the following subcases:

   (a) $l_1 \neq l$, $l' \neq l$, i.e. none of the redexes is self-referential. Suppose that $l_1 = l' = \tilde{l}$, i.e. all redexes occur in the same component. Then

   $$D'_2 = [\tilde{l} \mapsto \mathbb{A}\{\underline{l}, V, \underline{l}\}, l \mapsto V, \dots],$$

   where the first marked occurrence of l corresponds to the redex $(\mathbb{G}, \tilde{R})$, and the second to the redex $(\mathbb{D}', R')$. Note that the occurrence of $\underline{l}$ corresponding to the evaluation redex does not occur in V because it occurs in an evaluation context, and the one corresponding to $(\mathbb{D}', R')$ does not occur in V by the assumption that none of the redexes is self-referential. Then

   $$D'_1 = [\tilde{l} \mapsto \mathbb{A}\{\underline{l}, \underline{l}, \underline{l}\}, l \mapsto V, \dots],$$

   by lemma 5.2.37 $\mathbb{A}\{\Box, \underline{l}, \underline{l}\} \in \mathbf{EvalContext}_{\mathcal{T}}$, and the claim of the lemma easily follows.

   The cases when the three redexes occur in at least two different components are similar.

   (b) $l' = l$, i.e. the redex $(\mathbb{D}', R')$ is self-referential. Suppose $\tilde{l} = l_1$, then

   $$D'_2 = [l \mapsto \lambda x.\mathbb{C}\{\underline{l}\}, \tilde{l} \mapsto \mathbb{A}\{\underline{l}, \lambda x.\mathbb{C}\{l\}\}, \dots],$$
   $$D'_1 = [l \mapsto \lambda x.\mathbb{C}\{\underline{l}\}, \tilde{l} \mapsto \mathbb{A}\{\underline{l}, \underline{l}\}, \dots].$$

   As in the case 4(b) of the proof of lemma 5.2.39, the marked occurrence of l in the component bound to $\tilde{l}$ in D'_2 is independent from $\lambda x.\mathbb{C}\{l\}$, since it

occurs in an evaluation context. The rest of the proof is by the reduction in the case 2(b) of the proof of lemma 5.2.35. The case when the two non-self-referential redexes occur in different components is analogous.

(c) $l_1 = l$, i.e. $(\mathbb{D}, R)$ is self-referential. If $\tilde{l} = l'$, we have:

$$D'_2 = [l \mapsto \lambda x.\mathbb{C}\{\lambda x.\mathbb{C}\{l\}\}, \tilde{l} \mapsto \mathbb{A}\{\underline{l}, \underline{l}\}, \dots],$$
$$D'_1 = [l \mapsto \lambda x.\mathbb{C}\{\underline{l}\}, \tilde{l} \mapsto \mathbb{A}\{\underline{l}, \underline{l}\}, \dots].$$

The reduction in the case 2(c) of the proof of lemma 5.2.35. The case when the redex $(\mathbb{D}', R')$ occurs not in the same component as the evaluation redex is analogous.

□

Lemma 5.2.43 (Elementary Lift Diagram for $\mathcal{C}$). *If* $D'_1 \overset{(\mathbb{D},R)}{\underset{\gamma}{\circ\!\!\longrightarrow}} D'_2 \overset{(\mathbb{G},\tilde{R})}{\Longrightarrow} D'_4$, *then there exists* D'_3 *such that* $D'_1 \overset{(\mathbb{G}',\tilde{R}')}{\Longrightarrow} D'_3 \overset{(\mathbb{D},R)/(\mathbb{G}',\tilde{R}')}{\underset{\gamma}{\longrightarrow}{}^*} D'_4$ *such that* $(\mathbb{G}, \tilde{R}) = (\mathbb{G}', \tilde{R}')/(\mathbb{D}, R)$. □

Proof. The proof is by lemmas 5.2.39, 5.2.40, 5.2.41, and 5.2.42 analogous to the proof of lemma 5.2.36. □

It remains to show standardization of complete γ-evelopments. It follows from the lemmas 5.2.34 and 5.2.35 for the elementary project diagram and from lemmas 5.2.41 and 5.2.42 for the elementary lift diagram that when the self-referential redex is duplicated, the elementary diagrams commute if the self-referential copy of the redex is reduced after the non-self-referential one. For instance, if the two residuals of the self-referential redex in case 3 of lemma 5.2.41 are reduced in the

other order, the diagram does not commute:

$$
\begin{array}{ll}
[l \mapsto \lambda x.\mathbb{C}\{\underline{l}\}, \tilde{l} \mapsto \mathbb{E}\{l\}, l' \mapsto \mathbb{A}\{\underline{l}\}, \ldots] & \underset{\gamma}{\circ\!\!\longrightarrow} \\
[l \mapsto \lambda x.\mathbb{C}\{\lambda x.\mathbb{C}\{l\}\}, \tilde{l} \mapsto \mathbb{E}\{l\}, l' \mapsto \mathbb{A}\{\underline{l}\}, \ldots] & \Longrightarrow \\
[l \mapsto \lambda x.\mathbb{C}\{\lambda x.\mathbb{C}\{l\}\}, \tilde{l} \mapsto \mathbb{E}\{\lambda x.\mathbb{C}\{\lambda x.\mathbb{C}\{l\}\}\}, l' \mapsto \mathbb{A}\{\underline{l}\}, \ldots], &
\end{array}
$$

but the sequence below leads to a record different from the one above:

$$
\begin{array}{ll}
[l \mapsto \lambda x.\mathbb{C}\{\underline{l}\}, \tilde{l} \mapsto \mathbb{E}\{l\}, l' \mapsto \mathbb{A}\{\underline{l}\}, \ldots] & \Longrightarrow \\
[l \mapsto \lambda x.\mathbb{C}\{\underline{l}\}, \tilde{l} \mapsto \mathbb{E}\{\lambda x.\mathbb{C}\{\underline{l}\}\}, l' \mapsto \mathbb{A}\{\underline{l}\}, \ldots] & \underset{\gamma}{\circ\!\!\longrightarrow} \\
[l \mapsto \lambda x.\mathbb{C}\{\lambda x.\mathbb{C}\{l\}\}, \tilde{l} \mapsto \mathbb{E}\{\lambda x.\mathbb{C}\{\underline{l}\}\}, l' \mapsto \mathbb{A}\{\underline{l}\}, \ldots] & \underset{\gamma}{\circ\!\!\longrightarrow} \\
[l \mapsto \lambda x.\mathbb{C}\{\lambda x.\mathbb{C}\{l\}\}, \tilde{l} \mapsto \mathbb{E}\{\lambda x.\mathbb{C}\{\lambda x.\mathbb{C}\{\lambda x.\mathbb{C}\{l\}\}\}\}, l' \mapsto \mathbb{A}\{\underline{l}\}, \ldots]. &
\end{array}
$$

Note that it is not only the sets of marked redexes, but the records themselves are different.

Lemma 5.2.44. *If* $D'_1 \overset{(\mathbb{D},R)}{\underset{\gamma}{\circ\!\!\longrightarrow}} D'_2 \overset{(\mathbb{G},\tilde{R})}{\underset{\gamma}{\Longrightarrow}} D'_4$*, where* $(\mathbb{D}, R)$ *is not self-referential, then there exists* D'_3 *such that* $D'_1 \underset{\gamma}{\Longrightarrow} D'_3 \underset{\gamma}{\longrightarrow}^* D'_4$*.* □

Note that when $(\mathbb{D}, R)$ is a self-referential redex, then the claim of the above lemma does not hold. For instance, consider:

$$
\begin{array}{ll}
[l \mapsto \lambda x.\underline{l}, l' \mapsto \underline{l}] & \underset{\gamma}{\circ\!\!\longrightarrow} \\
[l \mapsto \lambda x.\lambda x.l, l' \mapsto \underline{l}] & \underset{\gamma}{\Longrightarrow} \\
[l \mapsto \lambda x.\lambda x.l, l' \mapsto \lambda x.\lambda x.l], & \text{but} \\
[l \mapsto \lambda x.\underline{l}, l' \mapsto \underline{l}] & \underset{\gamma}{\circ\!\!\longrightarrow} \\
[l \mapsto \lambda x.\underline{l}, l' \mapsto \lambda x.l] & \underset{\gamma}{\Longrightarrow} \\
[l \mapsto \lambda x.\lambda x.l, l' \mapsto \lambda x.l] &
\end{array}
$$

5.2.6 Weak Standardization of γ-developments in $\mathcal{C}$

Lemma 5.2.45 (Weak Standardization of γ-developments in $\mathcal{C}$). *$\mathcal{C}$ has weak standardization of γ-developments, as defined in definition 4.5.7. More precisely, for every D'_1, D'_2, D''_1, D''_2 such that the two evaluation sequences $S^e{}_1 : D'_1 \underset{\cup}{\Rightarrow}^* D''_1$ and $S_2 : D'_2 \Rightarrow^* D''_2$ are lockstep equivalent, there exists a record D' such that $D''_1 \underset{\gamma}{\Rightarrow}^* D' \underset{c\gamma}{\circ\!\!\rightarrow}^* D''_2$.* □

Proof. By definition of lockstep equivalent reductions (definition 4.5.6) $\text{marked}(D'_1) = \{(\mathbb{D}, R)\}$ for some non-evaluation redex $(\mathbb{D}, R)$. We have the following two cases:

Case 1. If $(\mathbb{D}, R)$ is not self-referential, then lemma 5.2.44 is applicable to any two marked redexes (since by lemma 5.2.13 all marked residuals of $(\mathbb{D}, R)$ are also non-self-referential). Since the γ-developments are bounded (by lemma 5.2.14), by lemma 4.3.3 we can show strong standardization of γ-developments (definition 4.5.5).

Case 2. If $(\mathbb{D}, R)$ is self-referential, then the proof is by induction on the number of steps in the sequence $D'_1 \Rightarrow^* D''_1$. Let n denote this number. We want to prove that for every such D''_1 and the corresponding D''_2 there exists a standard complete γ-development $D''_1 \underset{\gamma}{\Rightarrow}^* D' \underset{c\gamma}{\circ\!\!\rightarrow}^* D''_2$ such that the self-referential redex is reduced last.

Base case ($n = 0$). Then the complete γ-development consists of a single step (the step reducing $(\mathbb{D}, R)$), so it is trivially standard, and the self-referential redex $(\mathbb{D}, R)$ is reduced last.

Induction Step. Suppose $D'_1 \underset{\cup}{\Rightarrow}^* D''_1 \Rightarrow D'''_1$, $D'_2 \Rightarrow^* D''_2$. Note that $\text{marked}(D'_2) = \text{marked}(D''_2) = \emptyset$.

$(D_2, \emptyset) \Rightarrow^* (D'_4, \emptyset)$, the two evaluation sequences are related by the elementary diagrams, and for (D'_3, F'_3) and $(D'_4, \emptyset)$ the claim of the lemma holds. By the condition of the lemma there exists an evaluation sequence related by elementary di-

agrams to the sequence $(D_1, \{(\mathbb{D}, R)\}) \Rightarrow^* (D'_3, F'_3) \Rightarrow (D_3, F_3)$. This implies that there exist $(\tilde{D}_3, \tilde{F}_3)$ and $(\tilde{D}'_3, \tilde{F}'_3)$ such that $(D'_3, F'_3) \underset{\gamma}{\Rightarrow}^* (\tilde{D}'_3, \tilde{F}'_3) \overset{0/1}{\Longrightarrow} (\tilde{D}_3, \tilde{F}_3)$ and $(D_3, F_3) \underset{\gamma}{\Rightarrow}^* (\tilde{D}_3, \tilde{F}_3)$, and we have one of the two possibilities:

- $(\tilde{D}'_3, \tilde{F}'_3) \overset{0/1}{\Longrightarrow} (\tilde{D}_3, \tilde{F}_3)$ is a 0-step reduction. Since the two evaluation sequences are related by elementary diagrams, $(\tilde{D}'_3, \tilde{F}'_3) \underset{\gamma}{\circ\!\!\rightarrow}^* (D'_4, \emptyset)$. By the induction hypothesis the last redex reduced in the non-evaluation γ-development sequence is the self-referential residual of $(\mathbb{D}, R)$, therefore the sequence $(D_3, F_3) \underset{\gamma}{\Rightarrow}^* (\tilde{D}_3, \tilde{F}_3) \underset{\gamma}{\circ\!\!\rightarrow}^* (D'_4, \emptyset)$ satisfies the claim of the lemma.

- $(\tilde{D}'_3, \tilde{F}'_3) \overset{0/1}{\Longrightarrow} (\tilde{D}_3, \tilde{F}_3)$ is a 1-step reduction. The two evaluation sequences are related by elementary diagrams, therefore there exists $(D_4, \emptyset)$ such that $(\tilde{D}'_3, \tilde{F}_3) \underset{\gamma}{\circ\!\!\rightarrow}^* (D'_4, \emptyset) \Rightarrow (D_4, \emptyset)$, $(\tilde{D}_3, \tilde{F}_3) \underset{\gamma}{\rightarrow}^* (D_4, \emptyset)$, and the reductions are constructed from the elementary diagrams. By the inductive hypothesis the last redex in the sequence $(\tilde{D}'_3, \tilde{F}_3) \underset{\gamma}{\circ\!\!\rightarrow}^* (D'_4, \emptyset)$ is the self-referential residual of $(\mathbb{D}, R)$. Let $(\mathbb{D}', R')$ denote this redex. By lemma 5.2.13 the other redexes reduced in this sequence are not self-referential. Since the sequence $(\tilde{D}_3, \tilde{F}_3) \underset{\gamma}{\rightarrow}^* (D_4, \emptyset)$ is constructed from $(\tilde{D}'_3, \tilde{F}'_3) \underset{\gamma}{\circ\!\!\rightarrow}^* (D'_4, \emptyset)$ via the elementary diagrams, there exist (D'', F'') and (D''', F''') such that $(\tilde{D}'_3, \tilde{F}'_3) \underset{\gamma}{\circ\!\!\rightarrow}^* (D'', F'') \underset{\gamma}{\circ\!\!\xrightarrow{(\mathbb{D}', R')}} (D'_4, \emptyset)$, $(\tilde{D}_3, \tilde{F}_3) \underset{\gamma}{\rightarrow}^* (D''', F''') \underset{\gamma}{\rightarrow}^* (D_4, \emptyset)$, and $(D'', F'') \Rightarrow (D''', F''')$. Let $(\mathbb{D}'', R'')$ be the self-referential residual of $(\mathbb{D}', R')$. By lemma 5.2.34 $(\mathbb{D}'', R'')$ is reduced last in the sequence $(D''', F''') \underset{\gamma}{\rightarrow}^* (\hat{D}, \hat{F}) \underset{\gamma}{\circ\!\!\xrightarrow{(\mathbb{D}'', R'')}} (D_4, \emptyset)$. A self-referential redex is a non-evaluation redex, therefore the step $\underset{\gamma}{\circ\!\!\xrightarrow{(\mathbb{D}'', R'')}}$ is a non-evaluation step. Since no other redexes in the sequence $(\tilde{D}_3, \tilde{F}_3) \underset{\gamma}{\rightarrow}^* (D''', F''') \underset{\gamma}{\rightarrow}^* (\hat{D}, \hat{F})$ are self-referential, by lemma 5.2.44 there exists $(\hat{D}', \hat{F}')$ such that $(\tilde{D}_3, \tilde{F}_3) \underset{\gamma}{\Rightarrow}^* (\hat{D}', \hat{F}') \underset{\gamma}{\circ\!\!\rightarrow}^* (\hat{D}, \hat{F})$, which proves the claim of the lemma.

□

5.2.7 Class Preservation and Results

Recall that classification of records is defined as

$$Cl_{\mathcal{C}}(D) = \begin{cases} \mathbf{evaluatable}_{\mathcal{C}} & \text{if there exists } D' \text{ such that } D \Longrightarrow_{\mathcal{C}} D' \\ [l_i \overset{n}{\underset{i=1}{\mapsto}} Cl_{\mathcal{T}}(V_i)] & \text{if } D = [l_i \overset{n}{\underset{i=1}{\mapsto}} V_i], \\ \mathbf{error}_{\mathcal{C}} & \text{otherwise} \end{cases}$$

We omit an obvious proof of the following lemma:

Lemma 5.2.46 (Class Preservation of $\mathcal{C}$). *If $D_1 \circ\!\!\rightarrow_{\mathcal{C}} D_2$, then $Cl_{\mathcal{C}}(D_1) = Cl_{\mathcal{C}}(D_2)$.* □

We can also show class preservation for the classification of the core record calculus defined in [MT00]:

$$Cl_{\mathcal{C}}(D) = [l_i \overset{n}{\underset{i=1}{\mapsto}} Cl_{\mathcal{T}}(M_i)], \text{ where } D = [l_i \overset{n}{\underset{i=1}{\mapsto}} M_i]. \quad \text{([MT00] definition)}$$

Theorem 5.2.47 (Computational Soundness of $\mathcal{C}$). *If $D_1 \leftrightarrow_{\mathcal{C}} D_2$, then $Outcome_{\mathcal{C}}(D_1) = Outcome(D_2)$.* □

Proof. By theorems 4.8.4 and 4.8.5 $\mathcal{C}$ has lift and project properties, since it satisfies the elementary diagrams (lemmas 5.2.36 and 5.2.43), has weak γ-confluence of evaluation (lemma 5.2.30), and weak standardization of γ-developments (lemma 5.2.45). These results are shown for marked records, and by lemma 4.8.3 they hold for modules with no marked redexes. $\mathcal{C}$ also has confluence of evaluation (lemma 5.2.16) and class preservation (lemma 5.2.46), therefore by theorem 3.4.5 it is computationally sound. □

Chapter 6

Overview of the Linking Calculus

6.1 Module and Linking Calculi

The linking calculus is built on top of the module calculus. The module calculus is very similar to the calculus of records, with the following differences:

1. Unlike the record calculus, the module calculus has two kinds of labels: visible ones and hidden ones. Visible labels represented exported and imported comments of a module, they serve as wiring to facilitate "information exchange" between modules. Hidden labels represent private components of modules. Hidden components of one module are not accessible and not even visible to any other module. We sometimes refer to the module calculus as the *core module calculus*.

   Terms of the linking calculus are called *linking expressions*. We use v to range over the set **Visible** of visible labels, and h to range over **Hidden** – the set of hiddens. We require that $\mathbf{Visible} \cap \mathbf{Hidden} = \emptyset$. The set of labels **Label** is defined as $\mathbf{Visible} \cup \mathbf{Hidden}$.

   In examples we use upper-case letters to range over visible labels, and lower-

case letters to range over hidden ones. For instance, in the following module the label A is visible, but a is hidden.

$$[A \mapsto 2 + a, a \mapsto 5]$$

While visible labels may appear not bound in a module, in which case we say that they are imported, hidden labels must be bound. For instance, the first of the two modules below is a valid module in the calculus, but the second one is not because the hidden label b is not bound in the module:

$$[A \mapsto a, a \mapsto B]$$
$$[A \mapsto a, a \mapsto b]$$

2. In addition to being identified up to reordering of components, modules are identified up to consistent renaming of hiddens throughout the module. For instance, the following two modules are α-equivalent:

$$[A \mapsto a, a \mapsto b, b \mapsto a]$$
$$[A \mapsto c, c \mapsto a, a \mapsto c]$$

We also formalize the notion of α-renaming of variables in a term and identify modules up to the term-level α-renaming of their components.

3. In the presence of hidden labels we define classification of modules in such a

way that hidden labels are not observable in the module:

$$Cl_{\mathcal{C}}(D) = \begin{cases} \textbf{evaluatable} & \text{if there exists } D' \text{ s.t. } D \Longrightarrow_{\mathcal{C}} D' \\ [v_i \overset{n}{\underset{i=1}{\mapsto}} Cl(V_i)] & \text{if } D = [v_i \overset{n}{\underset{i=1}{\mapsto}} V_i, h_j \overset{m}{\underset{j=1}{\mapsto}} V'_j], \\ \textbf{error} & \text{otherwise} \end{cases}$$

4. In one version of the module calculus we have introduced a garbage collection operation which removes hidden bindings not referenced in the rest of the module. This operation is a non-evaluation step in the calculus. A non-evaluation step must not change the observable behavior of a module, therefore garbage collection is restricted to removing hiddens bound to values.

The details of definition of the module calculus, of α-renaming at the term and at the module level, and of definition and properties of the calculus with garbage collection are given in [MT02].

6.2 Syntax of the Linking Calculus

The syntax for the linking calculus and definitions of its calculus reduction and evaluation are given in figure 6.1. We use $\mathcal{L}$ to denote the linking calculus. In particular, we use this symbol as a subscript for reductions in linking calculus.

In the definition of the linking calculus $BL(D)$ stands for bound labels of a module, i.e. those appearing on the left-hand side of a module binding. $Imports(D)$ stands for imported labels, i.e. those which are not bound in the module. As we have mentioned earlier, hidden labels cannot be imported.

A linking expression and a linking context are defined in figure 6.1. We introduce the following additional notations and definitions for linking expressions.

$$
\begin{array}{rcll}
I \in \mathbf{ModIdent} & ::= & \textit{module identifiers} & \\
L \in \mathbf{Term}_{\mathcal{L}} & ::= & D \mid I \mid L_1 \oplus L_2 \mid L[v \overset{\text{ren}}{\leftarrow} v'] \mid L\{\text{hide } v\} & \\
 & & \mid \textbf{let } I = L_1 \textbf{ in } L_2 & \\
\mathbb{L} \in \mathbf{Context}_{\mathcal{L},\mathcal{L}} & ::= & \Box \mid \mathbb{L} \oplus L \mid L \oplus \mathbb{L} \mid \mathbb{L}[v \overset{\text{ren}}{\leftarrow} v'] \mid \mathbb{L}\{\text{hide } v\} & \\
 & & \mid \textbf{let } I = \mathbb{L} \textbf{ in } L \mid \textbf{let } I = L \textbf{ in } \mathbb{L} & \\
D & \Longrightarrow_{\mathcal{L}} & D', \text{ where } D \Longrightarrow_{\mathcal{C}} D' & \text{(mod-ev)} \\
\mathbb{L}\{[k_i \overset{n}{\underset{i=1}{\mapsto}} M_i] \oplus [l_j \overset{m}{\underset{j=1}{\mapsto}} N_j]\} & \Longrightarrow_{\mathcal{L}} & \mathbb{L}\{[k_i \overset{n}{\underset{i=1}{\mapsto}} M_i, l_j \overset{m}{\underset{j=1}{\mapsto}} N_j]\}, & \text{(link)} \\
 & & \text{where } [k_i \overset{n}{\underset{i=1}{\mapsto}} M_i], [l_j \overset{m}{\underset{j=1}{\mapsto}} N_j] \in \mathbf{HTerm}_{\mathcal{C}} & \\
 & & \text{and } (\cup_{i=1}^{n} k_i) \cap (\cup_{j=1}^{m} l_j) = \emptyset. & \\
\mathbb{L}\{D[v \overset{\text{ren}}{\leftarrow} v']\} & \Longrightarrow_{\mathcal{L}} & \mathbb{L}\{D[v := v']\}, & \text{(rename)} \\
 & & \text{where } v \in \mathit{BL}(D) \text{ implies } v' \notin \mathit{BL}(D). & \\
\mathbb{L}\{D\{\text{hide } v\}\} & \Longrightarrow_{\mathcal{L}} & \mathbb{L}\{D[v := h]\}, & \text{(hide)} \\
 & & \text{where } v \notin \mathit{Imports}(D),\ h \notin \mathit{Hid}(D). & \\
\mathbb{L}\{\textbf{let } I = D \textbf{ in } L\} & \Longrightarrow_{\mathcal{L}} & \mathbb{L}\{L[I := D]\}, & \text{(let)} \\
\mathbb{L}\{D\} & \circ\!\!\rightarrow_{\mathcal{L}} & \mathbb{L}\{D'\}, \text{ where } D \rightarrow_{\mathcal{C}} D' & \text{(mod-nev)} \\
 & & \text{and } \mathbb{L} \neq \Box \text{ or } D \circ\!\!\rightarrow_{\mathcal{C}} D' &
\end{array}
$$

Figure 6.1: The linking calculus

A module identifier I is *free* in an expression L if it is not bound by a **let**. The set

$FMI(L)$ of free module identifiers of L is inductively defined as follows:

$$\begin{aligned}
FMI(D) &= \emptyset \\
FMI(I) &= \{I\} \\
FMI(L_1 \oplus L_2) &= FMI(L_1) \cup FMI(L_2) \\
FMI(L[v \overset{\text{ren}}{\leftarrow} v']) &= FMI(L) \\
FMI(L\{\text{hide } v\}) &= FMI(L) \\
FMI(\textbf{let } I = L_1 \textbf{ in } L_2) &= FMI(L_1) \cup (FMI(L_2) \setminus \{I\})
\end{aligned}$$

A linking expression L is *well-formed* if $FMI(L) = \emptyset$. In this presentation we assume all top-level linking expressions to be well-formed (expressions that are in the scope of a **let** may have free identifiers).

We define a substitution of a linking expression for a free identifier $L[I := L']$ similarly to substitution $M[x := N]$ in the term calculus (see definition **??**).

$$\begin{aligned}
I[I := L] &= L \\
I_1[I := L] &= I_1 \text{ if } I \neq I_1 \\
D[I := L] &= D \\
L_1 \oplus L_2[I := L] &= L_1[I := L] \oplus L_2[I := L] \\
L_1\{\text{hide } v\}[I := L] &= (L_1[I := L])\{\text{hide } v\} \\
L_1[v \overset{\text{ren}}{\leftarrow} v'][I := L] &= (L_1[I := L])[v \overset{\text{ren}}{\leftarrow} v'] \\
(\textbf{let } I = L_1 \textbf{ in } L_2)[I := L] &= \textbf{let } I = L_1[I := L] \textbf{ in } L_2 \\
(\textbf{let } I_1 = L_1 \textbf{ in } L_2)[I := L] &= \textbf{let } I_1 = L_1[I := L] \textbf{ in } L_2[I := L] \\
&\quad \text{if } I \neq I_1 \text{ and } I_1 \notin FMI(L) \text{ or } I \notin FMI(L_2) \\
(\textbf{let } I_1 = L_1 \textbf{ in } L_2)[I := L] &= \textbf{let } I_2 = L_1[I := L] \textbf{ in } L_2[I_1 := I_2][I := L] \\
&\quad \text{if } I \neq I_1,\ I_1 \in FMI(L),\ I \in FMI(L_2), \\
&\quad \text{and } I_2 \notin FMI(L) \cup FMI(L_2)
\end{aligned}$$

The next section gives details of relations in the linking calculus.

6.3 Linking Relations

Intuitively, the linking of modules D_1 and D_2, written $D_1 \oplus D_2$, takes the union of their bindings. To avoid naming conflicts between both visible and hidden labels, $BL(D_1)$ and $BL(D_2)$ must be disjoint[1]. The fact that the import labels of a module are non-hidden prevents the components of one module from accessing hidden components of the other one when they are linked.

The renaming operator renames a visible module label (an import or an export) to another visible label. The proviso "$v \in BL(D)$ implies $v' \notin BL(D)$" guarantees that the resulting module is well-formed[2], i.e. does not have two components bound to the same label name. Renaming import and export labels is the way to connect an exported component of one module to an import site in another.

Hiding renames a visible label to a fresh (i.e. not appearing in the module) hidden. The choice of this hidden name does not matter when we consider α-equivalence classes of modules. The restriction $v \notin \mathit{Imports}(D)$ prevents renaming an import to a hidden label.

The binding operator **let** $I = L_1$ **in** L_2 names the result of evaluating the definition term L_1 and uses the name within the body term L_2. This models situations in which the same module is used multiple times in different contexts (see examples below). In this presentation we require that the definition term L_1 is first evaluated to a module D, and then D is substituted into the body L_2. This is consistent with the call-by-value semantics of other reductions in our calculus (e.g. the

[1]The extension of linking to $\alpha_{\mathcal{L}}$-equivalence classes introduced later in this section removes the requirement that hidden labels of the two modules are disjoint.

[2]An attempt to rename one bound label in a module to another one causes a linking error.

term reductions and substitution at the module level).

The definition of $\rightarrow_{\mathcal{L}}$ lifts core module reduction steps to the linking level. The lifted core module reduction steps are only considered evaluation steps if they are not surrounded by any link-level operators; this forces all link-level steps to be performed first in a "link-time stage", followed by a "run-time stage" of core module steps.

The structure of a linking context allows the link-level operators to be evaluated in any order, as we see in the examples below.

Example 6.3.1 (Evaluation of a Linking Expression). The example shows reuse of a module via **let**, as well as linking, hiding, and module evaluation:

$$
\begin{array}{ll}
\textbf{let } A = [X \mapsto 2] \textbf{ in } (A \oplus [Y \mapsto X + Z]) \oplus ((A \oplus [Z \mapsto X * 5])\{\text{hide } X\}) & \Rightarrow_{\mathcal{L}} \\
([X \mapsto 2] \oplus [Y \mapsto X + Z]) \oplus (([X \mapsto 2] \oplus [Z \mapsto X * 5])\{\text{hide } X\}) & \Rightarrow^*_{\mathcal{L}} \\
[X \mapsto 2, Y \mapsto X + Z] \oplus [h \mapsto 2, Z \mapsto h * 5] & \Rightarrow_{\mathcal{L}} \\
[X \mapsto 2, Y \mapsto X + Z, h \mapsto 2, Z \mapsto h * 5] & \Rightarrow^*_{\mathcal{L}} \\
[X \mapsto 2, Y \mapsto 12, h \mapsto 2, Z \mapsto 10] &
\end{array}
$$

The first evaluation step reduces the **let**. The next 3 steps (shown as one $\Rightarrow^*_{\mathcal{L}}$ sequence) perform two linking operations and hiding. The two (link) reductions may be performed in any order. The hiding can be performed only after the linking $[X \mapsto 2] \oplus [Z \mapsto X * 5]$, since the argument of hiding must be a module. Hiding renames a visible label X to a new hidden label h. After this renaming has been performed, the component is no longer accessible from outside of the module in which it is defined. The final linking reduces the expression to a single module, and the following $\Rightarrow^*_{\mathcal{L}}$ sequence evaluates the module to a module value (all components are bound to values). □

We use the following abbreviation for a sequence of nested **let**s:

$$\textbf{let}\ \ I_1 = L_1\ \ I_2 = L_2 \ldots I_n = L_n\ \ \textbf{in}\ \ L_{n+1}$$

is syntactic sugar for

$$\textbf{let}\ \ I_1 = L_1\ \ \textbf{in}\ \ (\textbf{let}\ \ I_2 = L_2\ \ \textbf{in} \ldots (\textbf{let}\ \ I_n = L_n\ \ \textbf{in}\ \ L_{n+1}) \ldots).$$

The next example shows possibilities of connecting module components via renaming.

Example 6.3.2 (Connecting Module Components Via Renaming).

$$\begin{array}{ll}
\textbf{let} & A = [X \mapsto 0] \\
 & B = [Y \mapsto Z + 1] \\
 & C = A \oplus B \\
\textbf{in} & C[Y \overset{\text{ren}}{\leftarrow} Y_1][Z \overset{\text{ren}}{\leftarrow} X] \oplus B[Z \overset{\text{ren}}{\leftarrow} Y_1] \\
\Longrightarrow^*_{\mathcal{L}} & [X \mapsto 0, Y_1 \mapsto X + 1, Y \mapsto Y_1 + 1] \\
\Longrightarrow^*_{\mathcal{L}} & [X \mapsto 0, Y_1 \mapsto 1, Y \mapsto 2].
\end{array}$$

Note that the first renaming applied to C renames an exported label, and the second one (as well as the renaming of Z in B) renames an imported label. □

Cases of $\circ\!\!\longrightarrow_{\mathcal{L}}$ include an evaluation of a module in a non-empty linking context and a non-evaluation step on a module in an empty context, as shown in the following example:

Example 6.3.3 (Non-Evaluation Steps in $\mathcal{L}$). Even though the substitution is an evaluation step at the module level, it is a non-evaluation step at the linking level,

since the module appears in a non-empty context:

$$[X \mapsto 2, Z \mapsto X + 3] \oplus [P \mapsto X + Z] \circ\!\!\rightarrow_{\mathcal{L}} [X \mapsto 2, Z \mapsto 2 + 3] \oplus [P \mapsto X + Z]$$

A non-evaluation step on a module is a non-evaluation linking step even if the module appears in an empty context:

$$[F \mapsto \lambda x.Y + 3, Y \mapsto 2] \circ\!\!\rightarrow_{\mathcal{L}} [F \mapsto \lambda x.2 + 3, Y \mapsto 2].$$

□

6.4 Overview of α-renaming

As we have mentioned in section 6.1, we formalize α-renaming for term and module calculus: α- renaming of bound variables in a term and renaming of hiddens in a module. At the linking level we add α-renaming of module identifiers in linking expressions to our list of α-renaming.

Each calculus level inherits α-renaming from the previous level. Linking expressions are identified up to all three α-renamings. However, the most interesting and useful of the three α-renamings at the linking level is renaming of hiddens in a module. By identifying modules up to renaming of hiddens we are able to define linking of modules in such a way that it is independent of the particular names of the hiddens in the two modules. The linking operation is defined on α-equivalence classes of modules. Thus linking of two modules is well-defined as long as there is no conflict between the names of the *visible* components of the two modules, regardless of possible conflicts between names of *hidden* components of the modules.

This approach leads to defining classification of linking expressions not on

individual expressions, but on α-equivalence classes of linking expressions. Classification defined in this manner does not depend on names of hiddens in modules, so conflicts in names of hiddens do not prevent linking of two modules. $L_{\alpha_\mathcal{L}}$ denotes the α-equivalence class of a linking expression L. Note that $Cl(D)$ is the classification of a module D in the module calculus.

$$Cl(L_{\alpha_\mathcal{L}}) = \begin{cases} \textbf{evaluatable} & \text{if there exists } {L'}_{\alpha_\mathcal{L}} \text{ s.t. } L_{\alpha_\mathcal{L}} \Longrightarrow_{\mathcal{L}\setminus\alpha} {L'}_{\alpha_\mathcal{L}}, \\ Cl(D) & \text{if there exists } D \in L_{\alpha_\mathcal{L}} \text{ s.t. } D \text{ is an } \Longrightarrow_\mathcal{C} \text{ normal form}, \\ \textbf{error} & \text{otherwise.} \end{cases}$$

The proviso "D is an $\Longrightarrow_\mathcal{C}$ normal form" in the second case of the definition is necessary to guarantee that the classes are disjoint.

The class **error** captures the case when a linking expression never evaluates to a module because it contains a $\oplus$ operation whose arguments export the same names, for instance:

$$Cl([A \mapsto 2 + 3] \oplus [A \mapsto \lambda x.x]) = \textbf{error}.$$

Another example of a linking error is an attempt to rename a bound label in a module to another bound label, e.g.

$$Cl([A \mapsto \lambda x.x, B \mapsto 2][A \overset{\text{ren}}{\leftarrow} B]) = \textbf{error}.$$

See [MT02] for more detailed discussion of the α-renaming issues.

6.5 Results

Even though reductions in $\mathcal{L}$ are defined on α-equivalence classes of linking expressions, as explained in the previous section, we state the respective results for concrete linking expressions for simplicity. For details and for proofs of the results see [MT02].

The linking calculus inherits non-confluence from the module calculus, as shown by the following example:

Example 6.5.1 (Non-confluence of Linking Calculus).

$$\begin{aligned}
&[A \mapsto \lambda x.B, B \mapsto \lambda x.A] \oplus [C \mapsto 2] \qquad \circ\!\!\longrightarrow_{\mathcal{L}} \\
&[A \mapsto \lambda x.\lambda x.A, B \mapsto \lambda x.A] \oplus [C \mapsto 2], \\
&[A \mapsto \lambda x.B, B \mapsto \lambda x.A] \oplus [C \mapsto 2] \qquad \circ\!\!\longrightarrow_{\mathcal{L}} \\
&[A \mapsto \lambda x.B, B \mapsto \lambda x.\lambda x.B] \oplus [C \mapsto 2].
\end{aligned}$$

As in the record calculus, and consequently in the module, calculus, there is no term both of these expressions reduce to. Note that, as in the record calculus, both reductions are non-evaluation steps. □

Despite the lack of confluence of $\longrightarrow$, we are still able to show the following:

Theorem 6.5.2. *$\Longrightarrow_{\mathcal{L}}$ is confluent.* □

We show that it cannot be the case that $\Longrightarrow_{\mathcal{L}}$ diverges on one path and leads to a normal form on another.

Lemma 6.5.3. *If $L \Longrightarrow_{\mathcal{L}}^{*} Eval_{\mathcal{L}}(L)$, then there is no infinite sequence of $\Longrightarrow_{\mathcal{L}}$ steps originating at L.* □

We show that every linking expression evaluates to a module, unless the evaluation encounters a linking error:

Theorem 6.5.4 (Non-error Linking Expressions Evaluate to a Module). *If* $Outcome(L) \neq$ **error**, *then there exists* D *such that* $L \Longrightarrow^*_{\mathcal{L}} D$. □

Since module evaluation steps are evaluation steps at the linking level only if they are performed in an empty linking context, any evaluation sequence in $\mathcal{L}$ performs all link-level steps first, followed by module level evaluation. We call this property *staging*.

Theorem 6.5.5 (Staging). *Given a sequence* $L_1 \stackrel{S}{\Longrightarrow}^*_{\mathcal{L}} L_2$, *there exists* L' *such that* $L_1 \stackrel{S_1}{\Longrightarrow}^*_{\mathcal{L}} L' \stackrel{S_2}{\Longrightarrow}^*_{\mathcal{L}} L_2$, *where* $S = S_1; S_2$, S_1 *is a sequence of only link-level redexes, and* S_2 *is a sequence of only module-level ones.* □

We also show that $\mathcal{L}$ satisfies the lift, project, and the class preservation properties, and therefore is computationally sound.

6.6 Example of Cross-Module Transformation

Our linking calculus is powerful enough to justify some cross-module transformations, such as the cross-module lambda-splitting transformation below. Figure 6.2 presents a sequence of steps justifying lambda-splitting in the calculus with garbage collection which we briefly described in section 6.1 above. See [MT02] for the precise definition and properties. In the figure, we assume that $\lambda y.M'$ is a closed abstraction. Since the calculus with garbage collection is computationally sound, the sequence is a proof that cross-module lambda-splitting is meaning preserving in the calculus.

This result is an encouraging sign that our calculus can rigorously prove meaning preservation of some real-life transformations.

$$
\begin{array}{l}
[U \mapsto \lambda x.\mathbb{C}\{\lambda y.M'\}] \oplus [X \mapsto U \mathbin{@} N] \\
\quad \downarrow \ \textit{(linking)} \\
[U \mapsto \lambda x.\mathbb{C}\{\lambda y.M'\}, X \mapsto U \mathbin{@} N] \\
\quad \uparrow \ \textit{(GC)} \\
[U \mapsto \lambda x.\mathbb{C}\{\lambda y.M'\}, h \mapsto \lambda y.M', X \mapsto U \mathbin{@} N] \\
\quad \uparrow \ \textit{(subst)} \\
[U \mapsto \lambda x.\mathbb{C}\{h\}, h \mapsto \lambda y.M', X \mapsto U \mathbin{@} N] \\
\quad \downarrow \ \textit{(subst)} \\
[U \mapsto \lambda x.\mathbb{C}\{h\}, h \mapsto \lambda y.M', X \mapsto \lambda x.\mathbb{C}\{h\} \mathbin{@} N] \\
\quad \uparrow \ \textit{(hiding)} \\
[U \mapsto \lambda x.\mathbb{C}\{U_e\}, U_e \mapsto \lambda y.M', X \mapsto \lambda x.\mathbb{C}\{U_e\} \mathbin{@} N]\{\text{hide } U_e\} \\
\quad \uparrow \ \textit{(linking)} \\
([U \mapsto \lambda x.\mathbb{C}\{U_e\}, U_e \mapsto \lambda y.M'] \oplus [X \mapsto \lambda x.\mathbb{C}\{U_e\} \mathbin{@} N])\{\text{hide } U_e\}
\end{array}
$$

Figure 6.2: A sequence of calculus steps proving that lambda-splitting is meaning preserving in $\mathcal{L}_{GC}$.

Chapter 7

Related Work

7.1 Related Work on Computational Soundness

The approach has been introduced by Plotkin in [Plo75]. The paper proves computational soundness of the call-by-value and call-by-name λ-calculi with constants (the calculi that we have introduced sections 2.2.1 and 2.2.2) via confluence, standardization[1], and, implicitly, class preservation. Since the calculus relation in both calculi is compatibly closed (i.e. $M \longrightarrow N$ implies $\mathbb{C}\{M\} \longrightarrow \mathbb{C}\{N\}$ for any one-hole context $\mathbb{C}$), computational soundness implies *observational soundness*[2] of the calculus: any two terms equivalent in the calculus behave the same way in any context. Plotkin's framework differs slightly from ours in that semantics of terms is given via evaluation by SECD machine. Another difference is that Plotkin's definition of a standard reduction sequence is given by induction on both the length of the sequence and the structure of the term. The resulting sequences have the form $M_1 \Longrightarrow^* M_2 \circ\!\!\longrightarrow^* M_3$ only when M_3 (and therefore, M_2) is a value, but not in general. However, the only

[1] Plotkin uses a somewhat different definition of standardization than the one given here – see discussion below.

[2] "Observational soundness" is our term (see [MT00]), Plotkin referred to this property as "consistency".

case when standardization is used in the classical computational soundness framework is when the end term is a value, so this difference does not affect the proofs of computational soundness.

Other well-known examples of traditional proofs of computational soundness are the proofs for two similar versions of a call-by-need calculus: [AF97] and [MOW98]. Both papers define the evaluation relation and the calculus relation, and prove confluence and standardization in order to justify program transformation. The latter paper employs the technique of reductions in marked calculus and of developments for the proof of standardization, which has many similarities to the techniques we use for proving lift and project. Note that the need for explicit work with marked terms and developments arises for the same reason as in the calculus of records: the calculus rules are not left-linear (see section 4.1). The non-left-linear rule in [MOW98] is $\mathbf{let}\ x = V\ \mathbf{in}\ \mathbb{C}\{x\} \longrightarrow \mathbf{let}\ x = V\ \mathbf{in}\ \mathbb{C}\{V\}$ because of two references to x (cf. the substitution rules in figure 2.4).

As far as we know, computational soundness has been proven only for confluent calculi. Such calculi include the call-by-value and the call-by-name λ-calculi, the call-by-need calculi mentioned above, as well as the module calculi, such as [FRR00] mentioned in section 7.3.

7.2 Related Proof Techniques

Classical frameworks for proving confluence and standardization rely, among other properties, on left-linearity of the calculus rules, which does not hold in the calculus of records. However, these frameworks give insight into various properties of the calculus, such as parallel moves lemma and finiteness (or boundedness) of developments. In our definition of γ-developments we tried to adapt this intuition to the

calculus of records, despite the lack of left-linearity and of finiteness, confluence, and standardization of developments.

Huet and Levy in [HL91] present an elegant approach to proving confluence and standardization in first-order term rewriting systems. In particular, they show that reductions of all residuals of a redex (i.e. complete developments) form a sup-semilattice structure of permutation classes of derivations (the basic block of these diagrams is a parallel moves diagram like the one in figure 4.1). This result implies both confluence and standardization of the system, where the notion of a "standard" sequence is based on the notion of the "needed" redex. These results serve as a good intuition behind reductions in a calculus with marked terms and standardization, but do not directly generalize to higher-order systems, such as λ-calculi.

Barendregt in [Bar84] gives a proof of confluence and standardization of the call-by-name λ-calculus without constants based on finiteness (or, more exactly, boundedness – see section 4.2.3 for details) of complete developments, also using the technique of reductions of marked terms. His definition of standardization is by induction on a standard reduction sequence.

Felleisen and Friedman in [FF86] introduced the notion of evaluation context which is used in the definition of an evaluation, or "needed", redex.

Masako Takahashi in [Tak95] introduced the notion of parallel reductions which captures and generalizes the inductive (with respect to the structure of the term) nature of definition of complete developments. The resulting technique can be applied to a variety of confluence and standardization proofs. Unfortunately, this technique is not applicable to the calculus of records. Complete developments in the calculus of records are non-confluent, not bound, and do not (in general) have standardization. This makes it impossible to define parallel reductions in our record calculus.

Gonthier, Levy, and Mellies in [GLM92] develop an axiomatic approach to standardization for general rewrite systems. However, their approach depends on left-linearity of the rewrite rules. It also requires finiteness of developments, which in general does not hold in the calculus of records. The paper summarizes the properties of an evaluation redex which turn out to be the essential properties in our framework as well: an evaluation redex may not be removed or duplicated by reduction of any other redex, and it may not be created by reduction of a non-evaluation redex.

Wells and Muller in [WM00] give a framework for finding a standard order and proving standardization for certain orthogonal combinatory reduction systems. The paper also given an excellent overview of various definitions of standardization and summarizes, via the notion of a "freezing" and a "frozen" redex, the essential properties of a standard sequence. Interestingly, our definition of standardization, which is clearly sufficient for computational soundness proofs, abstracts over the structure of the terms and of the concrete redexes reduced in the sequence, and requires only that the sequence reduces all evaluation redexes before all the non-evaluation ones. While our definition assumes that evaluation and non-evaluation steps are already defined in the calculus, it significantly simplifies most known definitions of a standard sequence.

Our work gives, as far as we know, the first concise definition of the class preservation property: a non-evaluation step does not change the class of a term. This definition summarizes many variations of this property used in literature. [Plo75] shows that the classes of values and "evaluatables" are preserved by "non-evaluation" steps [3].[AF97] define classification which distinguishes between "evaluatables", answers (a class of evaluation normal form), and "stuck" terms (corresponding to errors in our terminology) and prove several lemmas analogous to most cases of class

[3]Plotkin uses a different terminology.

preservation. [Bar84] also shows similar results with respect to the head reduction (analogous to evaluation). [Tah00] defines classification of terms for MetaML using the call-by-name model (Taha uses the term "workables" for what we call "evaluatables") and shows computational soundness of MetaML with respect to the big-step operational semantics via confluence.

7.3 Module Calculi and Recursive Systems

Recently there has been a lot of work on module and linking systems, focusing on such issues as: developing a formal system for modules and linking [Car97], sophisticated type systems for modules [HL94, Ler94, Sha99, Rus99]; and the expressiveness of module systems, e.g. handling features like recursive modules [FF98, DS98, CHP99, AZ99], inheritance and mixins [DS96, DS98, AZ99, FRR00], and dynamic linking [FF98, WV99, Dug01, HWC01].

Our work uses an untyped framework and focuses on computational soundness as a tool for proving meaning preservation of transformations. Many formal systems for recursion and modules define just the calculus relation: [AK97, WV99, AZ01]. The system of *units* in [FF98] defines the notion of reduction, but does not specify contexts in which it can be applied. Such systems do not distinguish between program evaluation and program transformations.

Most formal systems with recursions have a confluent calculus relation. The systems [AK97, WV99] achieve confluence by avoiding cyclic substitution. Even though confluence in our calculus fails in the same way as in the unrestricted system in [AK97], our way of dealing with this problem is different. Instead of prohibiting cyclic substitutions in the calculus, as it is done in [AK97, WV99], we allow such substitutions in the calculus but, due to the call-by-value nature of the substitution,

such steps are not evaluation steps in our calculus, since the only way to form a substitution cycle in our calculus is by a substitution inside a value, which can only be done by a non-evaluation step. This turns out to be sufficient for achieving our goal of proving the computational soundness of the calculus. A different approach is taken in [AB97], which addresses the lack of confluence for unfolding operations on recursive terms by identifying such terms up to information content – i.e., a term has a unique infinite normal form.

Other confluent module calculi include [AZ01] and [FRR00]. The latter paper defines both small-step operational semantics and the calculus relation and is computationally sound. The confluence in this calculus comes from the fact that the entire module gets copied into each reference, so the non-confluence example of our record calculus cannot be constructed.

Our calculus is able to express such features as module reuse (in our case, via **let**), also expressed in [FF98, WV99, FRR00, AZ01], recursion within the module, independence of the order of components, linking of two modules (these features are also expressed in all of the above models), renaming of components (also in [AZ01]), hiding and equivalence of modules up to α-renaming of hidden components (in all of the above).

However, our calculus does not have two features that we consider an important part of a module language: first-class modules ([FF98, WV99, FRR00]) and dynamic linking ([FF98, WV99, Dug01, HWC01]). Adding these features without breaking computational soundness is a challenging future work.

Chapter 8

Summary and Future Work

8.1 Summary

This work presents a new technique for proving computational soundness which is capable of handling a non-confluent calculus. In addition to non-traditional features related to non-confluence, the calculus of records studied in this work presents other challenges, such as non-finiteness and non-confluence of developments and lack of general standardization of developments. These challenges make it impossible to adopt known techniques for proofs of properties similar to those we were interested in. For instance, a well-known technique of developments and complete developments could not be applied directly, or even with minor modifications, to the calculus of records. We have developed a significantly new approach of γ-developments for our proofs.

This presentation focuses on computational soundness aspects of our work. Much more work has been done by the author in collaboration with Franklyn Turbak on various aspects of meaning preservation. The work includes developing a three-level calculus for reasoning about modules and linking, proving computational

soundness at each of the three level of the calculus, studying issues of embedding one calculus into another via contexts, studying α-renaming and other related issues. The details of this other work are presented in [MT02].

8.2 Future Work

There are numerous directions in which this work can be extended. Below we list the ones we are mostly interested in pursuing.

8.2.1 Applications to Other Non-confluent Calculi

We plan to explore possibilities of applying the proof technique of lift and project to other non-confluent calculi. For instance, we are interested in exploring various calculi with the letrec rule, such as the one in [AB97]. Such calculi are interesting in themselves, but also as a model for programs with assignment and state. Since assignments can form cycles in the store, one might want to handle the store by some form of a letrec rule. Letrec-unfolding rules may be helpful for justifying program transformations in such calculi. However, these rules are likely to break confluence. Since it is possible to define evaluation in such calculi without the letrec-unfolding rules, and to add the unfolding rules as non-evaluation steps, such calculi look like a good potential application of our technique.

Non-confluent calculi with explicit substitution, in particular composition-free calculi, such as the calculus presented in [DL01], are another potential application of the new technique. Interestingly, the calculus mentioned above has the normalization property for the leftmost and head reductions, which seem to be somewhat related to the standardization property defined here, since the normalization requires that all leftmost (respectively head) reductions are done first, before other calculus reduction.

These properties gives us a hope that we might be able to prove computational soundness of the calculus via our technique.

8.2.2 Integrating Program Analyses

Combining our technique with other program analyses, such as termination analysis, is a promising direction of research.

One such analysis is termination analysis: if we can show by means of some analysis that a term eventually evaluates to a value, then we can justify more transformations. For instance, we can perform a beta reduction $(\lambda x.M)\ @\ N \longrightarrow M[x := N]$ if we can prove that N evaluates to a value (otherwise the beta reduction would be potentially changing the outcome of the term, since N may diverge or evaluate to an error, but may be unused M). Termination analysis would allow more possibilities for garbage collection, since it will allow to remove unreferenced bindings which are guaranteed to terminate.

8.2.3 Proving Other Cross-Module Transformations

We would like to extend the list of cross-module transformations which can be proven to be meaning-preserving in the linking calculus that we have developed. As far as we know, there is very little work done in formally proving cross-module transformations, even those which are widely used in existing compilers. The example of cross-module lambda-splitting transformation shows that our calculus is capable of representing (and therefore proving) some fairly non-trivial transformations.

8.2.4 Extending the Calculus

Another promising direction of future work is extending the calculus relation and, if needed, the evaluation relation to represent more language features and more transformations. Our proof of meaning-preservation of the garbage collection rule (see [MT02]) is an example of how this could be accomplished. The rule has been added to the existing calculus as a non-evaluation step.

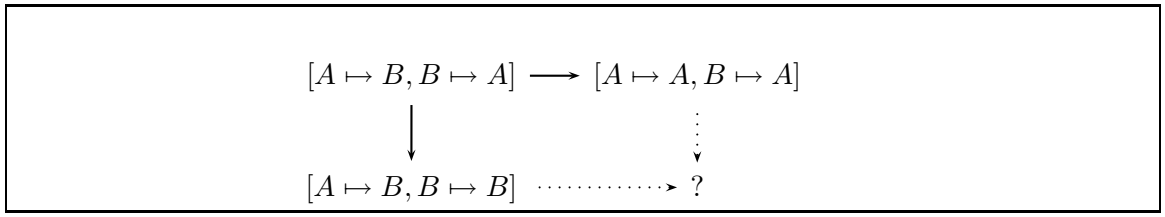


Figure 8.1: Labels as "values": non-confluence.

At the term level, we can add constructs for if statement and loops to be able to reason about loop unrolling and similar transformations.

At the module level, we can investigate adding a substitution of labels as "values" into another component. For instance, the following step is currently not a part of our calculus:

$$[A \mapsto B, B \mapsto C] \longrightarrow [A \mapsto C, B \mapsto C]$$

Such a step may be added only as a non-evaluation step, since taking two of such steps from the same term leads to non-confluence, see figure 8.1.

At the linking level, we would like to add such features as first class modules and dynamic linking.

8.2.5 Work on Classification

Studying various definitions of classification in a calculus is another interesting area of research: switching to a more fine-grained classification equates fewer terms, and therefore reduces the number of meaning-preserving transformations in the calculus. Using a more coarse classification makes more transformations meaning-preserving. We can think of a more refined classification as a subclassification of a more coarse one. Two classifications of the same calculus may also be unrelated to each other.

We would like to study the dependency of the class of meaning-preserving transformations on variations in classification, given fixed evaluation and non-evaluation relations.

8.2.6 Evaluation vs. Non-evaluation Steps

A generalization of a previous item is a study of various ways of breaking a given calculus relation into evaluation and non-evaluation steps and defining a classification which satisfies the class preservation property. During our work with the linking calculus we have explored many possibilities for evaluation and non-evaluation relations and for classification before we found those for which the proofs would go through. The process of finding just the right combination was largely a trial-and-error process. Structuring and formalizing this process would make the lift and project technique much more convenient to use.

Bibliography

[AB97] Z. M. Ariola and Stefan Blom. Cyclic lambda calculi. In *Theoretical Aspects of Computer Software: International Conference, TACS'97*, Sendai, Japan, 1997.

[AF97] Zena M. Ariola and Matthias Felleisen. The call-by-need lambda calculus. *Journal of Functional Programming*, 3(7), May 1997.

[AFM+95] Zena M. Ariola, Matthias Felleisen, John Maraist, Martin Odersky, and Philip Wadler. The call-by-need lambda calculus. In *Conference Record of the 22nd Annual ACM Symposium on the Principles of Programming Languages*, pages 233–246, 1995.

[AK97] Zena M. Ariola and Jan Willem Klop. Lambda calculus with explicit recursion. *Information and Computation*, 139(2):154–233, 15 December 1997.

[AZ99] Davide Ancona and Elena Zucca. A primitive calculus for module systems. In PPDP '99 [PPDP99], pages 62–79.

[AZ01] D. Ancona and E. Zucca. A calculus of module systems. *Journal of Functional Programming*, 2001. To appear.

[BA97] Matthias Blume and Andrew Appel. Lambda-splitting: A higher order approach to cross-module optimization. In *Proceedings of the 1997 International Conference on Functional Programming*, pages 112–124. ACM Press, 1997.

[Bar84] H[endrik] P[ieter] Barendregt. *The Lambda Calculus: Its Syntax and Semantics*. North-Holland, revised edition, 1984.

[Car97] Luca Cardelli. Program fragments, linking, and modularization. In *Conference Record of POPL '97: 24th ACM Symposium on the Principles of Programming Languages*, pages 266–277, 1997.

[CHP99] Karl Crary, Robert Harper, and Sidd Puri. What is a recursive module? In *Proceedings of the ACM SIGPLAN '99 Conference on Progrmaming Language Design and Implementation*, 1999.

[DL01] D. Dougherty and P. Lescanne. Reductions, intersection types, and explicit substitutions (extended abstract). In *5th International Conference on Typed Lambda Calculi and Applications*, volume 2044 of *Lecture Notes in Computer Science*, pages 121–135, Kraków, Poland, 2–5 May 2001. Springer-Verlag.

[DS96] Dominic Duggan and Constantinos Sourelis. Mixin modules. In *Proceedings of the 1996 International Conference on Functional Programming*, pages 262–273. ACM Press, 1996.

[DS98] Dominic Duggan and Constantinos Sourelis. Parameterized modules, recursive modules, and mixin modules. In *ACM SIGPLAN Workshop on ML*, pages 87–96, 1998.

[Dug01] Dominic Duggan. Sharing in typed module assembly language. In TIC '00 [TIC01].

[ESOP00] *Programming Languages and Systems, 9th European Symposium on Programming*, volume 1782 of *Lecture Notes in Computer Science*. Springer-Verlag, 2000.

[FF86] Matthias Felleisen and Daniel Friedman. Control operators, the SECD-machine, and the λ-calculus. In M. Wirsing, editor, *Formal Description of Programming Concepts — III*, pages 193–219. North-Holland, 1986.

[FF98] Matthew Flatt and Matthias Felleisen. Units: Cool modules for HOT languages. In *Proceedings of the ACM SIGPLAN '98 Conference on Programming Language Design and Implementation*, pages 236–248, 1998.

[FRR00] Kathleen Fisher, John Reppy, and Jon G. Riecke. A calculus for compiling and linking classes. In ESOP '00 [ESOP00], pages 135–149.

[GLM92] Georges Gonthier, Jean-Jacques Lévy, and Paul-André Melliès. An abstract standardisation theorem. In *Proceedings of the 7th Annual IEEE Symposium on Logic in Computer Science*, 1992.

[HL91] Gérard Huet and Jean-Jacques Lévy. Computations in orthogonal rewriting systems, I. In Jean-Louis Lassez and Gordon Plotkin, editors, *Computational Logic: Essays in Honor of Alan Robinson*, pages 395–414. MIT Press, 1991.

[HL94] Robert Harper and Mark Lillibridge. A type-theoretic approach to higher-order modules with sharing. In POPL '94 [POPL94], pages 123–137.

[HWC01] Michael Hicks, Stephanie Weirich, and Karl Crary. Safe and flexible dynamic linking of native code. In TIC '00 [TIC01].

[Ler94] Xavier Leroy. Manifest types, modules, and separate compilation. In POPL '94 [POPL94], pages 109–122.

[MOW98] John Maraist, Martin Odersky, and Philip Wadler. The call-by-need lambda calculus. *Journal of Functional Programming*, 8(3), May 1998.

[MT00] Elena Machkasova and Franklyn A. Turbak. A calculus for link-time compilation. In ESOP '00 [ESOP00], pages 260–274.

[MT02] Elena Machkasova and Franklyn Turbak. A computationally sound call-by-value module calculus. Technical report, Computer Science Department, Boston University, 2002.

[Plo75] G[ordon] D. Plotkin. Call-by-name, call-by-value and the lambda calculus. *Theoretical Computer Science*, 1:125–159, 1975.

[POPL94] *Conference Record of the 21st Annual ACM Symposium on the Principles of Programming Languages*, 1994.

[PPDP99] *Proceedings of the International Conference on the Principles and Practice of Declarative Programming*, volume 1702 of *Lecture Notes in Computer Science*. Springer-Verlag, 1999.

[Rus99] Claudio Russo. Non-dependent types for Standard ML modules. In PPDP '99 [PPDP99], pages 80–97.

[Sha99] Zhong Shao. Transparent modules with fully syntactic signatures. In *Proceedings of the 1999 International Conference on Functional Programming*, pages 220–232. ACM Press, 1999.

[Tah00] Walid Taha. A sound reduction semantics for untyped CBN multi-stage computation: Or, the theory of MetaML is non-trivial. In *Proceedings of the 2000 ACM SIGPLAN Workshop on Evaluation and Semantics-Based Program Manipulation (PEPM-00)*, pages 34–43, N.Y., January 22–23 2000. ACM Press.

[Tak95] Masako Takahashi. Parallel reductions in λ-calculus. *Information and Computation*, 118(1):120–127, April 1995.

[TIC01] *Types in Compilation, Third International Workshop, TIC 2000*, volume 2071 of *Lecture Notes in Computer Science*. Springer-Verlag, 2001.

[WM00] J. B. Wells and Robert Muller. Standardization and evaluation in Combinatory Reduction Systems. Unpublished draft to be submitted, 2000.

[WV99] J. B. Wells and René Vestergaard. Confluent equational reasoning for linking with first-class primitive modules (long version). A short version is [WV00]. Full paper, 3 appendices of proofs, August 1999.

[WV00] J. B. Wells and René Vestergaard. Equational reasoning for linking with first-class primitive modules. In ESOP '00 [ESOP00], pages 412–428. A long version is [WV99].

Curriculum Vitae

Elena Machkasova

Department of Computer Science

Wellesley College

Phone: office: (781) 283-3172

home: (617) 576-2483

e-mail: emachkas@wellesley.edu

Education:

Spring 1995-present: Boston University, PhD program in Computer Science. Advisor: Assaf Kfoury.

Fall 1994: University of Southern Maine, MS program in computer science.

June 1989: MS degree in Applied Mathematics awarded in Moscow Oil and Gas Institute, Department of Applied Mathematics.

Sept. 1984 - June 1989: Moscow Oil and Gas Institute, Department of Applied Mathematics, 5 year program analogous to BA/MS program.

Work Experience:

1989 - 1991: Software Engineer, Automated Solutions for Gas Industry, Moscow, Russia.

1987 - 1989: Computer Programmer (part-time), Moscow Oil and Gas Institute, Moscow, Russia.

Publications:

1. Elena Machkasova and Franklyn Turbak. A Calculus for Link-time Compilation. In *Programming Languages and Systems, 9th European Symposium*

on Programming, vol. 1782 of *Lecture Notes in Computer Science*, Springer-Verlag, 2000.

2. Elena Machkasova and Franklyn Turbak. A Computationally Sound Call-by-value Module Calculus. Technical Report, Computer Science Department, Boston University, to appear.

Teaching Experience:

Fall 2001 - present: Instructor in Computer Science Laboratory, Wellesley College. Courses:

- Introduction to e-commerce. An upper undergradute level course on e-commerce technology with an intensive Java programming component.
- Weekly laboratory classes for introductory level computer science course for majors. Introduction to Java programming.
- Fall 2001. Weekly laboratory classes for introductory level computer science course for non-majors. HTML, Java Script.

Spring 2001: Teaching Fellow for Introduction to CS II (Data Structures with C++) course. Responsibilities include preparing and teaching lab sections.

Fall 2000: Teaching Fellow for Introduction to Programming Languages course.

Fall 1998: Teaching Fellow for Introduction to CS II (Data Structures with C/C++) course.

Summer 1998: Instructor for Introduction to CS I (with C programming).

Fall 1997, Spring 1998: Teaching Fellow for Introduction to CS I (with C programming).

Summer 1997: Instructor for Introduction to CS (for non-majors).

Fall 1996, Spring 1997: Teaching Fellow for Introduction to CS I (with C

programming).

Summer 1996: Instructor for Introduction to CS (for non-majors).

Fall 1995, Spring 1996: Teaching Fellow for Introduction to CS I, II – intensive course.

Spring 1995: Grader for Introduction to CS (for non-majors).

1987 - 1989: Moscow High School #57. Co-taught advanced mathematics courses for high school students specializing in mathematics (part-time).

Teaching Awards:

1996, 1998: Teaching Fellow of the Department Award, Boston University.

Personal:

Born October 16, 1966 in Moscow, USSR.

US Permanent Resident.