Computational Soundness of a Call by Name Calculus of Recursively-scoped Records. UMM Working Papers Series, Volume 2, Num. 3.

Elena Machkasova

Contents

Abstract

The paper presents a calculus of recursively-scoped records: a two-level calculus with a traditional call-by-name λ-calculus at a lower level and unordered collections of labeled λ-calculus terms at a higher level. Terms in records may reference each other, possibly in a mutually recursive manner, by means of labels. We define two relations: a rewriting relation that models program transformations and an evaluation relation that defines a small-step operational semantics of records. Both relations follow a call-by-name strategy. We use a special symbol called a black hole to model cyclic dependencies that lead to infinite substitution.

Computational soundness is a property of a calculus that connects the rewriting relation and the evaluation relation: it states that any sequence of rewriting steps (in either direction) preserves the meaning of a record as defined by the evaluation relation. The computational soundness property implies that any program transformation that can be represented as a sequence of forward and backward rewriting steps preserves the meaning of a record as defined by the small step operational semantics.

In this paper we describe the computational soundness framework and prove computational soundness of the calculus. The proof is based on a novel inductive context-based argument for meaning preservation of substituting one component into another.

Chapter 1

Introduction and Related Work

1.1 Introduction

In this work we present a call-by-name calculus of *recursively-scoped records*. Recursively-scoped records (called *records* for the remainder of the paper) are unordered collections of labeled components that may reference each other, possibly in a mutually recursive manner. Representation of mutual dependencies arises in many calculi that model separate compilation, modules and linking, e.g. [1, 15], dynamic code manipulation, e.g. [6], `letrec`, e.g. [12]. While our system has a much more modest set of features, it captures the essence of mutual dependencies - substitution with a possibility of cyclic dependencies.

We use a common approach pioneered by G. Plotkin in [11] of defining two relations in a system: a rewriting relation that represents program transformations and an evaluation relation that defines the meaning of a term via small-step operational semantics. Computational soundness connects the two relations: it implies that any two terms that are equivalent with respect to the rewriting relation have the same meaning as defined by the evaluation relation. Thus any program transformation that can be constructed as a sequence of rewriting steps (forward and/or backward) preserves the meaning of a term.

In our work both the rewriting relation and the evaluation follow the call-by-name strategy. Since this strategy allows β-reduction and substitution of unevaluated terms, the repertoire of transformations represented by the rewriting relation is greatly expanded to include unrestricted common subexpression elimination within a record, specialization, etc. The computational soundness result implies that all such transformations on mutually dependent components are meaning preserving, and thus can be used in a variety of systems modeling modules and linking, mutual dependencies in a `letrec` binding, etc. Our future plan is to investigate whether the meaning preservation result holds if the evaluation is restricted to a more efficient call-by-value strategy, while transformations follow a more liberal call-by-name one.

As demonstrated in section 4.2, our calculus fails to satisfy properties required for some previously known proof methods for computational soundness: it lacks confluence of the rewriting relation required for Plotkin's original proof method and it fails to satisfy *lift* and *project* properties required for the approach in [10]. We use a novel context-based approach to complete the proof. It is an open question whether the proof can also be completed using an alternative diagram-based approach in [14].

The main contribution of the paper is the computational soundness proof of a call-by-name calculus of mutually dependent components in the framework of term meaning defined via a small-step operational semantics.

1.2 Related Work

G. Plotkin in [11] has proven a property equivalent to computational soundness for the call-by-name and the call-by-value term calculi. Z. M. Ariola and J. W. Klop studied issues of confluence and meaning preservation in similar systems of mutually dependent components. The straightforward definition of such a system breaks confluence (see [3]). In [4], in order to achieve confluence, substitution on cycles is disallowed. In [2] Z. M. Ariola and S. Blom show that unrestricted cyclic substitution is meaning preserving up to infinite unwindings of terms; their proof uses an approach that they call "skew-confluence".

In our earlier work [10, 7], we proved computational soundness of a non-confluent call-by-value calculus of records similar to the one considered here. We developed and used a diagram-based proof method based on properties that we called *lift* and *project*. This approach has been further extended and generalized to a collection of abstract diagram-based proof methods in [14]. However, the system considered here does not meet the requirements of the lift and project method (see Section 4.2) and it is unclear whether it can be handled using diagram-based methods presented in [14]. Nevertheless, the novel inductive context-based method presented here allows us to prove computational soundness of the call-by-name system.

A recent independent work [13] by M. Schmidt-Schauß presents a proof of correctness of a *copy* rule (analogous to our substitution rule) in a call-by-need and a call-by-name settings. The proof approach uses the machinery of infinite trees and is significantly different from our context-based approach. The relation between the applicability of the two proof methods is a subject of future research.

Chapter 2

Call-by-Name Calculus of Records

Records are unordered collections of labeled terms. Terms are elements of the traditional call-by-name λ-calculus [5], extended with constants, operations, and special symbols that represent interdependencies between terms. Each term in a record is marked by its unique label. The system can be viewed as a two-level calculus, with regular terms at the lower level and records at the upper level.

2.1 Term Level Calculus.

The term level of the calculus is defined below. We use prefix T for sets at the term level (such as $TTerm$), R is used at the level of records.

Definition 1 (Term-Level Calculus Syntax)**.**

$$\begin{array}{lll}
M, N \in TTerm & ::= & c \mid x \mid l \mid \bullet \mid \lambda x.M \mid M_1 \ @\ M_2 \mid M_1 \ +\ M_2 \\
\mathbb{C} \in TContext & ::= & \square \mid \lambda x.\mathbb{C} \mid \mathbb{C} \ @\ M \mid M \ @\ \mathbb{C} \mid \mathbb{C} \ +\ M \mid M \ +\ \mathbb{C} \\
\mathbb{E} \in TEvalContext & ::= & \square \mid \mathbb{E} \ @\ M \mid \mathbb{E} \ +\ M \mid c \ +\ \mathbb{E} \\
\mathbb{N} \in TNonEvalCntxt & & \mathbb{N} \in TContext, \mathbb{N} \notin TEvalContext
\end{array}$$

M, N denote terms, c stands for constants, such as numbers 1, 2, etc., x, y, z are variables (distinct from constants), l stands for labels (distinct from variables and constants), $\bullet$ is a special symbol that denotes a black hole, i.e. a cyclic dependency of a record component on itself, $\lambda x.M$ is a lambda abstraction, $M_1 \ @\ M_2$ is an application, $M_1 \ +\ M_2$ is a binary operation on terms. For simplicity we only use addition in our examples, but other operations can be added. The scope of a lambda binding extends as far to the right as possible, unless limited by parentheses. It is straightforward to extend the calculus with booleans, conditionals, and other features, but for simplicity they are not considered here.

The set of free variables of a term M, written $FV(M)$, is defined in Definition 2. Labels are distinct from variables, and thus are not included in $FV(M)$. We use $\equiv$ to denote syntactic equivalence of terms: if $M \equiv N$ then the two terms are identical. In particular they have the same names of all bound variables.

Definition 2 (Free variables)**.** *The set of* free variables of a term $M \in TTerm$, written FV(M), is defined as follows:

$$\begin{array}{rcl} FV(x) & = & \{x\}, \\ FV(c) & = & \emptyset, \\ FV(l) & = & \emptyset, \\ FV(\bullet) & = & \emptyset, \\ FV(\lambda x.M) & = & FV(M)\backslash\{x\}, \\ FV(M_1 \mathbin{@} M_2) & = & FV(M_1) \cup FV(M_2), \\ FV(M_1 \ + \ M_2) & = & FV(M_1) \cup FV(M_2). \end{array}$$

Following [5], we define α-equivalence of terms the following way:

Definition 3 (α-equivalence)**.** *1. An elementary α-renaming*[1] $\rightarrow_\alpha$ *is a relation on terms defined as* $\mathbb{C}\{\lambda x.N\} \rightarrow_\alpha \mathbb{C}\{\lambda y.N[x := y]\}$*, where y does not occur in N (neither bound, nor unbound).*

2. M is α-equivalent (or α-congruent) to N, denoted $M =_\alpha N$, if $M \rightarrow^*_\alpha N$, where $\rightarrow^*_\alpha$ is the reflexive transitive closure of $\rightarrow_\alpha$.

Note that $\rightarrow^*_\alpha$ is symmetric, therefore $\rightarrow^*_\alpha$ is the same as $=_\alpha$ and is an equivalence relation. We will also write $=$ to mean $=_\alpha$ and distinguish it from syntactic equivalence $\equiv$.

The following definition of substitution is also traditional (e.g., see [5]), with added cases for labels, black hole, and operations.

Definition 4 (Variable substitution in terms)**.** *The result of a capture-avoiding substitution of a term M' for a variable x in term M is written $M[x := M']$ and is defined as follows:*

$$\begin{array}{rcl} x[x := M] & = & M, \\ y[x := M] & = & y \text{ , if } x \neq y, \\ c[x := M] & = & c, \\ l[x := M] & = & l, \\ \bullet[x := M] & = & \bullet, \\ (N \mathbin{@} N')[x := M] & = & (N[x := M]) \mathbin{@} (N'[x := M]), \\ (N \ + \ N')[x := M] & = & (N[x := M]) \ + \ (N'[x := M]), \\ (\lambda x.N)[x := M] & = & \lambda x.N, \\ (\lambda y.N)[x := M] & = & \lambda y.(N[x := M]) \text{ if } x \neq y \text{ and } (x \notin FV(N) \text{ or } y \notin FV(M)), \\ (\lambda y.N)[x := M] & = & \lambda z.(N[y := z][x := M]), \text{where } z \notin (FV(M) \cup FV(N)), \\ & & \quad \text{if } x \neq y,\ x \in FV(N), \text{ and } y \in FV(M). \end{array}$$

[1] called a *change of bound variables* in [5]

Contexts are used as a way of specifying a particular subterm in a term. We use $\mathbb{C}$ as a metavariable for a term context, $\mathbb{E}$ as a metavariable for a subset of general contexts called *evaluation contexts*, and $\mathbb{N}$ for the complement of this subset called *non-evaluation contexts*. The symbol $\Box$ denotes a context hole. As an example, in the term $2 + \lambda x.3$ the subterm 2 appears in the context $\Box + \lambda x.3$ (an evaluation context) and 3 appears in $2 + \lambda x.\Box$ (a non-evaluation context, since $\Box$ is under a λ). Definition 7 uses evaluation contexts as means to specify a subterm to be evaluated according to the evaluation relation. If a subterm appears in a non-evaluation context, it will not be reduced by evaluation.

$\mathbb{C}\{M\}$ denotes the result of filling the hole in the context $\mathbb{C}$ with the term M. For instance, if $\mathbb{C} = \lambda x.\Box$ and $M = x + 2$ then $\mathbb{C}\{M\} = \lambda x.x + 2$. The notation for filling an evaluation context or a non-evaluation context is analogous. We can also fill a hole in a context with another context (denoted as $\mathbb{C}_1\{\mathbb{C}_2\}$), the result is a context. Note that it is possible to capture free variables of M when filling a context hole. Thus we do not introduce α-renaming of contexts. Definition 19 introduces contexts with multiple holes.

The following lemma states an important property of contexts:

Lemma 1. *Given one-hole contexts $\mathbb{C}_1$ and $\mathbb{C}_2$, $\mathbb{C}_1\{\mathbb{C}_2\} \equiv \mathbb{E}$ if and only if both $\mathbb{C}_1$ and $\mathbb{C}_2$ are evaluation contexts.*

Proof. By induction on the structure of an evaluation context. $\Box$

2.2 Record Level Calculus

Definition 5 (Record-Level Calculus)**.**

$$\begin{array}{lll}
D \in RTerm & ::= & [l_1 \mapsto M_1, ..., l_n \mapsto M_n],\ l_i \neq l_j \ for\ i \neq j \\
\mathbb{D} \in RContext & ::= & [l \mapsto \mathbb{C}, l_1 \mapsto M_1, \ldots, l_n \mapsto M_n], \mathbb{C} \in TContext \\
\mathbb{G} \in REvalContext & ::= & [l \mapsto \mathbb{E}, l_1 \mapsto M_1, \ldots, l_n \mapsto M_n], \mathbb{E} \in TEvalContext
\end{array}$$

D denotes a record with bindings of the form $l_i \mapsto M_i$. If $l_i \mapsto M_i$ occurs in a record, we say the term M is bound to the label l. We use notation $l \mapsto M \in D$ to indicate that the binding $l \mapsto M$ occurs in D. $L(D)$ denotes the set of all labels of D.

Assumption 1 (Closed Term)**.** *We assume that all terms in a record are closed, i.e. for any record $D = [l_1 \mapsto M_1, ..., l_n \mapsto M_n]$ we have $\cup_{i=1}^{n} FV(M_i) = \emptyset$.*

Recall that labels are separate from variables and are not included in $FV(M)$, thus labels are not affected by the assumption.

The following is an example of a record: $[l_1 \mapsto 2 + 3, l_2 \mapsto \lambda x.x, l_3 \mapsto l_2 \mathbin{@} l_1]$. It has three components, labeled by l_1, l_2, and l_3, respectively. The term $2 + 3$ is bound to l_1, $\lambda x.x$ is bound to l_2, and the component bound to l_3 references the first two by applying one to the other.

Definition 5 also introduces two record-level contexts: a general record context $\mathbb{D}$ and record evaluation context $\mathbb{G}$. For instance, $[l_1 \mapsto 2 + \Box, l_2 \mapsto \lambda x.x]$

is a record-level evaluation context (and also a general context since evaluation contexts are a subset of general contexts). Record-level contexts are filled with terms, not with records. For instance, one may fill the above context with a term 3 obtaining the record $[l_1 \mapsto 2+3, l_2 \mapsto \lambda x.x]$.

Record components are unordered, i.e two records that differ only in the order of their components are considered equivalent. We define α-renaming of records as α-renaming of bound variables in their components (recall that records consist of closed terms). Since records are intended to be embedded in larger systems, such as program modules, record components may be referenced from outside of a record. Thus there is no label renaming analogous to α-renaming of terms[2].

Definition 6 (Equivalence and Equality of Records)**.** *Two records D_1 and D_2 are considered syntactically equivalent ($D_1 \equiv D_2$) if the following holds: $l_i \mapsto M_i \in D_1$ if and only if $l_i \mapsto M'_i \in D_2$ and $M_i \equiv M'_i$. Likewise, two records are equal ($D_1 = D_2$) if $l_i \mapsto M_i \in D_1$ if and only if $l_i \mapsto M'_i \in D_2$ and $M_i = M'_i$.*

2.3 Calculus Relations

Both levels of the calculus follow the call-by-name reduction strategy. We define a *rewriting relation* $\rightarrow$ (which we also call *reduction*) and evaluation relation $\Rightarrow$ at the two levels of the calculus in definitions 7 and 9 respectively. Intuitively, the reduction relation represents transformations (i.e. "optimizations") of terms and records, and the evaluation relation represents the way records are evaluated at run-time by an evaluation engine (such as an interpreter). As discussed in section 4, the meaning of a record is defined by the result (called the *outcome*) of its evaluation.

At a more technical level, the difference between the two relations is that the rewriting relation reduces a redex in any context, while the evaluation reduces a redex in an evaluation context.

Definition 7 (Relations at the Term Level)**.** *The rewriting relation $\rightarrow$ and the evaluation relation $\Rightarrow$ at the term level are defined as follows:*

$$\begin{array}{lcll}
(\lambda x.M) @ N & \rightsquigarrow & M[x := N] & (\beta) \\
c_1 \ + \ c_2 & \rightsquigarrow & c_3 \textit{ where } c_3 \textit{ is the result of operation} & (op) \\
\mathbb{E}\{R\} & \Rightarrow & \mathbb{E}\{Q\} \textit{ where } R \rightsquigarrow Q & \\
\mathbb{C}\{R\} & \rightarrow & \mathbb{C}\{Q\} \textit{ where } R \rightsquigarrow Q &
\end{array}$$

The $\rightsquigarrow$ arrow denotes the "elementary" reduction, i.e. the basic operations at the term level of the calculus: a call-by-name β-reduction ($M[x := N]$ stands for the result of the capture-free substitution of N for x in M) and an operation (op) that replaces an operation on two constants by their result, also a constant. The

[2] It is possible to add hidden components to records that cannot be referenced from outside of a record (see [10]). Records then are identified up to renaming of hidden labels. However, here we focus on computational soundness of mutually recursive components which is independent from the issue of hidden labels.

rewriting relation → can perform an elementary reduction in any context $\mathbb{C}$, i.e. anywhere inside a term. The evaluation step ⇒ performs the same operations but only in an evaluation context. $TEvalContext \subseteq TContext$ implies $\Rightarrow \subseteq \rightarrow$.

Notations below are used at both the term and the record level:

Definition 8 (Non-evaluation Relation and Closures). *1. A non-evaluation relation $\hookrightarrow$ is defined as $\hookrightarrow = \rightarrow \setminus \Rightarrow$.*

2. $\rightarrow^*$, $\Rightarrow^*$, $\hookrightarrow^*$ denote reflexive transitive closures of the respective relations; $\leftrightarrow$, $\overset{n}{\leftrightarrow}$ denote the reflexive symmetric transitive closures of $\rightarrow$ and $\hookrightarrow$, respectively.

One can equivalently define $\hookrightarrow$ as a reduction in a non-evaluation context.

The term that matches the left-hand side of an elementary reduction rule is called *a term redex*. R denotes redexes. Intuitively, a redex is the subterm that gets reduced by the reduction. The redex is enclosed in a context that remains unchanged by the reduction[3]. As an example, in the reduction $\lambda x.2+3 \rightarrow \lambda x.5$ the redex is 2+3 and the context is $\lambda x.\square$. In the evaluation step $1 + (\lambda x.x) @ 3 \Rightarrow 1 + 3$ the redex is $(\lambda x.x) @ 3$ and the context is $1 + \square$.

Lemma 2. *If $\mathbb{E}_1\{R_1\} \equiv \mathbb{E}_2\{R_2\}$, where R_1, R_2 are redexes, then $\mathbb{E}_1 \equiv \mathbb{E}_2$ and $R_1 \equiv R_2$. If $M \equiv \mathbb{E}_1\{l_1\} \equiv \mathbb{E}_2\{l_2\}$ then $\mathbb{E}_1 \equiv \mathbb{E}_2$ and $l_1 = l_2$ and $M \neq \mathbb{E}\{R\}$ for any $\mathbb{E}$ and R.*

Recall that if two terms are equal then we can choose syntactically equivalent representatives of their α-equivalence classes. In this sense we generalize Lemma 2 to equality:

Lemma 3. *If $\mathbb{E}_1\{R_1\} = \mathbb{E}_2\{R_2\}$, where R_1, R_2 are redexes, then $\mathbb{E}_1 \equiv \mathbb{E}_2$ and $R_1 = R_2$. If $M = \mathbb{E}_1\{l_1\} = \mathbb{E}_2\{l_2\}$ then $\mathbb{E}_1 \equiv \mathbb{E}_2$ and $l_1 = l_2$ and $M \neq \mathbb{E}\{R\}$ for any $\mathbb{E}$ and R.*

Lemma 3 says that there may be at most one redex in an evaluation context in a term. It is not possible that the same term has a redex and a label both appearing in an evaluation context or two labels, both appearing in an evaluation context.

2.3.1 Relations at the level of records.

Following the call-by-name strategy, both a reduction of an individual component and a substitution from one component into another one may copy an unevaluated term.

[3] More precisely, it is possible to find such representatives M, N in the α-equivalence classes of the original and the reduced term, respectively, that $M \rightarrow N$ by reducing the given redex in the given context and the context remains unchanged.

Definition 9 (Relations at the Level of Records)**.**

$$\begin{array}{lcll}
\mathbb{D}\{R\} & \rightarrow & \mathbb{D}\{Q\} \ \textit{where } R \leadsto Q & (T) \\
\mathbb{D}\{l\} & \rightarrow & \mathbb{D}\{N\} \ \textit{where } l \mapsto N \in \mathbb{D}\{l\},\ \mathbb{D} \neq [l \mapsto \mathbb{E}, \dots] & (S) \\
\mathbb{G}\{R\} & \Rightarrow & \mathbb{G}\{Q\} \ \textit{where } R \leadsto Q & (TE) \\
\mathbb{G}\{l\} & \Rightarrow & \mathbb{G}\{N\} \ \textit{where } l \mapsto N \in \mathbb{G}\{l\},\ \mathbb{G} \neq [l \mapsto \mathbb{E}, \dots] & (SE) \\
[l_1 \mapsto \mathbb{E}\{l_1\}, ...] & \Rightarrow & [l_1 \mapsto \bullet, ...] & (B1) \\
[l_1 \mapsto \mathbb{E}\{\bullet\}, ...] & \Rightarrow & [l_1 \mapsto \bullet, ...] & (B2)
\end{array}$$

Definition 9 gives three kinds of reductions on records, two of which have an evaluation and a rewriting versions.

- A *term reduction* simply reduces a term redex in one of the record's components. It is a rewriting step (see rule T) when it happens in a general context and an evaluation step (rule TE) when it happens in an evaluation context. For example, $[l_1 \mapsto \lambda x.2 + 3] \rightarrow [l_1 \mapsto \lambda x.5]$ is a rewriting step, but not an evaluation step. Such steps are called *non-evaluation* steps (see Definition 8).

- A *substitution* replaces a label occurring in a component of a record by the term bound to that label in the record. Analogously to the term reduction, the substitution is a rewriting step (rule S) if the label occurs in a general context, and an evaluation step (rule SE) if it occurs in an evaluation context.

  For example, $[l_1 \mapsto 2 + 3, l_2 \mapsto l_1 + 1] \Rightarrow [l_1 \mapsto 2 + 3, l_2 \mapsto (2 + 3) + 1]$ is an evaluation step since $\Box + 1$ is an evaluation context. The following substitution is a reduction, but not an evaluation step, since l_1 appears under a lambda: $[l_1 \mapsto 2 + 3, l_2 \mapsto \lambda x.l_1] \rightarrow [l_1 \mapsto 2 + 3, l_2 \mapsto \lambda x.(2 + 3)]$. Note that, just like a term reduction, the substitution is call-by-name: the term that gets substituted does not have to be evaluated first.

  The side conditions $\mathbb{D}, \mathbb{G} \neq [l \mapsto \mathbb{E}, \dots]$ eliminate an ambiguity between substitution and the black hole rule (B1) by preventing a substitution into a label that directly depends on itself in an evaluation context. For instance, the following substitution is not allowed: $[l_1 \mapsto l_1 + 1] \Rightarrow [l_1 \mapsto l_1 + 1 + 1]$, the rule (B1) is applied instead (see below).

- A *black hole symbol* $\bullet$ denotes apparent infinite substitution cycles that cannot be meaningfully evaluated. The rule (B1) introduces a black hole to replace a label that depends on itself in an evaluation context. For instance, $[l_1 \mapsto l_1 + 1] \Rightarrow [l_1 \mapsto \bullet]$ instead of an infinite substitution $[l_1 \mapsto l_1 + 1] \Rightarrow [l_1 \mapsto l_1 + 1 + 1] \Rightarrow \dots$ The notion of a black hole was first introduced in [3]. In this work it is essential for confluence of $\Rightarrow$ on records.

  The rule (B2) turns a component that depend on a black hole into a black hole: $[l_1 \mapsto \bullet, l_2 \mapsto l_1 + 1] \xRightarrow{S} [l_1 \mapsto \bullet, l_2 \mapsto \bullet + 1] \xRightarrow{B2} [l_1 \mapsto \bullet, l_2 \mapsto \bullet]$.

> The black hole rules do not have analogous non-evaluation rules since a self-dependency in a non-evaluation context may be a legitimate recursion and does not always lead to infinite substitution cycle or it may be eliminated during evaluation.

Just as at the term level, relations on records are defined on α-equivalence classes:

Lemma 4. *If $D_1 \rightarrow D_2$ and $D_1 =_\alpha D_1'$ then there is D_2' s.t. $D_1' \rightarrow D_2'$ and $D_2 =_\alpha D_2'$. Moreover, the reduction is an evaluation step $D_1 \Rightarrow D_2$ if and only $D_1' \Rightarrow D_2'$.*

The following lemma summarizes an obvious, but important, observation:

Lemma 5. *In the substitution rules (S) and (SE) in definition 9 $FV(N) = \emptyset$.*

Proof. Assumption 1 states that any term bound to a label in a record is closed. □

A normal form of a term with respect to a relation R is a term that cannot be further reduced by R. The definition is applicable to both terms and records.

Definition 10 (Normal Form)**.** *Given a relation R on a set of terms, a normal form with respect to (w.r.t.) R is a term M for which there is no M' such that MRM'. The predicate $nf_R(M)$ is true if M is a normal form w.r.t. R and false otherwise. A term N is an R-normal form of M if MR^*N and $nf_R(N)$.*

As an example, $nf_\rightarrow(\lambda x.2+3) =$ false since the term can be further reduced: $\lambda x.2 + 3 \rightarrow \lambda x.5$. However $nf_\Rightarrow(\lambda x.2 + 3) =$ true since the term does not have a redex in an evaluation redex. $\lambda x.5$ is a normal form of $\lambda x.2 + 3$ w.r.t. $\rightarrow$.

Chapter 3

Confluence of Evaluation

It follows from Lemma 3 that there is at most one evaluation step in any record component. For instance, if a component is of the form $\mathbb{E}\{R\}$, i.e. it has a term evaluation redex, it may not have a label in an evaluation context.

However, there is no ordering on components in a record, so any component that has a term or a substitution redex may be evaluated. Thus it is possible to have multiple evaluation steps originating at the same record:

$$\begin{array}{lcl} [l_1 \mapsto 2+3, l_2 \mapsto l_1+1,] & \Rightarrow & [l_1 \mapsto 5, l_2 \mapsto l_1+1] \\ [l_1 \mapsto 2+3, l_2 \mapsto l_1+1,] & \Rightarrow & [l_1 \mapsto 2+3, l_2 \mapsto 2+3+1] \end{array}$$

This flexibility opens a way for modeling separate compilation and evaluation of modules: evaluation of known components may start before the entire record becomes available.

The calculus of records has the following property:

Lemma 6. *$\Rightarrow$ is confluent on records.*

Proof. Case analysis on pairs of evaluation redexes shows that evaluation satisfies the strip lemma (see [5], Ch. 11), i.e. given $D_1 \Rightarrow D_2$ and $D_1 \Rightarrow^* D_3$, there exists D_4 such that $D_2 \Rightarrow^* D_4$ and $D_3 \Rightarrow^* D_4$. The strip lemma implies confluence. See [9] for details of the proof [1]. □

The presence of a black hole in the calculus is essential for confluence of evaluation. Consider the following record: $[l_1 \mapsto 2+l_2, l_2 \mapsto l_1+1]$. Note that both labels are in evaluation contexts in both components. Without a black hole the substitution into the first component would yield $[l_1 \mapsto 2+l_1+1, l_2 \mapsto l_1+1]$, substitution into the second component gives $[l_1 \mapsto 2+l_2, l_2 \mapsto 2+l_2+1]$. In the first resulting record both components reference l_1, in the second one they both reference l_2, and any subsequent substitutions preserve these properties. This is a variation of a famous non-confluence example introduced in [3].

[1] The black hole rules in the system in [9] differ slightly from the rules presented here because of additional properties satisfied by the system that are not required here. However, the difference in rules does not affect the essence of the proof.

However, a black hole allows us to bring these two records together by a sequence of evaluation steps since both labels appear in an evaluation context, and thus represent an infinite cycle of substitutions:

$$\begin{array}{ll}
[l_1 \mapsto 2 + l_1 + 1, l_2 \mapsto l_1 + 1] & \Rightarrow \\
[l_1 \mapsto \bullet, l_2 \mapsto l_1 + 1] & \Rightarrow \\
[l_1 \mapsto \bullet, l_2 \mapsto \bullet + 1] & \Rightarrow \\
[l_1 \mapsto \bullet, l_2 \mapsto \bullet] &
\end{array}$$

The record $[l_1 \mapsto 2 + l_2, l_2 \mapsto 2 + l_2 + 1]$ also evaluates to $[l_1 \mapsto \bullet, l_2 \mapsto \bullet]$:

$$\begin{array}{ll}
[l_1 \mapsto 2 + l_2, l_2 \mapsto 2 + l_2 + 1] & \Rightarrow \\
[l_1 \mapsto 2 + l_2, l_2 \mapsto \bullet] & \Rightarrow \\
[l_1 \mapsto 2 + \bullet, l_2 \mapsto \bullet] & \Rightarrow \\
[l_1 \mapsto \bullet, l_2 \mapsto \bullet] & \Rightarrow
\end{array}$$

Confluence of evaluation guarantees uniqueness of a normal form w.r.t. $\Rightarrow$ if a term has one. However, in general confluence of evaluation does not prevent a term from having a normal form and diverging at the same time, as long as every term on a diverging path has an evaluation sequence leading to the normal form. The lemma below states that this situation does not occur in our calculus, i.e. a term may not have a normal form and diverge at the same time (this property is also known as *uniform normalization*).

Lemma 7. *If $D \Rightarrow^* D'$, $nf_{\Rightarrow}(D')$, and no component in D' is bound to $\bullet$, then there is no infinite sequence $D \Rightarrow D_1 \Rightarrow D_2 \Rightarrow^* \ldots$.*

Proof. Suppose there exists a record D such that $D \overset{l'}{\Longrightarrow} D'_1$, there is no diverging evaluation sequence starting at D'_1, and the normal form of D does not have a black hole. Also assume that $D \overset{l}{\Longrightarrow} D_1$ s.t. D_1 has a diverging evaluation sequence $D_1 \overset{l_1}{\Longrightarrow} D_2 \overset{l_2}{\Longrightarrow} D_3 \Rightarrow^* \ldots$. The labels over the arrows denote labels of components reduced in the given evaluation steps.

We show that we can transform the diverging sequence of D_1 into a diverging sequence of D'_1, thus obtaining a contradiction.

Suppose that l' does not participate in the given infinite evaluation sequence, i.e. the component bound to l' is not evaluated and not copied by a substitution. Then we can reduce components l, l_1, l_2, etc. in D'_1, following the diverging sequence for D_1. Since l' is not used in this sequence, all of the steps are exactly the same in D'_1 as they are in D_1, thus there exists a diverging sequence for D'_1 which contradicts the assumption.

Now suppose that l' is used in the infinite evaluation sequence. We have several possibilities:

1. The component bound to l' is of the form $\mathbb{E}\{R\}$, where R is a term redex. Suppose it is evaluated to $\mathbb{E}\{Q\}$ in the step $D \overset{l'}{\Longrightarrow} D'_1$. Note that R appears in an evaluation context, and thus any step in the given diverging

sequence that copies R into another component also places it in an evaluation context. To easier keep track of copies of R, let us assume that all such copies are marked D_1 (for instance, by underlining) and the marks are preserved by substitution. Note that if a marked copy of R reduces to a term that is syntactically identical to R, the new copy would correspond to Q, not to R, and thus would not be marked. As an example, consider $R = (\lambda x.x \mathbin{@} x) \mathbin{@} (\lambda x.x \mathbin{@} x)$. Then Q has the exact same form as R but was not obtained as a copy of R so it is not marked.

Let us construct the following evaluation sequence starting at D'_1: we follow the evaluation steps of the sequence $D \overset{l}{\Longrightarrow} D_1 \overset{l_1}{\Longrightarrow} D_2 \overset{l_2}{\Longrightarrow} D_3 \Rightarrow^* \ldots$. However, when R needs to be reduced in this sequence, we skip the first step of this evaluation, namely the step that reduces a marked copy of R to Q since it has been already performed in D_1.

The resulting sequence must be a diverging sequence in D'_1 since there may not be an infinite number of reductions of copies of R without an infinite number of copying steps. All copying steps are preserved in the sequence that starts at D'_1. Thus even if we remove an infinite number of reductions of R, the resulting sequence is still infinite.

2. The component bound to l' is of the form $\mathbb{E}\{l''\}$, where $l'' \neq l$. The argument is similar to that in the previous case, although it requires adding some steps in D'_1, not just skipping steps.

   Suppose the substitution changes the component to $\mathbb{E}\{M\}$ in D'_1. We mark the corresponding occurrence of l'' in D and also mark the copied term M in D'_1. We construct a diverging sequence for D'_1 by performing the same evaluation steps as in the sequence $D \overset{l}{\Longrightarrow} D_1 \overset{l_1}{\Longrightarrow} D_2 \overset{l_2}{\Longrightarrow} D_3 \Rightarrow^* \ldots$. Every time a substitution replaces a marked l'' in the original sequence, the constructed sequence skips a step. However, every time an evaluation step is performed on the l'' component in the original sequence, all marked copies of M are reduced accordingly. This guarantees that after each substitution into l'' the two records are the same, except for some copies of l'' in D_1 replaced by terms in D'_1.

   It is also not possible that D'_1 encounters a black hole when D_1 does not. Note that all teh copied redexes are in evaluation contexts. Therefore if any component with a copy of M references itself directly or indirectly, it must be the case that l'' also references that component. Thus the two terms (denoted by D_n and D'_n since they appear at some point on the evaluation path) would be as follows (we are showing a direct self-dependency):

$$\begin{array}{rl} D_n = & [l'' \mapsto \mathbb{E}_1\{l_1\}, l' \mapsto \mathbb{E}_2\{l''\}, l_1 \mapsto \mathbb{E}_3\{l_1\}] \\ D'_n = & [l'' \mapsto \mathbb{E}_1\{l_1\}, l' \mapsto \mathbb{E}'_2\{l_1\}, l_1 \mapsto \mathbb{E}_3\{l_1\}] \end{array}$$

   Both records evaluate to a black hole.

As in case 1, the constructed evaluation sequence must be infinite since even if the number of substitutions performed in D_1 that do not appear in D'_1 is infinite, there must be an infinite number of steps that remain since creating an infinite number of copies of a marked l'' would require an infinite number of steps.

3. The component bound to l' is of the form $\mathbb{E}\{l\}$. In this case D, D_1, and D'_1 are of the following form:

$$\begin{array}{lcl} D & = & [l \mapsto M, l' \mapsto \mathbb{E}\{l\}, \dots] \\ D_1 & = & [l \mapsto M', l' \mapsto \mathbb{E}\{l\}, \dots] \\ D'_1 & = & [l \mapsto M, l' \mapsto \mathbb{E}\{M\}, \dots] \end{array}$$

   Note that since l' references l, l cannot reference l' because this would reduce the record to a black hole. Therefore if M' is the result of a substitution of a variable into M, that variable cannot be l'. In this case we start the constructed evaluation sequence for D'_1 as

$$\begin{array}{lcll} D'_1 & = & [l \mapsto M, l' \mapsto \mathbb{E}\{M\}, \dots] & \Rightarrow \\ D'_2 & = & [l \mapsto M', l' \mapsto \mathbb{E}\{M\}, \dots] & \\ D'_3 & = & [l \mapsto M', l' \mapsto \mathbb{E}\{M'\}, \dots] & \end{array}$$

   Then D_1 and D'_3 are as in case 2 above, and the construction given in that case is applicable.

In all of the given cases the assumption that D_1 has an infinite evaluation sequence and D'_1 does not leads to a contradiction. Note that since the record has a black-hole-free normal form w.r.t. evaluation, there may not be evaluation self-dependencies in any of the labels or a mutual evaluation dependency between the two labels. Thus no more cases are possible.

We have shown that it is not possible that a record D has a normal form and a diverging evaluation sequence. □

3.1 An Efficient Evaluation Strategy

Confluence of evaluation guarantees that no matter what paths an evaluation of a record takes, all of the resulting records can be evaluated to the same one. We do not want to fix the order of evaluating components since we would like to have the flexibility of modeling systems where progress can be made on evaluating a record before all of its components become available or where components may be evaluated in parallel.

However, for proving properties of our calculus it is convenient to impose a particular order of evaluation that we call *efficient evaluation strategy.* Intuitively, this strategy requires that if a component bound to l_1 needs a component bound to l_2 (i.e. the component bound to l_1 is of the form $\mathbb{E}\{l_2\}$) then the term bound to l_2 must be completely evaluated (i.e. it has neither term redexes nor substitution redexes) before the substitution into the component bound to l_1

is made. This strategy imposes a partial order on components. The strategy stops if it discovers a cycle of mutual dependencies.

The formal definition depends on the partial function next(D, l) that defines the label of the component in which the next evaluation step takes place in order to make progress on evaluation of the component bound to l in D.

Definition 11 (Next Component To Be Evaluated)**.** *Let $l \mapsto M \in D$. A function next$(D, l) : RTerm \times L(D) \to L(D) \cup \{\bullet\}$ is defined as follows:*

1. *If $M = \mathbb{E}\{R\}$ then next$(D, l) = l$,*
2. *If $M = \mathbb{E}\{\bullet\}$ or $M = \mathbb{E}\{l\}$ then next$(D, l) = \bullet$,*
3. *If $M = \mathbb{E}\{l'\}$ then:*
   (a) *If next(D, l') is undefined, next$(D, l) = l$,*
   (b) *If next$(D, l') = \bullet$ or l' is bound to $\mathbb{E}'\{l\}$ or there is a sequence of labels $l_1, \ldots, l_n \in L(D)$, $n \geq 1$, such that D is of the form*
   $$[l \mapsto \mathbb{E}\{l'\}, l' \mapsto \mathbb{E}_1\{l_1\}, \ldots, l_i \mapsto \mathbb{E}_i\{l_{i+1}\}, \ldots, l_n \mapsto \mathbb{E}_n\{l\}, \ldots]$$
   then next$(D, l) = \bullet$,
   (c) *Otherwise next$(D, l) =$ next(D, l').*
4. *Otherwise next(D, l) is undefined.*

If next(D, l) is undefined then the component bound to l is fully evaluated.

Let $\mathcal{L}$ denote an ordered sequence of distinct labels; $\mathcal{L}_1 \preceq \mathcal{L}_2$ means that $\mathcal{L}_1$ is a prefix of $\mathcal{L}_2$ or $\mathcal{L}_1 = \mathcal{L}_2$. An efficient evaluation strategy follows the sequence of labels in $\mathcal{L}$ as a sequence of "goals".

Definition 12 (Efficient Evaluation Strategy)**.** *Given a record D and a label l, an efficient evaluation strategy starting at l is a sequence of evaluation steps $D_1 \Rightarrow D_2 \Rightarrow \ldots \Rightarrow D_n$ s.t. for all $i < n$ next(D_i, l) is defined and not equal to $\bullet$ and an evaluation step $D_i \Rightarrow D_{i+1}$ evaluates the component bound to next(D_i, l). We denote this sequence as $D \overset{l}{\underset{e}{\Longrightarrow}}{}^* D_n$.*

Given a sequence $\mathcal{L} = l_1, l_2, \ldots l_n$ s.t. $l_i \in L(D)$ for all i, an efficient strategy w.r.t. $\mathcal{L}$ is a sequence $D \overset{l_1}{\underset{e}{\Longrightarrow}}{}^* D_1 \overset{l_2}{\underset{e}{\Longrightarrow}}{}^* \ldots \overset{l_n}{\underset{e}{\Longrightarrow}}{}^* D_n$ s.t. next(D_i, l_j) is undefined for all $j < i$ for $1 \leq i \leq n$ (i.e. each component l_j in $\mathcal{L}$ is fully evaluated before evaluation of l_i starts). Note that it is possible that next(D_n, l_n) is not undefined. An efficient strategy w.r.t. $\mathcal{L}$ is denoted $\overset{\mathcal{L}}{\underset{e}{\Longrightarrow}}{}^*$.

The efficient evaluation strategy stops if it discovers that a record component evaluates to a black hole since such records represent divergence (see Definition 15). Thus, if next$(D, l) = \bullet$, no evaluation takes place.

The strategy is called "efficient" because it evaluates a component only once - the first time it is needed. Since no unevaluated components are copied, no computation is duplicated. This is similar to a call-by-value or a call-by-need strategy. However, unlike the call-by-value strategy, it does not require that a component evaluates to a value before it can be substituted (traditionally only constants, variables, and λ-abstractions are considered values), only to a substitution-free normal form which includes errors. This is also technically different from a call-by-need strategy since the evaluation of a λ-application still follows a call-by-value strategy, copying arguments rather than sharing them.

If a record D evaluates to a normal form D' then efficient evaluation strategy with any choice of $\mathcal{L}$ that includes all labels in $L(D)$ reaches D'. The strategy detects cycles of substitution as early as possible since the evaluation follows component dependencies as far as possible before evaluating any of them.

Below is an evaluation sequence that follows the efficient strategy w.r.t. l_1. On the left on line i we show the value of $\text{next}(D_i, l_1)$. For simplicity we write just $\text{next}(l)$ instead of $\text{next}(D, l)$ since the record on each line is obvious.

$$\begin{array}{lll}
\text{next}(l_1) = l_3 & [l_1 \mapsto l_2, l_2 \mapsto l_3 + 2, l_3 \mapsto 1 + 3] & \Rightarrow \\
\text{next}(l_1) = l_2 & [l_1 \mapsto l_2, l_2 \mapsto l_3 + 2, l_3 \mapsto 4] & \Rightarrow \\
\text{next}(l_1) = l_2 & [l_1 \mapsto l_2, l_2 \mapsto 4 + 2, l_3 \mapsto 4] & \Rightarrow \\
\text{next}(l_1) = l_1 & [l_1 \mapsto l_2, l_2 \mapsto 6, l_3 \mapsto 4] & \Rightarrow \\
\text{next}(l_1) \text{ is undefined} & [l_1 \mapsto 6, l_2 \mapsto 6, l_3 \mapsto 4] &
\end{array}$$

In contrast the following step does not follow the efficient strategy:

$$\begin{array}{ll}
[l_1 \mapsto l_2, l_2 \mapsto l_3 + 2, l_3 \mapsto 1 + 3] & \Rightarrow \\
[l_1 \mapsto l_3 + 2, l_2 \mapsto l_3 + 2, l_3 \mapsto 1 + 3] & \Rightarrow \ldots
\end{array}$$

The latter record eventually evaluates to the same one as the final record in the efficient evaluation strategy sequence. However, in this evaluation sequence the redex $l_3 + 2$ was duplicated so it will have to be evaluated twice, possibly duplicating evaluation of $1 + 3$ as well.

Lemma 9 states a fairly obvious but important property of an efficient evaluation strategy. In order to prove it we define two notations of interactions between evaluation steps that will be used in the proof of the lemma and prove a supplemental result in Lemma 8.

Definition 13. *Given an evaluation sequence $D_1 \overset{l_1}{\Longrightarrow} D_2 \overset{l_2}{\Longrightarrow} D_3$, where $\overset{l_i}{\Longrightarrow}$ evaluates a component bound to l_i ($i = 1, 2$), the steps $D_1 \overset{l_1}{\Longrightarrow} D_2$ and $D_2 \overset{l_2}{\Longrightarrow} D_3$ are called independent if $l_1 \neq l_2$ and $D_1 \neq \mathbb{E}\{l_2\}$ and $D_2 \neq \mathbb{E}\{l_1\}$.*

Given an evaluation sequence $D_1 \overset{l}{\Longrightarrow} D_2 \overset{l_1,\ldots l_m}{\Longrightarrow^*} D_3$, where the sequence $D_2 \Rightarrow^* D_3$ evaluates components according to the ordered sequence of labels $l_1, \ldots l_m$ (the labels $l_1, \ldots l_m$ are not required to be different), a step $D_1 \overset{l_1}{\Longrightarrow} D_2$ is called independent from a sequence $D_2 \overset{l_1,\ldots l_m}{\Longrightarrow^*} D_3$ if $l \neq l_i$ for all $1 \leq i \leq m$, $D_1 \neq \mathbb{E}\{l_i\}$ for all $1 \leq i \leq m$, and none of the records in the sequence $D_2 \Rightarrow^* D_3$, except possibly D_3, are of the form $\mathbb{E}\{l\}$.

Lemma 8. *The following properties hold for independent evaluation steps, as defined in Definition 13:*

1. *Given two independent evaluation steps $D_1 \xRightarrow{l_1} D_2 \xRightarrow{l_2} D_3$, there exists a record D_2' s.t. $D_1 \xRightarrow{l_2} D_2' \xRightarrow{l_1} D_3$.*

2. *Given an evaluation sequence $D_1 \xRightarrow{l} D_2 \overset{l_1,\dots l_m}{\Longrightarrow^*} D_3$, where the first step is independent from the subsequent sequence, there exists D_2' s.t. $D_1 \overset{l_1,\dots l_m}{\Longrightarrow^*} D_2' \xRightarrow{l} D_3$, where the number and the order of labels in $D_1 \overset{l_1,\dots l_m}{\Longrightarrow^*} D_2'$ is exactly the same as originally given.*

Proof. The first part follows from an easy observation that the two evaluations happen in two different components and do not affect each other. The second part follows from the first one by induction on m by moving $\xRightarrow{l}$ through the sequence. □

Lemma 9. *If a record D has an evaluation normal form D' with no components bound to a black hole then any evaluation sequence that follows the efficient evaluation strategy with any permutation $\mathcal{L}$ of all labels in $L(D)$ reaches D'.*

Proof. We will prove the claim by contradiction. Suppose D has an evaluation normal form D'. Let $\mathcal{L}$ be some sequence of all labels in $L(D)$. Suppose an evaluation sequence $D \Rightarrow D_1 \Rightarrow D_2 \dots$ follows the efficient strategy with respect to $\mathcal{L}$ and does not reach D', i.e. it cannot be continued as an efficient strategy. Since D has a normal form, by Lemma 7 the sequence cannot diverge. Let D'' denote the last record in the sequence.

By confluence of evaluation (Lemma 6) there exists an evaluation sequence $D'' \Rightarrow^* D'$. Since $D'' \neq D'$ by our assumption, the sequence is not empty. By our assumption $D \Rightarrow^* D'' \Rightarrow^* D'$ does not follow the efficient strategy w.r.t. $\mathcal{L}$ since we assumed that the efficient strategy stops before it reaches the normal form. Let $D'' \Rightarrow D_1'$ and let us denote the label of the component reduced in this step as l_i. Since $\mathcal{L}$ contains all labels of D, it contains l_i.

We show that we can transform $D \Rightarrow^* D'' \Rightarrow D_1'$ into an efficient strategy sequence by moving l_i towards the beginning of the sequence until it gets to its place according to $\mathcal{L}$. The component bound to l_i is not fully evaluated in $D \Rightarrow^* D''$, therefore it cannot be used in a substitution in a given sequence since efficient strategy allows substitution of fully evaluated components only. To give intuition behind the procedure, consider $D''' \Rightarrow D'' \Rightarrow D_1'$ There may be two cases:

- The two steps are independent.
- $D''' \Rightarrow D''$ reduces a component that gets substituted in the step $D'' \Rightarrow D_1'$.

If the evaluation of l_i is independent from another step in the given efficient evaluation sequence then by Lemma 8 the two reductions can be switched. If there is a dependency then the two steps are considered a sequence and togethe can be switched with the next (to the left) independent step by part 2 of Lemma 8. We formalize the transformation procedure below.

Let S be a subsequence of steps in a sequence $D \Rightarrow^* D_1 \xrightarrow{l} D_2 \overset{l_1,\ldots,l_m}{\Longrightarrow^*} D_3 \Rightarrow^* D_1'$, where $D \Rightarrow^* D_1$ is the yet unmodified part of the given efficient strategy evaluation sequence. We assume that there are no substitutions of any components reduced in S to the left of S, i.e. in the sequence $D \Rightarrow^* D_1 \xrightarrow{l} D_2$.

We initialize S to a single step $D'' \xrightarrow{l_i} D_1'$. To mark S in an evaluation sequence we will use a notation $\overset{S}{\Longrightarrow^*}$. The transformation proceeds as follows:

1. If the sequence $D \Rightarrow^* D_1 \xrightarrow{l} D_2 \overset{S}{\Longrightarrow^*} D_3 \Rightarrow^* D_1'$ is an efficient evaluation sequence w.r.t. $\mathcal{L}$, then stop. If the sequence cannot be represented in the given form because there are no more steps to the left of S, also stop.

2. If the sequence S is independent from $D_1 \xrightarrow{l} D_2$ (according to Definition 13) then change the sequence to $D \Rightarrow^* D_1 \overset{S'}{\Longrightarrow^*} D_2' \xrightarrow{l} D_3 \Rightarrow^* D_1'$ according to Lemma 8, where S' reduces the same sequence of labels as S. We set $S = D_1 \overset{S'}{\Longrightarrow^*} D_2'$. Note that it is still the case that there are no substitutions of any components reduced in S to the left of S. We proceed to step 1.

3. If the sequence$S$ is not independent from $D_1 \xrightarrow{l} D_2$ then the component l is used in a substitution in one of the steps in $\overset{S}{\Longrightarrow^*}$ (note that by the initial assumptionit cannot be the case that $D_1 \xrightarrow{l} D_2$ uses any of the components in S). Then we change S to $D_1 \xrightarrow{l} D_2 \overset{S}{\Longrightarrow^*} D_3$ and go to step 1. Note that since $D \Rightarrow^* D_1 \xrightarrow{l} D_2$ follows the efficient strategy, there may not be any substitutions of the component l to the left of S since the component is not fully evaluated at this point.

The procedure terminates with an efficient evaluation strategy sequence w.r.t. $\mathcal{L}$ since the sequence transformations guarantee that the no component is substituted before it is fully evaluated, and the evaluation of l_i is shifted to the left until it is moved to its place according to the label ordering.

We can repeat the transformation procedure for every evaluation step in $D'' \Rightarrow^* D'$, moving the step that is the closest to the efficient strategy sequence into that sequence. Thus we obtain an efficient evaluation sequence w.r.t. $\mathcal{L}$ leading to the normal form which contradicts our initial assumption. □

Chapter 4

Computational Soundness of the Calculus

4.1 Definition of Computational Soundness

Computational soundness states that rewriting relation in the calculus preserves the meaning of terms. A term's meaning is given by its normal form w.r.t. evaluation relation if such a normal form exists, otherwise the "meaning" is divergence. The notion of Outcome in Definition 15 formalizes this idea. Some normal forms may be syntactically different, but have the same "meaning". The classification function (Definition 14) groups terms based on their "meaning".

4.1.1 Classification function.

In order to define a term's meaning, we partition all terms into *equivalence classes*. The function that assigns a class to a term is called a *classification*. Two elements of the same class have the same meaning (however, they may be further distinguished by supplying a context that uses them). For instance, at the term level it is reasonable to make constants 2 and 3 be in different classes since their meaning is clearly different. However, it is common to group all lambda abstractions in the same class since a function by itself is not distinguishable from any other function until it is applied.

It is possible to define different classification functions for the same calculus. For instance, one may wish to distinguish between different types of errors by further subdividing the **error** class. On the other hand, if one is only concerned with proving termination equivalence then all normal forms may be placed into one class[1]. A calculus may be computationally sound for one choice of classification and unsound for another.

The classification function used in this work is defined in Definition 14. For simplicity we use the same notation *Cl* for the classification function at both

[1] Records with at least one component bound to a black hole should be classified as diverging

levels of the calculus. This function is very similar to the one used in [7], except for the inclusion of a black hole. We also do not have term-level classes for variables: since record components contain closed terms, a label bound to a variable would be considered an error. The function is well-defined on α-equivalence classes both at the term and at the record level.

Definition 14 (Classification)**.** *The classification function $Cl : TTerm \to S_T$, where S_T is a set of equivalence classes of terms, is defined as follows:*

- $Cl(M) = \mathbf{eval}$ *if $M = \mathbb{E}\{R\}$, R is a redex. Such terms are called evaluable.*
- $Cl(c) = \mathbf{const}(c)$*, where* $\mathbf{const}(c_1) = \mathbf{const}(c_2)$ *if and only if* $c_1 = c_2$
- $Cl(\bullet) = \bullet$
- $Cl(\lambda x.N) = \mathbf{abs}$
- $Cl(\mathbb{E}\{l\}) = \mathbf{stuck}(l)$*, where* $\mathbf{stuck}(l_1) = \mathbf{stuck}(l_2)$ *if and only if* $l_1 = l_2$
- $Cl(M) = \mathbf{error}$ *if M does not belong to any of the above categories*

$Cl : RTerm \to S_R$, where S_R is a set of equivalence classes of records, is defined on records as follows:

- $Cl([l_1 \mapsto M_1, \ldots l_n \mapsto M_n]) = [l_1 \mapsto Cl(M_1), \ldots l_n \mapsto Cl(M_n)]$ *if* $Cl(M_i) \neq \bullet$ *for all i s.t.* $1 \leq i \leq n$
- $Cl([\ldots, l_i \mapsto \bullet, \ldots]) = \bot$

An equivalence class of a record D with no label bound to a black hole is an unordered collection of labeled term-level classes corresponding to components of D. For instance, $Cl([l_1 \mapsto \lambda x.x, l_2 \mapsto l_1 \ @ \ 1]) = [l_1 \mapsto \mathbf{abs}, l_2 \mapsto \mathbf{stuck}(l_1)]$.

Since a black hole represents an infinite substitution, the class of a record with a black-hole-bound component is $\bot$. Note that a record with a black hole in a non-evaluation context does not necessarily diverge, and thus is not classified as $\bot$: consider $[l \mapsto (\lambda x.1) \ @ \ \bullet] \Rightarrow [l \mapsto 1]$, the latter record is a normal form.

The following property is important for proving computational soundness:

Lemma 10 (Class Preservation)**.** *If $D_1 \hookrightarrow D_2$ then $Cl(D_1) = Cl(D_2)$.*

Proof. Straigtforward, by cases of pairs classes and non-evaluation reduction steps. □

The given classification groups all abstractions in one class. However, this does not mean that replacing an abstraction by any other one is a meaning preserving transformation. One can always distinguish two semantically different abstractions by considering them in a record with a term that applies the abstraction to an argument. If a transformation is provably meaning preserving then its results are the same no matter what other components appear in a record. Since we can assume that any abstraction bound to a label is applied to arbitrary terms in other components, transformations must preserve the actual behavior of abstractions. [7] formalizes this notion via record contexts which we do not present here due to lack of space.

4.1.2 Outcome and Computational Soundness.

Classification characterizes a term at a given moment, while outcome characterizes the "ultimate fate" of the term - what happens to the term if it gets evaluated as far as possible.

Definition 15 (Outcome)**.** *The outcome of a record D, denoted Outcome(D), is $Cl(D')$ where D is the normal form of D w.r.t. $\Rightarrow$ if D has a normal form or a symbol $\perp$ if evaluation of D diverges.*

Lemmas 6 and 7 guarantee that the outcome to be well-defined since every record either has a unique normal form or diverges on all evaluation paths (we identify a label bound to a black hole with divergence). The outcome formalizes the notion that the meaning of a term is the result of its evaluation.

Definition 16 (Meaning Preservation and Computational Soundness)**.** *A relation R is meaning preserving if MRN implies that Outcome$(M) =$ Outcome(N). A calculus is computationally sound if $\leftrightarrow$ is meaning preserving.*

By confluence and uniform normalization (Lemmas 6, 7) the relation $\Rightarrow$ is meaning preserving.

4.2 Proof Methods and Their Applicability

Historically various methods have been used for proving computational soundness. A traditional method originating from Plotkin in [11] requires, in particular, *confluence of the rewriting relation* in the calculus. However, many recently developed calculi model such inherently non-confluent features of programming languages as mutually dependent components. The repertoire of proof methods has been expanded to relax requirements on the calculus. In this section we review some of these proof methods and discuss why they are not applicable to our calculus.

4.2.1 Failure of Confluence and Standardization Proof Method

The traditional method for computational soundness proofs has three requirements: confluence of the rewriting relation, standardization (a property that relates the rewriting relation and the evaluation relation), and the class preservation property defined in Lemma 10 (see [7] for detailed discussion). However, in our system $\rightarrow$ is non-confluent. The non-confluence example below is based on that in [3]. It also appears in the call-by-value version of our calculus described in [7, 10].

Example 1 (Non-confluence of $\rightarrow$)**.** *Consider a record $[l_1 \mapsto \lambda x.l_2, l_2 \mapsto \lambda y.l_1]$. It has two non-evaluation substitution redexes. By choosing each of the two redexes we obtain these two records: $[l_1 \mapsto \lambda x.\lambda y.l_1, l_2 \mapsto \lambda y.l_1]$ and $[l_1 \mapsto$*

$$\begin{tikzcd}
D_1 \arrow[r, Rightarrow, dashed, "*"] \arrow[d, hook] & D_4 \arrow[d, hook, dashed, "*"] \\
D_2 \arrow[r, Rightarrow, "*"] & D_3
\end{tikzcd}
\qquad
\begin{tikzcd}
D_1 \arrow[r, Rightarrow, "*"] \arrow[d, hook] & D_2 \arrow[r, Rightarrow, dashed, "*"] & D_4 \arrow[d, hook, dashed, "*"] \\
D_3 \arrow[rr, Rightarrow, dashed, "*"] & & D_5
\end{tikzcd}$$

Figure 4.1: Lift and Project properties.

$\lambda x.l_2, l_2 \mapsto \lambda y.\lambda x.l_2]$. *The only reductions that can originate from these records are substitutions. No matter what substitutions we perform on the records, they cannot be reduced to a common one since in the first one both components always reference l_1, and in the second record they reference l_2.*

It is important to observe that both reductions in this example are non-evaluation steps. In section 3 we showed a similar example, but with two evaluation step. Then both records evaluate to $[l_1 \mapsto \bullet, l_2 \mapsto \bullet]$, preserving confluence of $\Rightarrow$.

4.2.2 Failure of Lift and Project Method

In [7, 10] we have proposed a novel method for proving computational soundness and applied it to a call-by-value calculus of records. The call-by-value nature of the substitution disallows substituting non-evaluated components into other components.

In [7, 10] we use an approach based on three properties of the calculus: the *lift*, and *project* properties defined in Definition 17, and the class preservation property in Section 4.1 to prove computational soundness of a call-by-value calculus of recursively-scoped records. The project property "projects" a given evaluation sequence down, and the lift property "lifts" a given sequence up, according to the diagram layout in Figure 4.1[2].

Definition 17 (Lift and Project)**.** *A calculus has the lift property if, given $D_1 \hookrightarrow D_2 \Rightarrow^* D_3$, there exists D_4 s.t. $D_1 \Rightarrow^* D_4 \hookrightarrow^* D_3$. A calculus has the project property if, given $D_1 \Rightarrow^* D_2$ and $D_1 \hookrightarrow D_3$, there exist D_4, D_5 s.t. $D_2 \Rightarrow^* D_4 \hookrightarrow^* D_5$ and $D_3 \Rightarrow^* D_5$.*

The lift and project properties hold in the call-by-value system. However, despite the fact that the current system is very similar to the one considered in [7, 10], the call-by-name nature of substitution breaks the lift and the project properties.

Example 2 (Lack of Lift and Project Properties)**.** *The following example shows that the project property fails in our calculus: given the term reduction on top*

[2] In diagrams double arrows represent $\Rightarrow$, single arrows $\rightarrow$, arrows with a hook are $\hookrightarrow$. Solid arrows are the given relations, dashed arrows are those claimed to exist by the property. See Definition 8 for closure notations.

and the non-evaluation substitution, we cannot complete the diagram. The substitution (a non-evaluation step) has duplicated an evaluation term redex and thus requires a non-evaluation arrow to point "up" to complete the diagram, which contradicts the project property.

$$\begin{tikzcd}
{[l_1 \mapsto 2+3, l_2 \mapsto \lambda x.l_1]} \arrow[r, Rightarrow] \arrow[dd, hook] & {[l_1 \mapsto 5, l_2 \mapsto \lambda x.l_1]} \arrow[d, hook] \\
 & {[l_1 \mapsto 5, l_2 \mapsto \lambda x.5]} \\
{[l_1 \mapsto 2+3, l_2 \mapsto \lambda x.2+3]} \arrow[r, Rightarrow] & {[l_1 \mapsto 5, l_2 \mapsto \lambda x.2+3]} \arrow[u, hook]
\end{tikzcd}$$

This is also a counterexample to lift if we assume that we are given the sequence $[l_1 \mapsto 2+3, l_2 \mapsto \lambda x.l_1] \hookrightarrow [l_1 \mapsto 2+3, l_2 \mapsto \lambda x.2+3] \Rightarrow [l_1 \mapsto 5, l_2 \mapsto \lambda x.2+3]$ *as a premise, with the goal of completing the lift diagram.*

4.2.3 Applicability of Other Diagram-Based Methods.

The lift and project method has been extended and generalized in [14]. While it is possible that a form of the approach presented there, known as *lift/project when terminating* (or LPT), is applicable, we have not been able to construct such a proof.

A black hole, which is technically a normal form, may require a modification of the LPT approach. In our system a non-evaluation step may convert a record with a component evaluating to black hole to a diverging record, as shown below. Diagram-based methods generally do not equate diverging terms with normal forms. Note that the outcome of both records is $\perp$ so the meaning is preserved.

$$\begin{array}{lll}
[l_1 \mapsto l_2 \mathbin{@} 2, l_2 \mapsto \lambda x.l_1] & \Rightarrow & [l_1 \mapsto (\lambda x.l_1) \mathbin{@} 2, l_2 \mapsto \lambda x.l_1] \quad \Rightarrow \\
[l_1 \mapsto l_1, l_2 \mapsto \lambda x.l_1] & \Rightarrow^* & [l_1 \mapsto \bullet, l_2 \mapsto \lambda x.l_1] \\
[l_1 \mapsto l_2 \mathbin{@} 2, l_2 \mapsto \lambda x.l_2 \mathbin{@} 2] & \Rightarrow & [l_1 \mapsto (\lambda x.l_2 \mathbin{@} 2) \mathbin{@} 2, l_2 \mapsto \lambda x.l_2 \mathbin{@} 2] \quad \Rightarrow \\
[l_1 \mapsto l_2 \mathbin{@} 2, l_2 \mapsto \lambda x.l_2 \mathbin{@} 2] & \Rightarrow & \ldots
\end{array}$$

4.3 Proof of Meaning Preservation of the Term Reduction

Lemma 11. *$D_1 \overset{T}{\hookrightarrow} D_2$ implies Outcome(D_1) = Outcome(D_2).*

The meaning preservation property of a term reduction can be proven using the lift and project approach. The proof is similar to that for the call-by-value calculus in [7]. Below we outline the proof. The details can be easily checked.

A redex in a term M is any subterm such that $M = \mathbb{C}\{R\}$ and $\mathbb{C}\{R\}$ matches one of the calculus reduction rules. Given a reduction $M \rightarrow M'$ that reduces a redex R_1 and another redex R_2 in M, a *residual* of R_2 is any redex in M' that results from preserving, moving, or duplicating R_2. Note that R_2 may be somewhat transformed if the reduction copies an expression into one of free variables inside R_2. It is also possible that R_2 has no residuals (for instance, if it appeared inside an argument of a λ-abstraction that is not used in the body of the abstraction) See [5] for more details on the definition.

The proof uses the machinery of marked redexes and residuals. In order to prove that a term reduction $D_1 \overset{T}{\hookrightarrow} D_2$ preserves the meaning of a record, we mark the redex reduced by this reduction in D_1. The method of marking is unimportant. In this presentation we show marked redexes by underlining. After a reduction all residuals of marked redexes are still marked. A subterm that results from reducing a redex is not marked.

We use $\widetilde{M}, \widetilde{N}$, etc. to denote terms in which some term redexes are marked and $\widetilde{D}$, etc. to denote records with marked term redexes. We say that a term M is an erasure of a marked term $\widetilde{M}$ if M is obtained from M by removing all markings of redexes.

Given $D_1 \overset{T}{\hookrightarrow} D_2$, we start by marking the redex reduced by this reduction in D_1. Let $\widetilde{D}_1$ be the result of such marking. For instance,

$$\begin{array}{rl} \widetilde{D}_1 = & [l_1 \mapsto (\underline{(\lambda x.x) \mathbin{@} (\lambda y.y)}) \mathbin{@} 2, l_2 \mapsto 3 + l_1] \quad \hookrightarrow \\ D_2 = & [l_1 \mapsto (\lambda y.y) \mathbin{@} 2, l_2 \mapsto 3 + l_1] \end{array}$$

(here D_2 does not have any marked redexes, and this is not marked with a tilde)

When the first record gets evaluated, the marked redex may get duplicated, either within the same component, or, as illustrated below, it can be copied into a different component:

$$\begin{array}{rl} \widetilde{D}_1 = & [l_1 \mapsto (\underline{(\lambda x.x) \mathbin{@} (\lambda y.y)}) \mathbin{@} 2, l_2 \mapsto 3 + l_1] \ \Rightarrow \\ & [l_1 \mapsto (\underline{(\lambda x.x) \mathbin{@} (\lambda y.y)}) \mathbin{@} 2, l_2 \mapsto 3 + (\underline{(\lambda x.x) \mathbin{@} (\lambda y.y)}) \mathbin{@} 2] \end{array}$$

In the above example the original redex has two residuals in the resulting record. In general, an evaluation sequence starting at $\widetilde{D}_1$ may produce multiple copies of the marked redex in multiple components and marked redexes may be contained within other marked redexes. Since no redexes are marked in D_2, its evaluation does not produce any marked redexes either.

We use the well-known (see, e.g., [5]) notion of developments:

Definition 18 (Developments)**.** *A reduction step $\widetilde{D} \to \widetilde{D'}$ is called a development step and is denoted by $\underset{dev}{\to}$ if it reduces a marked redex. A sequence of development steps is called a development and is denoted by $\underset{dev}{\to}^*$. We extend these notations to the evaluation and non-evaluation relations in a straightforward way.*

A development $\widetilde{D} \underset{dev}{\to}^* \widetilde{D'}$ is called a complete development when $\widetilde{D'}$ does not have any marked redexes. We denote a complete development by $\underset{cd}{\to}^*$.

We use $\to$, $\Rightarrow$, etc. to denote the respective relations on marked terms. These relations may or may not reduce a marked redex, thus they may or may not be development steps. In other words, on marked terms $\underset{dev}{\to}\subset\to$, $\underset{dev}{\Rightarrow}\subset\Rightarrow$, etc.

The proof of Lemma 13 (lift and project properties of the calculus) uses the properties of developments given in Lemma 12.

Lemma 12. *The following properties of developments hold:*

1. *Developments are finite, i.e. given a record $\widetilde{D}$, there is no infinite sequence of $\underset{dev}{\to}$ steps originating at $\widetilde{D}$. More precisely, developments are bounded, i.e. for any record $\widetilde{D}$ there exists a number n such that all sequences $\underset{dev}{\to^*}$ starting at $\widetilde{D}$ are no longer than n steps.*

2. *Developments are confluent, i.e. if $\widetilde{D} \underset{dev}{\to^*} \widetilde{D}_1$ and $\widetilde{D} \underset{dev}{\to^*} \widetilde{D}_2$ then there exists $\widetilde{D}_3$ such that $\widetilde{D}_1 \underset{dev}{\to^*} \widetilde{D}_3$ and $\widetilde{D}_2 \underset{dev}{\to^*} \widetilde{D}_3$.*

3. **Elementary Lift:** *Given $\widetilde{D}_1 \underset{dev}{\hookrightarrow} \widetilde{D}_2$ and $\widetilde{D}_2 \Rightarrow \widetilde{D}'_2$, there exits $\widetilde{D}_1$ such that $\widetilde{D}_1 \Rightarrow \widetilde{D}'_1$ and $\widetilde{D}'_1 \underset{dev}{\to^*} \widetilde{D}'_2$.*

4. **Elementary Project:** *Given $\widetilde{D}_1 \underset{dev}{\hookrightarrow} \widetilde{D}_2$ and $\widetilde{D}_1 \Rightarrow \widetilde{D}'_1$, there exists $\widetilde{D}'_2$ such that $\widetilde{D}_2 \Rightarrow \widetilde{D}'_2$ and $\widetilde{D}'_1 \underset{dev}{\to^*} \widetilde{D}'_2$.*

5. **Standardization of Developments:** *given $\widetilde{D}_1 \underset{dev}{\to^*} \widetilde{D}_2$, there exists $\widetilde{D}_3$ s.t. $\widetilde{D}_1 \underset{dev}{\Rightarrow^*} \widetilde{D}_3 \underset{dev}{\hookrightarrow^*} \widetilde{D}_2$.*

Standardization of developments and the two elementary diagrams are illustrated in figure 4.2.

Proof. All of the above properties can be proven first for the term calculus, and then reduction sequences for individual components can be combined to obtain the corresponding properties for records.

Boundedness of developments is proven for terms analogously to the proof of finiteness of developments in [5]. The bound of developments in records is computed as the sum of bounds for each record component since developments of term redexes involve only term reductions, and term reductions in different components are independent.

The proof of confluence of developments is straightforward.

Properties 3 and 4 are proven by cases of all possible pairs of the given evaluation and non-evaluation development steps. Note that the starting term $\widetilde{D}_1$ may have more than one marked redex.

Standardization of developments is proven by showing that $\widetilde{D}_1 \underset{dev}{\hookrightarrow} \widetilde{D}_2 \underset{dev}{\Rightarrow} \widetilde{D}_3$ implies that there exists $\widetilde{D}_4$ such that $\widetilde{D}_1 \underset{dev}{\Rightarrow} \widetilde{D}_4 \underset{dev}{\to^*} \widetilde{D}_3$, i.e the evaluation step

$$\begin{tikzcd}
\widetilde{D}_1 \arrow[r, Rightarrow, dashed] \arrow[d, hook, "dev"'] & \widetilde{D}'_1 \arrow[d, dashed, "dev"', "*"] \\
\widetilde{D}_2 \arrow[r, Rightarrow] & \widetilde{D}'_2
\end{tikzcd}
\qquad
\begin{tikzcd}
\widetilde{D}_1 \arrow[r, Rightarrow] \arrow[d, hook, "dev"'] & \widetilde{D}'_1 \arrow[d, dashed, "dev"', "*"] \\
\widetilde{D}_2 \arrow[r, Rightarrow, dashed] & \widetilde{D}'_2
\end{tikzcd}
\qquad
\begin{tikzcd}
& \widetilde{D_3} \arrow[dr, hook, dashed, "dev"', "*"] & \\
\widetilde{D}_1 \arrow[ur, Rightarrow, dashed, "*", "dev"'] \arrow[rr, "dev"', "*"] & & \widetilde{D}_2
\end{tikzcd}$$

Figure 4.2: Elementary Lift and Project for Developments and Standardization of Developments.

is "pushed forward". Boundedness of developments implies that the process of "pushing forward" evaluation steps terminates. □

Evaluation of a transformed record parallels the evaluation of of the original one. More specifically, we state and prove the following two claims.

Lemma 13. *Assume that the only redex marked in $\widetilde{D}'_1$ is a non-evaluation redex and $\widetilde{D}'_1 \hookrightarrow D_2$ reduces that redex (note that then the reduction can also be written as $\widetilde{D'_1} \underset{cd}{\hookrightarrow} D_2$ since it reduces a marked redex and the resulting record does not have any marked redexes in it). Then the following two properties hold:*

1. **Lift.** *If $D_2 \Rightarrow^* D'_2$ then there exists $\widetilde{D}'_1$ s.t. $\widetilde{D}_1 \Rightarrow^* \widetilde{D}'_1$ and $\widetilde{D}'_1 \underset{cd}{\hookrightarrow^*} D'_2$.*

2. **Project.** *If $\widetilde{D}_1 \Rightarrow^* \widetilde{D}'_1$ then there exist $\widetilde{D}''_1$ and D'_2 s.t. $\widetilde{D}'_1 \Rightarrow^* \widetilde{D}''_1$, $D_2 \Rightarrow^* D'_2$, and $\widetilde{D}''_1 \underset{cd}{\hookrightarrow^*} D'_2$.*

Note that we do not put tilde on D_2 and D'_2 since they do not have any marked redexes.

Figure 4.3 illustrates the structure of the proof of lift. The proof is by induction on the number of steps in the evaluation sequence $D_2 \Rightarrow^* D'_2$. The inductive hypothesis guarantees the existence of the complete development sequence $\widetilde{D}'''_1 \underset{dev}{\hookrightarrow^*} D''_2$. Repeated application of the elementary lift property in Lemma 12.3 to each step of $\widetilde{D}'''_1 \underset{cd}{\hookrightarrow^*} D''_2$ allows us to "lift" the step $D'''_2 \Rightarrow D'_2$ all the way to $\widetilde{D}'''_1 \Rightarrow \widetilde{D}''''_1$. The standardization of developments property in Lemma 12.5 allows us to rewrite the sequence $\widetilde{D}''''_1 \underset{cd}{\hookrightarrow^*} D'_2$ as a sequence of evaluation development steps followed by a sequence of non-evaluation development steps. The erasure of all markings results in the desired lift property for a term redex in the calculus of records.

The proof of the project property is illustrated in Figure 4.4 and is similar to that of lift, except the fact that it uses the elementary project property in Lemma 12.4 to "project" the evaluation step $\widetilde{D}'''_1 \Rightarrow \widetilde{D}'_1$ down to $D''_2 \Rightarrow D'_2$ in place of analogous use of the elementary lift diagram 12.3 in the lift property. Again, by erasing all the markings we get the desired project property for a non-evaluation term reduction.

$$\begin{tikzcd}
\widetilde{D}_1 \arrow[r, Rightarrow, dashed, "*"] \arrow[dd, hook, "cd"'] \arrow[ddr, phantom, "\text{IH}"] & \widetilde{D}_1''' \arrow[r, Rightarrow, dashed] \arrow[dd, hook, dashed, "cd"', "*"] \arrow[ddr, phantom, "\text{Lemma 12.3,5}"] & \widetilde{D}_1'''' \arrow[d, Rightarrow, dashed, "dev"', "*"] \\
 & & \widetilde{D}_1'' \arrow[d, hook, dashed, "cd"', "*"] \\
D_2 \arrow[r, Rightarrow, "*"] & D_2'' \arrow[r, Rightarrow] & D_2'
\end{tikzcd}$$

Figure 4.3: Proof of Lift Property of Developments of Term Redexes

$$\begin{tikzcd}
\widetilde{D}_1 \arrow[r, Rightarrow, "*"] \arrow[dd, hook, "cd"'] \arrow[ddr, phantom, "\text{IH}"] & \widetilde{D}_1''' \arrow[r, Rightarrow] \arrow[dd, hook, dashed, "cd"', "*"] \arrow[ddr, phantom, "\text{Lemma 12.4,5}"] & \widetilde{D}_1' \arrow[d, Rightarrow, dashed, "dev"', "*"] \\
 & & \widetilde{D}_1'' \arrow[d, hook, dashed, "cd"', "*"] \\
D_2 \arrow[r, Rightarrow, dashed, "*"] & D_2'' \arrow[r, Rightarrow, dashed] & D_2'
\end{tikzcd}$$

Figure 4.4: Proof of Project Property of Developments of Term Redexes

Since non-evaluation term reductions satisfy lift and project diagrams, such reductions are meaning-preserving, as proven in [10] and [7].

4.4 Meaning Preservation of Substitution

We show that substitution preserves the outcome of a record. Our proof is inductive on the number of steps in the evaluation sequence ultimately leading to the record's $\Rightarrow$-normal form (if there is one). One of the key ideas of the proof is to use the efficient evaluation strategy (see Definition 12) to guarantee that each component is evaluated only once, the first time it is needed. Thus all of the record components that reference a label will use a substitution-free normal form of the component bound to the label if such a normal form exists, and no copy of the component would require any further evaluation. If there is no such normal form then the evaluation of the record diverges or one of the components becomes a black hole. We show that a substitution step preserves successful (i.e. black-hole free) termination of record evaluation.

4.4.1 Multi-hole Contexts and Similarity.

As a necessary formalism for the computational soundness proof we define multi-hole contexts, i.e. "terms" with several holes that can be filled in with subterms.

A zero-hole context is a term with no holes at all, i.e. just a regular term. Note that the one-hole context defined in Definition 1 is a partial case of a multi-hole context. However, it is more convenient to define it separately.

Definition 19 (Multi-hole contexts)**.** *A multi-hole context* $\mathbb{M}$ *is inductively defined as*

$$\mathbb{M} \quad ::= \quad \Box \mid M \mid \lambda x.\mathbb{M} \mid \mathbb{M} + \mathbb{M} \mid \mathbb{M} \mathbin{@} \mathbb{M}$$

A multi-hole context is an n*-hole context if the total number of holes in it is* n*. Note that* $n \geq 0$*.*

An n*-hole context* $\mathbb{M}$ *is called a non-evaluation multi-hole context if for any* $0 \leq i \leq n$ *and for any* $n-1$ *terms* $M_1, \dots M_{i-1}, M_{i+1}, \dots M_n$ *the one-hole context* $\mathbb{M}\{M_1, \dots, M_{i-1}, \Box, M_{i+1}, \dots, M_n\}$ *is a non-evaluation context. Such contexts are denoted by* $\mathbb{M}_{\mathbb{N}}$*.*

A multi-hole context $\mathbb{M}$ *is an evaluation context in* i*-th position if for every* $M_1, \dots, M_{i-1}, M_{i+1}, \dots M_n$ $\mathbb{M}\{M_1, \dots, M_{i-1}, \Box, M_{i+1}, \dots, M_n\}$ *is an evaluation context. We denote an evaluation multi-hole context by* $\mathbb{M}_{\mathbb{E}}$*. For evaluation multi-hole contexts in the further proofs we assume, without loss of genrality, that the evaluation hole is always in the first hole position.*

As an example, consider a 3-hole context $(\lambda x.\Box) \mathbin{@} (\Box + \Box)$. If we fill it with terms $5, (\lambda x.x) \mathbin{@} 1$, and 2, we get the term $(\lambda x.5) \mathbin{@} (((\lambda x.x) \mathbin{@} 1) + 2)$. The context is an evaluation context in the second position since no matter what terms fill the first and the third holes, the resulting context is an evaluation context. The following λ-abstraction is a non-evaluation 2-hole context: $\lambda x.\Box \mathbin{@} \Box$.

Lemma 14 (Multi-hole Context Cases)**.** *Given a multi-hole context* $\mathbb{M}$ *with* $n \geq 1$*, it is either a non-evaluation context or an evaluation context in exactly one position* i*.*

Proof. One may observe that a multi-hole context (for $n > 0$) is either a non-evaluation context or an evaluation context in exactly one position. Consider representation of a context as a tree and consider the path from the root down the tree that follows the structure of an evaluation context in Definition 1. Specifically, the path does not descend under a lambda and always chooses the left subterm of an application or an operation, unless the left subterm of an application is a lambda abstraction (then the right subtree is chosen) or the left subtree of an operation is a constant (also the right subtree is chosen). As the result of this traversal exactly one of the following two cases takes place:

1. The constructed path reaches a context hole. In this case no matter what terms fill in the other holes in the context, the resulting context is an evaluation context, i.e. the multi-hole context is an evaluation context in the position corresponding to the encountered hole.

2. The constructed path does not encounter a context hole. Since the path follows definition of an evaluation context, no hole in the multi-hole context forms an evaluation context, regardless of what terms the other contexts are filled with.

□

We use the notation $\mathbb{M} \equiv \mathbb{M}'$ to denote the fact that $\mathbb{M}$ and $\mathbb{M}'$ are identical contexts. Note that since a context may have bound variables appearing in the terms that fill it, it does not make sense to extend the notion of α-renaming to contexts. However, when we write $\mathbb{M}\{M_1, \ldots, M_n\} = \mathbb{M}'\{M_1', \ldots, M_n'\}$, we assume that the resulting terms belong to the same α-equivalence class. Note that in this case it is possible to apply a sequence of alpha-renamings to $\mathbb{M}\{M_1, \ldots, M_n\}$ to convert it to $\mathbb{M}''\{M_1'', \ldots, M_n''\}$ so that $\mathbb{M}' \equiv \mathbb{M}''$ and $M_i' \equiv M_i''$ for all $1 \leq i \leq n$ (this property follows directly from definitions of $=_\alpha$ and $\equiv$).

The following lemma formalizes the fact that terms in non-evaluation positions do not affect the class.

Lemma 15. *If $M_1 = \mathbb{M}_{\mathbb{N}}\{N_1, \ldots, N_n\}$ and $M_2 = \mathbb{M}_{\mathbb{N}}\{N_1', \ldots, N_n'\}$ then $Cl(M_1) = Cl(M_2)$.*

Proof. By cases of classification and by induction on the structure of an evaluation context. □

Definition 20. *A record D_1 is called (M_1, M_2)-similar to a record D_2 (denoted $D_1 \sim^{M_1}_{M_2} D_2$) if there exist multi-hole contexts $\mathbb{M}_1, \ldots, \mathbb{M}_n$ s.t.*

$$D_1 = [l_1 \mapsto \mathbb{M}_1\{M_1, \ldots, M_1\}, \ldots, l_n \mapsto \mathbb{M}_n\{M_1, \ldots, M_1\}],$$
$$D_2 = [l_1 \mapsto \mathbb{M}_1\{M_2, \ldots, M_2\}, \ldots, l_n \mapsto \mathbb{M}_n\{M_2, \ldots, M_2\}].$$

Note that if $D_1 \sim^{M_1}_{M_2} D_2$ then $D_2 \sim^{M_2}_{M_1} D_1$.

Definition 20 simply means that D_2 is the result of replacing some occurrences of M_1 in D_1 by M_2.

Similar to the convention on equality of contexts filled with terms, we assume that the representatives of the α-equivalence classes of all components in the two records are chosen in such a way that all $\mathbb{M}_i$ in D_1 are syntactically equivalent to the corresponding $\mathbb{M}_i$ in D_2. We also assume that M_1 and M_2 are closed terms, i.e. $FV(M_1) = FV(M_2) = \emptyset$. Since the notion of $\sim^{M_1}_{M_2}$ will be used only in cases when M_1 is a label and M_2 is a component bound to that label, the assumption holds in cases considered in this presentation. In addition, we assume that all copies of M_1 in D_1 are syntactically equivalent to each other and so are all copies of M_2 in D_2.

4.4.2 The Proof of Meaning Preservation of a Substitution Step.

The following lemma states the key property used in meaning preservation proof for a substitution step.

Lemma 16. *Let $D_1 = [l \mapsto M, l' \mapsto \mathbb{N}\{l\}, \ldots] \overset{S}{\hookrightarrow} [l \mapsto M, l' \mapsto \mathbb{N}\{M\}, \ldots] = D_2$ and $D_1 \Rightarrow^* D_1'$ and let $\mathcal{L} \preceq l, l', l_1, \ldots, l_n$, where $l_1 \ldots l_n$ is a sequence of labels in $L(D_1)$, $n \geq 0$, and $l \neq l_i$, $l' \neq l_i$ for all $1 \leq i \leq n$. Then*

$$\begin{tikzcd}
D_1 \arrow[r, Rightarrow, "*"] \arrow[d, hook] & D_1' \arrow[d, squiggly, dash, "\sim^l_M"] \\
D_2 \arrow[r, Rightarrow, dashed, "*"] & D_2'
\end{tikzcd}
\qquad
\begin{tikzcd}
D_1 \arrow[rr, Rightarrow, dashed, "*"] \arrow[d, hook] & & D_1' \arrow[d, squiggly, dash, "\sim^l_M"] \\
D_2 \arrow[r, Rightarrow, "*"] & D_2' \arrow[r, Rightarrow, dashed, "*"] & D_2''
\end{tikzcd}$$

Figure 4.5: The claims of Lemma 16.

- *If* $D_1 \overset{\mathcal{L}}{\underset{e}{\Longrightarrow}}^* D_1'$ *then there exists* D_2' *s.t.* $D_2 \overset{\mathcal{L}}{\underset{e}{\Longrightarrow}}^* D_2'$, $D_1' \sim^l_M D_2'$, *and* $\mathit{Outcome}(D_1') = \bot$ *if and only if* $\mathit{Outcome}(D_2') = \bot$.
- *If* $D_2 \overset{\mathcal{L}}{\underset{e}{\Longrightarrow}}^* D_2'$ *then there exist* D_1', D_2'' *s.t.* $D_1 \overset{\mathcal{L}}{\underset{e}{\Longrightarrow}}^* D_1'$ *and* $D_2' \overset{l''}{\underset{e}{\Longrightarrow}}^* D_2''$, $D_2'' \sim^l_M D_1'$, *and* $\mathit{Outcome}(D_1') = \bot$ *if and only if* $\mathit{Outcome}(D_2') = \bot$.

where l'' *is the last label in* $\mathcal{L}$. *Note that it may be the case that* $l = l'$.

Note that there is a slight asymmetry between the two cases: in the second case additional evaluation steps on the D_2' may be needed. This is because D_2' may be in the process of evaluating M where D_1' may have a single substitution step for l. Thus extra steps in D_2' are needed to finish evaluating M, and the resulting records D_1' and D_2'' satisfy the conditions of the lemma.

Before proving Lemma 16 we prove several properties used in the proof.

Lemma 17. *Let* M *be a closed term and let* $D_1 = [l_1 \mapsto \mathbb{E}\{M\}, l_2 \mapsto M_2, \ldots, l_n \mapsto M_n]$ *and suppose* $D \overset{l_1}{\underset{e}{\Longrightarrow}}^* [l_1 \mapsto \mathbb{E}\{M'\}, l_2 \mapsto M_2', \ldots, l_n \mapsto M_n'] = D_2$ *so that no further evaluation steps within* M' *(without modifying* $\mathbb{E}$*) are possible. We also assume that* $\mathbb{E}\{M'\} \neq \mathbb{E}'\{\bullet\}$ *and* $\mathbb{E}\{M'\} \neq \mathbb{E}'\{l_1\}$ *for any* $\mathbb{E}'$. *Suppose* $D_2 \underset{e}{\Longrightarrow}^* D_3$ *s.t.* $l \mapsto \mathbb{E}_1\{M\} \in D_3$, *i.e. a term equal to* M *needs to be evaluated in* D_3. *Then there exists* D_4 *s.t.* $D_3 \underset{e}{\Longrightarrow}^* D_4$ *and* $l \mapsto \mathbb{E}_1\{M'\} \in D_4$, *i.e.* M *evaluates to* M'.

Proof. The same term redexes in M are evaluated the same way in all copies. Thus we only need to show that substitution always substitutes the same results and it is not possible to encounter a black hole in evaluation of such copies.

The efficient evaluation strategy requires that all components that are substituted into M are fully evaluated. Thus when these components are needed again in subsequent evaluations of a copy of M, exactly the same terms will be substituted.

Now we show that none of the substitutions form a cycle that produces a black hole. Suppose such a cycle is possible, i.e. when evaluating a copy of M in the component bound to l we obtain a cycle. For simplicity we consider a cycle of length 2; longer cycles or cycles of length 1 are handled similarily. A cycle occurs when l is of the form $\mathbb{E}\{l'\}$ and l' is of the form $\mathbb{E}\{l\}$. This means that evaluation of M requires l'. However, then l' is also needed for evaluation of the original M in l_1, and that evaluation needs of l'', so a cycle is encountered in evaluation of M, but we assumed that evaluation of M does not produce a cycle.

Thus a cycle is impossible, and a copy of M successfully evaluates to M'. □

Lemma 18. *If $D_1 \sim^l_M D_2$ and*

$$\begin{aligned} D_1 &= [l_1 \mapsto \mathbb{M}_{\mathbb{N}1}\{l,\ldots,l\},\ldots,l_n \mapsto \mathbb{M}_{\mathbb{N}n}\{l,\ldots,l\}], \\ D_2 &= [l_1 \mapsto \mathbb{M}_{\mathbb{N}1}\{M,\ldots,M\},\ldots,l_n \mapsto \mathbb{M}_{\mathbb{N}n}\{M,\ldots,M\}] \end{aligned}$$

then $next(D_1, l) = next(D_2, l)$ for every $l \in L(D)$.

Proof. Terms in non-evaluation contexts do not affect $\text{next}(D, l)$ since the next component to be evaluated is determined only by terms in evaluation contexts. □

The following lemma serves as a basis for induction on the number of steps in a given evaluation sequence.

Lemma 19. *Suppose $D \sim^l_M D'$, l' is a label in D and D' such that $next(D, l') = l'$, $next(D', l') = l'$, the component bound to a label l' is of the form $\mathbb{M}_{\mathbb{N}}\{l,\ldots,l\}$ in D, and $\mathbb{M}_{\mathbb{N}}\{M,\ldots,M\}$ in D'. Let $\underset{e}{\overset{l'}{\Longrightarrow}}$ denote the first step in the efficient evaluation strategy that starts at l'. Then $D \underset{e}{\overset{l'}{\Longrightarrow}} D_1$, $D' \underset{e}{\overset{l'}{\Longrightarrow}} D'_1$, and $D_1 \sim^l_M D'_1$.*

Proof. Since $\text{next}(D, l') \neq \bullet$ (and the same for D'), the evaluation step is not a record-level rule (B1) or (B2). We have two cases:

1. The evaluation step is by rule (TE). Note that the context containing the terms that differ in the two records (i.e. the terms l in D and M in D') is a non-evaluation context. Thus M is not the term that gets evaluated, nor is the redex contained within any of the copies of M.

   Let $\mathbb{M}_1$ be a context that shows positions of all copies of x that are bound by λx and all the occurrences of l in D and, respectively, M in D' that participate in the similarity $D \sim^l_M D'$ that appear under the λx. Without loss of generality we show all copies of x preceding all copies of l (or, respectively, of M).

   We have the following evaluation step ($\mathbb{M}_1, \mathbb{M}_2$ are multi-hole contexts, possibly empty, and not necessarily non-evaluation ones):

$$\begin{array}{rll} D = & [l' \mapsto \mathbb{E}\{(\lambda x.\mathbb{M}_1\{x,\ldots,x,l,\ldots,l\}) @ \mathbb{M}_2\{l,\ldots,l\}\},\ldots] & \Rightarrow \\ D_1 = & [l' \mapsto \mathbb{E}\{\mathbb{M}_1\{\mathbb{M}_2\{l,\ldots,l\},\ldots,\mathbb{M}_2\{l,\ldots,l\},l,\ldots,l\}\},\ldots] & \\ & & \\ D' = & [l' \mapsto \mathbb{E}\{(\lambda x.\mathbb{M}_1\{x,\ldots,x,M,\ldots,M\}) @ \mathbb{M}_2\{M,\ldots,M\}\},\ldots] & \Rightarrow \\ D'_1 = & [l' \mapsto \mathbb{E}\{\mathbb{M}_1\{\mathbb{M}_2\{M,\ldots,M\},\ldots,\mathbb{M}_2\{M,\ldots,M\},M,\ldots,M\}\},\ldots] & \end{array}$$

   We observe that $D_1 \sim^l_M D'_1$.

2. The evaluation step is by rule (SE), i.e. it is a substitution from another component. The label of the component copied by the substitution is denoted by l_1. By the inductive hypothesis if $l_1 \mapsto \mathbb{M}_1\{l,\dots,l\} \in D$ then $l_1 \mapsto \mathbb{M}_1\{M,\dots,M\} \in D_1$. Recall the notation $\mathbb{M}_\mathbb{E}$ for a multi-hole context whose first position is an evaluation context regardless of the terms filling the other holes (see Definition 19). Below is the evaluation step (on both records):

$$\begin{array}{ll} D = & [l' \mapsto \mathbb{M}_\mathbb{E}\{l_1, l, \dots, l\}, l_1 \mapsto \mathbb{M}_1\{l,\dots,l\}, \dots] \qquad \Rightarrow \\ D_1 = & [l' \mapsto \mathbb{M}_\mathbb{E}\{\mathbb{M}_1\{l,\dots,l\}, l, \dots, l\}, l_1 \mapsto \mathbb{M}_1\{l,\dots,l\}, \dots] \end{array}$$

$$\begin{array}{ll} D' = & [l' \mapsto \mathbb{M}_\mathbb{E}\{l_1, M, \dots, M\}, l_1 \mapsto \mathbb{M}_1\{M,\dots,M\}, \dots] \qquad \Rightarrow \\ D'_1 = & [l' \mapsto \mathbb{M}_\mathbb{E}\{\mathbb{M}_1\{M,\dots,M\}, M, \dots, M\}, l_1 \mapsto \mathbb{M}_1\{M,\dots,M\}, \dots] \end{array}$$

Again we observe that $D_1 \sim^l_M D'_1$.

□

4.4.3 Proof of Lemma 16.

We assume the conditions and notations as in the statement of the lemma: $D_1 = [l \mapsto M, l' \mapsto \mathbb{N}\{l\}, \dots] \overset{S}{\hookrightarrow} [l \mapsto M, l' \mapsto \mathbb{N}\{M\}, \dots] = D_2$ and $D_1 \Rightarrow^* D'_1$ and let $\mathcal{L} \preceq l, l', l_1, \dots, l_n$, where $l_1 \dots l_n$ is any sequence of labels in $L(D_1)$ other than l and l'.

Part 1: $l \neq l'$. Suppose $D_1 \underset{e}{\overset{\mathcal{L}}{\Longrightarrow}}{}^* D'_1$, l_1 is the first label in $\mathcal{L}$ such that $\text{next}(D_1, l_1)$ is not undefined, and $\text{next}(D_1, l_1) = \text{next}(D_2, l_1)$. We prove by induction that there exists D'_2 such that $D_2 \underset{e}{\overset{\mathcal{L}}{\Longrightarrow}}{}^* D'_2$, $D'_1 \sim^l_M D'_2$ and that $\text{Outcome}(D'_1) = \bot$ if and only if $\text{Outcome}(D'_2) = \bot$.

The evaluation sequence starts with the component bound to l which is the same in the two records. Until the modified component (i.e. the one bound to l') is needed in the evaluation sequence, the only difference between the two records is in that component.

When then component bound to l' is needed by l or is required by the sequence $\mathcal{L}$, it gets evaluated. If the evaluation results in a term other than a black hole, this term may be substituted into other components. When this happens, the two records may differ in multiple components.

Thus in our proof we distinguish the following stages in an evaluation (assuming that none of the components is bound to a black hole):

- Stage I: the component bound to l' is not needed yet by any components that are being evaluated so far,
- Stage II: the component bound to l' is being evaluated,
- Stage III: the component bound to l' is completely evaluated.

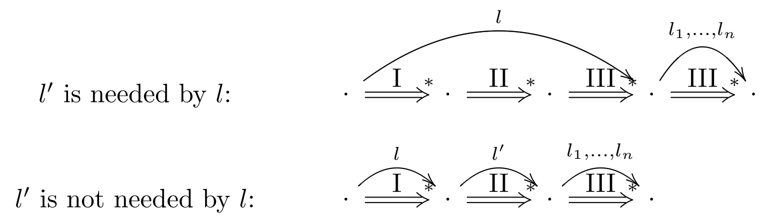

Figure 4.6: Stages of efficient evaluation with $\mathcal{L} = l, l', l_1, \ldots, l_n$, as defined in Part 1 of proof of Lemma 16

There are two cases, shown in Figure 4.6: when l' is needed during the evaluation of l (the top diagram) and when it is not needed for evaluating l (the bottom diagram). The arcs on top show the label in $\mathcal{L}$ that is currently being evaluated.

Base Case. It is easy to observe that $D_1 \sim^l_M D_2$.

Inductive Step. Suppose that the desired property holds after an n-step evaluation sequence $D_1 \overset{\mathcal{L}}{\underset{e}{\Longrightarrow}}{}^* D_1'$, i.e. there exists D_2' s.t. $D_2 \overset{\mathcal{L}}{\underset{e}{\Longrightarrow}}{}^* D_2'$ and $D_1' \sim^l_M D_2'$. Let $D_1' \Rightarrow D_1''$ be the next step in the efficient evaluation. Recall that the efficient evaluation sequence follows the sequence of labels $\mathcal{L} \preceq l, l', l_1, \ldots, l_n$. By convention labels l_i appearing anywhere in the proof are assumed to be different from l and l'.

We have the following cases (see Figure 4.6 for illustration of cases):

1. Stage I, i.e. the stage when l is being evaluated and l' is not needed yet. Thus D_1' and D_2' are the same except $l' \mapsto \mathbb{N}\{l\} \in D_1'$ and $l' \mapsto \mathbb{N}\{M\} \in D_2'$, and by Lemma 18 $\text{next}(D_1', l) = \text{next}(D_2', l)$.

   We have the following cases:

   (a) $\text{next}(D_1', l) = l$. Let $D_2' \Rightarrow D_2''$ by evaluating the component bound to l. Let M_l denote the term bound to l. Note that such terms are identical in D_1' and D_2'. There are four possibilities:

   - M_l is evaluatable,
   - $M_l = \mathbb{E}\{l_1\}$, where $l_1 \neq l', l_1 \neq l$,
   - $M_l = \mathbb{E}\{l\}$ (or, equivalently, $M_l = \mathbb{E}\{\bullet\}$), and
   - $M_l = \mathbb{E}\{l'\}$.

   In the first two cases the term bound to l remains identical in D_1'' and D_2'', so the only difference between the two terms is in the component bound to l'. Clearly $D_1'' \sim^l_M D_2''$. In the third case the component evaluates to $\bullet$ in both records and $\text{Outcome}(D_1) = \text{Outcome}(D_2) = \bot$.

   The last case is when $M_l = \mathbb{E}\{l'\}$. Since $\text{next}(D_1', l) = l$, the term $\mathbb{N}\{l\}$ bound to l' in D_1' is fully evaluated. By Lemma 15 $Cl(\mathbb{N}\{l\}) = Cl(\mathbb{N}\{M\})$, so the term $\mathbb{N}\{M\}$ bound to l' in D_2' also is fully evaluated. This case corresponds to the top part of Figure 4.6 since evaluation of l needs l', but Stage II is trivial (i.e. it does not contain any evaluation steps) since the term bound to l' is completely evaluated to begin with.

We have:

$$\begin{array}{rll} D_1' = & [l \mapsto \mathbb{E}\{l'\}, l' \mapsto \mathbb{N}\{l\}, \ldots] & \Rightarrow \\ & [l \mapsto \mathbb{E}\{\mathbb{N}\{l\}\}, l' \mapsto \mathbb{N}\{l\}, \ldots] & = D_1'' \\ D_2' = & [l \mapsto \mathbb{E}\{l'\}, l' \mapsto \mathbb{N}\{M\}, \ldots] & \Rightarrow \\ & [l \mapsto \mathbb{E}\{\mathbb{N}\{M\}\}, l' \mapsto \mathbb{N}\{M\}, \ldots] & = D_2'' \end{array}$$

It is easy to observe that $D_1'' \sim_M^l D_2''$. Note that in this case the evaluation sequence enters stage III.

(b) $\text{next}(D_1', l) = \bullet$. By Lemma 18 $\text{next}(D_2', l) = \bullet$, so both records diverge.

(c) $\text{next}(D_1', l) = l_1$. Since l' was not needed yet, the term bound to l_1 is the same in the two records. After the evaluation on the component l_1 the two resulting records also differ only in l' component.

(d) $\text{next}(D_1', l) = l'$, i.e. l' is needed for evaluation of l. The evaluation sequence enters Stage II on the top diagram in Figure 4.6. This situation is considered in Case 3.

2. Stage II (evaluation of l') in the case when l is fully evaluated and l' was not needed during evaluation of l. This corresponds to the bottom diagram in Figure 4.6.

   All components other than the one bound to l' are identical in the two records since the component bound to l' must be fully evaluated before it can be copied into other components, but it has not been evaluated yet. Below are the corresponding cases:

   (a) $\text{next}(D_1', l') = l'$, l does not appear in an evaluation context in the component bound to l'. In this case all copies of l in D_1' and all corresponding copies of M in D_2' appear in a non-evaluation context in the component bound to l'. By Lemma 19 in both cases $D_1'' \sim_M^l D_2''$.

   (b) $\text{next}(D_1', l') = l'$, l appears in an evaluation context in the component bound to l'. In this case D_2' may have l or M in the corresponding position. The former case is trivial (both records perform a substitution of l which is fully evaluated). In the latter case by Lemma 17 M evaluates to the same normal form as it did in the original evaluation of l, i.e. to the same term that gets substituted into l' in D_1'. The corresponding sequences are shown below:

$$\begin{array}{rlll} D_1' & \Rightarrow^* & [l \mapsto M', l' \mapsto \mathbb{M}_{\mathbb{E}}\{l, \ldots, l\}, \ldots] & \\ & \Rightarrow & [l \mapsto M', l' \mapsto \mathbb{M}_{\mathbb{E}}\{M', l, \ldots, l\}, \ldots] & = D_1'' \end{array}$$

$$\begin{array}{rlll} D_2' & \Rightarrow^* & [l \mapsto M', l' \mapsto \mathbb{M}_{\mathbb{E}}\{M, \ldots, M\}, \ldots] & \\ & \Rightarrow^* & [l \mapsto M', l' \mapsto \mathbb{M}_{\mathbb{E}}\{M', M, \ldots, M\}, \ldots] & = D_2'' \end{array}$$

Thus $D_1'' \sim_M^l D_2''$.

(c) $\text{next}(D_1', l') = l'' \neq l', l'' \neq l$. Since all the components in the two record other than the one bound to l' are identical, the case is trivial.

(d) $\text{next}(D_1', l') = \bullet$. In this case the component bound to l' either is of the form $\mathbb{E}\{l'\}$ or it depends on itself or on a black hole via a chain components of the form $\mathbb{E}\{l_i\}$. By Lemma 15 this means that the corresponding component in D_2' also either depends on itself directly or depends on a black hole or depends on itself via a chain of dependencies (note that components not bound to l or l' are identical in the two records).

(e) $\text{next}(D_1', l') = l$. This case is impossible since $\text{next}(D_1', l)$ is undefined.

(f) $\text{next}(D_1', l')$ is undefined. This means that the component bound to l' is already completely evaluated. Then no l appear in an evaluation context, and by Lemma 18 $\text{next}(D_2', l')$ is also undefined. In this case by the efficient strategy the next component to be evaluated is the one bound to $\text{next}(D_1', l_1)$ if it is defined and not a black hole. If $\text{next}(D_1', l_1)$ is undefined as well then the evaluation of D_1' moves to $\text{next}(D_1', l_2)$, etc. This corresponds to Stage III in Figure 4.6. We consider this situation in Case 5.

3. Stage II (l' is being evaluated) in the case when l' is needed by l, i.e. the case of the top diagram in Figure 4.6. Since l' is needed by l, the component l is of the form $\mathbb{E}\{l'\}$ in both records or there is a chain of component dependencies of the form $\mathbb{E}\{l_i\}$ from l to l'.

   Since the term bound to l' is currently being evaluated, it is of the form $\mathbb{M}\{l, \ldots, l\}$ in D_1' and of the form $\mathbb{M}\{M, \ldots, M\}$ in D_2'. According to the efficient evaluation strategy, no component can copy the term bound to l' until the term is completely evaluated. Thus all the other components are still identical in the two records during this stage.

   We have the following cases:

   (a) $\text{next}(D_1', l) = \text{next}(D_2', l) = l'$ and the multi-hole context $\mathbb{M}$ is a non-evaluation context. In this case the next evaluation step is performed on the component l'. By Lemma 19 after this step the term bound to l' is of the form $\mathbb{M}'\{l, \ldots, l\}$ in D_1'' and of the form $\mathbb{M}'\{M, \ldots, M\}$ in D_2''. All the other components are identical in D_1'' and D_2'', thus $D_1'' \sim_M^l D_2''$.

   (b) The multi-hole context $\mathbb{M} = \mathbb{M}_{\mathbb{E}}$ is an evaluation context for one of the holes. convention that, without loss of generality, the first hole in an evaluation multi-hole context is assumed to be the evaluation position). Then $l' \mapsto \mathbb{M}_{\mathbb{E}}\{l, \ldots, l\} \in D_1'$ and $l' \mapsto \mathbb{M}_{\mathbb{E}}\{M, \ldots, M\} \in D_2'$ We show that in this case $\text{Outcome}(D_1') = \text{Outcome}(D_2') = \bot$.

Substituting the component into l in D'_1 (possibly through a dependency chain), we get a black hole in the component bound to l since $\mathbb{E}\{\mathbb{M}_{\mathbb{E}}\{\square, \ldots, l\}\}$ is an evaluation context by Lemma 1.

$$\begin{array}{ll} D_1 \Rightarrow^* & [l \mapsto \mathbb{E}\{\mathbb{M}_{\mathbb{E}}\{l, \ldots, l\}\}, l' \mapsto \mathbb{M}_{\mathbb{E}}\{l, \ldots, l\}, \ldots] \quad \Rightarrow \\ & [l \mapsto \bullet, l' \mapsto \mathbb{M}_{\mathbb{E}}\{l, \ldots, l\}, \ldots] \end{array}$$

We show that the other record diverges in this case. Below is the result of the substitution into l:

$$D_2 \Rightarrow^* \quad [l \mapsto \mathbb{E}\{\mathbb{M}_{\mathbb{E}}\{M, \ldots, M\}\}, l' \mapsto \mathbb{M}_{\mathbb{E}}\{M, \ldots, M\}, \ldots]$$

The component bound to l is of the form $\mathbb{E}\{\mathbb{E}'\{M\}\}$, where $\mathbb{E}'$ denotes $\mathbb{M}_{\mathbb{E}}\{\square, M, \ldots, M\}$. Therefore M appears in an evaluation context in D'_2 and is the next term to be evaluated. However, recall that M is the term originally bound to l in D_2:

$$D_2 = \quad [l \mapsto M, l' \mapsto \mathbb{N}\{M\}, \ldots]$$

By Lemma 17 M evaluates to the same result as the first time it was evaluated:

$$\begin{array}{ll} [l \mapsto M, \ldots] & \Rightarrow^* \\ [l \mapsto \mathbb{E}\{\mathbb{E}'\{M\}\}, \ldots] & \Rightarrow^* \\ [l \mapsto \mathbb{E}\{\mathbb{E}'\{\mathbb{E}\{\mathbb{E}'\{M\}\}\}\}, \ldots] & \Rightarrow^* \ldots \end{array}$$

As demonstrated above, D_2 diverges. Note that by Lemma 7 if a record diverges on one evaluation path, it diverges on all paths.

(c) $\text{next}(D'_1, l') = \bullet$. Then $\text{next}(D'_2, l') = \bullet$ and both outcomes are $\bot$.

(d) $\text{next}(D'_1, l) = l'' \neq l$. Then the label appearing in the evaluation context in D'_1 cannot be l so all copies of l in D'_1 and the corresponding copies of M in D'_2 appear in non-evaluation contexts, and by Lemma 18 $\text{next}(D'_2, l') = l''$. A component bound to l'' is evaluated. All components other than l' are still the same in the two records so the results of the evaluation will clearly be the same. Note that the evaluation of the component bound to l'' may result in a fully evaluated term (which will be identical in both records) or in a circular dependency or a dependency on l. Again, this will happen in both records. These situations are considered in Cases 3c and 3e.

(e) $\text{next}(D'_1, l) = l$. The situation when the component bound to l' is of the form $\mathbb{M}_{\mathbb{E}}\{l, \ldots, l\}$ in D'_1 and of the form $\mathbb{M}_{\mathbb{E}}\{M, \ldots, M\}$ in D'_2 was already considered in 3b. Thus it must be the case that the components bound to l' are both of the form $\mathbb{E}\{l\}$ or both of the form $\mathbb{E}\{l''\}$ where l'' depends on l, i.e. $\text{next}(D'_2, l') = l$.

Since l needs l' and l' needs l in both records, the evaluation encounters a black hole in both records.

4. The first part of Stage III in the top diagram in Figure 4.6: l' was needed by l, the component bound to l' is completely evaluated, and the evaluation of l continues. This implies that in D'_1 the label l' is bound to $\mathbb{M}_{\mathbb{N}}\{l, \ldots, l\}$ and in D'_2 it is bound to $\mathbb{M}_{\mathbb{N}}\{M, \ldots, M\}$. It is also possible that the result of evaluating l' has been substituted into other components, thus the other components may differ in the two records. Recall that by the inductive hypothesis $D'_1 \sim^l_M D'_2$. We assume that no component is bound to a black hole at this point. We have the following cases:

   (a) $\mathrm{next}(D'_1, l) = l$. The component bound to l may be different for the two records. However, l may not appear in an evaluation context in D'_1 because $\mathrm{next}(D'_1, l) \neq \bullet$. Thus it must be the case that the component is in the form $\mathbb{M}_{\mathbb{N}}\{l, \ldots, l\}$ in D'_1 and by the inductive hypothesis this component is of the form $\mathbb{M}_{\mathbb{N}}\{M, \ldots, M\}$ in D'_2. By Lemma 19 the desired property holds after the step $D'_1 \overset{l}{\underset{e}{\Longrightarrow}} D''_1$.

   (b) $\mathrm{next}(D'_1, l) = l'' \neq l$. In this case the evaluation will switch to the component bound to l''. If this evaluation succeeds without reaching a black hole or a component bound to $\mathbb{E}\{l\}$ then this case is analogous to Case 5.

   Suppose the evaluation of l'' reaches $\mathbb{E}\{l\}$. The same component in D'_2 evaluates either to $\mathbb{E}'\{l\}$ or to $\mathbb{E}'\{M\}$. In the former case both records arrive at a black hole. In the latter case is somewhat similar to Case 3b: the component bound l in D'_1 evaluates to $\bullet$ right away and D'_2 repeats the evaluation of M. By Lemma 17 M evaluates to $\mathbb{E}\{l\}$. Thus the component bound to l'' evaluates to a black hole.

   Below we show the corresponding evaluation sequence. Without loss of generality we assume that l depends on l'' directly, i.e. without going through a chain of other components.

$$\begin{array}{ll} [l \mapsto M, \ldots] & \Rightarrow^* \\ [l \mapsto \mathbb{E}\{l''\}, l'' \mapsto \mathbb{E}'\{M\}, \ldots] & \Rightarrow^* \\ [l \mapsto \mathbb{E}\{l''\}, l'' \mapsto \mathbb{E}'\{\mathbb{E}\{l''\}\}, \ldots] & \Rightarrow^* \\ [l \mapsto \bullet, l'' \mapsto \bullet, \ldots] & \end{array}$$

   Another case is when evaluation of l'' reaches a black hole which is not cretaed by a reference to l. Then the corresponding component in D'_2 also evaluates to a black hole since the two records are the same except occurrences of l and M.

   (c) $\mathrm{next}(D'_1, l) = \bullet$. In this case the l component is of the form $\mathbb{E}\{l\}$ in D'_1. It may be of the form $\mathbb{E}'\{l\}$ in D'_2 in which case in both records the component evaluates to a black hole. It may also be the case that the component is of the form $\mathbb{E}\{l\}$ in D'_1 and of the form $\mathbb{E}'\{M\}$ in D'_2. This case is analogous to Case 3b: D'_1 evaluates to a record with a black hole and D'_2 diverges since the evaluation of M reaches $\mathbb{E}'\{M\}$ and by Lemma 17 M continuosly evaluates to $\mathbb{E}'\{M\}$ so D'_2 diverges.

(d) next(D_1', l) is undefined. Since l' is also completely evaluated, the next label to be evaluated is l_1. See Case 5 for this situation.

5. Stage III in both cases in Figure 4.6 (on the top diagram this is the second part of Stage III): components bound to l and l' are fully evaluated, and the evaluation proceeds with the rest of the labels in $\mathcal{L}$, i.e. with $l_1, \ldots, l_m$. Let next$(D_1', l_1) = l''$. Note that l is bound to $\mathbb{M}_{\mathbb{N}}\{l, \ldots, l\}$ in D_1' and to $\mathbb{M}_{\mathbb{N}}\{M, \ldots, M\}$ in D_2'. Again the only non-trivial case is

$$\begin{array}{lll} D_1' & \Rightarrow^* & [l \mapsto \mathbb{M}_{\mathbb{N}}\{l, \ldots, l\}, l'' \mapsto \mathbb{M}_{\mathbb{E}}\{l, \ldots, l\}, \ldots] \\ & \Rightarrow & [l \mapsto \mathbb{M}_{\mathbb{N}}\{l, \ldots, l\}, l'' \mapsto \mathbb{M}_{\mathbb{E}}\{\mathbb{M}_{\mathbb{N}}\{l, \ldots, l\}, l, \ldots, l\}, \ldots] \quad = D_1'' \end{array}$$

$$\begin{array}{lll} D_2' & \Rightarrow^* & [l \mapsto \mathbb{M}_{\mathbb{N}}\{M, \ldots, M\}, l'' \mapsto \mathbb{M}_{\mathbb{E}}\{M, \ldots, M\}, \ldots] \\ & \Rightarrow^* & [l \mapsto \mathbb{M}_{\mathbb{N}}\{M, \ldots, M\}, l'' \mapsto \mathbb{M}_{\mathbb{E}}\{\mathbb{M}_{\mathbb{N}}\{M, \ldots, M\}, M, \ldots, M\}, \ldots] \quad = D_2'' \end{array}$$

This case is similar to the Case 3b above: since M in the component bound to l evaluated to $\mathbb{M}_{\mathbb{N}}\{M, \ldots, M\}$ by the efficient strategy, any other copy of M in an evaluation context evaluates to the same term as well.

Since by the inductive hypothesis $D_1' \sim_M^l D_2'$, we have $D_1'' \sim_M^l D_2''$.

We have proven that for the case when $l \neq l'$ if $D_1 \overset{\mathcal{L}}{\underset{e}{\Longrightarrow}}{}^* D_1'$ then there exists D_2' s.t. $D_2 \overset{\mathcal{L}}{\underset{e}{\Longrightarrow}}{}^* D_2'$, $D_1' \sim_M^l D_2'$, and Outcome$(D_1') = \perp$ if and only if Outcome$(D_2') = \perp$.

Part 2. Now we consider the situation when $l = l'$. Suppose $D_1 \overset{\mathcal{L}}{\underset{e}{\Longrightarrow}}{}^* D_1'$. We would like to prove by induction that there exists D_2' s.t. $D_2 \overset{\mathcal{L}}{\underset{e}{\Longrightarrow}}{}^* D_2'$ and $D_1' \sim_M^l D_2'$ and Outcome$(D_1') = \perp$ if and only if Outcome$(D_1') = \perp$.

In this case the two records initially are of the form:

$$\begin{array}{l} D_1 = [l \mapsto \mathbb{N}\{l\}, \ldots,] \\ D_2 = [l \mapsto \mathbb{N}\{\mathbb{N}\{l\}\}, \ldots,] \end{array}$$

Let $M = \mathbb{N}\{l\}$. We use the efficient evaluation strategy starting at l followed by some labels $l_1, \ldots l_n$.

1. Suppose that the component bound to l is being evaluated. We have the following cases:

   (a) next(D_1', l) is undefined. By the inductive hypothesis the component bound to l is of the form $\mathbb{M}_{\mathbb{N}}\{l, \ldots, l\}$ in D_1' and $\mathbb{M}_{\mathbb{N}}\{M, \ldots, M\}$ in D_2', and the evaluation switches to label l_1. See Case 2.

   (b) next$(D_1', l) = l$. The evaluation of this component continues. Note that the context containing all occurrences of l for one record and all occurrences of M for the other one must be a non-evaluation context (otherwise next$(D_1', l) = \bullet$). By Lemma 19 the desired property holds after the next evaluation step.

(c) $\text{next}(D'_1, l) = l' \neq l$. Since l depends on l', it cannot be the case that l' depends on l (otherwise $\text{next}(D'_1, l) = \bullet$). Thus all the components other than the l component are the same in the two records, and the desired property holds after an evaluation step on l'.

(d) $\text{next}(D'_1, l) = \bullet$. This case can happen for several reasons:

   i. In both records the component bound to l is of the form $\mathbb{E}\{l\}$ (for possibly different $\mathbb{E}$). This means that the label in the context is not one of those that differ in the two records. In both records the component bound to l evaluates to a black hole.

   ii. In both records the component bound to l is of the form $\mathbb{E}\{l'\}$ where l' is either bound to a black hole or depends on a component bound to a black hole or depends, directly or via a dependency chain, on l. In all of these cases both records evaluate to components with black holes.

   iii. $D'_1 = \mathbb{M}_{\mathbb{E}}\{l, \ldots, l\}$, $D'_2 = \mathbb{M}_{\mathbb{E}}\{M, \ldots, M\}$. In this case $D'_1 \Rightarrow [l \mapsto \bullet, \ldots]$. By Lemma 17 evaluation of D'_2 proceeds as follows:

$$\begin{array}{ll} D_2 = & [l \mapsto M, , \ldots,] \\ \Rightarrow^* & [l \mapsto \mathbb{E}\{M\}, \ldots,] \\ \Rightarrow^* & [l \mapsto \mathbb{E}\{\mathbb{E}\{M\}\}, \ldots,] \quad \Rightarrow^* \ldots \end{array}$$

   We see that D_2 diverges, so $\text{Outcome}(D_1) = \text{Outcome}(D_2) = \perp$. This case is, once again, similar to Case 3b in Part 1.

2. The component bound to l is fully evaluated and the efficient strategy proceeds to other components. By the inductive hypothesis $D'_1 \sim^l_M D'_2$ and all copies of l and the corresponding copies of M appear in non-evaluation context positions.

   The only non-trivial case is when a component is bound to $\mathbb{M}_{\mathbb{E}}\{l, \ldots, l\}$ in D'_1 and to $\mathbb{M}_{\mathbb{E}}\{M, \ldots, M\}$ in D'_2. Because of the efficient strategy all substitutions during the evaluation of a copy of M are the same as during the evaluation of M bound to l. By Lemma 17 we have $D''_1 \sim^l_M D''_2$.

Part 3. The other direction of Lemma 16 for $l \neq l'$ is proven analogously. The order of evaluation is completely determined by the efficient strategy and is the same in both records. Thus if we are given an efficient evaluation sequence that starts at D_2, we observe that D_1 will follow the same evaluation sequence as in Part 1, so the same relation $D'_1 \sim^l_M D'_2$ takes place. However, note that the sequence that starts at D'_2 takes extra steps because duplication of a term M causes repeated evaluation of this term. Specifically, we have the following relation between D'_1 and D'_2:

1. either $\text{next}(D'_l, l'') = \text{next}(D'_l, l'')$ (where l'' may be equal to l or to l' or be different from both)

2. or the component being evaluated is of the form $\mathbb{M}_{\mathbb{E}}\{l, \ldots, l\}$ in D'_1 and of the form $\mathbb{M}_{\mathbb{E}}\{M, \ldots, M\}$ in D'_2. In this case additional steps may be

required for D_2' to complete the evaluation of M before the two resulting terms become (l, M)-similar, so $D_2' \Rightarrow^* D_2''$ s.t. $D_1' \sim_M^l D_2''$.

These extra steps in the sequence that starts at D_2' account for the additional steps $D_2' \Rightarrow^* D_2''$ in Lemma 16 (see also Figure 4.5). A simple example below illusrates the issue:

$$\begin{array}{rll}
D_1 = & [l \mapsto 2+3, l' \mapsto \lambda x.x \mathbin{@} l] & \Rightarrow \\
 & [l \mapsto 5, l' \mapsto \lambda x.x \mathbin{@} l] & \Rightarrow \\
 & [l \mapsto 5, l' \mapsto l] & \Rightarrow \\
D_1' = & [l \mapsto 5, l' \mapsto 5] & \Rightarrow
\end{array}$$

$$\begin{array}{rll}
D_1 = & [l \mapsto 2+3, l' \mapsto \lambda x.x \mathbin{@} l] & \to \\
D_2 = & [l \mapsto 2+3, l' \mapsto \lambda x.x \mathbin{@} 2+3] & \Rightarrow^* \\
D_2' = & [l \mapsto 5, l' \mapsto 2+3] & \Rightarrow^* \\
D_2'' = & [l \mapsto 5, l' \mapsto 5] &
\end{array}$$

Here it is not the case that $D_1' \sim_M^l D_2'$, an extra step to D_2'' is needed to "synchronize" the two records, and $D_1' \sim_M^l D_2''$.

Part 4. The other direction of Lemma 16 in the case when $l = l'$. Again, the proof is analogous to that of Part 2 above with possible extra steps as in Part 3 above.

Lemma 20 (Non-evaluation Substitution Step Preserves Meaning)**.** *If $D_1 \overset{S}{\hookrightarrow} D_2$ then Outcome$(D_1) =$ Outcome(D_2).*

Proof. By Lemma 16 if Outcome$(D_1) \neq \perp$ then Outcome$(D_2) \neq \perp$. Let D_1' be the normal form of D_1 and D_2' be the normal form of D_2. Since D_1' is a normal form, none of the copies of l appear in an evaluation context, thus all its components are bound to $\mathbb{M}_{\mathbb{N}}\{l, \ldots, l\}$. By Lemma 16 $D_1' \sim_M^l D_2'$, so all components of D_2' are of the form $\mathbb{M}_{\mathbb{N}}\{M, \ldots, M\}$. Thus by Lemma 15 $Cl(D_1') = Cl(D_2')$.

By a similar reasoning we can assume that Outcome$(D_2) \neq \perp$ and conclude that there is a normal form D_1' of D_1 and $Cl(D_2') = Cl(D_1')$. □

4.5 Computational Soundness Theorem

Theorem 1 (Computational Soundness)**.** *If $D_1 \leftrightarrow D_2$ then Outcome$(D_1) =$ Outcome(D_2).*

Proof. By Lemmas 6, 11, and 20 all steps in the calculus preserve the outcome. □

Chapter 5

Conclusions and Future Work

We have proven that the call-by-name calculus of recursively-scoped records is computationally sound. Our system captures the essential features of mutually recursive components. We plan to investigate applications of our proof method to more complex systems with possible cyclic dependencies, such as `letrec` calculi and more sophisticated systems that model modules and linking.

5.1 Acknowledgments

Many thanks to Emily Christiansen who participated in the early stages of this research, to Dr. Manfred Schmidt-Schauß for helpful discussions, and to Franklyn Turbak and Nicholas McPhee for helpful feedback on drafts of this work.

Bibliography

[1] Davide Ancona and Elena Zucca: A calculus of module systems. Vol. 12, 2002, pp. 91-132.

[2] Z. M. Ariola, Stefan Blom: Skew confluence and the lambda calculus with letrec. Annals of pure and applied logic 117/1-3, 97-170, 2002

[3] Z. M. Ariola and J. W. Klop: Equational Term Graph Rewriting. Fundamentae Informaticae, Vol. 26, Nrs. 3,4, June 1996. p. 207-240.

[4] Z. M. Ariola, J. W. Klop: Lambda calculus with explicit recursion. Journal of Information and Computation, Vol. 139 (2): 154-233, 1997.

[5] H. P. Barendregt: The Lambda Calculus, its Syntax and semantics. Studies in Logic, volume 103, Elsevier Science Publishers, 1984.

[6] Sonia Fagorzi and Elena Zucca: A Calculus for Reconfiguration: (Extended abstract). Electr. Notes Theor. Comput. Sci., Vol. 135, N. 3, 2006, pp. 49-59.

[7] E. Machkasova: Computational Soundness of Non-Confluent Calculi with Applications to Modules and Linking, Ph.D. dissertation, April 2002, Boston University

[8] E. Machkasova: Computational Soundness of a Call by Name Calculus of Records. Working Papers Series, University of Minnesota, Morris, Volume 2 Number 3, 2007. Available at http://cda.morris.umn.edu/~elenam/

[9] E. Machkasova, E. Christiansen: Call-by-name Calculus of Records and its Basic Properties. Working Papers Series, University of Minnesota, Morris, Volume 2 Number 2, 2006. Available at http://cda.morris.umn.edu/~elenam/

[10] E. Machkasova, F. Turbak: A calculus for link-time compilation. In Programming Languages & Systems, 9th European Symp. Programming, volume 1782 of LNCS, pages 260-274 Springer-Verlag, 2000

[11] G. D. Plotkin: Call-by-name, call-by-value and the lambda calculus. Theoret. Comput. Sci., 1, 1975.

[12] Manfred Schmidt-Schauß and Michael Huber: A lambda-calculus with letrec, case, constructors and non-determinism. In First International Workshop on Rule-Based Programming, 2000.

[13] M. Schmidt-Schauß: Correctness of copy in calculi with letrec, case and constructors. Frank report 28, Institut für Informatik. Fachbereich Informatik und Mathematik. J. W. Goethe-Universität Frankfurt am Main, February 2007.

[14] J. B. Wells, Detlef Plump, and Fairouz Kamareddine: Diagrams for meaning preservation. In Rewriting Techniques & Applications, 14th Int'l Conf., RTA 2003, volume 2706 of LNCS, pp. 88-106. Springer-Verlag, 2003

[15] J. B. Wells and RenVestergaard: Equational reasoning for linking with first-class primitive modules. n Programming Languages & Systems, 9th European Symp. Programming, volume 1782 of LNCS, pages 412-428. Springer-Verlag, 2000.