A Computationally Sound Call-By-Value Module Calculus

Elena Machkasova*
Department of Computer Science
Boston University
Boston, MA 02215, U.S.A.
elenam@cs.bu.edu
http://www.cs.bu.edu/~elenam

Franklyn A. Turbak†
Computer Science Department
Wellesley College
Wellesley, MA 02482, U.S.A.
fturbak@wellesley.edu
http://cs.wellesley.edu/~fturbak

Boston University Computer Science Department

Draft of Technical Report BUCS-TR-2001-XX

August 1, 2001

This document is a draft of a technical report that expands and improves upon [MT00]. While the main sections are very close to completion, the appendices are still work in progress. In particular, the proofs in the appendicies do not yet reflect the recent adoption of explicit alpha-renaming, and in some cases are incomplete. We have omitted appendices D and E entirely because they are currently in an inconsistent state.

*Supported by NSF grant EIA–9806745.
†Supported by NSF grant EIA–9806747.

Abstract

We present a call-by-value module calculus that serves as a framework for formal reasoning about simple module transformations. The calculus is stratified into three levels: a term calculus, a core module calculus, and a linking calculus. At each level, we define both a calculus reduction relation and a small-step operational semantics and relate them by a *computational soundness* property: if two terms are equivalent in the calculus, then they have the same observable outcome in the operational semantics. This result is interesting because recursive module bindings thwart confluence at two levels of our calculus and prohibit application of the traditional technique for showing computational soundness, which requires confluence (in addition to other properties, the most important being standardization). We show that it is sufficient to replace confluence by a weaker property that we call *confluence with respect to evaluation*. We introduce a new technique for proving computational soundness based on a pair of related properties that we call *lift* and *project*. The project property, under certain conditions, implies confluence with respect to evaluation, while the lift property is equivalent to standardization.

In our multi-layer calculus, terms at one level serve as components of terms at the same or higher level. This leads to a notion of *embedding*: calculus X is embedded in calculus Y if calculus steps of X are preserved when wrapped in a context of Y. Two terms in X have the same meaning with respect to Y if they have the same observational outcome in all contexts of Y. In our framework, computational soundess of Y implies an *observational soundness* property: two terms that are equivalent in X have the same meaning with respect to Y. Thus, calculus-based transformations in one calculus are meaning preserving in any embedding calculus.

Our modules have both public and private components. We formalize the notion of privacy by identifying modules up to alpha-renaming of hidden (i.e., private) labels. Because of this identification, module linking can be defined without the need to resolve naming conflicts between the hidden labels of two modules. In addition to alpha-renaming at the core module level, we also formalize alpha-renaming in namespaces at the term and linking levels of our calculus. This is important, because many properties of our module calculus hold only for alpha-equivalence classes of terms and not for concrete terms.

A particularly important domain for module transformations is link-time compilation, where many optimization opportunities are not apparent until two modules are linked together. We present a simple model of link-time compilation and introduce the *weak distributivity property* for a meaning-preserving transformation T operating on modules D_1 and D_2 linked by $\oplus$: $T(D_1 \oplus D_2) = T(T(D_1) \oplus T(D_2))$. We argue that this property finds promising candidates for link-time optimizations, and present simple conditions that imply weak distributivity. We use these to demonstrate that some meaning preserving module transformations are weakly distributive.

This reports expands upon our earlier work reported in [MT00]. We make several corrections to the earlier work, the most important of which concerns the rule for garbage collection at the core module level. Our rigorus treatment of alpha renaming in this report allows us to improve the treatment of several linking operations and simplify the characterization of term behavior.

1 Introduction

1.1 Meaning Preserving Module Transformations

Most modern programming languages provide some sort of *module system* that facilitates "programming in the large" – i.e., composing large, complex programs out of smaller units, *modules*, that can be independently specified, written, tested, debugged, and — ideally — reused.

While composing programs out of modules has numerous clear software engineering benefits, one drawback is that module boundaries often stand in the way of program optimizations. As an example of how modules can block optimizations, consider the following modules D_1 and D_2:

$$D_1 = [A \mapsto 3, B \mapsto A * C], \quad D_2 = [C \mapsto 4, D \mapsto A + C]$$

Here we use a notation for modules formally introduced in section 2. In this notation, modules are bracketed collections of bindings that associate labels with expressions. Consider the result of performing on each of these modules a constant propagation optimization (CP):

$$CP(D_1) = [A \mapsto 3, B \mapsto 3 * C], \quad CP(D_2) = [C \mapsto 4, D \mapsto A + 4]$$

Now suppose that the two optimized modules are linked via the linking operation $\oplus$, which in this case effectively takes the union of their bindings:

$$CP(D_1) \oplus CP(D_2) = [A \mapsto 3, B \mapsto 3 * C, C \mapsto 4, D \mapsto A + 4]$$

Linking exposes new opportunities for constant propagation that were unknown when the modules were independently optimized. Unless more optimizations are performed after linking, these opportunities are lost. In contrast, in non-modular *whole-program* approaches, optimizations such as CP would be performed on an entire program after merging all of its pieces:

$$\begin{aligned} CP(D_1 \oplus D_2) &= CP([A \mapsto 3, B \mapsto A * C, C \mapsto 4, D \mapsto A + C]) \\ &= [A \mapsto 3, B \mapsto 3 * 4, C \mapsto 4, D \mapsto 3 + 4] \end{aligned}$$

Our goal in this work is to present a formal framework for studying a simple class of module transformations for a module calculus for a purely functional language. Ideally, a module transformation should provably preserve the meaning of its input. Meaning preservation is usually phrased in terms of a notion of *observational equivalence*. Two program fragments are observationally equivalent if they behave indistinguishably in every context (modulo some notion of observable behavior). A program transformation is *meaning preserving* if the result of transforming a program fragment is observationally equivalent to the original fragment.

Meaning preservation is challenging to prove for general transformations. But for transformations expressible in the calculus of a language, meaning preservation holds if the language satisfies the following *computational soundness property*[1]: *if two fragments are equal in the calculus, then they have the same observable outcome relative to evaluation.* We show that this property holds at all three levels of our module calculus, and use it to prove that several classical program transformations are meaning preserving, both within and across modules: constant folding and propagation, function inlining, and a simple form of dead code elimination.

An example of a practical module transformation that can be shown to be meaning preserving in our framework is *cross-module lambda-splitting*, a transformation described by Blume and Appel in [BA97] and

[1] We will often abbreviate the name of this property as "soundness".

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

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

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

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

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

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

Intuitively, the original and transformed expressions have the same meaning. But how can this intuition be formalized? In our calculus we can formally prove this fact by showing that the expressions are equivalent in the calculus, and therefore have the same meaning due to computational soundness (see section 2.6).

It turns out to be rather tricky to show that our module calculus has computational soundness. The problem is that traditional techniques for showing this property (e.g., [Plo75, AF97]) require the language to be confluent, but the recursive nature of module bindings destroys confluence. In order to show that our module calculus has computational soundness, we introduce a new technique for proving this property based on a weaker notion of confluence that we call *confluence with respect to evaluation.* Our technique depends on pair of related properties that we call *lift* and *project.* After discussing computational soundness in the context of our three-level module calculus, section 3 reviews the traditional approach to proving computational soundness and introduces our new technique.

1.2 Link-time Compilation and Weak Distributivity

Our work was motivated in large part by an interest in *link-time compilation.* Link-time compilation is a model of compilation that lies in the relatively unexplored expanse between *whole-program compilation*, in which the entire source program is compiled to an executable, and *separate compilation*, in which source program modules are independently compiled into fragments, which are later linked to form an executable.[2] In the link-time compilation model (1) source program modules are first partially compiled into intermediate language modules; (2) intermediate modules are further compiled when they are combined, taking advantage of usage information exposed by the combination; and (3) when all intermediate modules have been combined into a final closed module, it is translated into an executable.

Link-time compilation is worth exploring because it can potentially provide more reusability than whole-program compilation and more efficiency than separate compilation. While separate compilation offers well-known benefits for program development and code reuse, a drawback is that the compilation of one module cannot take advantage of usage information in the modules with which it is later linked. In contrast, link-time compilation can use this information to perform optimizations and choose specialized data representations

[2] In our categorization, code for virtual machines (such as Java's JVM) is lumped together with with code for "real" machines in the "binary" category.

more efficient than the usual uniform representations for data passed across module boundaries. The link-time compilation model requires partitioning the work of program analysis and translation into pre-link-time, link-time, and post-link-time passes. How to perform this partitioning in a way that is both practical (i.e., the cost of linking is acceptable) and non-trivial (i.e., does not degenerate into the whole-program model or separate compilation model) is a fertile area for research.

In our study of module transformations, we have identified a property – *weak distributivity* – that suggests promising candidates for link-time optimizations. Suppose that T is a module transformation, D_1 and D_2 are modules, and $\oplus$ links two modules into one. Then we say that T is *weakly distributive* if and only if

$$T(D_1 \oplus D_2) = T(T(D_1) \oplus T(D_2)).$$

Why should such a T be a candidate for link-time optimization? Imagine that a program is a binary tree whose nodes are $\oplus$ and whose leaves are the modules of the program. For a weakly distributive transformation, the effect of transforming the whole program obtained by first linking all the modules together can instead be obtained by first transforming the leaf modules of the program, and then transforming the modules obtained after each link step. The intermediate modules are transformed to take advantage of information exposed in the combination that was not apparent in the individual components.

An example of a weakly distributive module transformation is the constant propagation transformation CP introduced in section 1.1. For the example given there, note that:

$$\begin{aligned}
CP(D_1 \oplus D_2) &= CP([A \mapsto 3, B \mapsto A * C, C \mapsto 4, D \mapsto A + C]) \\
&= [A \mapsto 3, B \mapsto 3 * 4, C \mapsto 4, D \mapsto 3 + 4] \\
&= CP([A \mapsto 3, B \mapsto 3 * C, C \mapsto 4, D \mapsto A + 4]) \\
&= CP([A \mapsto 3, B \mapsto 3 * C] \oplus [C \mapsto 4, D \mapsto A + 4]) \\
&= CP(CP(D_1) \oplus CP(D_2))
\end{aligned}$$

In this paper, we present simple conditions that imply weak distributivity for module transformations. Examples of such transformations include those mentioned for meaning preservation, except for function inlining, which fails to meet the conditions.

1.3 Summary of Contributions

The main contributions of this paper are as follows:

- We present an untyped call-by-value module calculus stratified into three levels: a term calculus, a core module calculus, and a linking calculus. At each level, we show that the reduction relation of the calculus is computationally sound with respect to a small-step operational semantics defined in terms of an evaluation relation that is a subset of the reduction relation.

- We develop a formal framework for reasoning about a multi-level calculus, which includes a notion of one calculus being embedded in another. This notion allows us to show that calculus-based transformations at one level of the calculus are meaning preserving with respect to another level.

- We formalize the privacy of module components via alpha-renaming and resolve a thorny interaction between alpha-renaming and module contexts with fixed private labels.

- We summarize the traditional technique for proving computational soundness based on confluence and standardization and introduce a new technique for proving it via a pair of related properties we call *lift* and *project*. This new technique employs a notion of *confluence with respect to evaluation* that is weaker than confluence, allowing us to handle the non-confluence of module evaluation in the presence of recursive module bindings.

- We sketch a simple model of link-time compilation and introduce the weak distributivity property as one way to find promising candidates for link-time optimizations. We show that module transformations satisfying certain conditions are weakly distributive, and demonstrate these conditions for some examples of meaning preserving transformations.

1.4 Differences from [MT00]

This technical report expands upon our earlier work presented in [MT00]. While the module calculus presented here is in essence very similar to that in [MT00], we have simplified several link-level operations by considering these operations at the level of α-equivalence classes of linking expressions, rather than on concrete expressions, improved and made consistent classification at the three level of our calculus, corrected an error of [MT00] related to garbage collection rule for the core module calculus, and introduced some new definitions which clarify reasoning about meaning preservation. Specifically, the changes are as follows:

- The theoretical framework presented in this work distinguishes between *computational soundness*, a property of a calculus that calculus equivalence of two terms implies their observational equivalence, and *observational soundness* of one calculus with respect to another, a property that two terms equivalent in a calculus X produce two observationally equivalent terms in X' when enclosed into the same context of X'. A simplified definition of observational equivalence of two terms in a calculus relates the terms by their outcomes in all contexts of the calculus, while [MT00] uses a more complex definition which takes into account only values as observables, but ignores other aspects of behavior of terms, for instance it does not distinguish between divergence and error. The new definition is consistent with the notion of computational soundness, which is the main focus of our study.

- In the current report we formally define confluence with respect to evaluation and show its relation to the properties lift and project introduced in [MT00]. See section 3.6 for a detailed discussion.

- The most significant difference of the calculus here from that of [MT00] is that terms at each level are identified up to a corresponding notion of α-renaming. While in the old version of the calculus the user had explicit control over the names of hidden components of a module, here modules are identified up to renaming of such components, and the user deals with entire α-equivalence classes of modules. Having introduced such α-renaming for modules, we have lifted it to the next level of the calculus – that of linking expressions. We have also identified terms up to renaming of λ-bound variables, and linking expressions up to renaming of **let**-bound module identifiers (in [MT00] such identification was assumed but not formalized).

  The new identification allows us, in particular, to define a linking of two modules that automatically resolves conflicts between the names of hidden components of the two modules by choosing a pair of representatives of the α-equivalence classes of the two modules which do not have a conflict. We also replaced a general (rename) rule of [MT00], which, in particular, allows explicit renaming of hidden labels and renaming of a visible label to a hidden, by a pair of rules: (rename) for renaming of one visible label to another, and (hide) for hiding a visible label. In the latter rule a new hidden name that the visible label gets renamed to is chosen automatically.

  It turns out that most claims that we have made about the calculus in [MT00] only hold at the level of α-equivalence classes, but not at the level of concrete terms (see examples 2.18 and 2.19). While the proofs of [MT00] are done with an implicit assumption that we are in fact working with α-equivalence classes rather than concrete terms, in this presentation we felt that a rigorous treatment of renaming of hiddens must be supported by the same level of formalism for the other α-renamings.

  This approach is similar to (and partly inspired by) the identification of modules up to structural rewrite rules in [WV99], which includes α-renaming of variables, i.e. internal names of components, corresponding to our hidden names. Other structural rewrite rules of [WV99] include reordering of module components (which in our calculus are unordered in a module by definition) and swapping arguments of a linking operation.

- In this presentation we correct the error of [MT00], where the rule (GC) for garbage collection of hidden module bindings not referenced elsewhere in the module was introduced as an evaluation step. It turns out that this reduction rule defined as an evaluation step destroys computational soundness of the calculus (see section 2.6.2 for details). We have corrected the error by defining garbage collection as a non-evaluation step. The change has prompted some changes in the classification functions for the core module calculus and the linking calculus (see below).

- In the current presentation we redefine a classification function for the core module calculus and the linking calculus. Unlike the componentwise classification defined in [MT00], the new classification of modules has three classes: evaluatable (for modules in the domain of the evaluation function), a value (for modules all of whose components are values), and an error (for all other modules). Module values are further characterized by the componentwise classifications of their visible components. Similar classes are introduced for linking expressions. We fixed the inconsistency of the classification of linking expressions in [MT00] which considered an expression "linkable" (we have replaced the name "linkable" by "evaluatable" in the current work) if it had a linking operation in it, regardless of whether there were linking operations that could actually be performed. For instance, linking of two modules cannot be performed if there is a conflict between visible labels of the two modules. In the new classification expressions in which linking operation are present, but cannot be performed, are correctly considered to be errors. Another significant change in classification of linking expressions is that it is defined at the level of α-equivalence classes of expressions. This allows us to ignore the conflicts between the names of hidden components in expressions invloving (link), because at the level of α-equivalence classes such conflicts are resolved automatically. We show that the new classification functions for modules and linking expressions are well-defined with respect to α-renaming.

- While in [MT00] the reduction of link-level (let) expression is allowed at any point of evaluation, the current rule requires that the **let**-bound expression is evaluated to a module before it gets substituted into the body of the **let**. The change makes the reduction of **let** more consistent with the call-by-value nature of our calculus. However, this is not a *truly* call-by-value reduction rule, since we do not require that a module evaluates to a module value before the reduction can take place. Such a requirement would have caused module evaluation steps and linking steps to be interleaved, thus destroying the staged behavior of evaluation at the linking level. See section 2.5.7 for a more detailed discussion.

- In addition to module-level contexts which can be filled with terms (defined in [MT00]), we also introduce module contexts filled with other modules (denoted $\mathbb{M}$). These contexts are convenient for formalizing α-renaming of modules and garbage collection rule (GC). A meaningful definition of filling such contexts with modules includes cases when the module that fills a context imports hidden labels. Thus we have extended the class of modules to include those with imported hidden labels. Modules that do not import hiddens are called h-closed, and are denoted by H. In the linking calculus we restore the restriction that all modules which form a linking expression are h-closed. The restriction guarantees that no capture of hidden labels occurs during linking. Besides the new module contexts, we have introduced other kinds of contexts for technical reasons.

- We have made several changes in notations, the most significant are that we have unified notations for reduction arrows on the diagrams and in the text (we use $\circ\!\!\longrightarrow$ instead of $\hookrightarrow$ for non-evaluation step). L and $\mathbb{L}$ are used instead of F and $\mathbb{F}$ for linking expressions and linking contexts, and the linking calculus is denoted $\mathcal{L}$ instead of $\mathcal{F}$. We also use pairs of the form $(\mathbb{C}, M)$, where $\mathbb{C}$ is a context and M is a term, to denote subterm occurrences in terms, which allows us to specify a particular redex in a term.

1.5 Relation to Other Work

Our work follows a long tradition of using untyped calculi for reasoning about programming languages features: e.g., call-by-name vs. call-by-value semantics [Plo75], call-by-need semantics [AFM+95, AF97], state and control [FH92], and sharing and cycles [AK97, AB97]. Our notion of confluence with respect to evaluation avoids cyclic substitutions in the operational semantics, and so is related to the acyclic substitution restriction of Ariola and Klop [AK97]. Even though confluence in our calculus fails in the same way as in [AK97], our way of dealing with this problem is different. Instead of prohibiting cyclic substitutions in the calculus, as it is done in [AK97, WV99], we allow such substitutions in the calculus but do not consider them to be evaluation steps in the operational semantics. This turns out to be sufficient for achieving our goal of proving the computational soundness of the calculus. A different approach is taken in [AB97], which addresses the lack of confluence for unfolding operations on recursive terms by identifying such terms up to information content – i.e., a term has a unique infinite normal form.

This work is part of a renewed interest in linking issues that was inspired by Cardelli's call to arms [Car97]. Recent work on module systems and linking has focused on such issues as: sophisticated type systems for modules [HL94, Ler94, Sha99, Rus99]; the expressiveness of module systems (e.g., handling features like recursive modules [FF98, DS98, CHP99, AZ99], inheritance and mixins [DS96, DS98, AZ99, FRR00], and dynamic linking [FF98, WV99, Dug01, HWC01]); module systems for typed assembly language [GM99, Dug01, HWC01] binary compatibility in the context of program modifications [SA93, DEW99]; and modularizing module systems [Ler96, AZ99]. There has been relatively little focus on issues related to link-time optimization and cross-module transformations; exceptions are Fernandez's work on link-time optimization [Fer95], techniques for cross-module optimization in ML [BA97, Sha98], and work on just-in-time compilers (e.g, [PC97]). While these projects consider module systems and transformations that are considerably more sophisticated than the ones we study, they skirt the issue of meaning preservation, which is one of our main goals.

Our work stands out from other work on modules in two important respects:

1. *Distinguishing the calculus from the operational semantics:* We partition the reduction relation of the calculus ($\longrightarrow$) into *evaluation* (sometimes called *standard*) steps ($\Longrightarrow$) that define a small-step operational semantics and *non-evaluation* (*non-standard*) steps ($\circ\!\!\longrightarrow$). While this partitioning is common in the calculus world (e.g., [Plo75, FH92, AF97]), it is rare in the module/linking world. Typical work on modules (e.g., [Car97, AZ99]) gives only an operational semantics for modules. Yet in the context of link-time compilation, the notion of reduction in a calculus is essential for justifying meaning preserving program transformations. Without non-evaluation steps, even simple transformations like transforming $[F \mapsto \lambda x.(1+2)]$ to $[F \mapsto \lambda x.3]$ or $[A \mapsto 4, F \mapsto \lambda x.x + A]$ to $[A \mapsto 4, F \mapsto \lambda x.x + 4]$ are difficult to prove meaning preserving. Meaning preservation is difficult to discuss in the context of module research that is based on a calculus with no associated operational semantics (e.g.,[WV99]), for then there is no well-defined notion of observational equivalence. The only recent work on linking that we are aware of that considers both a calculus *and* an operational semantics is [FRR00], which presents a calculus intended to serve as an intermediate language for compiling object-oriented languages.

2. *Untyped calculus*: Unlike most recent work on modules and linking (with the notable exceptions of [WV99, FRR00]), our work considers only an untyped module language. This is not because we think types are unimportant, but for other reasons. First, types are orthogonal to our focus on computational soundness and weak distributivity; considering types at this point would only complicate the presentation. Second, introducing types often requires imposing restrictions that we would like to avoid. For example, to add types to their system, [AZ99] need to impose several restrictions on their untyped language: no components with recursive types, and no modules as components to other modules. Finally, we do not yet have anything new to say in the type dimension. We believe that it is straightforward to adapt an existing simple module type system (e.g., [Car97, FF98, AZ99]) to our calculus, and that type soundness and subject reduction would hold. On the other hand, we think that enriching our module system with polymorphic types is a very interesting avenue for future exploration. In particular, polymorphism based on union and intersection types has modularity properties that may make it particularly well-suited to the area of link-time compilation (see [Jim96, Ban97, KW99]).

2 The Module Calculus

In this section, we present a stratified calculus with three levels: a term calculus $\mathcal{T}$, a core module calculus $\mathcal{C}$, and a linking calculus $\mathcal{L}$. The three calculi are summarized in figure 1 and described in detail in sections 2.3–2.5. We first consider a core module calculus without a garbage collection rule; section 2.6 discusses how to extend it with garbage collection. Before presenting the details of the three calculi, we define some mathematical concepts and notations in section 2.1 and introduce notational conventions and properties relevant to all three levels of the module calculus in section 2.2.

2.1 Mathematical Preliminaries

2.1.1 Relations

A *binary relation* R over sets S and T is a subset of $S \times T$, and the notation $x \; R \; y$ means $(x, y) \in R$. We say x is in the *domain* of R, written $x \in dom(R)$, iff there exists a y such that $x \; R \; y$. We say that R is *injective* if $x \; R \; y_1$ and $x \; R \; y_2$ implies $y_1 = y_2$

A binary relation R over $S \times S$ is *transitive* iff $x \; R \; y$ and $y \; R \; z$ imply $x \; R \; z$, *symmetric* iff $x \; R \; y$ implies $y \; R \; x$, and *reflexive* iff $x \; R \; x$ for all $x \in S$. We use $R^?$ to stand for the reflexive closure of R, R^+ to stand for the transitive closure of R, R^* to stand for the reflexive, transitive closure of R, and $R^=$ to stand for the reflexive, symmetric, and transitive closure of R. For a relation written as an arrow, $\rightarrow$, we also use $\leftrightarrow$ for its reflexive, symmetric, and transitive closure. A binary relation R over S is an *equivalence relation* iff R is reflexive, symmetric, and transitive; $R^=$ is necessarily an equivalence relation.

Below we formulate and prove some general properties of relations that will be used later in our presentation.

Lemma 2.1. *If R is symmetric, then xR^*y iff $xR^=y$.* □

Proof. If xR^*y, then by definition $xR^=y$.

Suppose $xR^=y$. Let $xR^{-1}y$ denote the fact that yRx, and let $R^{\pm} = R \cup R^{-1}$. $xR^=y$ by definition implies that there exist $x_1, \dots, x_n$, $n \geq 0$, such that $x = x_1 R^{\pm} x_2 R^{\pm} \dots R^{\pm} x_n = y$. By symmetry of R $R^{\pm} = R$, therefore $x = x_1 R x_2 R \dots R x_n = y$. □

Definition 2.2. We say that a binary relation R is *independent* from a binary relation S if xRy, ySz implies that there exists y' such that xSy', $y'Rz$. □

Lemma 2.3. *Let R_1 and R_2 be two symmetric binary relations such that R_1 is independent from R_2, and let $R = R_1 \cup R_2$. Then $xR^=y$ implies that there exists z such that xR_1^*z, zR_2^*y.* □

Proof. By symmetry of R_1 and R_2 $xR^=y$ implies that xR^*y by lemma 2.1. If the sequence is of the form $xR_1^*zR_2^*y$, then we are done, otherwise there exist x_1, x_2, x_3 such that $xR_1^*x_1R_2x_2R_1x_3R^*y$. Suppose the length of the sequence $xR_1^*x_1$ is n, where $n \geq 0$, and the length of the entire sequence xR^*y is m ($m \geq 2$). By definition 2.2 there exists x_2' such that $x_1R_1x_2'R_2x_3$, then in the sequence $xR_1^*x_1R_1x_2'R_2x_3R^*y$ the length of the subsequence $xR_1^*x_1R_1x_2'$ of only R_1 steps is $n+1$, and the length of the entire new sequence is still m. If the new sequence is of the form $xR_1^*zR_2^*y$, then we are done, otherwise we transform the sequence again in the same way. The length of the sequence of only R_1 steps (before the first R_2 step) originating at x increases by 1 after every transformation, but it cannot exceed m, therefore the process will terminate, and the resulting sequence is of the form $xR_1^*zR_2^*y$. □

2.1.2 Functions

A *(partial) function* is a triple (A, B, g), where A and B are sets, respectively called the *source* and *target* of the function, and g, the *graph* of the function, is an injective binary relation over $A \times B$. In the following definitions, let f be the function (A, B, g). We say that the *signature* of f is $f : A \rightarrow B$ and that $x \in dom(f)$

iff $x \in dom(g)$. If $x \in dom(f)$, we use the *function application* notation $f(x)$ to stand for the unique $y \in B$ such that $x\ g\ y$; otherwise, we say that f is *undefined at* x. We say that f is a *total function* if $dom(f) = A$.

Let $B_\perp$ denote $B \cup \{\perp_B\}$, where $\perp_B$ (pronounced "bottom") is a distinguished element not in B. Any function (A, B, g) that is not total can be extended to a total function $(A, B_\perp, g')$ where $g' = g \cup \{(x, \perp) \mid x \notin dom(g)\}$.

2.1.3 Sequences

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

Given a sequence S, let the length of S be 0 if $S(1) = \perp_X$, or the largest integer i such that $S(i) \neq \perp_X$, or ω if $S(i) \neq \perp_X$ for all $i \in \mathbb{N}$. Let X^ω be the set of all sequences over X, and X^* be the set of all finite sequences over X. Let $[x_1, \ldots, x_n]$ denote the sequence S such that $S(i) = x_i$ for $i \leq n$ and $S(i) = \perp_X$ otherwise. In particular, $[]$ is the empty sequence. Given $S \in X^*$ and $S' \in X^\omega$, let $S;S'$ be the sequence that begins according to S and then continues with S', i.e., $(S;S')(i) = S(i)$ if $i \leq |S|$ and $(S;S')(i) = S'(i - |S|)$ otherwise. We shall assume that, where appropriate, there is an implicit coercion from X into X^ω which maps every element $x \in X$ into the length 1 sequence $[x]$. For example, using this implicit coercion, when $S \in X^\omega$ and $x \in X$ the notation $S;x$ stands for $S;[x]$ and $x;S$ stands for $[x];S$.

2.2 Calculus Notations and Properties

Let $\mathcal{X}$ range over $\{\mathcal{T}, \mathcal{C}, \mathcal{L}\}$. The definition for each calculus $\mathcal{X}$ consists of the following:

- The syntax for calculus terms $\mathbf{Term}_\mathcal{X}$ and for one or more sets of one-hole contexts $\mathbf{Context}_{\mathcal{X}',\mathcal{X}}$, where $\mathcal{X}'$ may or may not be the same as $\mathcal{X}$. A set $\mathbf{Context}_{\mathcal{X}',\mathcal{X}}$ ranges over one-hole contexts $\mathbb{X}$ that are filled with terms $Y \in \mathbf{Term}_{\mathcal{X}'}$ such that the result, written $\mathbb{X}\{Y\}$, is an element of $\mathbf{Term}_\mathcal{X}$. Due to the hierarchical structure of our module calculus, $\mathcal{X}'$ may be different from $\mathcal{X}$. For instance, in our hierarchy $\mathcal{T}$ contexts are filled with $\mathcal{T}$ terms; $\mathcal{C}$ contexts are filled with $\mathcal{T}$ or $\mathcal{C}$ terms; and $\mathcal{L}$ contexts are filled with $\mathcal{L}$ terms (which include $\mathcal{C}$ terms as a special case). We assume that the notation $\mathbb{X}\{Y\}$ is only used with such $\mathbb{X}$ and Y that the result of the filling is a well-formed term in $\mathbf{Term}_\mathcal{X}$. For instance, given a module context $\mathbb{D}$ and a term M, the notation $\mathbb{D}\{M\}$ is defined only if the resulting module is well-defined element of $\mathbf{Term}_\mathcal{C}$.

  Given a set of contexts $C_{\mathcal{X}',\mathcal{X}} \subseteq \mathbf{Context}_{\mathcal{X}',\mathcal{X}}$ and a relation R on $\mathbf{Term}_{\mathcal{X}'}$, we say that a relation S on $\mathbf{Term}_\mathcal{X}$ is a *contextual closure* of R w.r.t. $C_{\mathcal{X}',\mathcal{X}}$ iff $S = \{(X, X') \mid X = \mathbb{X}\{Y\}, X' = \mathbb{X}\{Y'\}, \mathbb{X} \in C_{\mathcal{X}',\mathcal{X}}, (Y, Y') \in R\}$.

  Given a term $X \in \mathbf{Term}_\mathcal{X}$, a *subterm* of X is a term $Y \in \mathbf{Term}_{\mathcal{X}'}$ such that $X = \mathbb{X}\{Y\}$ for some $\mathbb{X} \in \mathbf{Context}_{\mathcal{X}',\mathcal{X}}$. It is often important to specify the "position" at which a subterm occurs in an enclosing term. For this purpose, we define a *subterm occurrence* of $X \in \mathbf{Term}_\mathcal{X}$ as a a pair $(\mathbb{X}, Y)$ such that $\mathbb{X} \in \mathbf{Context}_{\mathcal{X}',\mathcal{X}}$, $Y \in \mathbf{Term}_{\mathcal{X}'}$, and $X = \mathbb{X}\{Y\}$. We use $=$ to denote componentwise equality on subterm occurrences.

- A small-step operational semantics of $\mathcal{X}$ is defined via an evaluation step relation $\Rightarrow_\mathcal{X}$. We also define a complementary non-evaluation step relation $\circ\!\!\rightarrow_\mathcal{X}$, and for each of the three calculi we define a one-step reduction relation $\rightarrow_\mathcal{X} \stackrel{\text{def}}{=} \Rightarrow_\mathcal{X} \cup \circ\!\!\rightarrow_\mathcal{X}$.[3] A term $X \in \mathbf{Term}_\mathcal{X}$ is *reducible* if there is a term Y such that $X \rightarrow_\mathcal{X} Y$; otherwise it is an $\mathcal{X}$ *normal form*. Similarly, X is *evaluatable* if there is a term Y such that $X \Rightarrow_\mathcal{X} Y$; otherwise it is an $\mathcal{X}$ *eval normal form*. The set $NF_{\rightarrow_\mathcal{X}}$ contains exactly the $\mathcal{X}$ normal forms, and the set $NF_{\Rightarrow_\mathcal{X}}$ contains exactly the $\mathcal{X}$ eval normal forms.

  The relation $\Rightarrow_\mathcal{X}$ is often defined in terms of an *evaluation context* $\mathbf{EvalContext}_{\mathcal{X}',\mathcal{X}} \subseteq \mathbf{Context}_{\mathcal{X}',\mathcal{X}}$, along the lines of [FF86]. Note that since the redex of a calculus may be a term of another calculus

[3] Alternatively we could have defined the rules for $\rightarrow_\mathcal{X}$ explicitly and then set $\circ\!\!\rightarrow_\mathcal{X}$ to be $\rightarrow_\mathcal{X} \setminus \Rightarrow_\mathcal{X}$. However, giving explicit rules for $\circ\!\!\rightarrow_\mathcal{X}$ clarifies the presentation.

Syntax for the Term Calculus ($\mathcal{T}$):

$$
\begin{array}{l}
c \in \mathbf{Const} \ = \ \textit{constant values} \quad x \in \mathbf{Variable} \ = \ \textit{term variables} \\
v \in \mathbf{Visible} \ = \ \textit{external labels} \quad h \in \mathbf{Hidden} \ = \ \textit{internal labels} \\
k, l \in \mathbf{Label} \ = \ \mathbf{Visible} \cup \mathbf{Hidden} \\
M, N \in \mathbf{Term}_{\mathcal{T}} \ ::= \ c \mid x \mid l \mid (\lambda x.M) \mid M_1 \mathbin{@} M_2 \mid M_1 \mathit{\ op\ } M_2 \\
\mathbb{C} \in \mathbf{Context}_{\mathcal{T},\mathcal{T}} \ ::= \ \Box \mid (\lambda x.\mathbb{C}) \mid \mathbb{C} \mathbin{@} M \mid M \mathbin{@} \mathbb{C} \mid \mathbb{C} \mathit{\ op\ } M \mid M \mathit{\ op\ } \mathbb{C} \\
V \in \mathbf{Value}_{\mathcal{T}} \ ::= \ c \mid x \mid \lambda x.M
\end{array}
$$

Notion of Reduction on Terms:

$$
\begin{array}{rcll}
(\lambda x.M \mathbin{@} V) & \rightsquigarrow_{\mathcal{T}} & M[x := V] & (\beta) \\
c_1 \mathit{\ op\ } c_2 & \rightsquigarrow_{\mathcal{T}} & c, \text{ where } c = \delta(op, c_1, c_2) & (\delta)
\end{array}
$$

Evaluation and Non-evaluation Steps:

$$
\begin{array}{l}
\mathbb{E} \in \mathbf{EvalContext}_{\mathcal{T},\mathcal{T}} \ ::= \ \Box \mid \mathbb{E} \mathbin{@} M \mid (\lambda x.M) \mathbin{@} \mathbb{E} \mid \mathbb{E} \mathit{\ op\ } M \mid c \mathit{\ op\ } \mathbb{E} \\
\begin{array}{rcll}
\mathbb{E}\{R\} & \Rightarrow_{\mathcal{T}} & \mathbb{E}\{Q\} \ \text{, where } R \rightsquigarrow_{\mathcal{T}} Q, & \text{(term-ev)} \\
\overline{\mathbb{E}}\{R\} & \circ\!\!\rightarrow_{\mathcal{T}} & \overline{\mathbb{E}}\{Q\} \ \text{, where } R \rightsquigarrow_{\mathcal{T}} Q. & \text{(term-nev)}
\end{array}
\end{array}
$$

Syntax for the Core Module Calculus ($\mathcal{C}$):

$$
\begin{array}{l}
D \in \mathbf{Term}_{\mathcal{C}} \ ::= \ [l_1 \mapsto M_1, \ldots, l_n \mapsto M_n] \quad (\text{abbreviated } [l_i \overset{n}{\underset{i=1}{\mapsto}} M_i]), \\
\quad \text{provided } FV(D) = \emptyset \text{ and } l_i = l_j \text{ implies } i = j. \\
H \in \mathbf{HTerm}_{\mathcal{C}} = \{D \mid \mathit{Imports}(D) \cap \mathbf{Hidden} = \emptyset\} \\
\mathbb{M} \in \mathbf{Context}_{\mathcal{C},\mathcal{C}} \ ::= \ [l_i \overset{n}{\underset{i=1}{\mapsto}} M_i, \Box] \\
\mathbb{D} \in \mathbf{Context}_{\mathcal{T},\mathcal{C}} \ ::= \ \mathbb{M}\{[l \mapsto \mathbb{C}]\} \\
\mathbf{Value}_{\mathcal{C}} \ ::= \ [v_i \overset{n}{\underset{i=1}{\mapsto}} V_i, h_j \overset{m}{\underset{j=1}{\mapsto}} V'_j] \\
\text{Projection Notation: } [l_i \overset{n}{\underset{i=1}{\mapsto}} M_i] \downarrow l_j = M_j, \text{ if } 1 \le j \le n, \text{ and otherwise undefined.}
\end{array}
$$

Evaluation and Non-evaluation Steps:

$$
\mathbb{G} \in \mathbf{EvalContext}_{\mathcal{T},\mathcal{C}} ::= \mathbb{M}\{[l \mapsto \mathbb{E}]\}
$$

$$
\begin{array}{rcll}
\mathbb{G}\{R\} & \Rightarrow_{\mathcal{C}} & \mathbb{G}\{Q\}, \text{ where } R \rightsquigarrow_{\mathcal{T}} Q. & \text{(comp-ev)} \\
\mathbb{G}\{l\} & \Rightarrow_{\mathcal{C}} & \mathbb{G}\{V\}, \text{ where } \mathbb{G}\{l\} \downarrow l = V. & \text{(subst-ev)} \\
\overline{\mathbb{G}}\{R\} & \circ\!\!\rightarrow_{\mathcal{C}} & \overline{\mathbb{G}}\{Q\}, \text{ where } R \rightsquigarrow_{\mathcal{T}} Q. & \text{(comp-nev)} \\
\overline{\mathbb{G}}\{l\} & \circ\!\!\rightarrow_{\mathcal{C}} & \overline{\mathbb{G}}\{V\}, \text{ where } \overline{\mathbb{G}}\{l\} \downarrow l = V. & \text{(subst-nev)}
\end{array}
$$

Syntax for the Linking Calculus ($\mathcal{L}$):

$$
\begin{array}{rcl}
I \in \mathbf{ModIdent} & ::= & \textit{module identifiers} \\
L \in \mathbf{Term}_{\mathcal{L}} & ::= & H \mid I \mid L_1 \oplus L_2 \mid L[v \overset{\text{ren}}{\leftarrow} v'] \mid L\{\text{hide } v\} \mid \mathbf{let}\ I = L_1 \ \mathbf{in}\ L_2 \\
\mathbb{L} \in \mathbf{Context}_{\mathcal{L},\mathcal{L}} & ::= & \Box \mid \mathbb{L} \oplus L \mid L \oplus \mathbb{L} \mid \mathbb{L}[v \overset{\text{ren}}{\leftarrow} v'] \mid \mathbb{L}\{\text{hide } v\} \mid \mathbf{let}\ I = \mathbb{L}\ \mathbf{in}\ L \mid \mathbf{let}\ I = L\ \mathbf{in}\ \mathbb{L} \\
\mathbf{Value}_{\mathcal{L}} & = & \mathbf{Value}_{\mathcal{C}}
\end{array}
$$

Evaluation and Non-evaluation Steps:

$$
\begin{array}{rcll}
H & \Rightarrow_{\mathcal{L}} & H', \text{ where } H \Rightarrow_{\mathcal{C}} H' & \text{(mod-ev)} \\
\mathbb{L}\{[k_i \overset{n}{\underset{i=1}{\mapsto}} M_i] \oplus [l_j \overset{m}{\underset{j=1}{\mapsto}} N_j]\} & \Rightarrow_{\mathcal{L}} & \mathbb{L}\{[k_i \overset{n}{\underset{i=1}{\mapsto}} M_i, l_j \overset{m}{\underset{j=1}{\mapsto}} N_j]\}, & \text{(link)} \\
& & \text{where } [k_i \overset{n}{\underset{i=1}{\mapsto}} M_i], [l_j \overset{m}{\underset{j=1}{\mapsto}} N_j] \in \mathbf{HTerm}_{\mathcal{C}} & \\
& & \text{and } (\cup_{i=1}^{n} k_i) \cap (\cup_{j=1}^{m} l_j) = \emptyset. & \\
\mathbb{L}\{H[v \overset{\text{ren}}{\leftarrow} v']\} & \Rightarrow_{\mathcal{L}} & \mathbb{L}\{H[v := v']\}, & \text{(rename)} \\
& & \text{where } v \in BL(H) \text{ implies } v' \notin BL(H). & \\
\mathbb{L}\{H\{\text{hide } v\}\} & \Rightarrow_{\mathcal{L}} & \mathbb{L}\{H[v := h]\}, & \text{(hide)} \\
& & \text{where } v \notin \mathit{Imports}(H),\ h \notin \mathit{Hid}(H). & \\
\mathbb{L}\{\mathbf{let}\ I = H\ \mathbf{in}\ L\} & \Rightarrow_{\mathcal{L}} & \mathbb{L}\{L[I := H]\}, & \text{(let)} \\
\mathbb{L}\{H\} & \circ\!\!\rightarrow_{\mathcal{L}} & \mathbb{L}\{H'\}, \text{ where } H \rightarrow_{\mathcal{C}} H' & \text{(mod-nev)} \\
& & \text{and } \mathbb{L} \neq \Box \text{ or } H \circ\!\!\rightarrow_{\mathcal{C}} H' &
\end{array}
$$

Figure 1: The three levels of the module calculus without GC. See section 2.2 for notational conventions and sections 2.3–2.5 for details on each level.

(for instance, redexes of the core module calculus $\mathcal{C}$ range over terms of $\mathcal{T}$), an evaluation context has two subscripts: one for the calculus of the redex, and one for the calculus of the term being evaluated. If $\mathbb{X}$ ranges over $\mathbf{EvalContext}_{\mathcal{X}',\mathcal{X}}$, then $\overline{\mathbb{X}}$ ranges over $\mathbf{Context}_{\mathcal{X}',\mathcal{X}} \setminus \mathbf{EvalContext}_{\mathcal{X}',\mathcal{X}}$ (i.e. the set of *non-evaluation contexts*). For pairs of rules such as (comp-ev) and (comp-nev), which only differ by the use of an evaluation versus a non-evaluation context, we introduce a notation for the combined calculus rule. For instance, we say that $D \longrightarrow_{\mathcal{C}} D'$ by the rule (comp) if either $D \Longrightarrow_{\mathcal{C}} D'$ by (comp-ev) or $D \circ\!\!\longrightarrow_{\mathcal{C}} D'$ by (comp-nev).

In our calculi, all reduction steps are specified in terms of a basic rewriting step that takes place in an unchanging context. In a one-step reduction $Y \longrightarrow_{\mathcal{X}} Z$, it is sometimes necessary to indicate "where" in Y the basic rewriting step occurs. We indicate this location, known as a *redex*, by a subterm occurrence $(\mathbb{X}, R)$ of Y such that $(\mathbb{X}, Q)$ is a subterm occurrence of Z and the basic rewriting step rewrites R to Q. The notation $Y \overset{(\mathbb{X},R)}{\longrightarrow}_{\mathcal{X}} Z$ indicates the redex reduced by a reduction step. The notations $Y \overset{(\mathbb{X},R)}{\Longrightarrow}_{\mathcal{X}} Z$ and $Y \overset{(\mathbb{X},R)}{\circ\!\!\longrightarrow}_{\mathcal{X}} Z$ are used to highlight the redex reduced in evaluation and non-evaluation steps, respectively.

- A set $\mathbf{Value}_{\mathcal{X}} \subseteq NF_{\Longrightarrow_{\mathcal{X}}}$ of *values*, terms which intuitively are "results" for computations in $\mathcal{X}$.
- A set $\mathbf{Obs}_{\mathcal{X}}$ of *observables*, tokens that summarize the "observable properties" of a computation in $\mathcal{X}$. The set $\mathbf{Obs}_{\mathcal{X}}$ must contain the token $\mathbf{evaluatable}_{\mathcal{X}}$, which is used to classify evaluatable terms, and the token $\bot_{\mathcal{X}}$, which is used to classify a diverging evaluation sequence (and sometimes errors). $\mathbf{Obs}_{\mathcal{X}}$ differs from $\mathbf{Value}_{\mathcal{X}}$ in that (1) a single observable token may denote a set of values and (2) observable tokens can denote non-value terms, such as evaluatable terms and error terms.
- A classification function $Cl_{\mathcal{X}}$ that maps each term to a token in $\mathbf{Obs}_{\mathcal{X}}$ that summarizes its state w.r.t. evaluation. The $Cl_{\mathcal{X}}$ function must satisfy the following two properties:

  Property 2.4. *$Cl_{\mathcal{X}}$ maps every evaluatable term (and no eval normal form) to* $\mathbf{evaluatable}_{\mathcal{X}}$. □

  Property 2.5. *The image of* $\mathbf{Value}_{\mathcal{X}}$ *under $Cl_{\mathcal{X}}$ is disjoint from that of* $\mathbf{Term}_{\mathcal{X}} \setminus \mathbf{Value}_{\mathcal{X}}$. □

  We sometimes summarize the image of $\mathbf{Value}_{\mathcal{X}}$ under $Cl_{\mathcal{X}}$ using the single token $\mathbf{value}_{\mathcal{X}}$.

The following are general properties of calculi. As we shall see in section 3, they are important for proving that certain classes of program transformations are meaning preserving.

Definition 2.6 (Confluence). The $\longrightarrow$ relation is *confluent* if $M_1 \longrightarrow^* M_2$ and $M_1 \longrightarrow^* M_3$ implies the existence of M_4 such that $M_2 \longrightarrow^* M_4$, $M_3 \longrightarrow^* M_4$. A calculus X has confluence if $\longrightarrow_{\mathcal{X}}$ is confluent. □

Definition 2.7 (Standardization). A calculus $\mathcal{X}$ has the *standardization property* if for any sequence $M_1 \longrightarrow^*_{\mathcal{X}} M_2$ there exists M_3 such that $M_1 \Longrightarrow^*_{\mathcal{X}} M_3 \circ\!\!\longrightarrow^*_{\mathcal{X}} M_2$. A sequence of the form $M_1 \Longrightarrow^*_{\mathcal{X}} M_3 \circ\!\!\longrightarrow^*_{\mathcal{X}} M_2$ is called *standard*. □

Remark 2.8. Our definition of standardization is not the traditional one. Traditionally, a standard reduction sequence is defined as a sequence satisfying certain relation between redexes (see [CF58, Plo75, Bar84]). Our definition is based only on definition of $\Longrightarrow_{\mathcal{X}}$ and $\circ\!\!\longrightarrow_{\mathcal{X}}$ in a calculus $\mathcal{X}$. However, the definitions in the literature above, as well as our definition, capture the same idea: a standard sequence is such that it reduces first the redexes "needed" for a term to reach a normal form of certain kind, for instance a value, and after that the redexes that do not contribute to reaching a normal form, such as redexes under a lambda. See section 3.5.2 for a more detailed discussion. □

In a calculus with a confluent $\Longrightarrow_{\mathcal{X}}$ every non-diverging (w.r.t. $\Longrightarrow_{\mathcal{X}}$) term evaluates to a unique $\mathcal{X}$ eval normal form. This leads to the following definitions:

Definition 2.9 (Evaluation). Assume that $\Longrightarrow_{\mathcal{X}}$ is confluent. The partial function $Eval_{\mathcal{X}} : \mathbf{Term}_{\mathcal{X}} \to NF_{\Longrightarrow_{\mathcal{X}}}$ is defined so that $Eval_{\mathcal{X}}(X)$ is the unique term $Y \in NF_{\Longrightarrow_{\mathcal{X}}}$ (if it exists) such that $X \Longrightarrow^* Y$. If $Eval_{\mathcal{X}}(X)$ is undefined, X is said to *diverge*. □

Definition 2.10 (Outcome). Assume that $\Rightarrow_{\mathcal{X}}$ is confluent. The total function $Outcome_{\mathcal{X}} : \mathbf{Term}_{\mathcal{X}} \to \mathbf{Obs}_{\mathcal{X}}$ maps a term X to the classification of its $\mathcal{X}$ eval normal form $Cl_{\mathcal{X}}(Eval_{\mathcal{X}}(X))$ if it exists, and to the symbol $\perp_{\mathcal{X}}$ otherwise. □

REMARK 2.11. In some calculi with a confluent $\Rightarrow_{\mathcal{X}}$, it is possible that a term has a unique $\mathcal{X}$ eval normal form and also has a $\Rightarrow_{\mathcal{X}}$-diverging path. This does not contradict confluence of $\Rightarrow_{\mathcal{X}}$, since it is possible that every term on a diverging path can be reduced to the normal form. In this case, according to our definition, the outcome of the term is the classification of the $\mathcal{X}$ eval normal form, so $Outcome_{\mathcal{X}}(X) = \perp_{\mathcal{X}}$ only if a term diverges on *every* evaluation path. However, this is a technical issue which does not arise in our calculi: we show that for all calculi introduced in this presentation for which $\mathcal{X}$ eval normal form is defined ($\mathcal{X}$ eval normal form is not defined for calculi of concrete terms, only for calculi of α-equivalence classes), if a term has an $\mathcal{X}$ eval normal form, then it does not have a diverging evaluation path. See lemma 2.62 for the calculus $\mathcal{C}\backslash\alpha$ and lemma 2.113 for $\mathcal{L}\backslash\alpha$. □

We shall see that for all $\mathcal{X} \in \{\mathcal{T}, \mathcal{C}, \mathcal{L}\}$, $\Rightarrow_{\mathcal{X}}$ and $\rightarrow_{\mathcal{X}}$ are *not* confluent and $\mathcal{X}$ does *not* have the standardization property at the level of the concrete terms defined in figure 1. These properties fail to hold for technical reasons involving the choice of new names in certain reduction steps. However, it turns out that these properties *do* hold at the level of "α-equivalence" classes of terms in these calculi. That is, for all $\mathcal{X} \in \{\mathcal{T}, \mathcal{C}, \mathcal{L}\}$, there is a binary *$\alpha$-equivalence* relation $=^{\mathcal{X}}_{\alpha}$ on $\mathbf{Term}_{\mathcal{X}}$ that identifies terms in $\mathcal{X}$ modulo various kinds of renaming in $\mathcal{X}$. We shall show in each of the three calculi we study that notions like $\Rightarrow_{\mathcal{X}}$, $\circ\!\!\rightarrow_{\mathcal{X}}$, $\rightarrow_{\mathcal{X}}$, $\mathbf{Value}_{\mathcal{X}}$, and $Cl_{\mathcal{X}}$ can be "lifted" from the level of concrete terms to the level of α-equivalence classes of terms (i.e., terms modulo $=^{\mathcal{X}}_{\alpha}$). This means that we can sensibly consider each of the three calculi at the level of α-equivalence classes in addition to the level of concrete terms. We shall use the notation $\mathcal{X}\backslash\alpha$ to stand for the the calculus $\mathcal{X}$ considered at the α-equivalence level, in contradistinction with the notation $\mathcal{X}$, which we shall assume always indicates the calculus at the level of concrete terms. For instance, $\Rightarrow_{\mathcal{X}\backslash\alpha}$ stands for the lifting of $\Rightarrow_{\mathcal{X}}$ to the α-equivalence classes of a calculus $\mathcal{X}$.

REMARK 2.12. For all $\mathcal{X} \in \{\mathcal{T}, \mathcal{C}, \mathcal{L}\}$, the lack of confluence for $\Rightarrow_{\mathcal{X}}$ means that $Eval_{\mathcal{X}}$ and $Outcome_{\mathcal{X}}$ functions do not make sense at the level of concrete terms. However, we shall see that $\Rightarrow_{\mathcal{X}\backslash\alpha}$ is confluent for all three calculi, so that $Eval_{\mathcal{X}\backslash\alpha}$ and $Outcome_{\mathcal{X}\backslash\alpha}$ are defined. □

2.3 Term Calculus ($\mathcal{T}$)

2.3.1 Syntax of Terms

The module calculus is built on top of a term calculus $\mathcal{T}$, a typical call-by-value λ-calculus that includes constants (assumed to include integers) and binary operators (*op* is assumed to include standard arithmetic operations on integers). It could easily be extended to include other standard values and constructs (e.g., additional operators, binding constructs, conditionals, sums, and products) without affecting the results presented in this paper. Term-level recursion is unnecessary because recursion is supported at the core module calculus level.

The free variables of a term M, written $FV(M)$, are defined as usual.

Definition 2.13 (Free variables in $\mathcal{T}$). The set of *free variables* of a term $M \in \mathbf{Term}_{\mathcal{T}}$, written FV(M), is defined as follows:

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

□

The following definition of substitution is also traditional (e.g., see [Plo75]). We give it here because it is important for the definition of α-renaming in section 2.3.3.

Definition 2.14 (Substitution in $\mathcal{T}$). 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:

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

□

For interfacing with the module language in which it is embedded, the term syntax includes two disjoint classes of labels (**Visible** and **Hidden**) whose union, **Label**, is itself disjoint from **Variable**. Because there is no construct for declaring labels at the term level, labels always occur "free" in a term; the set of labels occurring in a term M is written $FL(M)$. The result of a substitution of a term M' for a label l in M is written $M[l := M']$. Even though in general such a substitution may capture free variables of M', we only use it in cases when $FV(M') = \emptyset$. Since $\mathcal{T}$ has no label declaration constructs, label capture is not an issue.

2.3.2 Evaluation and Reduction of Terms

Figure 1 defines $\Longrightarrow_{\mathcal{T}}$ and $\circ\!\!\longrightarrow_{\mathcal{T}}$ for terms with fixed names of bound variables. Later (see definition 2.37) we extend these definitions to $\alpha_{\mathcal{T}}$-equivalence classes of terms.

Both $\Longrightarrow_{\mathcal{T}}$ and $\circ\!\!\longrightarrow_{\mathcal{T}}$ are defined via a redex/contractum relation $\leadsto_{\mathcal{T}}$ specified by a call-by-value β rule and a δ rule (unspecified) for binary functions on constants. Terms in $dom(\leadsto_{\mathcal{T}})$ are called *term redexes*. The relations $\Longrightarrow_{\mathcal{T}}$ and $\circ\!\!\longrightarrow_{\mathcal{T}}$ are contextual closures of $\leadsto_{\mathcal{T}}$ with respect to an *evaluation context* $\mathbb{E}$ and a *non-evaluation context* $\overline{\mathbb{E}}$. It is easy to see that $\longrightarrow_{\mathcal{T}}$ (defined as $\Longrightarrow_{\mathcal{T}} \cup \circ\!\!\longrightarrow_{\mathcal{T}}$) is the contextual closure of $\leadsto_{\mathcal{T}}$ with respect to a general context $\mathbb{C}$.

Example 2.15. The following reduction sequence illustrates both evaluation and non-evaluation steps:[4]

$$
\begin{array}{ll}
\multicolumn{2}{l}{(\lambda x.(\lambda f.f \mathbin{@} (f \mathbin{@} x)) \mathbin{@} (\lambda y.y + (2 * 3))) \mathbin{@} 1} \\
\circ\!\!\xrightarrow{\delta}_{\mathcal{T}} & (\lambda x.(\lambda f.f \mathbin{@} (f \mathbin{@} x)) \mathbin{@} (\lambda y.y + 6)) \mathbin{@} 1 \\
\circ\!\!\xrightarrow{\beta}_{\mathcal{T}} & (\lambda x.(\lambda y.y + 6) \mathbin{@} ((\lambda y.y + 6) \mathbin{@} x)) \mathbin{@} 1 \\
\circ\!\!\xrightarrow{\beta}_{\mathcal{T}} & (\lambda x.(\lambda y.y + 6) \mathbin{@} (x + 6)) \mathbin{@} 1 \\
\stackrel{\beta}{\Longrightarrow}_{\mathcal{T}} & (\lambda y.y + 6) \mathbin{@} (1 + 6) \\
\stackrel{\delta}{\Longrightarrow}_{\mathcal{T}} & (\lambda y.y + 6) \mathbin{@} 7
\end{array}
$$

The first three steps take place in the body of an abstraction and are therefore non-evaluation steps. Note that $(\lambda y.y + 6) \mathbin{@} (x + 6)$ cannot reduce to $(x + 6) + 6$ because the β rule in $\mathcal{T}$ is call-by-value and $(x + 6)$ is not a value. However, $(\lambda y.y + 6) \mathbin{@} x$ can reduce to $x + 6$ because the variable x is a value.

As noted in section 2.2, the redex of an evaluation or non-evaluation step can be indicated by a subterm occurrence. For example:

[4] In examples, we adopt the usual convention that the body of a λ abstraction extends as far right as possible, to first unmatched closing parenthesis or to the end of the term, whichever comes first. Explicit parentheses are used to override this convention or to clarify the term structure.

- the redex in the first $\overset{\delta}{\circ\!\!\longrightarrow}_{\mathcal{T}}$ step above is $((\lambda x.(\lambda f.f \ @ \ (f \ @ \ x)) \ @ \ (\lambda y.y + \Box)) \ @ \ 1, (2 * 3))$,
- the redex in the first $\overset{\beta}{\circ\!\!\longrightarrow}_{\mathcal{T}}$ step is $((\lambda x.\Box) \ @ \ 1, (\lambda f.f \ @ \ (f \ @ \ x)) \ @ \ (\lambda y.y + 6))$, and
- the redex in the first $\overset{\beta}{\Longrightarrow}_{\mathcal{T}}$ step is $(\Box, (\lambda x.(\lambda y.y + 6) \ @ \ (x + 6)) \ @ \ 1)$.

□

A term M can be uniquely classified with respect to evaluation via $Cl_{\mathcal{T}}(M)$, defined as:

$\mathbf{const}(c)$	if $M = c$	$\mathbf{abs}$	if $M = \lambda x.N$	$\mathbf{evaluatable}_{\mathcal{T}}$	if $M = \mathbb{E}\{R\}$
$\mathbf{var}$	if $M = x$	$\mathbf{stuck}(l)$	if $M = \mathbb{E}\{l\}$	$\mathbf{error}_{\mathcal{T}}$	otherwise

The $\mathbf{const}(c)$, **var**, and **abs** classifications represent the "observable" aspects of values. The class $\mathbf{stuck}(l)$ contains those terms for which evaluation is "stuck" pending resolution of the label l. Our notion of "stuck" differs from that found elsewhere in the rewriting literature (e.g., [Plo81]), where the word is used to characterize error terms on which no progress can be made. We use the token $\mathbf{error}_{\mathcal{T}}$ to classify this latter kind of term.

The token $\mathbf{evaluatable}_{\mathcal{T}}$ classifies those terms on which an evaluation step may be taken. By the following lemma, every such term has a unique evaluation redex (recall that $=$ is defined on subterm occurrences as componentwise equality).

Lemma 2.16 (Uniqueness of $\Rightarrow_{\mathcal{T}}$-Redexes). *If* $\mathbb{E}\{R\} = \mathbb{E}'\{R'\}$, *then* $(\mathbb{E}, R) = (\mathbb{E}', R')$. □

Proof. By inspection of the definition of $\mathbb{E}$. □

It turns out that there are many desirable properties that do not hold at the level of "concrete" terms in $\mathcal{T}$ (i.e., terms with particular λ-bound variable names) because the fresh variable names that may be introduced by β-reduction are not unique. For instance, despite the fact that every evaluatable term has a unique redex (lemma 2.16), $\Rightarrow_{\mathcal{T}}$ is not a function on terms.

Example 2.17 ($\Rightarrow_{\mathcal{T}}$ is not a Function). Let $M_1 = (\lambda x.\lambda y.x) \ @ \ y$. Then $M_1 \Rightarrow_{\mathcal{T}} \lambda w.y = M_2$ and $M_1 \Rightarrow_{\mathcal{T}} \lambda z.y = M_3$, but $M_2 \neq M_3$. □

In the example, the bound y is renamed by the substitution process during β-reduction to avoid variable capture. Both w and z are valid choices of the new name, so the resulting term is not unique. However, as shown in the next section, all results of $\Rightarrow_{\mathcal{T}}$ are equal modulo a notion of α-equivalence, and $\Rightarrow_{\mathcal{T}\backslash\alpha}$ is a function. Recall that, according to the notation introduced in section 2.2, $\Rightarrow_{\mathcal{T}\backslash\alpha}$ denotes the "lifting" of $\Rightarrow_{\mathcal{T}}$ to the calculus $\mathcal{T}\backslash\alpha$ of $\alpha_{\mathcal{T}}$-equivalence classes of terms of $\mathcal{T}$.

The following examples show that the fresh variable names introduced by β-reduction also thwart the confluence of evaluation/reduction and standardization of $\mathcal{T}$ at the level of concrete terms. However, in the following subsection we show that these properties hold for α-equivalence classes of $\mathcal{T}$ terms.

Example 2.18 (Lack of Confluence of $\Rightarrow_{\mathcal{T}}$ / $\longrightarrow_{\mathcal{T}}$ for Concrete Terms). Using terms from example 2.17, we see that $M_1 \Rightarrow_{\mathcal{T}} M_2$ and $M_1 \Rightarrow_{\mathcal{T}} M_3$. But since M_2 and M_3 are both evaluation and reduction normal forms, there is no N such that (1) $M_2 \Rightarrow^*_{\mathcal{T}} N$ and $M_3 \Rightarrow^*_{\mathcal{T}} N$ or (2) $M_2 \longrightarrow^*_{\mathcal{T}} N$ and $M_3 \longrightarrow^*_{\mathcal{T}} N$ □

Example 2.19 (Lack of Standardization for Reduction on Concrete $\mathcal{T}$ Terms). Consider the following reduction sequence:

$$(\lambda x.((\lambda y.(y \ @ \ y)) \ @ \ \lambda z.x)) \ @ \ z \circ\!\!\longrightarrow_{\mathcal{T}} (\lambda x.((\lambda z.x) \ @ \ (\lambda z.x))) \ @ \ z \Rightarrow_{\mathcal{T}} (\lambda a.x) \ @ \ (\lambda b.x)$$

In the final β-reduction step, we assume that the process of substituting z for x in the two copies of $\lambda z.x$ renames the λ-bound z differently in the two copies. A standard sequence for the above reduction sequence must have the form

$$(\lambda x.((\lambda y.(y \ @ \ y)) \ @ \ \lambda z.x)) \ @ \ z \Rightarrow_{\mathcal{T}} (\lambda y.(y \ @ \ y)) \ @ \ (\lambda w.z) \circ\!\!\longrightarrow_{\mathcal{T}} (\lambda w.z) \ @ \ (\lambda w.z)$$

for some variable w. Since it is impossible for w to be both a and b, standardization is violated. □

2.3.3 $\mathcal{T}$ Terms Modulo α-Equivalence

Since variable names serve only to specify the "wiring" between the points of declaration and use for λ-bound variables, we intuitively expect that terms in $\mathcal{T}$ that differ only by renamings of their λ-bound variables should somehow be indistinguishable. This intuition is formalized below via a notion of *α-renaming* that gives rise to *α-equivalence* classes of $\mathcal{T}$ terms that are identical modulo α-renaming. A key motivation for formalizing α-equivalence in $\mathcal{T}$ is that several important properties shown above to fail at the level of concrete terms in $\mathcal{T}$ hold at the level of α-equivalence classes of $\mathcal{T}$ terms: (1) $\Rrightarrow_{\mathcal{T}}$ is a function; (2) confluence of $\Rrightarrow_{\mathcal{T}}$ and $\longrightarrow_{\mathcal{T}}$; and (3) standardization of $\mathcal{T}$.

Notions of α-renaming and α-equivalence are well-known in the λ calculus (e.g., [CF58, Plo75, Bar84]). Here we present a particular approach to α-renaming in $\mathcal{T}$ that is easy to extend to other kinds of renaming at the core module ($\mathcal{C}$) and linking ($\mathcal{L}$) levels of our module calculus. It turns out that, just as in $\mathcal{T}$, many properties of $\mathcal{C}$ and $\mathcal{L}$ only hold at the level of α-equivalence classes of terms, not at the level of concrete terms.

Definition 2.20 (Elementary $\alpha_{\mathcal{T}}$-Renaming). We say that $\lambda y.M$ reduces to $\lambda x.M'$ by *elementary $\alpha_{\mathcal{T}}$-renaming*, written $\lambda y.M \leadsto_{\alpha_{\mathcal{T}}} \lambda x.M'$, iff $x \neq y$, $M' = M[y := x]$ and the following two conditions hold:

1. $x \notin FV(M)$,
2. $M = \mathbb{C}\{\lambda x.N\}$ implies that either $y \notin FV(N)$, or $\mathbb{C} = \mathbb{C}_1\{\lambda y.\mathbb{C}_2\}$.

□

The first condition guarantees that no free variables named x in M are accidentally captured by renaming the bound variable of the outermost λ from y to x. The second condition guarantees that renaming a free y to x in M does not require renaming any bound occurrences of any variable in M. Without the second condition, the last clause of definition 2.14 can cause the substitution $M[x := y]$ to rename multiple bound variables in M, possibly some not even named x. So an elementary $\alpha_{\mathcal{T}}$-renaming guarantees all the free occurrences of y in M are renamed, but no other variables.

Lemma 2.21 (Unique Renaming Property of $\leadsto_{\alpha_{\mathcal{T}}}$). *If $\lambda y.N \leadsto_{\alpha_{\mathcal{T}}} \lambda x.M$, then the only variables renamed by the substitution $N[y := x]$ are free occurrences of y in N. In particular, the substitution does not rename any bound variables in N.* □

Proof. By induction on the structure of subterms P of N reached by the substitution process. They key cases are:

- $P = y$: $y \in FV(P)$ implying $y \in FV(N)$, and $y[y := x] = x$, so a free y in N is renamed to x.
- $P = z$, where $z \neq y$: $z \in FV(P)$ implying $z \in FV(N)$, and $z[y := x] = z$, so free variables of N other than y are not renamed.
- $P = \lambda y.Q$: $(\lambda y.Q)[y := x] = \lambda y.Q$, so the λ-bound y and any other variables occurring in Q are not renamed.
- $P = \lambda x.Q$: By condition 2 of definition 2.20, $y \notin FV(Q)$. (P cannot be enclosed in a λy within N because then substitution would not have descended inside the enclosing λy to reach P.) So $(\lambda x.Q)[y := x] = \lambda x.Q$; the λ-bound x and any other variables occurring in Q are not renamed. Here, condition 2 of definition 2.20 effectively prevents any renaming of bound variables implied by the last line of definition 2.14.
- $P = \lambda z.Q$, where $z \neq x$ and $z \neq y$: $(\lambda z.Q)[y := x] = \lambda z.(Q[y := x])$. Clearly, the λ-bound z is not renamed, and by the inductive hypothesis only free y in Q are renamed.

□

Lemma 2.22 (Symmetry of $\rightsquigarrow_{\alpha_{\mathcal{T}}}$). *$\rightsquigarrow_{\alpha_{\mathcal{T}}}$ is symmetric. I.e., if $\lambda y.M \rightsquigarrow_{\alpha_{\mathcal{T}}} \lambda x.N$, then $\lambda x.N \rightsquigarrow_{\alpha_{\mathcal{T}}} \lambda y.M$.* □

Proof. Assume that $\lambda y.M \rightsquigarrow_{\alpha_{\mathcal{T}}} \lambda x.N$. We wish to show $\lambda x.N \rightsquigarrow_{\alpha_{\mathcal{T}}} \lambda y.M$, which follows from showing the following three facts:

1. $y \notin FV(N)$: By the definition of $\rightsquigarrow_{\alpha_{\mathcal{T}}}$, $N = M[y := x]$. By lemma 2.21, N is syntactically identical to M modulo the renaming of all free occurrences y in M to x. So N has no free occurrences of y.

2. $N = \mathbb{C}\{\lambda y.P\}$ implies that either $x \notin FV(P)$ or $\mathbb{C} = \mathbb{C}_1\{\lambda x.\mathbb{C}_2\}$: Suppose that $N = \mathbb{C}\{\lambda y.P\}$ and $x \in FV(P)$. We show that a free occurrence of x in P cannot also be free in N; it must instead be bound by an enclosing λx in N. Assume otherwise – i.e., that a free x in P is also free in N. By condition (1) of definition 2.20 for $\lambda y.M \rightsquigarrow_{\alpha_{\mathcal{T}}} \lambda x.N$, we know $x \notin FV(M)$. This implies that all free occurrences of x in N must be the result of renaming a free y in M to x. But the given occurrence of x could not be the result of naming a free y in M because it occurs in the scope of λy, and such an occurrence of y is not free in M – a contradiction.

3. $M = N[x := y]$: From the proof of condition (1), we know that N is syntactically identical to M modulo the renaming of all free occurrences y in M to x. The substitution $M' = N[x := y]$ renames all these free occurrences of x back to free occurrences of y, so M' is the same as M except possibly for renaming of bound variables from the last clause of definition 2.14. As shown in the proof of lemma 2.21, condition (2) implies that the last clause in definition 2.14 is never enabled in the substitution $N[x := y]$, so no bound variables are renamed. Thus, $M = M'$.

□

Definition 2.23 ($\alpha_{\mathcal{T}}$-Renaming). We say that M reduces to M' by *one-step $\alpha_{\mathcal{T}}$-renaming*, written $M \longrightarrow_{\alpha_{\mathcal{T}}} M'$, iff $M = \mathbb{C}\{\lambda y.N\}$, $M' = \mathbb{C}\{\lambda x.N'\}$, and $\lambda y.N \rightsquigarrow_{\alpha_{\mathcal{T}}} \lambda x.N'$. We say that the triple $(\mathbb{C}, y, x)$ is the *$\alpha_{\mathcal{T}}$-redex* of the one-step $\alpha_{\mathcal{T}}$-renaming, and write $M \xrightarrow{(\mathbb{C},y,x)}_{\alpha_{\mathcal{T}}} M'$ to specify the $\alpha_{\mathcal{T}}$-redex of the $\alpha_{\mathcal{T}}$-renaming step. We write $M \xrightarrow{S}{}^{*}_{\alpha_{\mathcal{T}}} M'$ iff S is the sequence of $\alpha_{\mathcal{T}}$-redexes reduced in the multi-step $\alpha_{\mathcal{T}}$-renaming. □

The $\alpha_{\mathcal{T}}$-redex in a one-step $\alpha_{\mathcal{T}}$-renaming specifies which bound variable in the term has been renamed, and to which name. The old name (y in the definition above) is uniquely determined by the context $\mathbb{C}$, however we have added it to the syntax of an $\alpha_{\mathcal{T}}$-redex for convenience.

It follows from Lemma 2.22 that both $\longrightarrow_{\alpha_{\mathcal{T}}}$ and $\longrightarrow^*_{\alpha_{\mathcal{T}}}$ are symmetric, so $\longrightarrow^*_{\alpha_{\mathcal{T}}}$ is the same relation as $\longrightarrow^{=}_{\alpha_{\mathcal{T}}}$. We use the notation $=^{\mathcal{T}}_{\alpha}$ as a shorthand for this relation. Since $=^{\mathcal{T}}_{\alpha}$ is an equivalence relation, it induces $\alpha_{\mathcal{T}}$-equivalence classes of terms w.r.t. $=^{\mathcal{T}}_{\alpha}$:

Definition 2.24 ($\alpha_{\mathcal{T}}$-Equivalence Class). If $M \in \mathbf{Term}_{\mathcal{T}}$, then its *$\alpha_{\mathcal{T}}$-equivalence class*, written ${}_{\alpha_{\mathcal{T}}}\langle M\rangle$, is defined as ${}_{\alpha_{\mathcal{T}}}\langle M\rangle = \{N \in \mathbf{Term}_{\mathcal{T}} \mid N =^{\mathcal{T}}_{\alpha} M\}$. We say that any $N \in {}_{\alpha_{\mathcal{T}}}\langle M\rangle$ is a *representative* of ${}_{\alpha_{\mathcal{T}}}\langle M\rangle$. We use $M_{\alpha_{\mathcal{T}}}$ and $N_{\alpha_{\mathcal{T}}}$ as meta-variables that range over the set of $\alpha_{\mathcal{T}}$-equivalence classes of terms of $\mathcal{T}$. □

Assumption 2.25. We assume that all α-equivalence classes (w.r.t. all α-equivalence relations, such as $\longrightarrow_{\alpha_{\mathcal{T}}}$ defined here, $\longrightarrow_{\alpha_{\mathcal{C}}}$ defined in section 2.4, etc.) that we refer to in lemmas, examples, etc., are non-empty. The same holds for α-equivalence classes of entities other than terms, such as subterm occurrences and contexts, introduced later. □

The relation $=^{\mathcal{T}}_{\alpha}$ defined above is equivalent to traditional notions of $\alpha_{\mathcal{T}}$-equivalence on terms defined by induction on the structure of a term (e.g., see [Plo75]). However, the one-step relation $\longrightarrow_{\alpha_{\mathcal{T}}}$ is convenient for reasoning about the interaction (or lack thereof) between $\alpha_{\mathcal{T}}$-renaming and $\Longrightarrow_{\mathcal{T}}$, $\circ\!\!\longrightarrow_{\mathcal{T}}$, and $\longrightarrow_{\mathcal{T}}$, and between $\alpha_{\mathcal{T}}$-renaming and other kinds of renaming and reduction at the $\mathcal{C}$ and $\mathcal{L}$ levels of the calculus. Additionally, one-step alpha-renaming at the core module level allows us to distinguish results of renaming subterm occurrences which would be indistinguishable had we defined alpha-equivalence classes via the structure of terms (see section 2.4.3 for details) Note that since each $\longrightarrow_{\alpha_{\mathcal{T}}}$ renames exactly one bound variable, multiple $\longrightarrow_{\alpha_{\mathcal{T}}}$ steps are required to α-rename several bound variables.

Example 2.26 ($\alpha_{\mathcal{T}}$-Renaming of Terms). Let $M_1 = \lambda x.\lambda y.(y \mathbin{@} x)$ and $M_2 = \lambda y.\lambda x.(x \mathbin{@} y)$. Then $M_1 =_{\alpha}^{\mathcal{T}} M_2$, as shown by the following sequence of one-step $\alpha_{\mathcal{T}}$-renamings:

$$\lambda x.\lambda y.(y \mathbin{@} x) \xrightarrow{((\lambda x.\square),y,z)}_{\alpha_{\mathcal{T}}} \lambda x.\lambda z.(z \mathbin{@} x) \xrightarrow{(\square,x,y)}_{\alpha_{\mathcal{T}}} \lambda y.\lambda z.(z \mathbin{@} y) \xrightarrow{((\lambda y.\square),z,x)}_{\alpha_{\mathcal{T}}} \lambda y.\lambda x.(x \mathbin{@} y)$$

□

The notions of evaluation and non-evaluation steps, as well as the notion of classification, can be lifted from terms to $\alpha_{\mathcal{T}}$-equivalence classes of terms in a natural way. Below, we show that the redexes of $\Longrightarrow_{\mathcal{T}}$ and $\circ\!\!\longrightarrow_{\mathcal{T}}$ are preserved by $\alpha_{\mathcal{T}}$-renaming. This implies that evaluation and non-evaluation steps in $\mathcal{T}$ are well-defined on $\alpha_{\mathcal{T}}$-equivalence classes of terms.

To prove that redexes are preserved by $\alpha_{\mathcal{T}}$-renaming, we extend $\mathcal{T}$ with some marked syntax nodes. These markings allow tracing the image of a particular subterm occurrence of a term in an $\alpha_{\mathcal{T}}$-renaming of that term. Since it is sufficient for our purposes to trace only the images of redexes, we limit our extension to mark only those nodes at which a redex may be rooted: applications, operations, and labels (the latter are used in section 2.4 for defining a module substitution redex).

Definition 2.27 (The Marked Calculus $\underline{\mathcal{T}}$). Let $\underline{\mathcal{T}}$ denote an extension to $\mathcal{T}$ that augments the terms, contexts, values, and subterm occurrences of $\mathcal{T}$ with a marked variant $\underline{@}$ of @ and a marked variant $\underline{op}$ of *op* for each *op* in $\mathcal{T}$, as well as a marked label $\underline{l}$ for each $l \in \mathbf{Label}$. Assume that $\widetilde{M}, \widetilde{N}$ range over $\mathbf{Term}_{\underline{\mathcal{T}}}$, $\widetilde{\mathbb{A}}, \widetilde{\mathbb{B}}, \widetilde{\mathbb{C}}$ range over $\mathbf{Context}_{\underline{\mathcal{T}},\underline{\mathcal{T}}}$, and $\widetilde{V}$ ranges over $\mathbf{Value}_{\underline{\mathcal{T}}}$. A $\underline{\mathcal{T}}$-term of the form $\widetilde{M_1} \mathbin{\underline{@}} \widetilde{M_2}$, $\widetilde{M_1} \mathbin{\underline{op}} \widetilde{M_2}$, or $\underline{l}$ is called a *marked* term[5], and a subterm occurrence of the form $(\widetilde{\mathbb{C}}, \widetilde{N})$ is said to be marked if $\widetilde{N}$ is a marked term. For notions of reduction, $\underline{\mathcal{T}}$ treats $\underline{@}$ and $\underline{op}$ as @ and *op*, respectively. Substitution on $\underline{\mathcal{T}}$ terms preserves marked terms, as shown below:

$$\begin{aligned}(\widetilde{N} \mathbin{\underline{@}} \widetilde{N}')[x := \widetilde{M}] &= (\widetilde{N}[x := \widetilde{M}]) \mathbin{\underline{@}} (\widetilde{N}'[x := \widetilde{M}]),\\ (\widetilde{N} \mathbin{\underline{op}} \widetilde{N}')[x := \widetilde{M}] &= (\widetilde{N}[x := \widetilde{M}]) \mathbin{\underline{op}} (\widetilde{N}'[x := \widetilde{M}]),\\ (\underline{l})[x := \widetilde{M}] &= \underline{l}.\end{aligned}$$

□

Definition 2.28 (Mark Erasure). The *mark erasure* of a $\underline{\mathcal{T}}$ term $\widetilde{M}$, written $|\widetilde{M}|$, is the $\mathcal{T}$ term that is identical to $\widetilde{M}$ except that every $\underline{@}$ has been replaced by @, every $\underline{op}$ has been replaced by *op*, and every $\underline{l}$ has been replaced by l. We similarly define mark erasure on $\underline{\mathcal{T}}$ contexts. The mark erasure of a $\underline{\mathcal{T}}$ subterm occurrence $(\widetilde{\mathbb{C}}, \widetilde{M})$ is defined as $|(\widetilde{\mathbb{C}}, \widetilde{M})| = (|\widetilde{\mathbb{C}}|, |\widetilde{M}|)$. □

A one-step $\alpha_{\mathcal{T}}$-renaming is defined on $\mathbf{Term}_{\underline{\mathcal{T}}}$ exactly as it is defined on $\mathbf{Term}_{\mathcal{T}}$ (see definitions 2.20 and 2.23), using the extension of substitution to terms in $\mathbf{Term}_{\underline{\mathcal{T}}}$ introduced in definition 2.27. If $\widetilde{M} \xrightarrow{(\widetilde{\mathbb{A}},x,y)}_{\alpha_{\mathcal{T}}} \widetilde{N}$, then $M \xrightarrow{(\mathbb{A},x,y)}_{\alpha_{\mathcal{T}}} N$, where $|\widetilde{M}| = M$, $|\widetilde{N}| = N$, and $|\widetilde{\mathbb{A}}| = \mathbb{A}$.

Marked terms are used to define notions of $\alpha_{\mathcal{T}}$-renaming on (a subset of) subterm occurrences of both $\underline{\mathcal{T}}$ and $\mathcal{T}$ terms:

Definition 2.29 ($\alpha_{\mathcal{T}}$-Renaming of $\underline{\mathcal{T}}$ Subterm Occurrences). Let $(\widetilde{\mathbb{C}}, \widetilde{N})$ be a marked subterm of $\widetilde{\mathbb{C}}\{\widetilde{N}\} \in \mathbf{Term}_{\underline{\mathcal{T}}}$, where $\widetilde{\mathbb{C}}\{\widetilde{N}\}$ contains no other marked subterm occurrences. We say that $(\widetilde{\mathbb{C}}, \widetilde{N})$ *reduces to* $(\widetilde{\mathbb{C}}', \widetilde{N}')$ *by a one-step $\alpha_{\mathcal{T}}$-renaming* with the $\alpha_{\mathcal{T}}$-redex $(\widetilde{\mathbb{A}}, y, x)$, written $(\widetilde{\mathbb{C}}, \widetilde{N}) \xrightarrow{(\widetilde{\mathbb{A}},y,x)}_{\alpha_{\mathcal{T}}} (\widetilde{\mathbb{C}}', \widetilde{N}')$, iff $\widetilde{\mathbb{C}}\{\widetilde{N}\} \xrightarrow{(\widetilde{\mathbb{A}},y,x)}_{\alpha_{\mathcal{T}}} \widetilde{\mathbb{C}}'\{\widetilde{N}'\}$ and $(\widetilde{\mathbb{C}}', \widetilde{N}')$ is marked. □

Definition 2.30 ($\alpha_{\mathcal{T}}$-Renaming of $\mathcal{T}$ Subterm Occurrences). Let $\mathbb{C}\{N\} \in \mathbf{Term}_{\mathcal{T}}$, where $N = N_1 \mathbin{@} N_2$, $N = N_1 \mathbin{op} N_2$, or $N = l$. Then $(\mathbb{C}, N)$ *reduces to* $(\mathbb{C}', N')$ *by a one-step $\alpha_{\mathcal{T}}$-renaming* with the $\alpha_{\mathcal{T}}$-redex $(\mathbb{A}, y, x)$, written $(\mathbb{C}, N) \xrightarrow{(\mathbb{A},y,x)}_{\alpha_{\mathcal{T}}} (\mathbb{C}', N')$, iff there exist $\underline{\mathcal{T}}$ subterm occurrences $(\widetilde{\mathbb{C}}, \widetilde{N})$ and $(\widetilde{\mathbb{C}}', \widetilde{N}')$ such that $(\widetilde{\mathbb{C}}, \widetilde{N}) \xrightarrow{(\widetilde{\mathbb{A}},y,x)}_{\alpha_{\mathcal{T}}} (\widetilde{\mathbb{C}}', \widetilde{N}')$, $|(\widetilde{\mathbb{C}}, \widetilde{N})| = (\mathbb{C}, N)$, $|(\widetilde{\mathbb{C}}', \widetilde{N}')| = (\mathbb{C}', N')$, and $|\widetilde{\mathbb{A}}| = \mathbb{A}$. □

[5]This marking of redexes is introduced for reasoning about α-renaming, and is different from marking of redexes in appendix B.

Note that definition 2.29 considers the renaming of *marked* subterm occurrences, while definition 2.30 considers the renaming of *unmarked* subterm occurrences. In definition 2.29, a unique mark is used to guarantee that $\alpha_{\mathcal{T}}$-renaming of a marked subterm occurrence preserves the "position" of the renamed subterm relative to the renamed enclosing term. In definition 2.30, two unmarked subterm occurrences are only related by $\alpha_{\mathcal{T}}$-renaming if they are the erasures of two marked subterm occurrences related by $\alpha_{\mathcal{T}}$-renaming.

REMARK 2.31. Alternatively to marking subterm occurrences, we could have defined a notion of "shape" on terms and contexts that ignores all variable name information in equating two syntax trees[6], and defined $\alpha_{\mathcal{T}}$-renaming on subterm occurrences so that it preserves the shapes of the two components of a subterm occurrence. However, the "shape" approach turned out to be non-applicable to the calculus $\mathcal{C}$, where modules are unordered collections of components, and "shape" does not contain enough information to distinguish two such components (see example 2.51 for details). □

Definition 2.30 naturally extends to $(\mathbb{C}, N) \xrightarrow{S}{}^{*}_{\alpha_{\mathcal{T}}} (\mathbb{C}', N')$, where S is a sequence of $\alpha_{\mathcal{T}}$-redexes. The annotation $(\mathbb{A}, y, x)$ for a one-step $\alpha_{\mathcal{T}}$-renaming and S for a sequence of such renamings may be omitted. As with $\longrightarrow^{*}_{\alpha_{\mathcal{T}}}$ on terms, the $\longrightarrow^{*}_{\alpha_{\mathcal{T}}}$ relation on subterm occurrences is symmetric and therefore the same relation as $\longrightarrow^{=}_{\alpha_{\mathcal{T}}}$, and we use $=^{\mathcal{T}}_{\alpha}$ as a synonym for this relation. We also define the $\alpha_{\mathcal{T}}$-equivalence class of $(\mathbb{C}, N)$ as ${}_{\alpha_{\mathcal{T}}}\langle(\mathbb{C}, N)\rangle = \{(\mathbb{C}', N') \mid (\mathbb{C}', N') =^{\mathcal{T}}_{\alpha} (\mathbb{C}, N)\}$. We use $Sub_{\alpha_{\mathcal{T}}}$ as a meta-variable that ranges over $\alpha_{\mathcal{T}}$-equivalence classes of subterm occurrences.

EXAMPLE 2.32 ($\alpha_{\mathcal{T}}$-RENAMING OF SUBTERM OCCURRENCES). Consider the following context and term:

$$\begin{aligned} \mathbb{C} &= \lambda z.\Box \mathbin{@} \lambda x.z, \\ N &= ((\lambda y.y \mathbin{@} z) \mathbin{@} z), \end{aligned}$$

which form a subterm occurrence of the term $\lambda z.((\lambda y.y \mathbin{@} z) \mathbin{@} z) \mathbin{@} \lambda x.z$.

The following are $\alpha_{\mathcal{T}}$-renamings of the subterm occurrence $(\mathbb{C}, N)$:

1. $(\mathbb{C}, N) \xrightarrow{(\lambda z.((\lambda y.y \mathbin{@} z) \mathbin{@} z) \mathbin{@} \Box, x, a)}_{\alpha_{\mathcal{T}}} (\lambda z.\Box \mathbin{@} \lambda a.z, N)$. In this case, the renaming occurs only within the context $\mathbb{C}$.
2. $(\mathbb{C}, N) \xrightarrow{(\lambda z.(\Box \mathbin{@} z) \mathbin{@} \lambda x.z, y, b)}_{\alpha_{\mathcal{T}}} (\mathbb{C}, ((\lambda b.b \mathbin{@} z) \mathbin{@} z))$. In this case, the renaming occurs only within the subterm N.
3. $(\mathbb{C}, N) \xrightarrow{(\Box, z, c)}_{\alpha_{\mathcal{T}}} (\lambda c.\Box \mathbin{@} \lambda x.c, ((\lambda y.y \mathbin{@} c) \mathbin{@} c))$. In this case, the renaming occurs within both the context $\mathbb{C}$ and the subterm N.

The above examples emphasize that the context for the $\alpha_{\mathcal{T}}$-renaming and the context of the subterm occurrence being renamed are independent. □

Lemma 2.33 ($\alpha_{\mathcal{T}}$-Renaming Preserves Distinctness of Subterm Occurrences). *If* $\mathbb{C}_1\{N_1\} = \mathbb{C}_2\{N_2\} \in$ $\mathbf{Term}_{\mathcal{T}}$, $(\mathbb{C}_1, N_1) \xrightarrow{(\mathbb{A}, x, y)}_{\alpha_{\mathcal{T}}} (\mathbb{C}'_1, N'_1)$, $(\mathbb{C}_2, N_2) \xrightarrow{(\mathbb{A}, x, y)}_{\alpha_{\mathcal{T}}} (\mathbb{C}'_2, N'_2)$, *and* $(\mathbb{C}_1, N_1) \neq (\mathbb{C}_2, N_2)$, *then* $(\mathbb{C}'_1, N'_1) \neq (\mathbb{C}'_2, N'_2)$. □

Proof. Let $\widetilde{M} \in \mathbf{Term}_{\underline{\mathcal{T}}}$ be a term such that $|\widetilde{M}| = \widetilde{\mathbb{C}}_1\{\widetilde{N}_1\} = \widetilde{\mathbb{C}}_2\{\widetilde{N}_2\}$, where $|(\widetilde{\mathbb{C}}_1, \widetilde{N}_1)| = (\mathbb{C}_1, N_1)$, $|(\widetilde{\mathbb{C}}_2, \widetilde{N}_2)| = (\mathbb{C}_2, N_2)$, and both $(\widetilde{\mathbb{C}}_1, \widetilde{N}_1)$ and $(\widetilde{\mathbb{C}}_2, \widetilde{N}_2)$ are marked, but no other subterms are marked in $\widetilde{M}$. By definition of a one-step $\alpha_{\mathcal{T}}$-renaming on $\mathbf{Term}_{\underline{\mathcal{T}}}$ $\widetilde{M} \xrightarrow{(\widetilde{\mathbb{A}}, x, y)}_{\alpha_{\mathcal{T}}} \widetilde{M}' = \widetilde{\mathbb{C}}'_1\{\widetilde{N}'_1\} = \widetilde{\mathbb{C}}'_2\{\widetilde{N}'_2\}$, where $|\widetilde{\mathbb{A}}| = \mathbb{A}$. It is clear that the two marks that mark $(\widetilde{\mathbb{C}}_1, \widetilde{N}_1)$ and $(\widetilde{\mathbb{C}}_2, \widetilde{N}_2)$ remain distinct in $\widetilde{M}'$. Hence $(\widetilde{\mathbb{C}}'_1, \widetilde{N}'_1) \neq (\widetilde{\mathbb{C}}'_2, \widetilde{N}'_2)$, and therefore $|(\widetilde{\mathbb{C}}'_1, \widetilde{N}'_1)| = (\mathbb{C}'_1, N'_1) \neq (\mathbb{C}'_2, N'_2) = |(\widetilde{\mathbb{C}}'_2, \widetilde{N}'_2)|$. □

Lemma 2.34 ($\alpha_{\mathcal{T}}$-Renaming Preserves Redexes).

[6] For example, all of the following terms have the same shape: $\lambda x.\lambda y.(x \mathbin{@} y)$, $\lambda x.\lambda y.(y \mathbin{@} x)$, $\lambda x.\lambda y.(x \mathbin{@} x)$, and $\lambda x.\lambda y.(y \mathbin{@} y)$.

1. *If $(\mathbb{C}, R)$ is a redex and $(\mathbb{C}, R) \longrightarrow_{\alpha_\mathcal{T}} (\mathbb{C}', N)$, then $(\mathbb{C}', N)$ is a redex.*

2. *If $(\mathbb{C}, R)$ is an evaluation redex and $(\mathbb{C}, R) \longrightarrow_{\alpha_\mathcal{T}} (\mathbb{C}', N)$, then $(\mathbb{C}, N)$ is an evaluation redex.*

3. *If $(\mathbb{C}, R)$ is a non-evaluation redex and $(\mathbb{C}, R) \longrightarrow_{\alpha_\mathcal{T}} (\mathbb{C}', N)$, then $(\mathbb{C}, N)$ is a non-evaluation redex.*

$\square$

Proof. For statement (1), definition 2.30 implies the existence of $(\widetilde{\mathbb{C}}, \widetilde{R})$ and $(\widetilde{\mathbb{C}}', \widetilde{N})$ such that $(\widetilde{\mathbb{C}}, \widetilde{R}) \longrightarrow_{\alpha_\mathcal{T}} (\widetilde{\mathbb{C}}', \widetilde{N})$ where $|\widetilde{\mathbb{C}}| = \mathbb{C}$, $|\widetilde{R}| = R$, $|\widetilde{\mathbb{C}}'| = \mathbb{C}'$, and $|\widetilde{N}| = N$. Since R must have the form $(\lambda x.M) \mathbin{@} V$ or $c_1 \ op\ c_2$, definition 2.29 and definition 2.28 imply that $\widetilde{R}$ has the form $(\lambda x.\widetilde{M}) \mathbin{\underline{@}} \widetilde{V}$ or $c_1 \ \underline{op}\ c_2$. The result $\widetilde{N}$ of a one-step renaming has the form $(\lambda x.\widetilde{M}') \mathbin{\underline{@}} \widetilde{V}'$ or $c_1 \ op\ c_2$, respectively, which means that N must have the form $(\lambda x.M') \mathbin{@} V'$ or $c_1 \ op\ c_2$ — the left-hand side of a redex/contractum pair. Since $\mathbb{C}'$ is a context, $(\mathbb{C}', N)$ is a redex.

The proof of statement (2) is similar, except that it is necessary to argue that a one-step alpha-renaming of an evaluation context $\mathbb{C}$ yields a context $\mathbb{C}'$ that is also an evaluation redex. Indeed, we can show by induction on the structure of a redex that $\mathbb{C}'$ is an evaluation redex iff $\mathbb{C}'$ is one. So if $\mathbb{C}$ is a non-evaluation redex, then so is $\mathbb{C}'$, from which fact statement (3) follows. $\square$

Together, lemmas 2.33 and 2.34 imply that two distinct redexes in $\mathcal{T}$ remain two distinct redexes after $\alpha_\mathcal{T}$-renaming.

Lemma 2.35 (Properties of Terms Preserved by $\alpha_\mathcal{T}$-Renaming).

1. *If $M' =_\alpha^\mathcal{T} M$ then $FV(M') = FV(M)$ and $FL(M') = FL(M)$.*

2. *If $M \xRightarrow{(\mathbb{E},R)}_\mathcal{T} N$ and $M \xrightarrow{S}{}^*_{\alpha_\mathcal{T}} M'$ for some sequence of $\alpha_\mathcal{T}$-renamings S and $(\mathbb{E}, R) \xrightarrow{S}{}^*_{\alpha_\mathcal{T}} (\mathbb{E}', R')$, then there exists N' such that $M' \xRightarrow{(\mathbb{E}',R')}_\mathcal{T} N'$ and $N' =_\alpha^\mathcal{T} N$.*

3. *If $M \circ\!\!\xrightarrow{(\mathbb{C},R)}_\mathcal{T} N$ and $M \xrightarrow{S}{}^*_{\alpha_\mathcal{T}} M'$ for some sequence of $\alpha_\mathcal{T}$-renamings S and $(\mathbb{C}, R) \xrightarrow{S}{}^*_{\alpha_\mathcal{T}} (\mathbb{C}', R')$, then there exists N' such that $M' \circ\!\!\xrightarrow{(\mathbb{C}',R')}_\mathcal{T} N'$ and $N' =_\alpha^\mathcal{T} N$.*

4. *If $M' =_\alpha^\mathcal{T} M$ then $Cl_\mathcal{T}(M') = Cl_\mathcal{T}(M)$.*

$\square$

Parts 2 and 3 of lemma 2.35 imply that if an evaluation or a non-evaluation step is defined on one representative of an $\alpha_\mathcal{T}$-equivalence class, it is defined on all representatives of this class:

Lemma 2.36 ($\mathcal{T}$ Relations Well-defined w.r.t. $\alpha_\mathcal{T}$-Equivalence).

1. *If $M \xRightarrow{(\mathbb{E},R)}_\mathcal{T} N$ and $M =_\alpha^\mathcal{T} M'$, then there exist $(\mathbb{E}', R')$ and N' such that $M' \xRightarrow{(\mathbb{E}',R')}_\mathcal{T} N'$, $N' =_\alpha^\mathcal{T} N$, and $(\mathbb{E}, R) =_\alpha^\mathcal{T} (\mathbb{E}', R')$.*

2. *If $M \circ\!\!\xrightarrow{(\mathbb{C},R)}_\mathcal{T} N$ and $M =_\alpha^\mathcal{T} M'$, then there exist $(\mathbb{C}', R')$ and N' such that $M' \circ\!\!\xrightarrow{(\mathbb{C}',R')}_\mathcal{T} N'$, $N' =_\alpha^\mathcal{T} N$, and $(\mathbb{C}, R) =_\alpha^\mathcal{T} (\mathbb{C}', R')$.*

$\square$

Proof. Follows from parts 2 and 3 of lemma 2.35 by symmetry of $\longrightarrow_{\alpha_\mathcal{T}}$. $\square$

The above results allow us to lift $\Longrightarrow_\mathcal{T}$, $\circ\!\!\longrightarrow_\mathcal{T}$, and $Cl_\mathcal{T}$ to $\alpha_\mathcal{T}$-equivalence classes:

Definition 2.37 (Evaluation and Non-evaluation Steps in $\mathcal{T}\backslash\alpha$). $M_{\alpha_\mathcal{T}} \xRightarrow{Sub_{\alpha_\mathcal{T}}}_{\mathcal{T}\backslash\alpha} N_{\alpha_\mathcal{T}}$ (respectively $M_{\alpha_\mathcal{T}} \circ\!\!\xrightarrow{Sub_{\alpha_\mathcal{T}}}_{\mathcal{T}\backslash\alpha} N_{\alpha_\mathcal{T}}$) if there exists $M \in M_{\alpha_\mathcal{T}}$, $N \in N_{\alpha_\mathcal{T}}$, and $(\mathbb{E}, R) \in Sub_{\alpha_\mathcal{T}}$ such that $M \xRightarrow{(\mathbb{E},R)}_{\mathcal{T}} N$ (respectively $(\mathbb{C}, R) \in Sub_{\alpha_\mathcal{T}}$ such that $M \circ\!\!\xrightarrow{(\mathbb{C},R)}_{\mathcal{T}} N$). □

Definition 2.38 (Classification in $\mathcal{T}\backslash\alpha$). $Cl_{\mathcal{T}\backslash\alpha}(M_{\alpha_\mathcal{T}}) = Cl_\mathcal{T}(M)$, where $M \in M_{\alpha_\mathcal{T}}$. This is well-defined since, by part 4 of lemma 2.35, such classification does not depend on the choice of a representative of the $\alpha_\mathcal{T}$-equivalence class. Note that if $Cl_\mathcal{T}(M) = \mathbf{evaluatable}_\mathcal{T}$ then $Cl_{\mathcal{T}\backslash\alpha}(\alpha_\mathcal{T}\langle M\rangle) = \mathbf{evaluatable}_\mathcal{T}$. That is, there is no distinct token $\mathbf{evaluatable}_{\mathcal{T}\backslash\alpha}$; $Cl_{\mathcal{T}\backslash\alpha}$is a function from α-equivalence classes of $\mathcal{T}$ terms to $\mathbf{Obs}_\mathcal{T}$ (tokens in $\mathbf{Obs}_\mathcal{T}$ do not expose any information about bound variables of terms, and therefore can be used for classification of $\alpha_\mathcal{T}$-equivalence classes). □

We show that $\Rightarrow_{\mathcal{T}\backslash\alpha}$ is a function:

Lemma 2.39 ($\Rightarrow_{\mathcal{T}\backslash\alpha}$ is a Function). *If $Cl_{\mathcal{T}\backslash\alpha}(M_{\alpha_\mathcal{T}}) = \mathbf{evaluatable}_\mathcal{T}$, then there exists a unique $\alpha_\mathcal{T}$-equivalence class $N_{\alpha_\mathcal{T}}$ such that $M_{\alpha_\mathcal{T}} \Rightarrow_{\mathcal{T}\backslash\alpha} N_{\alpha_\mathcal{T}}$.* □

Since $\Rightarrow_{\mathcal{T}\backslash\alpha}$ is a function, it is trivially confluent, and therefore $Eval_{\mathcal{T}\backslash\alpha}$ and $Outcome_{\mathcal{T}\backslash\alpha}$ are well-defined. It also trivially follows that if an $\alpha_\mathcal{T}$-equivalence class $M_{\alpha_\mathcal{T}}$ has an eval normal form, then it cannot diverge w.r.t. $\Rightarrow_{\mathcal{T}\backslash\alpha}$.

Using classical techniques [Plo75, Bar84], it is straightforward to prove that $\rightarrow_{\mathcal{T}\backslash\alpha}$ is confluent and $\mathcal{T}\backslash\alpha$ has the standardization property. These properties are shown in appendix B.

Convention 2.40. Wherever unambiguous, we will use concrete terms and subterm occurrences to stand for the alpha-equivalence classes that they represent, i.e. we write $M \xrightarrow{(\mathbb{C},R)}_{\mathcal{T}\backslash\alpha} N$ to mean $_{\alpha_\mathcal{T}}\langle M\rangle \xrightarrow{\alpha_\mathcal{T}\langle(\mathbb{C},R)\rangle}_{\mathcal{T}\backslash\alpha}$ $_{\alpha_\mathcal{T}}\langle N\rangle$. The same convention holds for $Cl_{\mathcal{T}\backslash\alpha}$, $Eval_{\mathcal{T}\backslash\alpha}$, and $Outcome_{\mathcal{T}\backslash\alpha}$. □

Note that the subscript $\mathcal{T}\backslash\alpha$ on the reduction arrow indicates that the reduction is performed on $\alpha_\mathcal{T}$-equivalence classes, so there is no confusion with reductions on concrete terms. The following example illustrating standardization is written using the above convention.

Example 2.41. The following reduction sequence is a standard sequence corresponding to the sequence in example 2.15. Recall that a standard sequence is such where all evaluation steps precede all non-evaluation steps.

$$\begin{array}{ll}
\multicolumn{2}{l}{(\lambda x.(\lambda f.f \;@\; (f \;@\; x)) \;@\; (\lambda y.y + (2 * 3))) \;@\; 1} \\
\xRightarrow{\beta}_{\mathcal{T}\backslash\alpha} & (\lambda f.f \;@\; (f \;@\; 1)) \;@\; (\lambda y.y + (2 * 3)) \\
\xRightarrow{\beta}_{\mathcal{T}\backslash\alpha} & (\lambda y.y + (2 * 3)) \;@\; ((\lambda y.y + (2 * 3)) \;@\; 1) \\
\xRightarrow{\beta}_{\mathcal{T}\backslash\alpha} & (\lambda y.y + (2 * 3)) \;@\; (1 + (2 * 3)) \\
\xRightarrow{\delta}_{\mathcal{T}\backslash\alpha} & (\lambda y.y + (2 * 3)) \;@\; (1 + 6) \\
\xRightarrow{\delta}_{\mathcal{T}\backslash\alpha} & (\lambda y.y + (2 * 3)) \;@\; 7 \\
\circ\!\!\xrightarrow{\delta}_{\mathcal{T}\backslash\alpha} & (\lambda y.y + 6) \;@\; 7
\end{array}$$

□

2.4 Core Module Calculus ($\mathcal{C}$)

2.4.1 Syntax of Modules

In our module calculus, modules are unordered collections of labeled terms. There are two disjoint classes of labels: *visible* and *hidden*. Visible labels name components to be exported to other modules, and also name import sites within a component, while hidden labels name components that can only be referenced within the module itself (this distinction is similar to distinction between deferred variables and expression names on one hand and local variables on the other in [AZ99]). Intuitively, a module is a fragment of a recursively scoped record that can be dynamically constructed by linking, where visible labels serve to "wire" the definitions in one module to the uses in another.

A *module binding* is written $l \mapsto M$. We say that M is the *component bound to* l, or that l *binds* M. A *module* is a bracketed set of such bindings. The notation $l_i \overset{n}{\underset{i=1}{\mapsto}} M_i$ stands for the bindings $l_1 \mapsto M_1 \ldots l_n \mapsto M_n$. If $D = [l_i \overset{n}{\underset{i=1}{\mapsto}} M_i]$, then the set of free variables of D is $FV(D) = \cup_{i=1}^n FV(M_i)$. We require that a module does not have any free variables, i.e. $FV(D) = \emptyset$, and that the labels of any two bindings are distinct.

Suppose that $D = [l_i \overset{n}{\underset{i=1}{\mapsto}} M_i]$. We introduce the following notations and definitions. The set of *bound labels* in D is defined as $BL(D) = \cup_{i=1}^n l_i$, while the set of *free labels* is $FL(D) = (\cup_{i=1}^n FL(M_i))/BL(D)$. $Lab(D)$ denotes the set $FL(D) \cup BL(D)$ of all labels in a module. The *exported labels* of D are those that are both bound and visible ($Exports(D) = BL(D) \cap \mathbf{Visible}$), while the *imported labels* are just the free ones ($Imports(D) = FL(D)$). We also define the set of hidden labels of a module as $Hid(D) = Lab(D) \cap \mathbf{Hidden}$. Since all module bindings are distinct, the following notation is well-defined: $D \downarrow l$ extracts the component M bound to l in D if it exists, otherwise $D \downarrow l$ is undefined.

Unlike [MT00], where the condition $Imports(D) \cap \mathbf{Hidden} = \emptyset$ was imposed on all modules, in this presentation we allow modules to import hiddens. Modules that do not import hiddens, i.e. modules D such that $Imports(D) \cap \mathbf{Hidden} = \emptyset$, are called *h-closed* and form a set $\mathbf{HTerm}_{\mathcal{C}}$. We use H to range over the set $\mathbf{HTerm}_{\mathcal{C}}$. Since by definition $Imports(D) = FL(D)$, h-closed modules do not have free hidden labels, and thus are analogous to closed terms, i.e. terms with no free variables. Note that for h-closed modules $Hid(D) = BL(D) \cap \mathbf{Hidden}$. This change from [MT00] was mainly prompted by introduction of contexts $\mathbb{M} \in \mathbf{Context}_{\mathcal{C},\mathcal{C}}$ (see below for details).

Removing the condition $Imports(D) \cap \mathbf{Hidden} = \emptyset$ changes the set $\mathbf{Term}_{\mathcal{C}}$ from that in [MT00], but does not change the results for $\mathcal{C}$ shown in there (see section C). Note that at the linking level the restriction that all modules in a linking expression are h-closed makes the set $\mathbf{Term}_{\mathcal{L}}$ of linking expressions exactly the same as in [MT00] and guarantees that no capture of a hidden label is possible during linking.

We define two sets of one-hole contexts for the core module calculus. The set $\mathbf{Context}_{\mathcal{C},\mathcal{C}}$ contains module contexts obtained from modules by removing all or some of the bindings. As the subscripts suggest, such contexts are filled with other modules. These contexts are used in section 2.4.3 to define $\alpha_{\mathcal{C}}$-renaming and in section 2.6 to define the garbage-collection reduction rule (GC). The rule for filling contexts in $\mathbf{Context}_{\mathcal{C},\mathcal{C}}$ is as follows:

$$[l_i \overset{n}{\underset{i=1}{\mapsto}} M_i, \Box]\{[l_j \overset{m}{\underset{j=1}{\mapsto}} N_j]\} = [l_i \overset{n}{\underset{i=1}{\mapsto}} M_i, l_j \overset{m}{\underset{j=1}{\mapsto}} N_j],$$

provided the result is a module, i.e. it does not have free term variables and all its bindings have distinct labels. For instance, the result of filling a context $[A \mapsto \lambda x.B, h \mapsto 2, \Box]$ with a module $[B \mapsto \lambda y.h]$ is a module $[A \mapsto \lambda x.B, h \mapsto 2, B \mapsto \lambda y.h]$. Note that the module $[B \mapsto \lambda y.h]$, which fills the context, is not h-closed (it exports a hidden label h). This illustrates why we have extended the set of modules (compared to those in [MT00]) to include modules that are not h-closed.

If we fill a context $\mathbb{M}$ with a binding $l \mapsto \mathbb{C}$ which binds a label to a term-level context $\mathbb{C}$, we get a context that can be filled with a term $M \in \mathbf{Term}_{\mathcal{T}}$, and the result of such filling is a module. These contexts form the set $\mathbf{Context}_{\mathcal{T},\mathcal{C}}$. The filling of a context $\mathbb{D} \in \mathbf{Context}_{\mathcal{T},\mathcal{C}}$ with a term $M \in \mathbf{Term}_{\mathcal{T}}$ is defined if the result of this filling is in $\mathbf{Term}_{\mathcal{C}}$, i.e. it satisfies the two conditions in the definition of a module. Unless specified otherwise, the notation $\mathbb{D}\{M\}$ implies that the result of the filling is in $\mathbf{Term}_{\mathcal{C}}$. The context set $\mathbf{Context}_{\mathcal{T},\mathcal{C}}$ is used to define core module reductions (comp) and (subst) (see below for more details). A subterm occurrence $(\mathbb{D}, M)$ such that $\mathbb{D}\{M\} \in \mathbf{Term}_{\mathcal{C}}$ is called a *$\mathcal{T}$-term occurrence*.

The label renaming substitution $D[l := k]$, where $D = [l_i \overset{n}{\underset{i=1}{\mapsto}} M_i]$, yields $[l'_i \overset{n}{\underset{i=1}{\mapsto}} M_i[l := k]]$, where $l'_i = k$ if $l_i = l$ and $l'_i = l_i$ otherwise. See section 2.3.1 for definition of the substitution $M[l := k]$.

2.4.2 Evaluation and Reduction of Modules

The evaluation relation $\Longrightarrow_{\mathcal{C}}$ is defined in figure 1 via a *module evaluation context* $\mathbb{G} \in \mathbf{Context}_{\mathcal{T},\mathcal{C}}$ which lifts term-level evaluation context $\mathbb{E}$ to the module level. The rules of $\Longrightarrow_{\mathcal{C}}$ allow the following reductions:

- (comp-ev) lifts $\Longrightarrow_{\mathcal{T}}$ to the module level;
- (subst-ev) substitutes a labeled value for a label occurrence in the module in an evaluation context. The requirement that $FV(D) = \emptyset$ in definition of a module implies that for every module component $l \mapsto M$ $FV(M) = \emptyset$. Therefore the value that gets substituted for the label in (subs-ev) rule is a closed term, and no variable capture occurs.

The complementary relation $\circ\!\!\longrightarrow_{\mathcal{C}}$ has two rules (comp-nev) and (subst-nev) which differ from their evaluation analogs by using a non-evaluation context in place of an evaluation context. Note that in (subst-nev) rule no variable capture is possible for the same reason as for (subst-ev). The corresponding $\longrightarrow_{\mathcal{C}}$ rules (comp) and (subst) are the unions of the respective $\Longrightarrow_{\mathcal{C}}$ and $\circ\!\!\longrightarrow_{\mathcal{C}}$ rules. They can be formulated by replacing an evaluation context in the $\Longrightarrow_{\mathcal{C}}$-rules by a general context $\mathbb{D} \in \mathbf{Context}_{\mathcal{T},\mathcal{C}}$.

Example 2.42. The following reduction sequence illustrates the different kinds of module-level reductions:[7]

$$
\begin{array}{ll}
\multicolumn{2}{l}{[A \mapsto 2+3, B \mapsto (\lambda x.x * (4+A)) \mathbin{@} A]} \\
\xRightarrow{\text{comp-ev}}_{\mathcal{C}} & [A \mapsto 5, B \mapsto (\lambda x.x * (4+A)) \mathbin{@} A] \\
\circ\!\!\xrightarrow{\text{subst-nev}}_{\mathcal{C}} & [A \mapsto 5, B \mapsto (\lambda x.x * (4+5)) \mathbin{@} A] \\
\circ\!\!\xrightarrow{\text{comp-nev}}_{\mathcal{C}} & [A \mapsto 5, B \mapsto (\lambda x.x * 9) \mathbin{@} A] \\
\xRightarrow{\text{subst-ev}}_{\mathcal{C}} & [A \mapsto 5, B \mapsto (\lambda x.x * 9) \mathbin{@} 5] \\
\xRightarrow{\text{comp-ev}}_{\mathcal{C}} & [A \mapsto 5, B \mapsto 5 * 9] \\
\xRightarrow{\text{comp-ev}}_{\mathcal{C}} & [A \mapsto 5, B \mapsto 45]
\end{array}
$$

□

Note that (subst-nev) allows some abstractions to be substituted into themselves:

$$[F \mapsto \lambda x.F] \circ\!\!\longrightarrow_{\mathcal{C}} [F \mapsto \lambda x.(\lambda x.F)] \circ\!\!\longrightarrow_{\mathcal{C}} [F \mapsto \lambda x.(\lambda x.(\lambda x.(\lambda x.F)))]$$

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

Classification of modules is defined as follows:

$$
Cl_{\mathcal{C}}(D) = \begin{cases}
\mathbf{evaluatable}_{\mathcal{C}} & \text{if there exists } D' \text{ s.t. } D \Longrightarrow_{\mathcal{C}} D' \\
[v_i \overset{n}{\underset{i=1}{\mapsto}} Cl_{\mathcal{T}}(V_i)] & \text{if } D = [v_i \overset{n}{\underset{i=1}{\mapsto}} V_i, h_j \overset{m}{\underset{j=1}{\mapsto}} V_j'], \\
\mathbf{error}_{\mathcal{C}} & \text{otherwise}
\end{cases}
$$

In the second case (i.e. when $D = [v_i \overset{n}{\underset{i=1}{\mapsto}} V_i, h_j \overset{m}{\underset{j=1}{\mapsto}} V_j']$) we also say that D is a module value ($D \in \mathbf{Value}_{\mathcal{C}}$). Note that the classification of a module value does not expose information about hidden components, but requires that all such components are values. This restriction avoids a clash between evaluatables and values: without such a restriction a module $[A \mapsto 2, a \mapsto 3+4]$ could be both an evaluatable and a value, which would contradict the requirement that classifications are disjoint. The restriction also makes it possible to lift classification directly to $\alpha_{\mathcal{C}}$-equivalence classes, without changing the set $\mathbf{Obs}_{\mathcal{C}}$ of observables, since observables do not expose the names of bound variables or bound hidden labels in a module, so they are preserved by $\alpha_{\mathcal{C}}$-renaming (see lemma 2.55).

[7] In examples, we adopt the convention that visible labels have uppercase names while hidden labels have lowercase names.

2.4.3 $\alpha_{\mathcal{C}}$-Renaming of Modules

Simplifying the presentation in [MT00], modules are identified up to renaming of λ-bound variables in labeled terms and a consistent renaming of hidden components throughout the entire module, similar to $\alpha_{\mathcal{T}}$-renaming of λ-bound variables in terms. With the latter identification, the explicit renaming of hidden variables required in [MT00] is unnecessary. For instance, consider the following two modules:

$$[A \mapsto h_1, h_1 \mapsto 2, h_2 \mapsto h_1] \quad \text{and} \quad [A \mapsto h_3, h_3 \mapsto 2, h_4 \mapsto h_3].$$

The hidden labels of the first module can be renamed to those of the second: h_1 to h_3 and h_2 to h_4. We say that these two modules are two *representatives* of the same *$\alpha_{\mathcal{C}}$-equivalence class*. The notion of $\alpha_{\mathcal{C}}$-equivalence is formalized below.

Definition 2.43 ($\alpha_{\mathcal{C}}$-Renaming). D reduces to D' by a *one-step $\alpha_{\mathcal{C}}$-renaming* iff one of the following is true:

- $D = \mathbb{M}\{l \mapsto M\}$, $D' = \mathbb{M}\{l \mapsto M'\}$, and $M \xrightarrow{(\mathbb{C},x,y)}_{\alpha_{\mathcal{T}}} M'$, in which case we write $D \xrightarrow{(\mathbb{D},x,y)}_{\alpha_{\mathcal{C}}} D'$, where $\mathbb{D} = \mathbb{M}\{l \mapsto \mathbb{C}\}$, or
- there exists $h \in BL(D) \cap \mathbf{Hidden}$ and $h' \notin \mathit{Hid}(D)$ such that $D' = D[h := h']$. In this case we write $D \xrightarrow{(h,h')}_{\alpha_{\mathcal{C}}} D'$.

The triple $(\mathbb{D}, x, y)$ in the first case and the pair (h, h') in the second case is called the $\alpha_{\mathcal{C}}$-redex of the reduction. As usual, the redex annotation may be omitted.

As for the term calculus, we write $D \xrightarrow{S}{}^{*}_{\alpha_{\mathcal{C}}} D'$, where S is a sequence of $\alpha_{\mathcal{C}}$-redexes, if D is reduced to D' by a sequence of one-step $\alpha_{\mathcal{C}}$-renamings reducing the $\alpha_{\mathcal{C}}$-redexes in S left-to-right. ☐

The condition $h \in BL(D) \cap \mathbf{Hidden}$ guarantees that the label being renamed is bound in D. Since the new name h' does not appear in D at all by the condition $h' \notin \mathit{Hid}(D)$, a free hidden label cannot be captured by the $\alpha_{\mathcal{C}}$-renaming, and also all the bound hidden labels of the module remain distinct after $\alpha_{\mathcal{C}}$-renaming.

Example 2.44 ($\alpha_{\mathcal{C}}$-renaming of modules).

$$\begin{array}{rl} & [A \mapsto \lambda x.h_1, h_1 \mapsto \lambda x.h_2] \\ \xrightarrow{(h_1,h_3)}_{\alpha_{\mathcal{C}}} & [A \mapsto \lambda x.h_3, h_3 \mapsto \lambda x.h_2] \\ \xrightarrow{([A\mapsto\square, h_3\mapsto\lambda x.h_2],x,y)}_{\alpha_{\mathcal{C}}} & [A \mapsto \lambda y.h_3, h_3 \mapsto \lambda x.h_2]. \end{array}$$

The first step renames a hidden label h to h'. The condition that the new hidden name does not appear in the module prevents us from choosing h_2 as the new name for h_1. Any name except for h_2 is a legal choice. The second step renames a bound variable in the first module component. ☐

Lemma 2.45 ($\to_{\alpha_{\mathcal{C}}}$ is Symmetric). *$\to_{\alpha_{\mathcal{C}}}$ is symmetric, and $D \xrightarrow{(h,h')}_{\alpha_{\mathcal{C}}} D'$ iff $D' \xrightarrow{(h',h)}_{\alpha_{\mathcal{C}}} D$.* ☐

Proof. Since $\to_{\alpha_{\mathcal{T}}}$ is symmetric on $\mathbf{Term}_{\mathcal{T}}$, its lifting is symmetric on $\mathbf{Term}_{\mathcal{C}}$. Renaming of hidden labels of a module $D \xrightarrow{(h,h')}_{\alpha_{\mathcal{C}}} D'$ is symmetric, with the reverse step $D' \xrightarrow{(h',h)}_{\alpha_{\mathcal{C}}} D$: by the condition $h \in BL(D)$, $h' \notin \mathit{Hid}(D)$, and $D' = D[h := h']$, therefore in $h' \in BL(D')$, and $h \notin \mathit{Hid}(D')$ (all occurrences of h has been renamed to h'). ☐

As for the term calculus $\mathcal{T}$, $\to^*_{\alpha_{\mathcal{C}}}$ is symmetric by lemma 2.1, so $\to^*_{\alpha_{\mathcal{C}}}$ is the same as $\to^{=}_{\alpha_{\mathcal{C}}}$, also abbreviated as $=^{\mathcal{C}}_{\alpha}$. We define $\alpha_{\mathcal{C}}$-equivalence classes of modules as follows:

Definition 2.46 ($\alpha_{\mathcal{C}}$-Equivalence Class). If $D \in \mathbf{Term}_{\mathcal{C}}$, then its $\alpha_{\mathcal{C}}$-equivalence class, written ${}_{\alpha_{\mathcal{C}}}\langle D \rangle$, is defined as ${}_{\alpha_{\mathcal{C}}}\langle D \rangle = \{D' \in \mathbf{Term}_{\mathcal{C}} \mid D' =^{\mathcal{C}}_{\alpha} D\}$. We say that any $D' \in {}_{\alpha_{\mathcal{C}}}\langle D \rangle$ is a *representative* of ${}_{\alpha_{\mathcal{C}}}\langle D \rangle$. We use $D_{\alpha_{\mathcal{C}}}$ as a meta-variable that ranges over the set of $\alpha_{\mathcal{C}}$-equivalence classes of modules. ☐

We have mentioned in section 2.3.3 that, alternatively to one-step α-renaming, we could have defined α-equivalence of two terms by induction on the structure of the terms, as in [Plo75]. While this approach works for the term calculus, it causes problems at the core module level. The example below illustrates why we have chosen to use one-step α-renaming instead of just defining $=_\alpha^{\mathcal{C}}$.

Example 2.47 (Hidden components indistinguishable w.r.t. $\alpha_{\mathcal{C}}$-renaming). Consider the following $\alpha_{\mathcal{C}}$-renaming sequence:

$$
\begin{array}{ll}
\multicolumn{2}{l}{[A \mapsto 7, h_1 \mapsto 2 + h_2, h_2 \mapsto 2 + h_1]} \\
\xrightarrow{(h_1,h_3)}_{\alpha_{\mathcal{C}}} & [A \mapsto 7, h_3 \mapsto 2 + h_2, h_2 \mapsto 2 + h_3] \\
\xrightarrow{(h_2,h_1)}_{\alpha_{\mathcal{C}}} & [A \mapsto 7, h_3 \mapsto 2 + h_1, h_1 \mapsto 2 + h_3] \\
\xrightarrow{(h_3,h_2)}_{\alpha_{\mathcal{C}}} & [A \mapsto 7, h_2 \mapsto 2 + h_1, h_1 \mapsto 2 + h_2].
\end{array}
$$

In this example we cannot distinguish the two module components at the level of the $\alpha_{\mathcal{C}}$-equivalence class of the module, because they can be $\alpha_{\mathcal{C}}$-renamed to one another. However, the sequence of one-step $\alpha_{\mathcal{C}}$-renamings $(h_1, h_3), (h_2, h_1), (h_3, h_2)$ shows that the components have been switched, so that the image of the binding $h_1 \mapsto 2 + h_2$ is $h_2 \mapsto 2 + h_1$. Note that the hidden components are not referenced in the visible part of the module. If they were, we could have been able to distinguish them by the visible component(s) in which they are referenced. For instance, if we rename the module $[A \mapsto h_1, B \mapsto h_2, h_1 \mapsto 2 + h_2, h_2 \mapsto 2 + h_1]$ to $[A \mapsto h_2, B \mapsto h_1, h_2 \mapsto 2 + h_1, h_1 \mapsto 2 + h_2]$, we know just by examining the original and the renamed module, i.e. without specifying the sequence of $\alpha_{\mathcal{C}}$-renamings, that the component bound to h_1 is the one that used to be bound to h_2, because it is referenced in the binding $B \mapsto h_2$ (recall that visible labels cannot be $\alpha_{\mathcal{C}}$-renamed). □

We extend the calculus $\underline{\mathcal{T}}$ of marked terms to the calculus $\underline{\mathcal{C}}$ of modules over marked terms as follows:

Definition 2.48 (The Marked Calculus $\underline{\mathcal{C}}$). Let $\underline{\mathcal{C}}$ denote an extension to $\mathcal{C}$, where modules are defined the same way as in $\mathcal{C}$, with the difference that module bindings are of the form $l \mapsto \widetilde{M}$, where $\widetilde{M} \in \mathbf{Term}_{\underline{\mathcal{T}}}$, and, similarly, context bindings are of the form $l \mapsto \widetilde{\mathbb{C}}$, $\widetilde{\mathbb{C}} \in \mathbf{Context}_{\underline{\mathcal{T}},\underline{\mathcal{T}}}$. Let $\widetilde{D}, \widetilde{\mathbb{M}}$, and $\widetilde{\mathbb{D}}$ range over $\mathbf{Term}_{\underline{\mathcal{C}}}, \mathbf{Context}_{\underline{\mathcal{C}},\underline{\mathcal{C}}}$, and $\mathbf{Context}_{\underline{\mathcal{T}},\underline{\mathcal{C}}}$, respectively. Similarly to $\underline{\mathcal{T}}$, reduction in $\underline{\mathcal{C}}$ reduces marked redexes as if they were unmarked.

The appropriate extension of label substitution to marked modules is defined below. In $\underline{\mathcal{C}}$ a marked term occurs only on the right-hand side of a binding, and therefore l and k are unmarked in the substitution $\widetilde{M}[l := k]$. The result of the substitution $\widetilde{D}[l := k]$, where $\widetilde{D} = [l_i \overset{n}{\underset{i=1}{\mapsto}} \widetilde{M_i}]$, is $[l'_i \overset{n}{\underset{i=1}{\mapsto}} \widetilde{M'_i}]$, where $l'_i = k$ if $l_i = l$, and $l'_i = l_i$ otherwise, and $\widetilde{M'_i} = \widetilde{M_i}[l := k]$, where the substitution is extended to marked labels as follows:

$$
\begin{array}{rcll}
\underline{l'}[l := k] & = & \underline{l'} & \text{if } l' \neq l, \\
\underline{l}[l := k] & = & \underline{k} & \text{otherwise.}
\end{array}
$$

□

Definition 2.49 (Mark Erasure in $\underline{\mathcal{C}}$). The *mark erasure* of a module $\widetilde{D} = [l_i \overset{n}{\underset{i=1}{\mapsto}} \widetilde{M_i}] \in \mathbf{Term}_{\underline{\mathcal{C}}}$, written $|\widetilde{D}|$, is a module $[l_i \overset{n}{\underset{i=1}{\mapsto}} |\widetilde{M_i}|] \in \mathbf{Term}_{\mathcal{C}}$. The *mark erasure* on contexts in $\mathbf{Context}_{\underline{\mathcal{T}},\underline{\mathcal{C}}}$ and $\mathbf{Context}_{\underline{\mathcal{C}},\underline{\mathcal{C}}}$ is defined analogously. As for $\underline{\mathcal{T}}$, the mark erasure of a $\mathcal{T}$-term occurrence $(\widetilde{\mathbb{D}}, \widetilde{M})$ is defined as $|(\widetilde{\mathbb{D}}, \widetilde{M})| = (|\widetilde{\mathbb{D}}|, |\widetilde{M}|)$. □

Definition 2.50 ($\alpha_{\mathcal{C}}$-Renaming of $\mathcal{T}$-Term Occurrences). Let $\mathbb{D}\{M\} \in \mathbf{Term}_{\mathcal{C}}$, where $M = M_1 \ @ \ M_2$, $M = M_1 \ op \ M_2$, or $M = l$. Then $(\mathbb{D}, M)$ *reduces to* $(\mathbb{D}', M')$ *by a one-step $\alpha_{\mathcal{C}}$-renaming* with the $\alpha_{\mathcal{C}}$-redex $(\mathbb{D}_1, x, y)$ (respectively with the $\alpha_{\mathcal{C}}$-redex (h, h')), written $(\mathbb{D}, M) \xrightarrow{(\mathbb{D}_1,x,y)}_{\alpha_{\mathcal{C}}} (\mathbb{D}', M')$ (respectively $(\mathbb{D}, M) \xrightarrow{(h,h')}_{\alpha_{\mathcal{C}}} (\mathbb{D}', M')$) iff there exist $(\widetilde{\mathbb{D}}, \widetilde{M})$ and $(\widetilde{\mathbb{D}}', \widetilde{M}')$, where $(\widetilde{\mathbb{D}}, \widetilde{M})$ is the only marked $\mathcal{T}$-term occurrence in $\widetilde{\mathbb{D}}\{\widetilde{M}\}$, such that $\widetilde{\mathbb{D}}\{\widetilde{M}\} \xrightarrow{(\widetilde{\mathbb{D}}_1,x,y)}_{\alpha_{\mathcal{C}}} \widetilde{\mathbb{D}}'\{\widetilde{M}'\}$ (respectively $\widetilde{\mathbb{D}}\{\widetilde{M}\} \xrightarrow{(h,h')}_{\alpha_{\mathcal{C}}} \widetilde{\mathbb{D}}'\{\widetilde{M}'\}$) and $|(\widetilde{\mathbb{D}}, \widetilde{M})| = (\mathbb{D}, M)$, $|(\widetilde{\mathbb{D}}', \widetilde{M}')| = (\mathbb{D}', M')$, and $|\widetilde{\mathbb{D}}_1| = \mathbb{D}_1$ (the latter in the case of the $\alpha_{\mathcal{C}}$-redex $(\mathbb{D}_1, x, y)$). □

We mostly use the above definition for cases when $(\mathbb{D}, M)$ is a redex. Using one-step $\alpha_\mathcal{C}$-renaming and specifying a particular $\alpha_\mathcal{C}$-redex allows us to trace calculus redexes in the hidden part of the module along sequences of $\alpha_\mathcal{C}$-renamings. The following example illustrates how we can distinguish two redexes by giving the sequence of $\alpha_\mathcal{C}$-renamings.

Example 2.51 (Tracing Redexes by a Sequence of $\alpha_\mathcal{C}$-Renamings). Given two modules $D_1 = [A \mapsto 2, h_1 \mapsto 3+4, h_2 \mapsto 3+4]$ and $D_2 = [A \mapsto 2, h'_1 \mapsto 3+4, h'_2 \mapsto 3+4]$, we see that $D_1 =^\mathcal{C}_\alpha D_2$, but, without specifying the sequence of renamings from D_1 to D_2, we cannot uniquely match the redexes of D_1 to those of D_2. However, given a particular sequence S such that $D_1 \xrightarrow{S}{}^*_{\alpha_\mathcal{C}} D_2$, we know which redex in D_1 corresponds to which redex in D_2. For instance, if $S = (h_1, h'_1), (h_2, h'_2)$, then the redex $3+4$ bound to h_1 in D_1 is bound to h'_1 in D_2. Note that a sequence $S' = (h_1, h'_2), (h_2, h'_1)$ gives a different correspondence between the redexes.

We observe that the notion of a "shape" of a context introduced in remark 2.31 for the term calculus does not help us to distinguish the two redexes in the module D_1, since the contexts $[A \mapsto 2, h_1 \mapsto 3+4, h_2 \mapsto \Box]$ and $[A \mapsto 2, h_2 \mapsto 3+4, h_1 \mapsto \Box]$ have the same "shape". Thus we use one-step $\alpha_\mathcal{C}$-renaming to identify calculus redexes before and after a renaming. □

The example above shows that to be able to say that $\alpha_\mathcal{C}$-renaming of a module preserves distinctness of its calculus redexes, we need to specify particular $\alpha_\mathcal{C}$-renaming sequences between modules. Otherwise we cannot prove that two distinct calculus redexes in one module do not map to the same redex in the other module. The two lemmas below formalize this intuition. Together they imply that any two distinct redexes remain two distinct redexes after $\alpha_\mathcal{C}$-renaming. Note that in lemma 2.53 it is essential that the step $\longrightarrow_{\alpha_\mathcal{C}}$ reduces the same $\alpha_\mathcal{C}$-redex for both $\alpha_\mathcal{C}$-renamings. If we replace this step by $=^\mathcal{C}_\alpha$, the lemma no longer holds, because, as we have seen in the example above, two different sequences may reduce different $\mathcal{T}$-terms in one module to the same $\mathcal{T}$-term in another.

We extend the definition of $\alpha_\mathcal{C}$-equivalence classes to $\mathcal{T}$-terms, since calculus redexes in the core module calculus are $\mathcal{T}$-terms.

Definition 2.52 ($\alpha_\mathcal{C}$-Equivalence of $\mathcal{T}$-terms). By definition ${}_{\alpha_\mathcal{C}}\langle(\mathbb{D}_1, M_1)\rangle = \{(\mathbb{D}, M) \mid (\mathbb{D}, M) =^\mathcal{C}_\alpha (\mathbb{D}_1, M_1)\}$. We use a meta-variable $Sub_{\alpha_\mathcal{C}}$ to range over the set of $\alpha_\mathcal{C}$-equivalence classes of $\mathcal{T}$-terms. □

Lemma 2.53 ($\alpha_\mathcal{C}$-Renaming Preserves Distinctness of $\mathcal{T}$-subterms). *If* $\mathbb{D}_1\{M_1\} = \mathbb{D}_2\{M_2\} \in \mathbf{Term}_\mathcal{C}$, $(\mathbb{D}_1, M_1) \neq (\mathbb{D}_2, M_2)$, *and*

- *either* $(\mathbb{D}_1, M_1) \xrightarrow{(\mathbb{D},x,y)}_{\alpha_\mathcal{C}} (\mathbb{D}'_1, M'_1)$ *and* $(\mathbb{D}_2, M_2) \xrightarrow{(\mathbb{D},x,y)}_{\alpha_\mathcal{C}} (\mathbb{D}'_2, M'_2)$,
- *or* $(\mathbb{D}_1, M_1) \xrightarrow{(h,h')}_{\alpha_\mathcal{C}} (\mathbb{D}'_1, M'_1)$ *and* $(\mathbb{D}_2, M_2) \xrightarrow{(h,h')}_{\alpha_\mathcal{C}} (\mathbb{D}'_2, M'_2)$,

then $(\mathbb{D}'_1, M'_1) \neq (\mathbb{D}'_2, M'_2)$. □

Lemma 2.54 ($\alpha_\mathcal{C}$-Renaming Preserves Calculus Redexes).

- *If* $(\mathbb{D}, M)$ *is a redex and* $(\mathbb{D}, M) \longrightarrow_{\alpha_\mathcal{C}} (\mathbb{D}', M')$, *then* $(\mathbb{D}', M')$ *is a redex.*
- *If* $(\mathbb{G}, M)$ *is an evaluation redex and* $(\mathbb{G}, M) \longrightarrow_{\alpha_\mathcal{C}} (\mathbb{D}, M')$, *then* $(\mathbb{D}, M')$ *is an evaluation redex.*
- *If* $(\mathbb{D}, M)$ *is a non-evaluation redex and* $(\mathbb{D}, M) \longrightarrow_{\alpha_\mathcal{C}} (\mathbb{D}', M')$, *then* $(\mathbb{D}', M')$ *is a non-evaluation redex.*

□

The proofs of lemmas 2.53 and 2.54 are analogous to those of lemmas 2.33 and 2.34.

The following lemma gives the properties of modules preserved by $\alpha_\mathcal{C}$-renaming.

Lemma 2.55 (Properties Preserved by $\alpha_\mathcal{C}$-Renaming). *If* $D \xrightarrow{S}{}^*_{\alpha_\mathcal{C}} D'$, *then:*

1. $FL(D) = FL(D')$ *(and hence* $Imports(D) = Imports(D')$, *since* $FL(D) = Imports(D)$*),*

2. *Exports*$(D) =$ *Exports*(D'),

3. $D \in \mathbf{HTerm}_{\mathcal{C}}$ *iff* $D' \in \mathbf{HTerm}_{\mathcal{C}}$.

4. *If* $D \xRightarrow{(\mathbb{G},R)}_{\mathcal{C}} D_1$, *then there exists* $D'_1 \in {}_{\alpha_{\mathcal{C}}}\langle D_1 \rangle$ *such that* $D' \xRightarrow{(\mathbb{G}',R')}_{\mathcal{C}} D'_1$, *where* $(\mathbb{G}, R) \xrightarrow{S}{}^{*}_{\alpha_{\mathcal{C}}} (\mathbb{G}', R')$.

5. *If* $D \circ\!\!\xrightarrow{(\overline{\mathbb{G}},R)}_{\mathcal{C}} D_1$, *then there exists* $D'_1 \in {}_{\alpha_{\mathcal{C}}}\langle D_1 \rangle$ *such that* $D' \circ\!\!\xrightarrow{(\overline{\mathbb{G}}',R')}_{\mathcal{C}} D'_1$, *where* $(\overline{\mathbb{G}}, R) \xrightarrow{S}{}^{*}_{\alpha_{\mathcal{C}}} (\overline{\mathbb{G}}', R')$.

6. $Cl_{\mathcal{C}}(D) = Cl_{\mathcal{C}}(D')$.

□

Proof. We show that the properties are preserved by a one-step $\alpha_{\mathcal{C}}$-renaming. Then they are preserved by $\xrightarrow{S}{}^{*}_{\alpha_{\mathcal{C}}}$. Suppose $D \longrightarrow_{\alpha_{\mathcal{C}}} D'$, then:

1. The lifting of $\longrightarrow_{\alpha_{\mathcal{T}}}$ to the module level preserves all labels. Renaming of hiddens renames only bound hidden labels. The set $FL(D)$ consists of free visible and free hidden labels, which are not renamed. The condition $h' \notin Hid(D)$ in the definition of $\alpha_{\mathcal{C}}$-renaming guarantees that a free hidden label is not captured.

2. By definition *Exports*$(D) = BL(D) \cap \mathbf{Visible}$. Since $\alpha_{\mathcal{C}}$-renaming renames only bound hiddens, *Exports*$(D) =$ *Exports*(D').

3. By part 1 $FL(D) = FL(D')$. Therefore $FL(D) \cap \mathbf{Hidden} = \emptyset$ if and only if $FL(D') \cap \mathbf{Hidden} = \emptyset$.

4. Straightforward proof by cases of the calculus evaluation redex $(\mathbb{G}, R)$ and the $\alpha_{\mathcal{C}}$-redex of the $\alpha_{\mathcal{C}}$-renaming $D \longrightarrow_{\alpha_{\mathcal{C}}} D'$. We show that if $D \xrightarrow{(\mathbb{D}_1,x,y)}_{\alpha_{\mathcal{C}}} D'$ or $D \xrightarrow{(h,h')}_{\alpha_{\mathcal{C}}} D'$, and $D \xRightarrow{(\mathbb{G},R)}_{\mathcal{C}} D_1$, then there exists D'_1 such that $D' \xRightarrow{(\mathbb{G}',R')}_{\mathcal{C}} D'_1$, where $(\mathbb{G}, R) \xrightarrow{(\mathbb{D}_1,x,y)}_{\alpha_{\mathcal{C}}} (\mathbb{G}', R')$ in the first case and $(\mathbb{G}, R) \xrightarrow{(h,h')}_{\alpha_{\mathcal{C}}} (\mathbb{G}', R')$ in the second, and $D_1 =^{\mathcal{C}}_{\alpha} D'_1$.

5. Analogous to part 4.

6. $Cl_{\mathcal{C}}(D) = \mathbf{evaluatable}_{\mathcal{C}}$ iff $Cl_{\mathcal{C}}(D') = \mathbf{evaluatable}_{\mathcal{C}}$ by part 4. Suppose $Cl_{\mathcal{C}}(D) = \mathbf{Value}_{\mathcal{C}}$, i.e. $D = [l_i \overset{n}{\underset{i=1}{\mapsto}} V_i]$. If $D \xrightarrow{(\mathbb{D}_1,x,y)}_{\alpha_{\mathcal{C}}} D'$, then by part 4 of lemma 2.35 all $D' = [l_i \overset{n}{\underset{i=1}{\mapsto}} V'_i]$. If $D \xrightarrow{(h,h')}_{\alpha_{\mathcal{C}}} D'$, then $D' = [l'_i \overset{n}{\underset{i=1}{\mapsto}} V'_i]$, where $l'_i = l_i$ if $l \neq h$, and $l'_i = h'$ otherwise, and $V'_i = V_i[h := h'] \in \mathbf{Value}_{\mathcal{T}}$. Therefore $Cl_{\mathcal{C}}(D') = \mathbf{Value}_{\mathcal{C}}$. The claim follows by symmetry of $\longrightarrow_{\alpha_{\mathcal{C}}}$.

□

Parts 4 and 5 of lemma 2.55 and the symmetry of $\longrightarrow_{\alpha_{\mathcal{C}}}$ imply the following:

Lemma 2.56 ($\mathcal{C}$ Relations Well-defined w.r.t. $\alpha_{\mathcal{C}}$-Equivalence).

1. *If* $D_1 \xRightarrow{(\mathbb{G},R)}_{\mathcal{C}} D_2$ *and* $D_1 \longrightarrow_{\alpha_{\mathcal{C}}} D'_1$, *then there exist* $(\mathbb{G}', R')$ *and* D'_2 *such that* $D'_1 \xRightarrow{(\mathbb{G}',R')}_{\mathcal{C}} D'_2$, $D_2 =^{\mathcal{C}}_{\alpha} D'_2$, *and* $(\mathbb{G}, R) =^{\mathcal{C}}_{\alpha} (\mathbb{G}', R')$.

2. *If* $D_1 \circ\!\!\xrightarrow{(\overline{\mathbb{G}},R)}_{\mathcal{C}} D_2$ *and* $D_1 \longrightarrow_{\alpha_{\mathcal{C}}} D'_1$, *then there exist* $(\overline{\mathbb{G}}', R')$ *and* D'_2 *such that* $D'_1 \circ\!\!\xrightarrow{(\overline{\mathbb{G}}',R')}_{\mathcal{C}} D'_2$, $D_2 =^{\mathcal{C}}_{\alpha} D'_2$, *and* $(\overline{\mathbb{G}}, R) =^{\mathcal{C}}_{\alpha} (\overline{\mathbb{G}}', R')$.

□

The above results allow us to define the extensions $\Longrightarrow_{\mathcal{C}\backslash\alpha}$ and $\circ\!\!\longrightarrow_{\mathcal{C}\backslash\alpha}$ of $\Longrightarrow_{\mathcal{C}}$ and $\circ\!\!\longrightarrow_{\mathcal{C}}$, respectively, to $\alpha_{\mathcal{C}}$-equivalence classes.

Definition 2.57 (Evaluation and Non-evaluation on $\alpha_\mathcal{C}$-Equivalence Classes). $D_{\alpha_\mathcal{C}} \xrightarrow{Sub_{\alpha_\mathcal{C}}}\!\!\!\!\Rightarrow_{\mathcal{C}\backslash\alpha} D'_{\alpha_\mathcal{C}}$ (respectively $D_{\alpha_\mathcal{C}} \circ\!\xrightarrow{Sub_{\alpha_\mathcal{C}}}_{\mathcal{C}\backslash\alpha} D'_{\alpha_\mathcal{C}}$) if there exist $D \in D_{\alpha_\mathcal{C}}$, $D' \in D'_{\alpha_\mathcal{C}}$, and $(\mathbb{G}, R) \in Sub_{\alpha_\mathcal{C}}$ such that $D \overset{(\mathbb{G},R)}{\Longrightarrow}_\mathcal{C} D'$ (respectively $(\overline{\mathbb{G}}, R) \in Sub_{\alpha_\mathcal{C}}$ s.t. $D \circ\!\xrightarrow{(\overline{\mathbb{G}},R)}_\mathcal{C} D'$). $\square$

By part 6 of lemma 2.55 classification of modules in the same $\alpha_\mathcal{C}$-equivalence class is the same. Therefore we can define:

Definition 2.58 (Classification of $\alpha_\mathcal{C}$-Equivalence Classes). By definition $Cl_\mathcal{C}(D_{\alpha_\mathcal{C}}) = Cl_\mathcal{C}(D)$, where $D \in D_{\alpha_\mathcal{C}}$. $\square$

We adopt the same notational convention for $\mathcal{C}\backslash\alpha$ as we did for $\mathcal{T}\backslash\alpha$ (see convention 2.40), i.e. we use concrete terms to denote the $\alpha_\mathcal{C}$-equivalence classes that they represent. The examples below are written according to this convention.

Unlike $\Rightarrow_{\mathcal{T}\backslash\alpha}$, $\Rightarrow_{\mathcal{C}\backslash\alpha}$ is not a function, because it can perform an evaluation step on any component, as shown below.

Example 2.59 ($\Rightarrow_{\mathcal{C}\backslash\alpha}$ not a Function).

$$\begin{array}{lll} [A \mapsto 1+2, B \mapsto 3*4] & \Rightarrow_{\mathcal{C}\backslash\alpha} & [A \mapsto 3, B \mapsto 3*4], \\ [A \mapsto 1+2, B \mapsto 3*4] & \Rightarrow_{\mathcal{C}\backslash\alpha} & [A \mapsto 1+2, B \mapsto 12], \end{array}$$

but ${}_{\alpha_\mathcal{C}}\langle[A \mapsto 3, B \mapsto 3*4]\rangle \neq {}_{\alpha_\mathcal{C}}\langle[A \mapsto 1+2, B \mapsto 12]\rangle$. $\square$

Nevertheless, we have the following result:

Theorem 2.60. *$\Rightarrow_{\mathcal{C}\backslash\alpha}$ is confluent.* $\square$

The confluence of $\Rightarrow_{\mathcal{C}\backslash\alpha}$ gives rise to a partial function $Eval_{\mathcal{C}\backslash\alpha}$ which, when defined, returns the result of evaluating an $\alpha_\mathcal{C}$-equivalence class $D_{\alpha_\mathcal{C}}$ to the $\alpha_\mathcal{C}$-equivalence class of its evaluation normal form. Recall that a module is in an evaluation normal form if it does not have substitution evaluation redexes and its components are all $\Rightarrow_\mathcal{T}$-normal forms. By lemma 2.55 if a module is in an evaluation normal form, then so are all modules that it is $\alpha_\mathcal{C}$-equivalent to, so $Eval_{\mathcal{C}\backslash\alpha}$ is well-defined w.r.t. $\alpha_\mathcal{C}$-equivalence classes. The related function $Outcome_{\mathcal{C}\backslash\alpha}$ (see definition 2.10) is also well-defined. Note that since $\Rightarrow_\mathcal{C}$ is not confluent on individual modules (in particular, it inherits non-confluence of $\Rightarrow_\mathcal{T}$), $Eval_\mathcal{C}$ and $Outcome_\mathcal{C}$ are not defined on concrete modules[8]. The following example illustrates $Eval_{\mathcal{C}\backslash\alpha}(D_{\alpha_\mathcal{C}})$.

Example 2.61 (Evaluation of Modules).

$$Eval_{\mathcal{C}\backslash\alpha}([F \mapsto f \;@\; 2, G \mapsto \lambda y.y+g, f \mapsto \lambda x.x, g \mapsto 3]) = [F \mapsto 2, G \mapsto \lambda y.y+g, f \mapsto \lambda x.x, g \mapsto 3].$$

Note that the result is an $\Rightarrow_{\mathcal{C}\backslash\alpha}$-normal form, even though it has a $\rightarrow_{\mathcal{C}\backslash\alpha}$-redex: g in the second term is a substitution redex, but it occurs under a lambda, and therefore not in an evaluation context. $\square$

We also have the following result for $\Rightarrow_{\mathcal{C}\backslash\alpha}$ which ensures that a module's observable behavior with respect to $\Rightarrow_{\mathcal{C}\backslash\alpha}$ is the same regardless of which evaluation path it takes:

Lemma 2.62. *If $D_{\alpha_\mathcal{C}} \Rightarrow^*_{\mathcal{C}\backslash\alpha} Eval_\mathcal{C}(D_{\alpha_\mathcal{C}})$, then there is no infinite sequence of $\Rightarrow_{\mathcal{C}\backslash\alpha}$ steps originating at $D_{\alpha_\mathcal{C}}$.* $\square$

Proofs of both Theorem 2.60 and Lemma 2.62 are given in appendix C and are based on the fact that $\Rightarrow_{\mathcal{C}\backslash\alpha}$ satisfies the *diamond property*: if $D_{1\alpha_\mathcal{C}} \Rightarrow_{\mathcal{C}\backslash\alpha} D_{2\alpha_\mathcal{C}}$ and $D_{1\alpha_\mathcal{C}} \Rightarrow_{\mathcal{C}\backslash\alpha} D_{3\alpha_\mathcal{C}}$, where the two evaluation steps do not reduce the same redex, then there exists a $D_{4\alpha_\mathcal{C}}$ such that $D_{2\alpha_\mathcal{C}} \Rightarrow_{\mathcal{C}\backslash\alpha} D_{4\alpha_\mathcal{C}}$ and $D_{3\alpha_\mathcal{C}} \Rightarrow_{\mathcal{C}\backslash\alpha} D_{4\alpha_\mathcal{C}}$.

Interestingly, even though $\Rightarrow_{\mathcal{C}\backslash\alpha}$ is confluent, $\rightarrow_{\mathcal{C}\backslash\alpha}$ is *not*, due to the possibility of mutually recursive (subst) redexes that appear under a λ and therefore not in an evaluation context, as shown by the following example:

[8] However, we will sometimes write $Eval_\mathcal{C}(D)$ to denote a module $D' \in Eval_{\mathcal{C}\backslash\alpha}({}_{\alpha_\mathcal{C}}\langle D\rangle)$, and similarly with $Outcome_\mathcal{C}(D)$.

Example 2.63 (Non-confluence of $\rightarrow_{\mathcal{C}}$). The example is due to [AK97].

$$\begin{array}{lclcl} D_0 & = & [F \mapsto \lambda x.G, G \mapsto \lambda y.F] & \circ\!\!\rightarrow_{\mathcal{C}\backslash\alpha} & [F \mapsto \lambda x.\lambda y.F, G \mapsto \lambda y.F] & = & D_1, \\ D_0 & = & [F \mapsto \lambda x.G, G \mapsto \lambda y.F] & \circ\!\!\rightarrow_{\mathcal{C}\backslash\alpha} & [F \mapsto \lambda x.G, G \mapsto \lambda y.\lambda x.G] & = & D_2. \end{array}$$

D_1 (resp. D_2) has an even (resp. odd) number of λs for F and an odd (resp. even) number for G, and in every reduction sequence starting with D_1 (resp. D_2), all terms will have this property. Clearly, reduction sequences starting at D_1 and D_2 can never meet at a common term. □

The fact that $\rightarrow_{\mathcal{C}\backslash\alpha}$ is not confluent is problematic for the traditional approach for proving computational soundness – i.e., that a calculus is sound relative to an operational semantics. We have developed an approach for proving the computational soundness of the core module calculus that does not require the confluence of $\rightarrow_{\mathcal{C}\backslash\alpha}$. See section 3 for details.

We also have the following result (shown in appendix C):

Theorem 2.64. *$\mathcal{C}\backslash\alpha$ has the standardization property.* □

Remark 2.65 (Changes from [MT00]). The core module calculus defined here differs from the one defined in [MT00] in the following ways:

- In this presentation we have removed the requirement that a module does not import any hiddens. This requirement still exists at the linking level: only h-closed modules, i.e. modules that do not import hiddens, can form a linking expression. The reason for removing this requirement at the core module level is that we use a new set of contexts $\mathbf{Context}_{\mathcal{C},\mathcal{C}}$. A meaningful definition of filling such contexts with modules includes cases when the module that fills a context is not h-closed. See discussion in section 2.4.1 for more details.

- The module classification in section 2.3.2 simplifies the presentation in [MT00], where a classification of a module exposed the classes of all of its components, including the hidden ones:

  $$Cl_{\mathcal{C}}(D) = [l_i \overset{n}{\underset{i=1}{\mapsto}} Cl_{\mathcal{T}}(M_i)], \text{ where } D = [l_i \overset{n}{\underset{i=1}{\mapsto}} M_i]. \quad \text{([MT00] definition)}$$

  The reason why a less fine-grained classification is sufficient is because we only use it to show that two modules are observationally equivalent, i.e. that in any linking context if one of them evaluates to a module value of a certain class, then so does the other (see definition 3.2). It also turns out that the more specific classification gets in the way of defining garbage collection (GC) as a non-evaluation step (see discussion in section 2.6).

- Another change from [MT00] is the definition of $\mathbf{Value}_{\mathcal{C}}$. In [MT00], we required that all components of a module value are visible. Here, we allow module values to contain hidden components, as long as they are values.

  There are two reasons for this change:

  - In the core module calculus of [MT00] we defined (GC) to be an evaluation step, so it was natural to expect that all (GC) redexes in a module would be removed by evaluation. In the current presentation, we consider (GC) to be a non-evaluation step (see section 2.6), and therefore (GC) redexes can be removed only as an optimization (i.e. a non-evaluation step), but not as part of evaluation. Therefore, evaluation normal forms may contain (GC) redexes, which involve hidden value components.
  - There are module values in which references to a hidden component cannot be removed, even by non-evaluation steps, and so the component can not be GCed. As an example of such module, consider $[A \mapsto \lambda x.h, h \mapsto \lambda y.h]$. References to h in the component bound to the visible label A cannot be removed. The new definition is more flexible for handling such situations than the one in [MT00]. Note, however, that it is possible to place a module with hidden components referenced

in the visible part of the module into a linking context that will hide the components referencing the hidden part, so that all such components can be garbage collected. For instance, in the case of the above module (see definitions of the module operations $\oplus$ and $\{\text{hide } l\}$ in section 2.5):

$$\begin{array}{ll}([A \mapsto \lambda x.h, h \mapsto \lambda y.h] \oplus [B \mapsto (\lambda z.\lambda w.w) @ A])\{\text{hide } A\} & \Rightarrow^*_{\mathcal{L}} \\ {[h' \mapsto \lambda x.h, h \mapsto \lambda y.h, B \mapsto \lambda w.w]} & \end{array}$$

There are no references to the hidden components bound to h and h' in the visible part of the module, and therefore these components are garbage collectable. Therefore if GC step were an evaluation step, the module could have been evaluated to a value as defined in [MT00].

□

2.5 Linking Calculus ($\mathcal{L}$)

2.5.1 Syntax of Linking Expressions

The linking calculus extends the core module calculus with four module operators: linking, renaming, hiding, and binding. Figure 1 defines these operations for concrete linking expressions. In section 2.5.6 we introduce the notion of $\alpha_{\mathcal{L}}$-equivalence and extend these operations to $\alpha_{\mathcal{L}}$-equivalence classes.

A linking expression and a linking context are defined in figure 1. All modules that appear in a linking expression are h-closed, i.e. they do not import hidden labels. We introduce the following additional notations and definitions for linking expressions. A module identifier I is *free* in an expression L if it is not bound by a **let**. The set $FMI(L)$ of free module identifiers of L is inductively defined as follows:

$$\begin{array}{rcl}
FMI(H) & = & \emptyset \\
FMI(I) & = & \{I\} \\
FMI(L_1 \oplus L_2) & = & FMI(L_1) \cup FMI(L_2) \\
FMI(L[v \overset{\text{ren}}{\leftarrow} v']) & = & FMI(L) \\
FMI(L\{\text{hide } v\}) & = & FMI(L) \\
FMI(\mathbf{let}\ I = L_1\ \mathbf{in}\ L_2) & = & FMI(L_1) \cup (FMI(L_2) \setminus \{I\})
\end{array}$$

A linking expression L is *well-formed* if $FMI(L) = \emptyset$. In this presentation we assume all top-level linking expressions to be well-formed (expressions that are in the scope of a **let** may have free identifiers).

We define a substitution of a linking expression for a free identifier $L[I := L']$ similarly to substitution $M[x := N]$ in the term calculus (see definition 2.14).

$$\begin{array}{rcl}
I[I := L] & = & L \\
I_1[I := L] & = & I_1 \text{ if } I \neq I_1 \\
H[I := L] & = & H \\
L_1 \oplus L_2[I := L] & = & L_1[I := L] \oplus L_2[I := L] \\
L_1\{\text{hide } v\}[I := L] & = & (L_1[I := L])\{\text{hide } v\} \\
L_1[v \overset{\text{ren}}{\leftarrow} v'][I := L] & = & (L_1[I := L])[v \overset{\text{ren}}{\leftarrow} v'] \\
(\mathbf{let}\ I = L_1\ \mathbf{in}\ L_2)[I := L] & = & \mathbf{let}\ I = L_1[I := L]\ \mathbf{in}\ L_2 \\
(\mathbf{let}\ I_1 = L_1\ \mathbf{in}\ L_2)[I := L] & = & \mathbf{let}\ I_1 = L_1[I := L]\ \mathbf{in}\ L_2[I := L] \\
& & \text{if } I \neq I_1 \text{ and } I_1 \notin FMI(L) \text{ or } I \notin FMI(L_2) \\
(\mathbf{let}\ I_1 = L_1\ \mathbf{in}\ L_2)[I := L] & = & \mathbf{let}\ I_2 = L_1[I := L]\ \mathbf{in}\ L_2[I_1 := I_2][I := L] \\
& & \text{if } I \neq I_1,\ I_1 \in FMI(L),\ I \in FMI(L_2), \\
& & \text{and } I_2 \notin FMI(L) \cup FMI(L_2)
\end{array}$$

In order to define redexes of module reductions lifted to the linking level and redexes of α-renaming of linking expressions, it is convenient to introduce a set of contexts that can be filled with a term so that the result of the filling is a linking expression.

Definition 2.66. Let $\mathbf{Context}_{\mathcal{T},\mathcal{L}}$ denote the set of contexts $\{\mathbb{L}\{\mathbb{D}\} \mid \mathbb{L} \in \mathbf{Context}_{\mathcal{L},\mathcal{L}}, \mathbb{D} \in \mathbf{Context}_{\mathcal{T},\mathcal{C}}\}$, and let $\mathbb{F}$ range over $\mathbf{Context}_{\mathcal{T},\mathcal{L}}$. The result of filling a context $\mathbb{F} = \mathbb{L}\{\mathbb{D}\}$ with a term $M \in \mathbf{Term}_{\mathcal{T}}$ is defined as $\mathbb{L}\{\mathbb{D}\{M\}\}$ if $\mathbb{D}\{M\} \in \mathbf{HTerm}_{\mathcal{C}}$, otherwise it is undefined. □

Example 2.67 (Contexts in the Set $\mathbf{Context}_{\mathcal{T},\mathcal{L}}$). The context $\mathbb{F}_1 = [A \mapsto \square, B \mapsto 2]\{\text{hide } A\}$ is obtained by filling a linking context $\square\{\text{hide } A\}$ with the module context $[A \mapsto \square, B \mapsto 2]$. The context $\mathbb{F}_2 = [A \mapsto \square]$ is the result of filling $\square$ with $[A \mapsto \square] \in \mathbf{Context}_{\mathcal{T},\mathcal{C}}$. The latter example demonstrates that $\mathbf{Context}_{\mathcal{T},\mathcal{C}} \subset \mathbf{Context}_{\mathcal{T},\mathcal{L}}$. □

2.5.2 Evaluation and Reduction of Linking Expressions

Intuitively, the linking of modules H_1 and H_2, written $H_1 \oplus H_2$, takes the union of their bindings. To avoid naming conflicts between both visible and hidden labels, $BL(H_1)$ and $BL(H_2)$ must be disjoint[9]. The fact that the import labels of an h-closed module are non-hidden prevents the components of one module from accessing hidden components of the other one when they are linked.

The renaming operator renames a visible module label (an import or an export) to another visible label. The proviso "$v \in BL(H)$ implies $v' \notin BL(H)$" guarantees that the resulting module is well-formed[10], i.e. does not have two components bound to the same label name. Renaming import and export labels is the way to connect an exported component of one module to an import site in another.

Hiding renames a visible label to a fresh (i.e. not appearing in the module) hidden. As we show below, the choice of this hidden name does not matter when we consider $\alpha_{\mathcal{C}}$-equivalence classes of modules. The restriction $v \notin \mathit{Imports}(H)$ prevents renaming an import to a hidden label.

The binding operator **let** $I = L_1$ **in** L_2 names the result of evaluating the definition term L_1 and uses the name within the body term L_2. This models situations in which the same module is used multiple times in different contexts (see examples below). In this presentation we require that the definition term L_1 is first evaluated to an h-closed module H, and then H is substituted into the body L_2. This is consistent with the call-by-value semantics of other reductions in our calculus (e.g. the term reductions and substitution at the module level). In section 2.5.6 we discuss the more general non-deterministic (let) reduction introduced in [MT00]. All the results shown for the non-deterministic (let) hold for (let) restricted to the call-by-value case.

The definition of $\rightarrow_{\mathcal{L}}$ lifts core module reduction steps to the linking level. The lifted core module reduction steps are only considered evaluation steps if they are not surrounded by any link-level operators; this forces all link-level steps to be performed first in a "link-time stage", followed by a "run-time stage" of core module steps.

Similarly to the calculi $\mathcal{T}$ and $\mathcal{C}$, a redex is a subterm occurrence which specifies exactly where in the term the reduction has been performed. Observe that there are two kinds of reductions at the linking level: reductions of link-level operations, e.g. (link) or (rename), and reductions of modules (mod-ev) and (mod-nev). These two kinds of reductions require different kinds of redexes: a link-level reduction is uniquely specified by its linking context and the linking expression, but for a reduction on a module we need to specify which of the (possibly several) module redexes is reduced. Therefore for the module reductions we need to use contexts of the set $\mathbf{Context}_{\mathcal{T},\mathcal{L}}$ to specify the redex. The definition below formalizes the notion of a linking redex for these two cases.

Definition 2.68 (Linking Redex).

- *Case of the rules (link), (rename), (hide), and (let).* A subterm occurrence $(\mathbb{L}, L)$ is called a *linking redex* if its parsing matches the left-hand side of one of the reduction rules (link), (rename), (hide), or (let), and $\mathbb{L}\{L\}$ satisfies all the requirements for the rule. We use a variable K to denote the

[9] The extension of linking to $\alpha_{\mathcal{C}}$-equivalence classes introduced later in this section removes the requirement that hidden labels of the two modules are disjoint.

[10] An attempt to rename one bound label in a module to another one causes a linking error.

expression L filling the context $\mathbb{L}$, i.e. the notation $(\mathbb{L}, K)$ denotes a linking redex in the case of these four reduction rules. A linking redex $(\mathbb{L}, K)$ is an evaluation redex.

- *Case of the rules (mod-ev) and (mod-nev).* A subterm occurrence $(\mathbb{F}, R)$ is called a *linking redex* if $\mathbb{F} = \mathbb{L}\{\mathbb{D}\}$, and $(\mathbb{D}, R)$ is a module redex. The linking redex is an evaluation redex if $\mathbb{L} = \square$ and $\mathbb{D} \in \mathbf{EvalContext}_{\mathcal{T},\mathcal{C}}$, otherwise it is a non-evaluation redex.

$\square$

Example 2.69 (Linking Redexes). The following are evaluation redexes:

$$\begin{array}{ll} (\square[B \overset{\text{ren}}{\leftarrow} C], [A \mapsto 2, B \mapsto 3]\{\text{hide } A\}), & \text{(hide)} \\ ([A \mapsto \square, B \mapsto 4 * 5, C \mapsto \lambda x.C], 2+3), & \text{(mod-ev)} \\ ([A \mapsto 2+3, B \mapsto \square, C \mapsto \lambda x.C], 4 * 5). & \text{(mod-ev)} \end{array}$$

The first redex is a (hide) redex appearing in the linking context $\square[B \overset{\text{ren}}{\leftarrow} C]$. The second and the third redexes form the same module $[A \mapsto 2+3, B \mapsto 4 * 5, C \mapsto \lambda x.C]$. By specifying the module context containing each of the two redexes we are able to distinguish them.

The following are non-evaluation redexes:

$$\begin{array}{ll} ([A \mapsto 2+3, B \mapsto 4 * 5, C \mapsto \lambda x.\square], C), & \text{(mod-nev)} \\ ([A \mapsto \square] \oplus [h \mapsto 3], 2+3). & \text{(mod-nev)} \end{array}$$

The first redex is in the same module as the two evaluation redexes above. The second example is of a module evaluation step which occurs in a non-empty linking context, and therefore is a non-evaluation linking step.

The following subterm occurrence is not a redex: $(\square, [A \mapsto 2, B \mapsto 3][A \overset{\text{ren}}{\leftarrow} B])$ (even though it matches the left-hand side of the rule (rename), it does not satisfy the conditions for the rule, see figure 1). $\square$

The structure of $\mathbf{Context}_{\mathcal{L},\mathcal{L}}$ allows the link-level operators to be evaluated in any order, as we see in the examples below. We show that $\Longrightarrow_{\mathcal{L}\backslash\alpha}$, i.e. the "lifting" of $\Longrightarrow_{\mathcal{L}}$ to $\alpha_{\mathcal{L}}$-equivalence classes introduced in section 2.5.6, is confluent, and therefore at the level of $\alpha_{\mathcal{L}}$-equivalence classes the result is the same for every order of evaluation.

Example 2.70 (Evaluation of a Linking Expression). The example illustrates reuse of a module via **let**, as well as linking, hiding, and module evaluation:

$$\begin{array}{ll} \mathbf{let}\ A = [X \mapsto 2]\ \mathbf{in}\ (A \oplus [Y \mapsto X + Z]) \oplus ((A \oplus [Z \mapsto X * 5])\{\text{hide } X\}) & \Longrightarrow_{\mathcal{L}} \\ ([X \mapsto 2] \oplus [Y \mapsto X + Z]) \oplus (([X \mapsto 2] \oplus [Z \mapsto X * 5])\{\text{hide } X\}) & \Longrightarrow^*_{\mathcal{L}} \\ [X \mapsto 2, Y \mapsto X + Z] \oplus [h \mapsto 2, Z \mapsto h * 5] & \Longrightarrow_{\mathcal{L}} \\ [X \mapsto 2, Y \mapsto X + Z, h \mapsto 2, Z \mapsto h * 5] & \Longrightarrow^*_{\mathcal{L}} \\ [X \mapsto 2, Y \mapsto 12, h \mapsto 2, Z \mapsto 10] & \end{array}$$

The first evaluation step reduces the **let**. The next 3 steps (shown as one $\Longrightarrow^*_{\mathcal{L}}$ sequence) perform two linking operations and hiding. The two (link) reductions may be performed in any order. The hiding can be performed only after the linking $[X \mapsto 2] \oplus [Z \mapsto X * 5]$, since the argument of hiding must be a module. Hiding renames a visible label X to a new hidden label h. After this renaming has been performed, the component is no longer accessible from outside of the module in which it is defined. The final linking reduces the expression to a single module, and the following $\Longrightarrow^*_{\mathcal{L}}$ sequence evaluates the module to a module value (all components are bound to values). $\square$

In the example above the label h in the resulting module value is not referenced in any other component. In [MT00] we have introduced a core module calculus rule (GC) for garbage-collecting such hidden components. In this presentation a (GC) rule is added to the core module calculus as a non-evaluation step (see section 2.6). If we apply this rule, the resulting module is reduced to $[X \mapsto 2, Y \mapsto 12, Z \mapsto 10]$. However, since (GC) is a non-evaluation step, the result of the evaluation above is an evaluation normal form even before applying (GC).

We use the following abbreviation for a sequence of nested **let**s: **let** $I_1 = L_1$ $I_2 = L_2 \ldots I_n = L_n$ **in** L_{n+1} is syntactic sugar for **let** $I_1 = L_1$ **in** (**let** $I_2 = L_2$ **in**...(**let** $I_n = L_n$ **in** $L_{n+1}) \ldots$).

The next example shows possibilities of connecting module components via renaming.

Example 2.71 (Connecting Module Components Via Renaming).

$$
\begin{array}{ll}
\textbf{let} & A = [X \mapsto 0] \\
 & B = [Y \mapsto Z + 1] \\
 & C = A \oplus B \\
\textbf{in} & C[Y \overset{\text{ren}}{\leftarrow} Y_1][Z \overset{\text{ren}}{\leftarrow} X] \oplus B[Z \overset{\text{ren}}{\leftarrow} Y_1] \\
\Rightarrow^*_{\mathcal{L}} & [X \mapsto 0, Y_1 \mapsto X + 1, Y \mapsto Y_1 + 1] \\
\Rightarrow^*_{\mathcal{L}} & [X \mapsto 0, Y_1 \mapsto 1, Y \mapsto 2].
\end{array}
$$

Note that the first renaming applied to C renames an exported label, and the second one (as well as the renaming of Z in B) renames an imported label. □

Cases of $\circ\!\!\longrightarrow_{\mathcal{L}}$ include an evaluation of a module in a non-empty linking context and a non-evaluation step on a module in an empty context, as shown in the following example:

Example 2.72 (Non-Evaluation Steps in $\mathcal{L}$). Even though the substitution is an evaluation step at the module level, it is a non-evaluation step at the linking level, since the module appears in a non-empty context:

$$[X \mapsto 2, Z \mapsto X + 3] \oplus [P \mapsto X + Z] \circ\!\!\longrightarrow_{\mathcal{L}} [X \mapsto 2, Z \mapsto 2 + 3] \oplus [P \mapsto X + Z]$$

A non-evaluation step on a module is a non-evaluation linking step even if the module appears in an empty context:

$$[F \mapsto \lambda x.Y + 3, Y \mapsto 2] \circ\!\!\longrightarrow_{\mathcal{L}} [F \mapsto \lambda x.2 + 3, Y \mapsto 2].$$

□

Unlike in $\mathcal{T}$ and $\mathcal{C}$, classification of linking expressions is defined only on their $\alpha_{\mathcal{L}}$-equivalence classes, but not on concrete linking expressions. The reasons for this are discussed later (see section 2.5.6).

2.5.3 Lifting of $\alpha_{\mathcal{C}}$-equivalence to $\mathcal{L}$ Terms

We lift the notion of $\alpha_{\mathcal{C}}$-equivalence classes from modules to linking expressions and, since modules are used as elements of linking contexts, to such contexts. In section 2.5.5 we introduce α-renaming of **let**-bound module identifiers in linking expressions, denoted $\longrightarrow_{\alpha_I}$, which induces an equivalence relation $=^I_\alpha$. Finally, we join the two notions of α-equivalence of linking expressions (induced by $\longrightarrow_{\alpha_{\mathcal{C}}}$ and $\longrightarrow_{\alpha_I}$) to form α-equivalence relation $=^{\mathcal{L}}_\alpha$ in section 2.5.6. As for the lifting of $\longrightarrow_{\alpha_{\mathcal{T}}}$ to the module level in section 2.4.3, we use the same notation $\longrightarrow_{\alpha_{\mathcal{C}}}$ for the lifting of $\longrightarrow_{\alpha_{\mathcal{C}}}$ to linking expressions and linking contexts, rather than introduce a separate notation.

Definition 2.73 ($\alpha_{\mathcal{C}}$-Renaming of Linking Expressions).

- We say that:
  - L reduces to L' in a *one-step $\alpha_{\mathcal{C}}$-renaming* with the $\alpha_{\mathcal{C}}$-redex $(\mathbb{F}, x, y)$, written $L \xrightarrow{(\mathbb{F},x,y)}_{\alpha_{\mathcal{C}}} L'$, if there exist $\mathbb{L}, \mathbb{D}, H$, and H' such that $L = \mathbb{L}\{H\}$, $L' = \mathbb{L}\{H'\}$, $H \xrightarrow{(\mathbb{D},x,y)}_{\alpha_{\mathcal{C}}} H'$, and $\mathbb{F} = \mathbb{L}\{\mathbb{D}\}$.
  - L reduces to L' in a *one-step $\alpha_{\mathcal{C}}$-renaming* with the $\alpha_{\mathcal{C}}$-redex $(\mathbb{L}, h, h')$, written $L \xrightarrow{(\mathbb{L},h,h')}_{\alpha_{\mathcal{C}}} L'$, if there exist $\mathbb{L}, H$, and H' such that $L = \mathbb{L}\{H\}$, $L' = \mathbb{L}\{H'\}$, and $H \xrightarrow{(h,h')}_{\alpha_{\mathcal{C}}} H'$.

  As for the other relations, the redex annotation may be omitted.

- We write $L \xrightarrow{S}{}^{*}_{\alpha_{\mathcal{C}}} L'$ if L reduces to L' by a sequence of one-step $\alpha_{\mathcal{C}}$-renamings reducing $\alpha_{\mathcal{C}}$-redexes in S left-to-right.

$\square$

Since $\longrightarrow_{\alpha_{\mathcal{C}}}$ is symmetric on $\mathbf{Term}_{\mathcal{C}}$, its lifting to $\mathbf{Term}_{\mathcal{L}}$ is also symmetric. The relations $\longrightarrow^{*}_{\alpha_{\mathcal{C}}}$ and $=^{\mathcal{C}}_{\alpha}$ are defined as usual, and, since $\longrightarrow_{\alpha_{\mathcal{C}}}$ is symmetric, coincide. $\alpha_{\mathcal{C}}$-equivalence classes of linking expressions are defined as usual:

Definition 2.74 ($\alpha_{\mathcal{C}}$-Equivalence Classes of Linking Expressions). If $L \in \mathbf{Term}_{\mathcal{L}}$, then its $\alpha_{\mathcal{C}}$-equivalence class, written ${}_{\alpha_{\mathcal{C}}}\langle L\rangle$, is defined as ${}_{\alpha_{\mathcal{C}}}\langle L\rangle = \{L' \in \mathbf{Term}_{\mathcal{L}} \mid L =^{\mathcal{C}}_{\alpha} L'\}$. We use $L_{\alpha_{\mathcal{C}}}$ as a meta-variable that ranges over $\alpha_{\mathcal{C}}$-equivalence classes of linking expressions. $\square$

Unlike in $\mathcal{T}$ and $\mathcal{C}$, we also extend the notion of $\alpha_{\mathcal{C}}$-equivalence classes to linking contexts. It is possible, since linking contexts contain entire modules, therefore $\alpha_{\mathcal{C}}$-renaming a module in a linking context (i.e. renaming its hiddens or λ-bound variables) does not affect any other modules, in particular those of a linking expression that fills the hole of the context. Note that this is not the case in the other calculi. For instance, extending $\alpha_{\mathcal{T}}$-renaming to context in $\mathcal{T}$ would cause the following problem: suppose we α-rename the context $\lambda x.\square$ to $\lambda y.\square$, then filling this context with x will cause variable capture for the former context, but not for the latter, and the resulting terms will have different meaning. However, no variable or label capture can happen when we fill a linking context with a linking expression, therefore $\alpha_{\mathcal{C}}$-renaming a module which is a part of a linking context cannot change the meaning of the linking expression.

In order to describe $\alpha_{\mathcal{C}}$-redexes of linking contexts, we introduce the notion of two-hole contexts:

Definition 2.75. A set of two-hole linking contexts is defined as follows:

$$\begin{array}{rcl} {}^2\mathbb{L} \in \mathbf{Context}_{\mathcal{L}\times\mathcal{L},\mathcal{L}} & ::= & \mathbb{L} \oplus \mathbb{L} \mid {}^2\mathbb{L} \oplus L \mid L \oplus {}^2\mathbb{L} \mid {}^2\mathbb{L}[v \overset{\text{ren}}{\longleftarrow} v'] \mid {}^2\mathbb{L}\{\text{hide } v\} \mid \\ & & \mathbf{let}\ I = \mathbb{L}\ \mathbf{in}\ \mathbb{L} \mid \mathbf{let}\ I = {}^2\mathbb{L}\ \mathbf{in}\ L \mid \mathbf{let}\ I = L\ \mathbf{in}\ {}^2\mathbb{L} \end{array}$$

The result of filling of a two-hole context ${}^2\mathbb{L}$ with two linking expressions L_1 and L_2 is denoted as ${}^2\mathbb{L}\{L_1, L_2\}$, where the two expressions fill the holes of the context left-to-right. Note that it is also meaningful to fill the holes of a context ${}^2\mathbb{L}$ with other contexts, s.a. $\mathbb{L}$ or $\mathbb{F}$. Contexts obtained from ${}^2\mathbb{L}$ by filling one of the holes with a context $\mathbb{F} \in \mathbf{Context}_{\mathcal{T},\mathcal{L}}$ form the set $\mathbf{Context}_{\mathcal{T}\times\mathcal{L},\mathcal{L}}$. We use ${}^2\mathbb{F}$ to range over this set. Contexts in $\mathbf{Context}_{\mathcal{T}\times\mathcal{L},\mathcal{L}}$ are filled with a term of $\mathbf{Term}_{\mathcal{T}}$ and a linking expression. $\square$

Example 2.76 (Contexts of $\mathbf{Context}_{\mathcal{L}\times\mathcal{L},\mathcal{L}}$). Consider a two-hole linking context ${}^2\mathbb{L} = (\square \oplus \square)\{\text{hide } A\}$. The context can be filled with two linking expressions:

$${}^2\mathbb{L}\{[A \mapsto 2], [A \mapsto 3][A \overset{\text{ren}}{\longleftarrow} B]\} \;=\; ([A \mapsto 2] \oplus [A \mapsto 3][A \overset{\text{ren}}{\longleftarrow} B])\{\text{hide } A\}.$$

The result of filling ${}^2\mathbb{L}$ with contexts $[A \mapsto \square] \in \mathbf{Context}_{\mathcal{T},\mathcal{L}}$ and $\square[A \overset{\text{ren}}{\longleftarrow} B] \in \mathbf{Context}_{\mathcal{L},\mathcal{L}}$ is

$${}^2\mathbb{L}\{[A \mapsto \square], \square[A \overset{\text{ren}}{\longleftarrow} B]\} \;=\; ([A \mapsto \square] \oplus \square[A \overset{\text{ren}}{\longleftarrow} B])\{\text{hide } A\}.$$

Let ${}^2\mathbb{F} = ([A \mapsto \square] \oplus \square[A \overset{\text{ren}}{\longleftarrow} B])\{\text{hide } A\} \in \mathbf{Context}_{\mathcal{T}\times\mathcal{L},\mathcal{L}}$. Note that ${}^2\mathbb{F} \in \mathbf{Context}_{\mathcal{T}\times\mathcal{L},\mathcal{L}}$. We can fill this context with a term and a linking expression as follows:

$${}^2\mathbb{F}\{2, [A \mapsto 3]\} \;=\; ([A \mapsto 2] \oplus [A \mapsto 3][A \overset{\text{ren}}{\longleftarrow} B])\{\text{hide } A\}.$$

$\square$

Definition 2.77 ($\alpha_{\mathcal{C}}$-Renaming of Linking Contexts). We say that a linking context $\mathbb{L}$ reduces by a *one-step $\alpha_{\mathcal{C}}$-renaming* to a linking context $\mathbb{L}'$ with the $\alpha_{\mathcal{C}}$-redex $({}^2\mathbb{F}, x, y)$, written $\mathbb{L} \xrightarrow{({}^2\mathbb{F},x,y)}_{\alpha_{\mathcal{C}}} \mathbb{L}'$, if there exist H, H', and ${}^2\mathbb{L}$ such that $H \xrightarrow{(\mathbb{D},x,y)}_{\alpha_{\mathcal{C}}} H'$, and either $\mathbb{L} = {}^2\mathbb{L}\{H, \square\}$, $\mathbb{L}' = {}^2\mathbb{L}\{H', \square\}$, and ${}^2\mathbb{F} = {}^2\mathbb{L}\{\mathbb{D}, \square\}$, or $\mathbb{L} = {}^2\mathbb{L}\{\square, H\}$, $\mathbb{L}' = {}^2\mathbb{L}\{\square, H'\}$, and ${}^2\mathbb{F} = {}^2\mathbb{L}\{\square, \mathbb{D}\}$.

We say that a linking context $\mathbb{L}$ reduces by a *one-step $\alpha_{\mathcal{C}}$-renaming* to a linking context $\mathbb{L}'$ with the $\alpha_{\mathcal{C}}$-redex $({}^2\mathbb{L}, h, h')$, written $\mathbb{L} \xrightarrow{({}^2\mathbb{L},h,h')}_{\alpha_{\mathcal{C}}} \mathbb{L}'$, if there exist H, H' such that $H \xrightarrow{(h,h')}_{\alpha_{\mathcal{C}}} H'$, and either $\mathbb{L} = {}^2\mathbb{L}\{H, \Box\}$ and $\mathbb{L}' = {}^2\mathbb{L}\{H', \Box\}$, or $\mathbb{L} = {}^2\mathbb{L}\{\Box, H\}$ and $\mathbb{L}' = {}^2\mathbb{L}\{\Box, H'\}$.

We extend $\longrightarrow^*_{\alpha_{\mathcal{C}}}$ and $=^{\mathcal{C}}_{\alpha}$ to linking contexts in the usual way. $\Box$

We define $\alpha_{\mathcal{C}}$-equivalence of linking contexts as follows:

Definition 2.78 ($\alpha_{\mathcal{C}}$-Equivalence Classes of Linking Contexts). *$\alpha_{\mathcal{C}}$-equivalence class* of a linking context $\mathbb{L}$, denoted ${}_{\alpha_{\mathcal{C}}}\langle\mathbb{L}\rangle$, is a set $\{\mathbb{L}' \mid \mathbb{L}' =^{\mathcal{C}}_{\alpha} \mathbb{L}\}$. We use $\mathbb{L}_{\alpha_{\mathcal{C}}}$ to range over $\alpha_{\mathcal{C}}$-equivalence classes of linking contexts. $\Box$

We extend the operation of filling a context with a linking expression to $\alpha_{\mathcal{C}}$-equivalence classes as follows: ${}_{\alpha_{\mathcal{C}}}\langle\mathbb{L}\rangle\{{}_{\alpha_{\mathcal{C}}}\langle L\rangle\} = {}_{\alpha_{\mathcal{C}}}\langle\mathbb{L}_1\{L_1\}\rangle$, where $\mathbb{L}_1 \in {}_{\alpha_{\mathcal{C}}}\langle\mathbb{L}\rangle$, $L_1 \in {}_{\alpha_{\mathcal{C}}}\langle L\rangle$. Filling of a context is well-defined for $\alpha_{\mathcal{C}}$-equivalence classes:

Lemma 2.79. *If $\mathbb{L} =^{\mathcal{C}}_{\alpha} \mathbb{L}'$ and $L =^{\mathcal{C}}_{\alpha} L'$, then ${}_{\alpha_{\mathcal{C}}}\langle\mathbb{L}\{L\}\rangle = {}_{\alpha_{\mathcal{C}}}\langle\mathbb{L}'\{L'\}\rangle$.* $\Box$

Unlike the calculi $\mathcal{T}$ and $\mathcal{C}$, where we needed to add markings to the calculus syntax in order to define α-equivalence classes of subterm occurrences, in the case of $\mathcal{L}$ the definition of an $\alpha_{\mathcal{C}}$-equivalence class of a subterm occurrence is much simpler, because it uses the notion of $\alpha_{\mathcal{C}}$-equivalence class of a linking context. We define $\alpha_{\mathcal{C}}$-renaming for both subterm occurrences of the form $(\mathbb{L}, L)$ and of the form $(\mathbb{F}, M)$, since these two kinds of subterm occurrences correspond to two kinds of linking redexes (see definition 2.68).

Definition 2.80 ($\alpha_{\mathcal{C}}$-Renaming of Subterm Occurrences in $\mathcal{L}$). We say that $(\mathbb{L}, L)$ reduces to $(\mathbb{L}', L')$ by a one-step $\alpha_{\mathcal{C}}$-renaming with the $\alpha_{\mathcal{C}}$-redex $(\mathbb{F}, x, y)$, written $(\mathbb{L}, L) \xrightarrow{(\mathbb{F},x,y)}_{\alpha_{\mathcal{C}}} (\mathbb{L}', L')$, if

- either $\mathbb{L} = \mathbb{L}'$ and there exists $\mathbb{F}_1 \in \mathbf{Context}_{\mathcal{T},\mathcal{L}}$ such that $L \xrightarrow{(\mathbb{F}_1,x,y)}_{\alpha_{\mathcal{C}}} L'$ and $\mathbb{F} = \mathbb{L}\{\mathbb{F}_1\}$,
- or $L = L'$ and there exists ${}^2\mathbb{F}$ such that $\mathbb{L} \xrightarrow{({}^2\mathbb{F},x,y)}_{\alpha_{\mathcal{C}}} \mathbb{L}'$, where $\mathbb{F} = {}^2\mathbb{F}\{L, \Box\}$ or $\mathbb{F} = {}^2\mathbb{F}\{\Box, L\}$.

We say that $(\mathbb{L}, L)$ reduces to $(\mathbb{L}', L')$ by a one-step $\alpha_{\mathcal{C}}$-renaming with the $\alpha_{\mathcal{C}}$-redex $(\mathbb{L}_1, h, h')$, written $(\mathbb{L}, L) \xrightarrow{(\mathbb{L}_1,h,h')}_{\alpha_{\mathcal{C}}} (\mathbb{L}', L')$, if

- either $\mathbb{L} = \mathbb{L}'$ and there exists $\mathbb{L}_2$ such that $L \xrightarrow{(\mathbb{L}_2,h,h')}_{\alpha_{\mathcal{C}}} L'$, where $\mathbb{L}_1 = \mathbb{L}\{\mathbb{L}_2\}$,
- or $L = L'$ and there exists ${}^2\mathbb{L}$ such that $\mathbb{L} \xrightarrow{({}^2\mathbb{L},h,h')}_{\alpha_{\mathcal{C}}} \mathbb{L}'$, where $\mathbb{L}_1 = {}^2\mathbb{L}\{L, \Box\}$ or $\mathbb{L}_1 = {}^2\mathbb{L}\{\Box, L\}$.

We say that $(\mathbb{F}, M)$ reduces to $(\mathbb{F}', M')$ by a one-step $\alpha_{\mathcal{C}}$-renaming with the $\alpha_{\mathcal{C}}$-redex $(\mathbb{F}_1, x, y)$, written $(\mathbb{F}, M) \xrightarrow{(\mathbb{F}_1,x,y)}_{\alpha_{\mathcal{C}}} (\mathbb{F}', M')$, if $\mathbb{F} = \mathbb{L}\{\mathbb{D}\}$, $\mathbb{F}' = \mathbb{L}'\{\mathbb{D}'\}$, and

- either $\mathbb{L} = \mathbb{L}'$ and there exists $\mathbb{D}_1$ such that $(\mathbb{D}, M) \xrightarrow{(\mathbb{D}_1,x,y)}_{\alpha_{\mathcal{C}}} (\mathbb{D}', M')$, where $\mathbb{F}_1 = \mathbb{L}\{\mathbb{D}_1\}$,
- or $(\mathbb{D}, M) = (\mathbb{D}', M')$ and there exists ${}^2\mathbb{F} \in \mathbf{Context}_{\mathcal{T}\times\mathcal{L},\mathcal{L}}$ such that $\mathbb{L} \xrightarrow{({}^2\mathbb{F},x,y)}_{\alpha_{\mathcal{C}}} \mathbb{L}'$, where $\mathbb{F}_1 = {}^2\mathbb{F}\{\mathbb{D}\{M\}, \Box\}$ or $\mathbb{F}_1 = {}^2\mathbb{F}\{\Box, \mathbb{D}\{M\}\}$.

We say that $(\mathbb{F}, M)$ reduces to $(\mathbb{F}', M')$ by a one-step $\alpha_{\mathcal{C}}$-renaming with the $\alpha_{\mathcal{C}}$-redex $(\mathbb{L}, h, h')$, written $(\mathbb{F}, M) \xrightarrow{(\mathbb{L}_1,h,h')}_{\alpha_{\mathcal{C}}} (\mathbb{F}', M')$, if $\mathbb{F} = \mathbb{L}\{\mathbb{D}\}$, $\mathbb{F}' = \mathbb{L}'\{\mathbb{D}'\}$, and

- either $\mathbb{L} = \mathbb{L}' = \mathbb{L}_1$ and $(\mathbb{D}, M) \xrightarrow{(h,h')}_{\alpha_{\mathcal{C}}} (\mathbb{D}', M')$,
- or $(\mathbb{D}, M) = (\mathbb{D}', M')$, and there exists ${}^2\mathbb{L}$ such that $\mathbb{L} \xrightarrow{({}^2\mathbb{L},h,h')}_{\alpha_{\mathcal{C}}} \mathbb{L}'$, where $\mathbb{L}_1 = {}^2\mathbb{L}\{\mathbb{D}\{M\}, \Box\}$ or $\mathbb{L}_1 = {}^2\mathbb{L}\{\Box, \mathbb{D}\{M\}\}$.

$\Box$

We define $\alpha_{\mathcal{C}}$-equivalence classes of both kinds of subterm occurrences as usual. We use a meta-variable $Sub^{\mathcal{L},\mathcal{L}}_{\alpha_{\mathcal{C}}}$ to range over the set of subterm occurrences of the form $(\mathbb{L}, L)$, and a meta-variable $Sub^{\mathcal{T},\mathcal{L}}_{\alpha_{\mathcal{C}}}$ to range over those of the form $(\mathbb{F}, M)$.

For both kinds of subterm occurrences we have the following result:

Lemma 2.81 ($\alpha_{\mathcal{C}}$-Renaming Preserves Distinctness of Subterms in $\mathcal{L}$). *If $\mathbb{L}_1\{L_1\} = \mathbb{L}_2\{L_2\} \in \mathbf{Term}_{\mathcal{L}}$, $(\mathbb{L}_1, L_1) \neq (\mathbb{L}_2, L_2)$, and either $(\mathbb{L}_1, L_1) \xrightarrow{(\mathbb{F},x,y)}_{\alpha_{\mathcal{C}}} (\mathbb{L}'_1, L'_1)$ and $(\mathbb{L}_2, L_2) \xrightarrow{(\mathbb{F},x,y)}_{\alpha_{\mathcal{C}}} (\mathbb{L}'_2, L'_2)$, or $(\mathbb{L}_1, L_1) \xrightarrow{(\mathbb{L},h,h')}_{\alpha_{\mathcal{C}}} (\mathbb{L}'_1, L'_1)$ and $(\mathbb{L}_2, L_2) \xrightarrow{(\mathbb{L},h,h')}_{\alpha_{\mathcal{C}}} (\mathbb{L}'_2, L'_2)$, then $(\mathbb{L}'_1, L'_1) \neq (\mathbb{L}'_2, L'_2)$.*

Similarly, if $\mathbb{F}_1\{M_1\} = \mathbb{F}_2\{M_2\} \in \mathbf{Term}_{\mathcal{L}}$, $(\mathbb{F}_1, M_1) \neq (\mathbb{F}_2, M_2)$, and either $(\mathbb{F}_1, M_1) \xrightarrow{(\mathbb{F},x,y)}_{\alpha_{\mathcal{C}}} (\mathbb{F}'_1, M'_1)$ and $(\mathbb{F}_2, M_2) \xrightarrow{(\mathbb{F},x,y)}_{\alpha_{\mathcal{C}}} (\mathbb{F}'_2, M'_2)$, or $(\mathbb{F}_1, M_1) \xrightarrow{(\mathbb{L},h,h')}_{\alpha_{\mathcal{C}}} (\mathbb{F}'_1, M'_1)$ and $(\mathbb{F}_2, M_2) \xrightarrow{(\mathbb{L},h,h')}_{\alpha_{\mathcal{C}}} (\mathbb{F}'_2, M'_2)$, then $(\mathbb{F}'_1, M'_1) \neq (\mathbb{F}'_2, M'_2)$. □

Recall that in $\mathcal{T}$ and $\mathcal{C}$ it is the case that if two subterm occurrences are $\alpha_{\mathcal{C}}$-equivalent and one is a redex, then so is the other (see lemmas 2.34 and 2.54). This property holds for linking redexes of the form $(\mathbb{F}, M)$:

Lemma 2.82 (Properties of $\alpha_{\mathcal{C}}$-Equivalent Linking Redexes $(\mathbb{F}, M)$).

- *If $(\mathbb{F}, M)$ is a linking redex and $(\mathbb{F}, M) \longrightarrow_{\alpha_{\mathcal{C}}} (\mathbb{F}', M')$, then $(\mathbb{F}', M')$ is a linking redex.*
- *If $L_1 \xRightarrow{(\mathbb{F},M)}_{\mathcal{L}} L_2$, $L_1 \xrightarrow{(\mathbb{L},h,h')}_{\alpha_{\mathcal{C}}} L'_1$, and $(\mathbb{F}, M) \xrightarrow{(\mathbb{L},h,h')}_{\alpha_{\mathcal{C}}} (\mathbb{F}', M')$ (respectively $L_1 \xrightarrow{(\mathbb{F}_1,x,y)}_{\alpha_{\mathcal{C}}} L'_1$ and $(\mathbb{F}, M) \xrightarrow{(\mathbb{F}_1,x,y)}_{\alpha_{\mathcal{C}}} (\mathbb{F}', M')$), then there exists $L'_2 \in {}_{\alpha_{\mathcal{C}}}\langle L_2\rangle$ such that $L'_1 \xRightarrow{(\mathbb{F},M)}_{\mathcal{L}} L'_2$.*
- *If $L_1 \circ\!\!\xrightarrow{(\mathbb{F},M)}_{\mathcal{L}} L_2$, $L_1 \xrightarrow{(\mathbb{L},h,h')}_{\alpha_{\mathcal{C}}} L'_1$, and $(\mathbb{F}, M) \xrightarrow{(\mathbb{L},h,h')}_{\alpha_{\mathcal{C}}} (\mathbb{F}', M')$ (respectively $L_1 \xrightarrow{(\mathbb{F}_1,x,y)}_{\alpha_{\mathcal{C}}} L'_1$ and $(\mathbb{F}, M) \xrightarrow{(\mathbb{F}_1,x,y)}_{\alpha_{\mathcal{C}}} (\mathbb{F}', M')$), then there exists $L'_2 \in {}_{\alpha_{\mathcal{C}}}\langle L_2\rangle$ such that $L'_1 \circ\!\!\xrightarrow{(\mathbb{F},M)}_{\mathcal{L}} L'_2$.*

□

It turns out that the analogous property does not hold for linking redexes of the form $(\mathbb{L}, L)$. Recall that linking of two concrete modules is defined when bound labels of the modules (both visible and hidden) are disjoint (see figure 1). By lemma 2.55 if $H_1 =^{\mathcal{C}}_{\alpha} H_2$, then $\mathit{Exports}(H_1) = \mathit{Exports}(H_2)$, so one representative of an $\alpha_{\mathcal{C}}$-equivalence class ${}_{\alpha_{\mathcal{C}}}\langle H \oplus H'\rangle$ satisfies the condition $\mathit{Exports}(H) \cap \mathit{Exports}(H') = \emptyset$ if and only if any other representative does. On the other hand, hidden labels are different for different representatives of the same class, so it may be the case that some representatives of ${}_{\alpha_{\mathcal{C}}}\langle H \oplus H'\rangle$ satisfy the condition $\mathit{Hid}(H) \cap \mathit{Hid}(H') = \emptyset$, and some do not:

Example 2.83 ($\oplus$ Defined on Some But Not All Represenatives of an $\alpha_{\mathcal{C}}$-Equivalence Class). Two linking expressions below are $\alpha_{\mathcal{C}}$-equivalent. The first expression does not have a conflict between the names of hidden labels. In the second expression both modules define a hidden label h, so linking cannot be performed.

$$\begin{array}{lll} [A \mapsto \lambda x.h, h \mapsto 5] \oplus [B \mapsto A \mathbin{@} h', h' \mapsto 7] & \Rightarrow_{\mathcal{L}} & [A \mapsto \lambda x.h, h \mapsto 5, B \mapsto A \mathbin{@} h', h' \mapsto 7] \\ [A \mapsto \lambda x.h, h \mapsto 5] \oplus [B \mapsto A \mathbin{@} h, h \mapsto 7] & \not\Rightarrow_{\mathcal{L}} & (h \text{ defined in both modules}) \end{array}$$

□

However, we are able to show the following weaker property:

Lemma 2.84 (Properties of $\alpha_{\mathcal{C}}$-Equivalent Linking Redexes). *Let $(\mathbb{L}, K)$ and $(\mathbb{L}', K')$ be two linking redexes such that $(\mathbb{L}, K) \longrightarrow_{\alpha_{\mathcal{C}}} (\mathbb{L}', K')$, $L_1 \xrightarrow{(\mathbb{L},K)}_{\mathcal{L}} L_2$, and $L'_1 \xrightarrow{(\mathbb{L},K)}_{\mathcal{L}} L'_2$, then $L_2 =^{\mathcal{C}}_{\alpha} L'_2$ (recall that $L_1 =^{\mathcal{C}}_{\alpha} L'_1$ by lemma 2.79).* □

Recall that all redexes of the form $(\mathbb{L}, K)$ are evaluation redexes, therefore we do not need to consider evaluation and non-evaluation redexes separately in this case.

The following lemma states that the $\alpha_{\mathcal{C}}$-equivalence class of the result of the hiding operation $\mathbb{L}\{D\{\text{hide } v\}\}$ does not depend on the choice of the new hidden name.

Lemma 2.85. *If $h_1, h_2 \notin Hid(H)$, then for all $\mathbb{L}$ we have $\mathbb{L}\{H[v := h_1]\} =_\alpha^{\mathcal{C}} \mathbb{L}\{H[v := h_2]\}$.* □

2.5.4 Canonical Linking Expressions

Recall that by lemma 2.56 for the core module calculus if $\Rightarrow_{\mathcal{C}}$ or $\circ\!\!\rightarrow_{\mathcal{C}}$ is defined on one representative of an $\alpha_{\mathcal{C}}$-equivalence class, then it is defined on all representatives, and all the results of the reduction are in one $\alpha_{\mathcal{C}}$-equivalence class. This significantly simplifies proofs of properties involving $\Rightarrow_{\mathcal{C}}$ and $\circ\!\!\rightarrow_{\mathcal{C}}$. For instance, to check whether ${}_{\alpha_{\mathcal{C}}}\langle D\rangle$ reduces to ${}_{\alpha_{\mathcal{C}}}\langle D'\rangle$, it is enough to take any $D \in {}_{\alpha_{\mathcal{C}}}\langle D\rangle$ and check if there exists $D' \in {}_{\alpha_{\mathcal{C}}}\langle D'\rangle$ such that D reduces to D', so proofs do not involve $\alpha_{\mathcal{C}}$-renaming.

Not surprisingly, the analog of lemma 2.56 holds for $\circ\!\!\rightarrow_{\mathcal{L}}$, as implied by lemma 2.82, since this reduction is just a lifting of $\Rightarrow_{\mathcal{C}}$ (in a non-empty context) and $\circ\!\!\rightarrow_{\mathcal{C}}$ (in any context) to the linking level.

However, as we have seen in example 2.83, $\Rightarrow_{\mathcal{L}}$ does not have this property, because $\oplus$ may be defined on some representatives of ${}_{\alpha_{\mathcal{C}}}\langle H_1 \oplus H_2\rangle$, but not on others. Therefore to check if ${}_{\alpha_{\mathcal{C}}}\langle L\rangle$ reduces to ${}_{\alpha_{\mathcal{C}}}\langle L'\rangle$ it is not enough to check just any $L \in {}_{\alpha_{\mathcal{C}}}\langle L\rangle$: we need to consider arbitrary $\alpha_{\mathcal{C}}$-renamings of L, which may be quite complicated. To minimize dealing with $\alpha_{\mathcal{C}}$-renaming, we introduce the notion of a *canonical linking expression*: a linking expression in which the names of all hidden labels in all modules are distinct. In such expressions there is no conflict between names of hidden components of operands of $\oplus$.

Definition 2.86 (Canonical Linking Expression). A linking expression L is called a *canonical linking expression*, and is denoted as $\hat{L}$, if for any H_1, H_2 such that $L = \mathbb{L}_1\{H_1\} = \mathbb{L}_2\{H_2\}$, where $\mathbb{L}_1 \neq \mathbb{L}_2$, it is true that $Hid(H_1) \cap Hid(H_2) = \emptyset$. □

Definition 2.87. We define the set of hidden labels of L as $Hid(L) = \cup\{Hid(H) \mid L = \mathbb{L}\{H\}\}$. □

The following lemma formalizes some important properties of canonical linking expressions:

Lemma 2.88.

1. *For any linking redex $(\mathbb{L}, K)$ if $(\mathbb{L}, K) \xrightarrow{(\mathbb{F},x,y)}_{\alpha_{\mathcal{C}}} (\mathbb{L}', L)$, then $(\mathbb{L}', L)$ is a linking redex.*
2. *If $(\mathbb{L}, K)$ is a (rename), (hide), or (let) redex and $(\mathbb{L}, K) \xrightarrow{(\mathbb{L}_1,h,h')}_{\alpha_{\mathcal{C}}} (\mathbb{L}', L)$, then $(\mathbb{L}', L)$ is a linking redex.*
3. *For every $L \in \mathbf{Term}_{\mathcal{L}}$ there exists a canonical linking expression $\hat{L} \in \mathbf{Term}_{\mathcal{L}}$ and a sequence of $\alpha_{\mathcal{C}}$-renamings $L = L_1 \xrightarrow{(\mathbb{L}_1,h_{11},h_{12})}_{\alpha_{\mathcal{C}}} L_2 \xrightarrow{(\mathbb{L}_2,h_{21},h_{22})}_{\alpha_{\mathcal{C}}} \dots \xrightarrow{(\mathbb{L}_n,h_{n1},h_{n2})}_{\alpha_{\mathcal{C}}} L_n = \hat{L}$ such that for every linking redex $(\mathbb{L}, K)$ in L for every i such that $1 \leq i \leq n$ $(\mathbb{L}, K) \xrightarrow{(\mathbb{L}_1,h_{11},h_{12})}_{\alpha_{\mathcal{C}}} \dots \xrightarrow{(\mathbb{L}_i,h_{i1},h_{i2})}_{\alpha_{\mathcal{C}}} (\mathbb{L}', L')$ implies that $(\mathbb{L}', L')$ is a linking redex in L_i.*
4. *For every $L_{\alpha_{\mathcal{C}}}$ there exists a canonical $\hat{L} \in L_{\alpha_{\mathcal{C}}}$.*
5. *If $\hat{L} \xrightarrow{(\mathbb{L},h,h')}_{\alpha_{\mathcal{C}}} \hat{L}'$, $(\mathbb{L}, K)$ is a linking redex in $\hat{L}$, and $(\mathbb{L}, K) \xrightarrow{(\mathbb{L}_1,h,h')}_{\alpha_{\mathcal{C}}} (\mathbb{L}', L)$, then $(\mathbb{L}', L)$ is a linking redex in $\hat{L}'$.*
6. *If $L_1 =_\alpha^{\mathcal{C}} L_2$, then there exists L' s.t. $L_1 \xrightarrow{S_1}{}^{*}_{\alpha_{\mathcal{C}}} L' \xrightarrow{S_2}{}^{*}_{\alpha_{\mathcal{C}}} L_2$, where the sequence S_1 consists only of $\alpha_{\mathcal{C}}$-redexes of the form $(\mathbb{F}, x, y)$, and the sequence S_2 has only those of the form $(\mathbb{L}, h, h')$.*
7. *If $\hat{L} =_\alpha^{\mathcal{C}} \hat{L}'$, then there exist canonical expressions $\hat{L}_1, \hat{L}_2, \dots, \hat{L}_n$ such that $\hat{L} = \hat{L}_1 \rightarrow_{\alpha_{\mathcal{C}}} \hat{L}_2 \rightarrow_{\alpha_{\mathcal{C}}} \dots \rightarrow_{\alpha_{\mathcal{C}}} \hat{L}_n = \hat{L}'$.*

8. *If $\hat{L} \in {}_{\alpha_{\mathcal{C}}}\langle L_1 \rangle$ is canonical, then there exist a sequence of $\alpha_{\mathcal{C}}$-renamings S such that $L_1 \xrightarrow{S}{}^{*}_{\alpha_{\mathcal{C}}} \hat{L}$ and for every linking redex $(\mathbb{L}, K)$ and L_2 such that $L_1 \overset{(\mathbb{L},K)}{\Longrightarrow}_{\mathcal{L}} L_2$ there exists $(\mathbb{L}', K')$ such that $(\mathbb{L}, K) \xrightarrow{S}{}^{*}_{\alpha_{\mathcal{C}}} (\mathbb{L}', K')$ and $\hat{L} \overset{(\mathbb{L}',K')}{\Longrightarrow}_{\mathcal{L}} L_2'$.*

9. *If $\hat{L} \in {}_{\alpha_{\mathcal{C}}}\langle L_1 \rangle$ is canonical, then for every sequence of $\alpha_{\mathcal{C}}$-renamings S such that $L_1 \xrightarrow{S}{}^{*}_{\alpha_{\mathcal{C}}} \hat{L}$ and for every linking redex $(\mathbb{F}, M)$ and L_2 such that $L_1 \overset{(\mathbb{F},M)}{\Longrightarrow}_{\mathcal{L}} L_2$ there exists $(\mathbb{F}', M')$ such that $(\mathbb{F}, M) \xrightarrow{S}{}^{*}_{\alpha_{\mathcal{C}}} (\mathbb{F}', M')$ and $\hat{L} \overset{(\mathbb{F}',M')}{\Longrightarrow}_{\mathcal{L}} L_2'$.*

□

Proof.

1. $(\mathbb{L}, K)$ is a (link), (rename), (hide), or (let) redex. A step $\xrightarrow{(\mathbb{F},x,y)}_{\alpha_{\mathcal{C}}}$ renames a λ-bound variable in one of the modules H in the expression $\mathbb{L}\{K\}$. Clearly the result of such renaming is also a linking redex.

2. The rules (rename), (hide), and (let) do not impose any conditions on hidden labels in a module. Therefore (rename), (hide), and (let) redexes are preserved by $\alpha_{\mathcal{C}}$-renaming.

3. If L is canonical, then the claim trivially holds. Otherwise $L = \mathbb{L}_1\{H_1\} = \mathbb{L}_2\{H_2\}$, where $\mathbb{L}_1 \neq \mathbb{L}_2$, and $\mathit{Hid}(H_1) \cap \mathit{Hid}(H_2) = \emptyset$. Let $h \in \mathit{Hid}(H_1) \cap \mathit{Hid}(H_2)$, $h' \notin \mathit{Hid}(L)$, and let $L \xrightarrow{(\mathbb{L}_1,h,h')}_{\alpha_{\mathcal{C}}} L'$, then $L' = \mathbb{L}_1\{H_1'\}$, where $H_1' = H[h := h']$.

   Let $(\mathbb{L}, K)$ be a linking redex in L, and suppose $(\mathbb{L}, K) \xrightarrow{(\mathbb{L}_1,h,h')}_{\alpha_{\mathcal{C}}} (\mathbb{L}', L'')$. If it is a (rename), (hide), or (let) redex, then by part 1 $(\mathbb{L}', L'')$ is a linking redex. Suppose $(\mathbb{L}, K)$ is a (link) redex, then $K = H \oplus H'$, where $\mathit{Hid}(H) \cap \mathit{Hid}(H') = \emptyset$. If $H_1 \neq H$, $H_1 \neq H'$, then the hidden labels of the two modules have not changed, so $(\mathbb{L}', H \oplus H')$ is still a redex in L'. Otherwise H_1 is one of the modules in the linking redex. Without loss of generality suppose $H = H_1$. Then $(\mathbb{L}, H_1 \oplus H') \xrightarrow{(\mathbb{L}_1,h,h')}_{\alpha_{\mathcal{C}}} (\mathbb{L}, H_1' \oplus H')$. Since $h' \notin \mathit{Hid}(L)$, $h' \notin \mathit{Hid}(H')$, and we have $\mathit{Hid}(H_1') \cap \mathit{Hid}(H') = \emptyset$, so $(\mathbb{L}, H_1' \oplus H')$ is a (link) redex in L'.

   If L' is canonical, we are done, otherwise we repeat the procedure described above. Note that $\mathit{Hid}(H_1') \cap \mathit{Hid}(H_2) = \mathit{Hid}(H_1) \cap \mathit{Hid}(H_2) \setminus \{h\}$, therefore we have reduced the number of name conflicts in L. The process stops when all hidden labels in the expression become distinct. The result is a canonical expression, and since the redex $(\mathbb{L}, K)$ has been preserved at every step, its image in the resulting expression is still a redex.

4. By assumption 2.25 there exists $L_1 \in L_{\alpha_{\mathcal{C}}}$. Therefore by the previous result there exists a canonical $\hat{L} \in L_{\alpha_{\mathcal{C}}}$.

5. The claim holds for redexes (rename), (hide), and (let) by part 2. Suppose $(\mathbb{L}, K)$ is a (link) redex, i.e. $K = H_1 \oplus H_2$, where, in particular, $\mathit{Hid}(H_1) \cap \mathit{Hid}(H_2) = \emptyset$. If $\mathbb{L}_1 \neq \mathbb{L}\{\Box \oplus H_2\}$ and $\mathbb{L}_1 \neq \mathbb{L}\{H_1 \oplus \Box\}$, i.e. the $\alpha_{\mathcal{C}}$-renaming does not rename either H_1 or H_2, then $(\mathbb{L}, K) \xrightarrow{(\mathbb{L}_1,h,h')}_{\alpha_{\mathcal{C}}} (\mathbb{L}', K)$, and $(\mathbb{L}', K)$ is a (link) redex. Suppose (without loss of generality) that $\mathbb{L}_1 = \mathbb{L}\{\Box \oplus H_2\}$, i.e. H_1 gets renamed. Let $H_1' = H_1[h := h']$. Since $\hat{L}'$ is canonical, $h' \notin \mathit{Hid}(H_2)$, and therefore $(\mathbb{L}, H_1' \oplus H_2)$ is a (link) redex in $\hat{L}'$.

6. To prove the claim, we first show that $\alpha_{\mathcal{C}}$-renaming with $\alpha_{\mathcal{C}}$-redex of the form $(\mathbb{L}, h, h')$ is independent from $\alpha_{\mathcal{C}}$-renaming with $\alpha_{\mathcal{C}}$-redex of the form $(\mathbb{F}, x, y)$ (see definition 2.2).

   Suppose $L_1 \xrightarrow{(\mathbb{L},h,h')}_{\alpha_{\mathcal{C}}} L_2 \xrightarrow{(\mathbb{F},x,y)}_{\alpha_{\mathcal{C}}} L_3$. Suppose $\mathbb{F} = \mathbb{L}'\{\mathbb{D}\}$ (recall that this parsing is unique). We have the following three cases:

(a) there exists ${}^2\mathbb{L}$ such that $L_1 = {}^2\mathbb{L}\{H_1, H_2\}$, $\mathbb{L} = {}^2\mathbb{L}\{\Box, H_2\}$, $H_1 \xrightarrow{(h,h')}_{\alpha_{\mathcal{C}}} H_1'$, and $\mathbb{L}' = {}^2\mathbb{L}\{H_1', \Box\}$. Let H_2' be such that $H_2 \xrightarrow{(\mathbb{D},x,y)}_{\alpha_{\mathcal{C}}} H_2'$, and let $\mathbb{F} = {}^2\mathbb{L}\{H_1, \Box\}$, $\mathbb{L}' = {}^2\mathbb{L}\{\Box, H_2'\}$. Then $L_1 = {}^2\mathbb{L}\{H_1, H_2\} \xrightarrow{(\mathbb{F}',x,y)}_{\alpha_{\mathcal{C}}} {}^2\mathbb{L}\{H_1, H_2'\} \xrightarrow{(\mathbb{L}',h,h')}_{\alpha_{\mathcal{C}}} {}^2\mathbb{L}\{H_1', H_2'\} = L_3$.

(b) there exists ${}^2\mathbb{L}$ such that $L_1 = {}^2\mathbb{L}\{H_2, H_1\}$, $\mathbb{L} = {}^2\mathbb{L}\{H_2, \Box\}$, $H_1 \xrightarrow{(h,h')}_{\alpha_{\mathcal{C}}} H_1'$, and $\mathbb{L}' = {}^2\mathbb{L}\{\Box, H_1'\}$. The case is completely analogous to the previous one.

(c) $L_1 = \mathbb{L}\{H_1\}$, $H_1 \xrightarrow{(h,h')}_{\alpha_{\mathcal{C}}} H_2$, and $\mathbb{L}' = \mathbb{L}$. Suppose $H_1 = [l_i \overset{n}{\underset{i=1}{\mapsto}} M_i]$, then $H_1' = [l_i' \overset{n}{\underset{i=1}{\mapsto}} M_i']$, where $M_i' = M_i[h := h']$, and $l_i' = h'$ if $l_i = h$, otherwise $l_i' = l_i$. Suppose $\mathbb{D} = \mathbb{M}\{l' \mapsto \mathbb{C}\}$, where $\mathbb{M} = [l_i' \overset{n-1}{\underset{i=1}{\mapsto}} M_i', \Box]$. We have $\mathbb{D}\{\lambda x.N\} \xrightarrow{(\mathbb{D},x,y)}_{\alpha_{\mathcal{C}}} \mathbb{D}\{\lambda y.N[x := y]\}$.

By the symmetry of $\rightarrow_{\alpha_{\mathcal{C}}}$ $H_1 \xrightarrow{(h,h')}_{\alpha_{\mathcal{C}}} H_2$ implies $H_2 \xrightarrow{(h',h)}_{\alpha_{\mathcal{C}}} H_1$. In particular, let $l = l'$ if $l' \neq h'$, and $l = h$ otherwise, and let M be the component bound to l in H_1. Then $M = (\mathbb{C}\{\lambda x.N\})[h' := h]$, and by definition of substitution $M = \mathbb{C}_1\{N_1\}$, where $\mathbb{C}_1$ is obtained from $\mathbb{C}$ by replacing all occurrences of h' by h, and $N_1 = N[h' := h]$. Let $\mathbb{D}_1 = [l_i \overset{n-1}{\underset{i=1}{\mapsto}} M_i, l \mapsto \mathbb{C}_1]$, then $H_1 \xrightarrow{(\mathbb{D}_1,x,y)}_{\alpha_{\mathcal{C}}} [l_i \overset{n-1}{\underset{i=1}{\mapsto}} M_i, l \mapsto \mathbb{D}_1\{\lambda y.N_1[x := y]\}] = H_1'$. Then $H_1' \xrightarrow{(h,h')}_{\alpha_{\mathcal{C}}} [l_i' \overset{n-1}{\underset{i=1}{\mapsto}} M_i', l' \mapsto \mathbb{C}_1\{\lambda y.N_1[x := y]\}[h := h']] = H_2'$. We observe the following: $\mathbb{C}_1\{\lambda y.N_1[x := y]\}[h := h'] = \mathbb{C}\{\lambda y.(N_1[x := y])[h := h']\} = \mathbb{C}\{\lambda y.(N_1[h := h'])[x := y]\} = \mathbb{C}\{\lambda y.N[x := y]\}$ (we have used the fact that $\mathbb{C}$ is the result of replacing all occurrences of h by h'). Then $H_2' = [l_i' \overset{n-1}{\underset{i=1}{\mapsto}} M_i', l' \mapsto \mathbb{C}\{\lambda y.N[x := y]\}] = H_2$.

We have shown that $\alpha_{\mathcal{C}}$-renaming with $\alpha_{\mathcal{C}}$-redex of the form $(\mathbb{L}, h, h')$ is independent from $\alpha_{\mathcal{C}}$-renaming with $\alpha_{\mathcal{C}}$-redex of the form $(\mathbb{F}, x, y)$. Both relations are symmetric, and the claim follows from lemma 2.3.

7. By the previous result there exists L s.t. $\hat{L} \xrightarrow{S_1}{}^{*}_{\alpha_{\mathcal{C}}} L \xrightarrow{S_2}{}^{*}_{\alpha_{\mathcal{C}}} \hat{L}'$, where S_1 contains only $\alpha_{\mathcal{C}}$-redexes of the form $(\mathbb{F}, x, y)$, and S_2 only those of the form $(\mathbb{L}, h, h')$. Since $\hat{L}$ is canonical, and renaming λ-bound variables does not change hidden labels of any H contained in $\hat{L}$ (as follows from part 1 of lemma 2.35), L is canonical. Let $\hat{L}_0 = L$.

   It is left to show that there exists a sequence of $\alpha_{\mathcal{C}}$-renamings of hidden labels from $\hat{L}_0$ to $\hat{L}'$. Suppose $\mathit{Hid}(\hat{L}_0) = \{h_1, \ldots, h_n\}$, and $\mathit{Hid}(\hat{L}') = \{h_1', \ldots, h_n'\}$. Note that the number of elements in the two sets is the same, since $\hat{L}_0 =_\alpha^{\mathcal{C}} \hat{L}'$, and both expressions are canonical. In general it is possible that $\mathit{Hid}(\hat{L}_0) \cap \mathit{Hid}(\hat{L}') \neq \emptyset$, so renaming hiddens of $\hat{L}_0$ directly to those of $\hat{L}'$ may lead to a non-canonical expression among intermediate results. To avoid this problem, we choose n hiddens $h_1'', \ldots, h_n''$ s.t. $h_i'' \notin \mathit{Hid}(\hat{L}_0) \cup \mathit{Hid}(\hat{L}')$ for all $1 \leq i \leq n$. Then we can construct a sequence $\hat{L}_0 \xrightarrow{(\mathbb{L}_1,h_1,h_1'')}_{\alpha_{\mathcal{C}}} L_1 \ldots \xrightarrow{(\mathbb{L}_n,h_n,h_n'')}_{\alpha_{\mathcal{C}}} L_n$. Clearly L_i is canonical for all $1 \leq i \leq n$. Similarly, there exists a sequence $L_n \xrightarrow{(\mathbb{L}_1',h_1'',h_1')}_{\alpha_{\mathcal{C}}} L_1' \ldots \xrightarrow{(\mathbb{L}_n',h_n'',h_n')}_{\alpha_{\mathcal{C}}} L_n' = \hat{L}'$. Note that in constructing the first sequence we were using the names $h_1'', \ldots, h_n''$ as just a set of fresh names, but in constructing the other sequence we need to rename each of the h_i'' to a particular corresponding h_i' in $\mathit{Hid}(\hat{L}')$. Each of L_i' is canonical, and, combining the three sequences, we get the desired sequence: $\hat{L} \xrightarrow{S_1}{}^{*}_{\alpha_{\mathcal{C}}} \hat{L}_0 \xrightarrow{(\mathbb{L}_1,h_1,h_1'')}_{\alpha_{\mathcal{C}}} L_1 \ldots \xrightarrow{(\mathbb{L}_n,h_n,h_n'')}_{\alpha_{\mathcal{C}}} L_n \xrightarrow{(\mathbb{L}_1',h_1'',h_1')}_{\alpha_{\mathcal{C}}} L_1' \ldots \xrightarrow{(\mathbb{L}_n',h_n'',h_n')}_{\alpha_{\mathcal{C}}} \hat{L}'$.

8. By part 3 there exists $\hat{L}'$ such that $L_1 \xrightarrow{S_1}{}^{*}_{\alpha_{\mathcal{C}}} \hat{L}'$, and for an arbitrary linking redex $(\mathbb{L}, K)$ there exists a linking redex $(\mathbb{L}', K')$ such that $(\mathbb{L}, K) \xrightarrow{S_1}{}^{*}_{\alpha_{\mathcal{C}}} (\mathbb{L}', K')$. By part 7 there exists a sequence $\hat{L}' \rightarrow_{\alpha_{\mathcal{C}}} \hat{L}_1 \rightarrow_{\alpha_{\mathcal{C}}} \ldots \rightarrow_{\alpha_{\mathcal{C}}} \hat{L}_n = \hat{L}$, and by parts 1 and 5 each of the steps $\hat{L}_i \rightarrow_{\alpha_{\mathcal{C}}} \hat{L}_{i+1}$ preserves linking redexes, i.e. $(\mathbb{L}', K') \xrightarrow{S_2}{}^{*}_{\alpha_{\mathcal{C}}} (\mathbb{L}'', K'')$, where S_2 is the sequence $\hat{L}' \rightarrow_{\alpha_{\mathcal{C}}} \hat{L}_1 \rightarrow_{\alpha_{\mathcal{C}}} \ldots \rightarrow_{\alpha_{\mathcal{C}}} \hat{L}$ above, and $(\mathbb{L}, K) \xrightarrow{S_1;S_2}{}^{*}_{\alpha_{\mathcal{C}}} (\mathbb{L}'', K'')$. By lemma 2.84 $\hat{L} \xRightarrow{((\mathbb{L}'',K''))}_{\mathcal{L}} L_2' =_\alpha^{\mathcal{C}} L_2$.

9. Let S be a sequence such that $L_1 \xrightarrow{S}^*_{\alpha_\mathcal{C}} \hat{L}$. Suppose $(\mathbb{F}, M) \xrightarrow{S}^*_{\alpha_\mathcal{C}} (\mathbb{F}', M')$. By lemma 2.82 there exists $L'_2 \in {}_{\alpha_\mathcal{C}}\langle L_2 \rangle$ such that $\hat{L} \xRightarrow{(\mathbb{F}',M')}_\mathcal{L} L'_2$, where $(\mathbb{F}, M) \xrightarrow{S}^*_{\alpha_\mathcal{C}} (\mathbb{F}', M')$.

□

The last two claims of Lemma 2.88 are the key properties of canonical linking expressions. They state that for any reduction on a representative of an $\alpha_\mathcal{C}$-equivalence class there is a corresponding reduction on a canonical representative of the class.

We would like to work with just canonical representatives of classes and not to deal with $\alpha_\mathcal{C}$-renaming at all. A property that would have allowed us to do so is "if $L_1 \Rightarrow_\mathcal{L} L_2$, then there exist canonical expressions $\hat{L}_1 \in {}_{\alpha_\mathcal{C}}\langle L_1 \rangle$ and $\hat{L}_2 \in {}_{\alpha_\mathcal{C}}\langle L_2 \rangle$ such that $\hat{L}_1 \Rightarrow_\mathcal{L} \hat{L}_2$. Unfortunately, this property does not hold in the linking calculus because of the rule (let):

$$\begin{array}{ll}\textbf{let } I = [A \mapsto 2, h \mapsto 3] \textbf{ in } I[A \overset{\text{ren}}{\leftarrow} B] \oplus I & \Rightarrow_\mathcal{L} \\ [A \mapsto 2, h \mapsto 3][A \overset{\text{ren}}{\leftarrow} B] \oplus [A \mapsto 2, h \mapsto 3] & \Rightarrow_\mathcal{L} \\ [B \mapsto 2, h \mapsto 3] \oplus [A \mapsto 2, h \mapsto 3]. & \end{array}$$

Despite the fact that the original linking expression is canonical, the result of the reduction is not, due to duplication of the module bound to I. After the renaming $[A \overset{\text{ren}}{\leftarrow} B]$ it is necessary to perform $\alpha_\mathcal{C}$-renaming to enable linking. This example shows that in some cases $\alpha_\mathcal{C}$-renaming is unavoidable.

However, (let) is the only problematic case: for the rules (mod-ev), (link), rename, and (mod-nev) it is the case that the result of reducing a canonical representative is itself canonical. For the rule (hide) it depends on the new hidden name: if this name does not appear elsewhere in the linking expression, then the result is canonical. The following definition adds such a restriction to (hide). Note that by lemma 2.85 the equivalence class of the result does not depend on the choice of h.

Definition 2.89. We say that a hiding operation $L = \mathbb{L}\{H\{\text{hide } v\}\} \Rightarrow_\mathcal{L} \mathbb{L}\{H[h := v]\}$ is *canonical* if $h \notin \mathit{Hid}(L)$. □

Lemma 2.90. *If $L \xRightarrow{(\mathbb{L},H\{hide\ v\})}_\mathcal{L} L'$, then there exists $L'_1 \in {}_{\alpha_\mathcal{C}}\langle L' \rangle$ such that $L \xRightarrow{(\mathbb{L},H\{hide\ v\})}_\mathcal{L} L'_1$ by canonical hiding.* □

Proof. Let the result of the given reduction be $\mathbb{L}\{H[v := h]\}$. If $h \notin \mathit{Hid}(L)$, then the given reduction is itself canonical, otherwise let h' be such that $h' \notin \mathit{Hid}(L)$, and let $L'_1 = \mathbb{L}\{H[v := h']\}$. Then $L \xRightarrow{(\mathbb{L},H\{\text{hide } v\})}_\mathcal{L} L'_1$, and $L_1 \xrightarrow{(\mathbb{L},h,h')}_{\alpha_\mathcal{C}} L'_1$, therefore $L'_1 \in {}_{\alpha_\mathcal{C}}\langle L' \rangle$. □

The above observations allow us to introduce a reduction on canonical representatives that performs $\alpha_\mathcal{C}$-renaming only in the case of duplication of modules by the rule (let).

Definition 2.91 (Canonical Reductions). Let $\hat{L}_1$ be a canonical expression. We say that $\hat{L}_1$ reduces to L_2 by a *canonical evaluation* with the redex $(\mathbb{L}, K)$ (respectively with the redex $(\mathbb{F}, M)$, and write $\hat{L}_1 \underset{\sim}{\xRightarrow{(\mathbb{L},K)}}_\mathcal{L} L_2$ (respectively $\hat{L}_1 \underset{\sim}{\xRightarrow{(\mathbb{F},M)}}_\mathcal{L} L_2$), if one of the following takes place:

- $\hat{L}_1 \xRightarrow{(\mathbb{L},K)}_\mathcal{L} \hat{L}_2$ by rules (link) or (rename),
- $\hat{L}_1 \xRightarrow{(\mathbb{L},K)}_\mathcal{L} \hat{L}_2$ by canonical hiding (see definition 2.89),
- $\hat{L}_1 \xRightarrow{(\mathbb{L},K)}_\mathcal{L} L'_2 \rightarrow^*_{\alpha_\mathcal{C}} L_2$, and L_2 is canonical.
- $\hat{L}_1 \xRightarrow{(\mathbb{F},M)}_\mathcal{L} L_2$ by the rule (mod-ev).

We say that $\hat{L}_1$ reduces to L_2 by a *canonical non-evaluation* (and write $\hat{L}_1 \underset{\sim}{\overset{(\mathbb{F},M)}{\circ\!\!\longrightarrow}}_{\mathcal{L}} L_2$) if $\hat{L}_1 \overset{(\mathbb{F},M)}{\circ\!\!\longrightarrow}_{\mathcal{L}} L_2$ by the rule (mod-nev).

We say that $\hat{L}_1$ reduces to L_2 by a *canonical (calculus) reduction*, denoted $\hat{L}_1 \underset{\sim}{\overset{(\mathbb{L},K)}{\longrightarrow}}_{\mathcal{L}} L_2$ or $\hat{L}_1 \underset{\sim}{\overset{(\mathbb{F},M)}{\longrightarrow}}_{\mathcal{L}} L_2$, depending on the redex, if $\hat{L}_1$ reduces to L_2 by a canonical evaluation or a canonical non-evaluation. □

Lemma 2.92 (The Result of a Canonical Reduction is Canonical). *If $\hat{L}_1 \underset{\sim}{\longrightarrow}_{\mathcal{L}} L_2$, then L_2 is canonical.* □

The following lemma states an important property of canonical reduction, namely that any reduction from an element of one $\alpha_{\mathcal{C}}$-equivalence class to an element of another one has a corresponding reduction for all canonical representatives of the first $\alpha_{\mathcal{C}}$-equivalence class.

Lemma 2.93. *If $L_1 \overset{(\mathbb{L},K)}{\Longrightarrow}_{\mathcal{L}} L_2$ (respectively $L_1 \overset{(\mathbb{F},M)}{\Longrightarrow}_{\mathcal{L}} L_2$ or $L_1 \overset{(\mathbb{F},M)}{\circ\!\!\longrightarrow}_{\mathcal{L}} L_2$), then for every canonical $\hat{L}_1 \in {}_{\alpha_{\mathcal{C}}}\langle L_1\rangle$ there exists a canonical $\hat{L}_2 \in {}_{\alpha_{\mathcal{C}}}\langle L_2\rangle$ such that $\hat{L}_1 \underset{\sim}{\overset{(\mathbb{L}',K')}{\Longrightarrow}}_{\mathcal{L}} \hat{L}_2$ (respectively $\hat{L}_1 \underset{\sim}{\overset{(\mathbb{F}',M')}{\Longrightarrow}}_{\mathcal{L}} \hat{L}_2$ or $\hat{L}_1 \underset{\sim}{\overset{(\mathbb{F}',M')}{\circ\!\!\longrightarrow}}_{\mathcal{L}} \hat{L}_2$) and $(\mathbb{L},K) \overset{S}{\longrightarrow}{}^{*}_{\alpha_{\mathcal{C}}} (\mathbb{L}',K')$ (respectively $(\mathbb{F},M) \overset{S}{\longrightarrow}{}^{*}_{\alpha_{\mathcal{C}}} (\mathbb{F}',M')$), where S is such that $L_1 \overset{S}{\longrightarrow}{}^{*}_{\alpha_{\mathcal{C}}} \hat{L}_1$.* □

Proof. Suppose $L_1 \overset{(\mathbb{L},K)}{\Longrightarrow}_{\mathcal{L}} L_2$. By part 8 of lemma 2.88 for every canonical $\hat{L}_1 \in {}_{\alpha_{\mathcal{C}}}\langle L_1\rangle$ there exist a sequence S and $L_2' \in {}_{\alpha_{\mathcal{C}}}\langle L_2\rangle$ such that $L_1 \overset{S}{\longrightarrow}{}^{*}_{\alpha_{\mathcal{C}}} \hat{L}_1$ and $\hat{L}_1 \overset{(\mathbb{L}',K')}{\Longrightarrow}_{\mathcal{L}} L_2'$, where $(\mathbb{L},K) \overset{S}{\longrightarrow}{}^{*}_{\alpha_{\mathcal{C}}} (\mathbb{L}',K')$. If the rule of the reduction is (link) or (rename), then by definition 2.91 the reduction is canonical. If the rule is (hide), then by lemma 2.90 there exists $L_2'' \in {}_{\alpha_{\mathcal{C}}}\langle L_2\rangle$ such that $\hat{L} \overset{(\mathbb{L}',K')}{\Longrightarrow}_{\mathcal{L}} L_2''$ is a canonical hiding, and the reduction is canonical. If the rule is (let), then by part 4 of lemma 2.88 there exists a canonical $L_2'' \in {}_{\alpha_{\mathcal{C}}}\langle L_2\rangle$, and $L_2' \longrightarrow^{*}_{\alpha_{\mathcal{C}}} L_2''$, since $=_{\alpha}^{\mathcal{C}}$ is the same relation as $\longrightarrow^{*}_{\alpha_{\mathcal{C}}}$. The reduction is also canonical. Hence in all four cases of the reduction rule there exists $L_2'' \in {}_{\alpha_{\mathcal{C}}}\langle L_2\rangle$ such that $\hat{L} \underset{\sim}{\overset{(\mathbb{L}',K')}{\Longrightarrow}}_{\mathcal{L}} L_2''$ (in the first two cases $L_2'' = L_2'$), and by lemma 2.92 L_2'' is canonical.

The cases $L_1 \overset{(\mathbb{F},M)}{\Longrightarrow}_{\mathcal{L}} L_2$ and $L_1 \overset{(\mathbb{F},M)}{\circ\!\!\longrightarrow}_{\mathcal{L}} L_2$ are similar. We use part 9 of lemma 2.88 and part 3 of lemma 2.82, respectively, for these two reductions in place of part 8 of lemma 2.88 in the above case. Then the claim follows directly from definition of a canonical reduction and lemma 2.92. □

2.5.5 Renaming let-Bound Module Identifiers

Similarly to identifying terms in $\mathcal{T}$ up to renaming of λ-bound variables, we identify linking expressions up to renaming of **let**-bound module identifiers. For instance, the following two linking expressions are considered to be α-equivalent:

$$\begin{aligned} L_1 &= \textbf{let } I_1 = D_1 \oplus I_2 \textbf{ in let } I_3 = D_2 \textbf{ in } I \oplus I_3[v \overset{\text{ren}}{\longleftarrow} v'], \\ L_2 &= \textbf{let } I_3 = D_1 \oplus I_2 \textbf{ in let } I_1 = D_2 \textbf{ in } I_3 \oplus I_1[v \overset{\text{ren}}{\longleftarrow} v']. \end{aligned}$$

Note that I_2 appears free in the expressions, and therefore cannot be renamed.

The α-renaming of **let**-bound module identifiers illustrated above is very similar to $\alpha_{\mathcal{T}}$-renaming of λ-bound variables in $\mathcal{T}$. The similarity becomes clear if we think of a **let**-expression **let** $I = L_1$ **in** L_2 as synonymous to a λ-expression $\lambda I.L_2 \mathbin{@} L_1$. Even though the operators in the linking calculus $\mathcal{L}$ are different from those in $\mathcal{T}$, the analogy holds as far as the scope rules are concerned. Most of the results shown for $\alpha_{\mathcal{T}}$-renaming hold for the renaming of module identifiers (called α_I-renaming below), and the proofs of these results are also very similar. Therefore in many cases we just refer to the respective proofs of $\mathcal{T}$ rather than spell out the proofs here. To observe the similarity of the two renamings, compare the following definition with definition 2.20 of $\alpha_{\mathcal{T}}$-renaming.

Definition 2.94 (α_I-Renaming in $\mathcal{L}$). We say that **let** $I = L_1$ **in** L_2 reduces to **let** $I' = L_1$ **in** L_2' by *elementary α_I-renaming*, and write **let** $I = L_1$ **in** $L_2 \rightsquigarrow_{\alpha_I}$ **let** $I' = L_1$ **in** L_2', if $I \neq I'$, $L_2' = L_2[I := I']$, and the following two conditions hold:

1. $I' \notin FMI(L_2)$,
2. $L_2 = \mathbb{L}\{\textbf{let } I' = L_3 \textbf{ in } L_4\}$ implies that either $I \notin FMI(L_4)$ or $\mathbb{L} = \mathbb{L}_1\{\textbf{let } I = L_5 \textbf{ in } \mathbb{L}_2\}$ for some $\mathbb{L}_1, \mathbb{L}_2$, and L_5.

We say that L_1 reduces to L_2 by a *one-step α_I-renaming* with α_I-redex $(\mathbb{L}, I, I')$, and write $L_1 \xrightarrow{(\mathbb{L},I,I')}_{\alpha_I} L_2$, if $L_1 = \mathbb{L}\{\textbf{let } I = L_2 \textbf{ in } L_3\}$, $L_2 = \mathbb{L}\{\textbf{let } I' = L_2 \textbf{ in } L_3'\}$, and **let** $I = L_2$ **in** $L_3 \rightsquigarrow_{\alpha_I}$ **let** $I' = L_2$ **in** L_3'.

As usual, $\xrightarrow{S}{}^{*}_{\alpha_I}$, where S is a sequence of α_I-redexes, denotes a sequence of α_I-renaming steps reducing α_I-redexes in S left-to-right. □

As for $\rightsquigarrow_{\alpha_\mathcal{T}}$, the first condition for $\rightsquigarrow_{\alpha_I}$ guarantees that there is no identifier capture, and the second – that exactly one identifier gets renamed by $\rightsquigarrow_{\alpha_I}$. We omit the proof of the latter fact, which is analogous to that of lemma 2.21. We can also show that $\rightsquigarrow_{\alpha_I}$, and therefore $\longrightarrow_{\alpha_I}$, is symmetric. Again, the proof is analogous to the respective proof for $\rightsquigarrow_{\alpha_\mathcal{T}}$ (see lemma 2.22) and is omitted. Since $\longrightarrow_{\alpha_I}$ is symmetric, so is $\longrightarrow^*_{\alpha_I}$, and the latter equals to $=^I_\alpha$ by lemma 2.1.

We define α_I-equivalence classes of linking expressions in the usual way:

Definition 2.95 (α_I-Equivalence Classes of Terms in $\mathcal{L}$). If $L \in \textbf{Term}_\mathcal{L}$, then its α_I-equivalence class, denoted ${}_{\alpha_I}\langle L\rangle$, is defined as ${}_{\alpha_I}\langle L\rangle = \{L' \in \textbf{Term}_\mathcal{L} \mid L =^I_\alpha L'\}$. We use L_{α_I} as a meta-variable for α_I-equivalence classes. □

We define α_I-equivalence classes of subterm occurrences (so that we can consider α_I-equivalence classes of linking redexes) using the same technique as we used for $\mathcal{T}$ and $\mathcal{C}$, i.e. marking a subterm occurrence before α-renaming and finding the marked occurrence in the α-renamed term. This technique is more cumbersome than defining α-equivalence classes of contexts and filling them with α-equivalence classes of terms, as we did for $\alpha_\mathcal{C}$-renaming of linking expressions in section 2.5.3. However, we cannot meaningfully define α_I-equivalence classes of linking contexts, since some representatives of such α_I-equivalence classes may capture a free identifier, whereas others may not (this property of α_I-renaming also is common with $\alpha_\mathcal{T}$-renaming). As an example, consider:

Example 2.96. Let $\mathbb{L} = \textbf{let } I = [A \mapsto 4] \textbf{ in } \square$. If we fill $\mathbb{L}$ with the module identifier I', no capture occurs. However, if we allow α_I-renaming of $\mathbb{L}$ to $\mathbb{L}' = \textbf{let } I' = [A \mapsto 4] \textbf{ in } \square$, then I' is captured. □

Since the calculus of marked linking expression is analogous to the calculus of marked terms $\underline{\mathcal{T}}$, we only sketch the construction here without giving all the details, which can be found in section 2.3.3.

Let $\underline{\mathcal{L}}$ denote calculus $\mathcal{L}$ augmented with the following marked module operations: $\widetilde{\oplus}$, $\widetilde{[v \overset{\text{ren}}{\leftarrow} v']}$, $\widetilde{\{\text{hide } v\}}$, $\widetilde{\textbf{let}}$, and also with module bindings of the form $l \mapsto \widetilde{M}$, where $\widetilde{M} \in \textbf{Term}_{\underline{\mathcal{T}}}$ (the latter are used to mark (mod-ev) and (mod-nev) redexes). We use $\widetilde{L}, \widetilde{\mathbb{L}}$, and $\widetilde{\mathbb{F}}$ to denote terms and contexts in $\underline{\mathcal{L}}$. We also use $(\widetilde{\mathbb{L}}, \widetilde{L})$ and $(\widetilde{\mathbb{F}}, \widetilde{M})$ to denote the two kinds of subterm occurrences in $\underline{\mathcal{L}}$. Similarly to the case of $\mathcal{T}$, we say that a subterm occurrence $(\widetilde{\mathbb{L}}, \widetilde{L})$ or $(\widetilde{\mathbb{F}}, \widetilde{M})$ is *marked* if $\widetilde{L}$ is of the form $L_1 \widetilde{\oplus} L_2$, $L\widetilde{[v \overset{\text{ren}}{\leftarrow} v']}$, $L\widetilde{\{\text{hide } v\}}$, or $\widetilde{\textbf{let}} I = L_1 \textbf{in} L_2$ for the former subterm occurrence, and $\widetilde{M}$ is marked (as defined in section 2.3.3) for the latter. We extend the definition of a substitution (see section 2.5.1) to the marked linking expressions so that it preserves the markings, and define the mark erasure that maps marked terms, contexts, and subterm occurrences to unmarked ones by replacing all marked operations and labels by their unmarked analogs. The mark erasure is denoted by $|\ |$.

As for $\alpha_\mathcal{T}$-renaming in the term calculus, we define α_I-renaming of subterm occurrences first in the calculus of marked linking expressions, and then in $\mathcal{L}$. The definitions below consider both kinds of subterm occurrences.

Definition 2.97 (α_I-Renaming of $\underline{\mathcal{L}}$ Subterm Occurrences). Let $(\widetilde{\mathbb{L}}, \widetilde{L})$ be a marked subterm occurrence of $\widetilde{\mathbb{L}}\{\widetilde{L}\} \in \mathbf{Term}_{\underline{\mathcal{L}}}$ (respectively let $(\widetilde{\mathbb{F}}, \widetilde{M})$ be a marked subterm of $\widetilde{\mathbb{F}}\{\widetilde{M}\} \in \mathbf{Term}_{\underline{\mathcal{L}}}$), where $\widetilde{\mathbb{L}}\{\widetilde{L}\}$ (respectively $\widetilde{\mathbb{F}}\{\widetilde{M}\}$) does not contain any other marked subterm occurrences. We say that $(\widetilde{\mathbb{L}}, \widetilde{L})$ reduces to $(\widetilde{\mathbb{L}}', \widetilde{L}')$ (respectively $(\widetilde{\mathbb{F}}, \widetilde{M})$ reduces to $(\widetilde{\mathbb{F}}', \widetilde{M}')$) by a *one-step α_I-renaming with the α_I-redex* $(\widetilde{\mathbb{L}}_1, I, I')$, written as $(\widetilde{\mathbb{L}}, \widetilde{L}) \xrightarrow{(\widetilde{\mathbb{L}}_1, I, I')}_{\alpha_I} (\widetilde{\mathbb{L}}', \widetilde{L}')$ (respectively $(\widetilde{\mathbb{F}}, \widetilde{M}) \xrightarrow{(\widetilde{\mathbb{L}}_1, I, I')}_{\alpha_I} (\widetilde{\mathbb{F}}', \widetilde{M}')$), iff $\widetilde{\mathbb{L}}\{\widetilde{L}\} \xrightarrow{(\widetilde{\mathbb{L}}_1, I, I')}_{\alpha_I} \widetilde{\mathbb{L}}'\{\widetilde{L}'\}$ and $\widetilde{\mathbb{L}}'\{\widetilde{L}'\}$ is marked (respectively $\widetilde{\mathbb{F}}\{\widetilde{M}\} \xrightarrow{(\widetilde{\mathbb{L}}_1, I, I')}_{\alpha_I} \widetilde{\mathbb{F}}'\{\widetilde{M}'\}$ and $\widetilde{\mathbb{F}}'\{\widetilde{M}'\}$ is marked). $\square$

Definition 2.98 (α_I-Renaming of $\mathcal{L}$ Subterm Occurrences). We say that $(\mathbb{L}, L)$ reduces to $(\mathbb{L}', L')$ (respectively $(\mathbb{F}, M)$ reduces to $(\mathbb{F}', M')$) by a *one-step α_I-renaming with the α_I-redex* $(\mathbb{L}_1, I, I')$, written as $(\mathbb{L}, L) \xrightarrow{(\mathbb{L}_1, I, I')}_{\alpha_I} (\mathbb{L}', L')$ (respectively $(\mathbb{F}, M) \xrightarrow{(\mathbb{L}_1, I, I')}_{\alpha_I} (\mathbb{F}', M')$), iff there exist $\underline{\mathcal{L}}$ subterm occurrences $(\widetilde{\mathbb{L}}, \widetilde{L})$ and $(\widetilde{\mathbb{L}}', \widetilde{L}')$ (respectively $(\widetilde{\mathbb{F}}, \widetilde{M})$ and $(\widetilde{\mathbb{F}}', \widetilde{M}')$) and a context $\widetilde{\mathbb{L}}_1$ such that $(\widetilde{\mathbb{L}}, \widetilde{L}) \xrightarrow{(\widetilde{\mathbb{L}}_1, I, I')}_{\alpha_I} (\widetilde{\mathbb{L}}', \widetilde{L}')$, $|(\widetilde{\mathbb{L}}, \widetilde{L})| = (\mathbb{L}, L)$, $|(\widetilde{\mathbb{L}}', \widetilde{L}')| = (\mathbb{L}', L')$ (respectively $(\widetilde{\mathbb{F}}, \widetilde{M}) \xrightarrow{(\widetilde{\mathbb{L}}_1, I, I')}_{\alpha_I} (\widetilde{\mathbb{F}}', \widetilde{M}')$, $|(\widetilde{\mathbb{F}}, \widetilde{M})| = (\mathbb{F}, M)$, $|(\widetilde{\mathbb{F}}', \widetilde{M}')| = (\mathbb{F}', M')$), and $|\widetilde{\mathbb{L}}_1| = \mathbb{L}_1$. $\square$

We define α_I-equivalence classes of subterm occurrences as usual, and use a meta-variable $Sub_{\alpha_I}^{\mathcal{L},\mathcal{L}}$ to range over α_I-equivalence classes of subterm occurrences of the form $(\mathbb{L}, L)$, and a meta-variable $Sub_{\alpha_I}^{\mathcal{T},\mathcal{L}}$ to range over α_I-equivalence classes of subterm occurrences of the form $(\mathbb{F}, M)$.

We can show that α_I-renaming of subterm occurrences preserves distinctness of subterm occurrences, as well as both evaluation and non-evaluation redexes. The results and the proofs are analogous to those of lemmas 2.33 and 2.34 in section 2.3.3.

We show the following important properties of α_I-renaming:

Lemma 2.99 (Properties of α_I-Renaming). *Suppose $L_1 \xrightarrow{S}{}^{*}_{\alpha_I} L_1'$, then:*

1. $FMI(L_1) = FMI(L_1')$,
2. $Hid(L_1) = Hid(L_1')$,
3. *L_1 is canonical iff L_1' is,*
4. *If $L_1 \xRightarrow{(\mathbb{L}, K)}_{\mathcal{L}} L_2$, then there exists $L_2' \in {}_{\alpha_I}\langle L_2 \rangle$ such that $L_1' \xRightarrow{(\mathbb{L}', K')}_{\mathcal{L}} L_2'$, where $(\mathbb{L}, K) \xrightarrow{S}{}^{*}_{\alpha_I} (\mathbb{L}', K')$,*
5. *If $L_1 \xRightarrow{(\mathbb{F}, M)}_{\mathcal{L}} L_2$, then there exists $L_2' \in {}_{\alpha_I}\langle L_2 \rangle$ such that $L_1' \xRightarrow{(\mathbb{F}', M')}_{\mathcal{L}} L_2'$, where $(\mathbb{F}, M) \xrightarrow{S}{}^{*}_{\alpha_I} (\mathbb{F}', M')$,*
6. *If $L_1 \circ\!\xrightarrow{(\mathbb{F}, M)}_{\mathcal{L}} L_2$, then there exists $L_2' \in {}_{\alpha_I}\langle L_2 \rangle$ such that $L_1' \circ\!\xrightarrow{(\mathbb{F}', M')}_{\mathcal{L}} L_2'$, where $(\mathbb{F}, M) \xrightarrow{S}{}^{*}_{\alpha_I} (\mathbb{F}', M')$,*
7. *If $L_1 \underset{\sim}{\xRightarrow{(\mathbb{L}, K)}}_{\mathcal{L}} L_2$, then there exists $L_2' \in {}_{\alpha_I}\langle L_2 \rangle$ such that $L_1' \underset{\sim}{\xRightarrow{(\mathbb{L}', K')}}_{\mathcal{L}} L_2'$, where $(\mathbb{L}, K) \xrightarrow{S}{}^{*}_{\alpha_I} (\mathbb{L}', K')$,*
8. *If $L_1 \underset{\sim}{\xRightarrow{(\mathbb{F}, M)}}_{\mathcal{L}} L_2$, then there exists $L_2' \in {}_{\alpha_I}\langle L_2 \rangle$ such that $L_1' \underset{\sim}{\xRightarrow{(\mathbb{F}', M')}}_{\mathcal{L}} L_2'$, where $(\mathbb{F}, M) \xrightarrow{S}{}^{*}_{\alpha_I} (\mathbb{F}', M')$,*
9. *If $L_1 \circ\!\underset{\sim}{\xrightarrow{(\mathbb{F}, M)}}_{\mathcal{L}} L_2$, then there exists $L_2' \in {}_{\alpha_I}\langle L_2 \rangle$ such that $L_1' \circ\!\underset{\sim}{\xrightarrow{(\mathbb{F}', M')}}_{\mathcal{L}} L_2'$, where $(\mathbb{F}, M) \xrightarrow{S}{}^{*}_{\alpha_I} (\mathbb{F}', M')$,*

$\square$

We do not define calculus reductions of $\mathcal{L}$ on α_I-equivalence classes, because in the next section we define $\alpha_{\mathcal{L}}$-renaming which subsumes α_I-renaming, and define reductions on $\alpha_{\mathcal{L}}$-equivalence classes.

2.5.6 $\mathcal{L}$ Terms Modulo α-Equivalence

Note that a linking expression has two α-equivalence classes in the linking calculus: the one induced by $\longrightarrow_{\alpha_\mathcal{C}}$ and the one induced by $\longrightarrow_{\alpha_I}$. The following lemma states that the two relations are independent (see definition 2.2):

Lemma 2.100. *The relation $\longrightarrow_{\alpha_\mathcal{C}}$ is independent from $\longrightarrow_{\alpha_I}$, i.e. if $L_1 \longrightarrow_{\alpha_\mathcal{C}} L_2$ and $L_2 \longrightarrow_{\alpha_I} L_3$, then there exists L_4 such that $L_1 \longrightarrow_{\alpha_I} L_4$ and $L_4 \longrightarrow_{\alpha_\mathcal{C}} L_3$.* □

Proof. By definition of $\longrightarrow_{\alpha_\mathcal{C}}$ there exist $\mathbb{L}, H$, and H' such that $L_1 = \mathbb{L}\{H\}$, $L_2 = \mathbb{L}\{H'\}$, and $H \longrightarrow_{\alpha_\mathcal{C}} H'$. By definition of $\longrightarrow_{\alpha_I}$ there exist $\mathbb{L}', I, I', L', L'_1$, and L'_2 such that $L_2 = \mathbb{L}'\{\textbf{let } I = L' \textbf{ in } L'_1\}$, $L_3 = \mathbb{L}'\{\textbf{let } I' = L' \textbf{ in } L'_2\}$, and $L'_2 = L'_1[I := I']$. Combining the two results, we get $L_2 = \mathbb{L}\{H'\} = \mathbb{L}'\{\textbf{let } I = L' \textbf{ in } L'_1\}$. We have the following 4 cases:

- There exists ${}^2\mathbb{L}$ such that $L_2 = {}^2\mathbb{L}\{H', \textbf{let } I = L' \textbf{ in } L'_1\}$, then

$$\begin{array}{ll} {}^2\mathbb{L}\{H, \textbf{let } I = L' \textbf{ in } L'_1\} & \longrightarrow_{\alpha_I} \\ {}^2\mathbb{L}\{H, \textbf{let } I' = L' \textbf{ in } L'_2\} & \longrightarrow_{\alpha_\mathcal{C}} \\ {}^2\mathbb{L}\{H', \textbf{let } I' = L' \textbf{ in } L'_2\}, & \end{array}$$

  and the resulting expression is L_3.

- There exists ${}^2\mathbb{L}$ such that $L_2 = {}^2\mathbb{L}\{\textbf{let } I' = L' \textbf{ in } L'_2, H'\}$. Analogous to the previous case.

- There exists $\mathbb{L}_1$ such that $L' = \mathbb{L}_1\{H'\}$. Then $L_1 = \mathbb{L}'\{\textbf{let } I = \mathbb{L}_1\{H\} \textbf{ in } L'_1\}$, and

$$\begin{array}{ll} \mathbb{L}'\{\textbf{let } I = \mathbb{L}_1\{H\} \textbf{ in } L'_1\} & \longrightarrow_{\alpha_I} \\ \mathbb{L}'\{\textbf{let } I' = \mathbb{L}_1\{H\} \textbf{ in } L'_2\} & \longrightarrow_{\alpha_\mathcal{C}} \\ \mathbb{L}'\{\textbf{let } I' = \mathbb{L}_1\{H'\} \textbf{ in } L'_2\}, & \end{array}$$

  and the result is L_3.

- There exists $\mathbb{L}_1$ such that $L'_1 = \mathbb{L}_1\{H'\}$. Then $L'_2 = (\mathbb{L}_1\{H'\})[I := I'] = \mathbb{L}'_1\{H'\}$. We have $L_2 = \mathbb{L}'\{\textbf{let } I = L' \textbf{ in } \mathbb{L}_1\{H'\}\}$, and $L_1 = \mathbb{L}'\{\textbf{let } I = L' \textbf{ in } \mathbb{L}_1\{H\}\}$. Then

$$\begin{array}{ll} \mathbb{L}'\{\textbf{let } I = L' \textbf{ in } \mathbb{L}_1\{H\}\} & \longrightarrow_{\alpha_I} \\ \mathbb{L}'\{\textbf{let } I' = L' \textbf{ in } (\mathbb{L}_1\{H\})[I := I']\} & = \\ \mathbb{L}'\{\textbf{let } I' = L' \textbf{ in } \mathbb{L}'_1\{H\}\} & \longrightarrow_{\alpha_\mathcal{C}} \\ \mathbb{L}'\{\textbf{let } I' = L' \textbf{ in } \mathbb{L}'_1\{H'\}\}, & \end{array}$$

  and again the resulting linking expression is L_3.

□

Now we combine the two α-renaming relations into one, which we call $\alpha_\mathcal{L}$-renaming:

Definition 2.101 ($\alpha_\mathcal{L}$-Renaming in $\mathcal{L}$). We say that L_1 reduces to L_2 by $\alpha_\mathcal{L}$-renaming with the $\alpha_\mathcal{L}$-redex $(\mathbb{L}, h, h')$ (respectively with the $\alpha_\mathcal{L}$-redex $(\mathbb{F}, x, y)$), written $L_1 \xrightarrow{(\mathbb{L},h,h')}_{\alpha_\mathcal{L}} L_2$ (respectively $L_1 \xrightarrow{(\mathbb{F},x,y)}_{\alpha_\mathcal{L}} L_2$) if $L_1 \xrightarrow{(\mathbb{L},h,h')}_{\alpha_\mathcal{C}} L_2$ (respectively $L_1 \xrightarrow{(\mathbb{F},x,y)}_{\alpha_\mathcal{C}} L_2$).

We say that L_1 reduces to L_2 by $\alpha_\mathcal{L}$-renaming with the $\alpha_\mathcal{L}$-redex $(\mathbb{L}, I, I')$, written $L_1 \xrightarrow{(\mathbb{L},I,I')}_{\alpha_\mathcal{L}} L_2$, if $L_1 \xrightarrow{(\mathbb{L},I,I')}_{\alpha_I} L_2$.

As usual, we sometimes omit $\alpha_\mathcal{L}$-redexes in the notation. The notations $\longrightarrow^*_{\alpha_\mathcal{L}}$, $\xrightarrow{S}{}^*_{\alpha_\mathcal{L}}$, and $=^\mathcal{L}_\alpha$ are all standard. □

Lemma 2.102. *If $L_1 =^\mathcal{L}_\alpha L_2$, then there exists L' such that $L_1 \longrightarrow^*_{\alpha_I} L' \longrightarrow^*_{\alpha_\mathcal{C}} L_2$.* □

Proof. The result follows from symmetry of $\longrightarrow_{\alpha_C}$ and $\longrightarrow_{\alpha_I}$ and from the fact that $\longrightarrow_{\alpha_C}$ is independent from $\longrightarrow_{\alpha_I}$ (see lemma 2.100) by lemma 2.3. $\square$

Definition 2.103 ($\alpha_{\mathcal{L}}$-Renaming of Subterm Occurrences). A subterm occurrence $(\mathbb{L}, L)$ *reduces by a one-step $\alpha_{\mathcal{L}}$-renaming* to a subterm occurrence $(\mathbb{L}', L')$ with an $\alpha_{\mathcal{L}}$-redex $(\mathbb{L}_1, I, I')$ (written $(\mathbb{L}, L) \xrightarrow{(\mathbb{L}_1,I,I')}_{\alpha_{\mathcal{L}}} (\mathbb{L}', L')$) if $(\mathbb{L}, L) \xrightarrow{(\mathbb{L}_1,I,I')}_{\alpha_I} (\mathbb{L}', L')$. $(\mathbb{L}, L)$ *reduces by a one-step $\alpha_{\mathcal{L}}$-renaming* to $(\mathbb{L}', L')$ with an $\alpha_{\mathcal{L}}$-redex $(\mathbb{L}_1, h, h')$ or $(\mathbb{F}, x, y)$ if $(\mathbb{L}, L) \xrightarrow{(\mathbb{L}_1,h,h')}_{\alpha_C} (\mathbb{L}', L')$ or $(\mathbb{L}, L) \xrightarrow{(\mathbb{F},x,y)}_{\alpha_C} (\mathbb{L}', L')$.

The definition for a one step $\alpha_{\mathcal{L}}$-renaming of a subterm occurrence $(\mathbb{F}, M)$ to $(\mathbb{F}, M)$ is completely analogous.

$\alpha_{\mathcal{L}}$-equivalence classes of the two kinds of subterm occurrences are defined as usual. We use a meta-variable $Sub^{\mathcal{L},\mathcal{L}}_{\alpha_{\mathcal{L}}}$ to range over $\alpha_{\mathcal{L}}$-equivalence classes of the former kind of subterm occurrences, and $Sub^{\mathcal{T},\mathcal{L}}_{\alpha_{\mathcal{L}}}$ to range over the $\alpha_{\mathcal{L}}$-equivalence classes of the latter kind. $\square$

As for the other α-renamings, e.g. lemma 2.81, $\longrightarrow_{\alpha_{\mathcal{L}}}$ preserves distinctness of subterm occurrences (for both kinds of subterms). Also, as it is the case for $\longrightarrow_{\alpha_C}$, $\longrightarrow_{\alpha_{\mathcal{L}}}$ preserves redexes of the form $(\mathbb{F}, M)$ (see lemma 2.82). Since redexes of the form $(\mathbb{L}, K)$ are not preserved by $\longrightarrow_{\alpha_C}$, they are not preserved by $\longrightarrow_{\alpha_{\mathcal{L}}}$. However, the weaker property stated in lemma 2.84 holds for $\longrightarrow_{\alpha_C}$ as well:

Lemma 2.104 (Property of $\alpha_{\mathcal{L}}$-Equivalent Linking Redexes). *Let $(\mathbb{L}, K)$ and $(\mathbb{L}', K')$ be two linking redexes such that $(\mathbb{L}, K) \longrightarrow_{\alpha_{\mathcal{L}}} (\mathbb{L}', K')$, $\mathbb{L}\{K\} \xRightarrow{(\mathbb{L},K)}_{\mathcal{L}} L_2$, and $\mathbb{L}'\{K'\} \xRightarrow{(\mathbb{L}',K')}_{\mathcal{L}} L_2'$, then $\mathbb{L}\{K\} \longrightarrow_{\alpha_{\mathcal{L}}} \mathbb{L}'\{K'\}$ and $L_2 =^{\mathcal{L}}_{\alpha} L_2'$.* $\square$

Proof. Follows from lemma 2.84 if $\longrightarrow_{\alpha_{\mathcal{L}}}$ is a α_C-renaming step, and from the fact that $\longrightarrow_{\alpha_I}$ preserves linking redexes if $\longrightarrow_{\alpha_{\mathcal{L}}}$ is a α_I-renaming step. $\square$

The lemma below shows that for every calculus reduction there is a corresponding canonical reduction:

Lemma 2.105 (Existence of a Redex-preserving Sequence of $\alpha_{\mathcal{L}}$-Renamings). *If $L =^{\mathcal{L}}_{\alpha} \hat{L}$, where $\hat{L}$ is canonical, then there exists S such that $L \xrightarrow{S}{}^{*}_{\alpha_{\mathcal{L}}} \hat{L}$ and for every linking redex $(\mathbb{L}, K)$ and L' such that $L \xRightarrow{(\mathbb{L},K)}_{\mathcal{L}} L'$ (respectively $L \xRightarrow{(\mathbb{F},M)}_{\mathcal{L}} L'$ or $L \circ\!\xrightarrow{(\mathbb{F},M)}_{\mathcal{L}} L'$) there exists $L_1' \in {}_{\alpha_{\mathcal{L}}}\langle L' \rangle$ such that $\hat{L} \underset{\sim}{\xRightarrow{(\mathbb{L}',K')}}_{\mathcal{L}} L_1'$ (respectively $\hat{L} \underset{\sim}{\xRightarrow{(\mathbb{F}',M')}}_{\mathcal{L}} L_1'$ or $\hat{L} \circ\!\underset{\sim}{\xrightarrow{(\mathbb{F}',M')}}_{\mathcal{L}} L_1'$), where $(\mathbb{L}, K) \xrightarrow{S}{}^{*}_{\alpha_{\mathcal{L}}} (\mathbb{L}', K')$ (respectively $(\mathbb{F}, M) \xrightarrow{S}{}^{*}_{\alpha_{\mathcal{L}}} (\mathbb{F}', M')$).* $\square$

Proof. By lemma 2.102 $L =^{\mathcal{L}}_{\alpha} \hat{L}$ implies that there exists $\hat{L}_1$ such that $L \xrightarrow{S_1}{}^{*}_{\alpha_C} \hat{L}_1 \xrightarrow{S_1}{}^{*}_{\alpha_I} \hat{L}$, and by part 3 of lemma 2.99 $\hat{L}_1$ is canonical. Then by part 8 of lemma 2.88 there exists a sequence S_1' such that $L \xrightarrow{S_1'}{}^{*}_{\alpha_C} \hat{L}_1$ and for every $(\mathbb{L}, K)$ such that $L \xRightarrow{(\mathbb{L},K)}_{\mathcal{L}} L'$ there exists $L_1' \in {}_{\alpha_C}\langle L' \rangle$ such that $\hat{L}_1 \underset{\sim}{\xRightarrow{(\mathbb{L}',K')}}_{\mathcal{L}} L_1'$, where $(\mathbb{L}, K) \xrightarrow{S_1'}{}^{*}_{\alpha_C} (\mathbb{L}', K')$. Then by part 7 of lemma 2.99 there exist $\hat{L}' \in {}_{\alpha_I}\langle \hat{L}_1' \rangle$ such that $\hat{L} \underset{\sim}{\xRightarrow{(\mathbb{L}',K')}}_{\mathcal{L}} \hat{L}'$, where $(\mathbb{L}', K') \xrightarrow{S_2}{}^{*}_{\alpha_C} (\mathbb{L}', K')$. Then $L \xrightarrow{S_1';S_2}{}^{*}_{\alpha_{\mathcal{L}}} \hat{L}$, $(\mathbb{L}, K) \xrightarrow{S_1';S_2}{}^{*}_{\alpha_{\mathcal{L}}} (\mathbb{L}', K')$, and $\hat{L}' \in {}_{\alpha_{\mathcal{L}}}\langle L' \rangle$.

The proof for redexes of the form $(\mathbb{F}, M)$ is similar. We use part 9 of lemma 2.88 for evaluation $(\mathbb{F}, M)$ redexes, and part 3 of lemma 2.82 for non-evaluation redexes. $\square$

We extend the calculus relations of $\mathcal{L}$ to $\alpha_{\mathcal{L}}$-equivalence classes in the usual way:

Definition 2.106. $L_{\alpha_{\mathcal{L}}} \xRightarrow{Sub^{\mathcal{L},\mathcal{L}}_{\alpha_{\mathcal{L}}}}_{\mathcal{L}\backslash\alpha} L'_{\alpha_{\mathcal{L}}}$ if there exist $L_1 \in L_{\alpha_{\mathcal{L}}}$, $L_2 \in L'_{\alpha_{\mathcal{L}}}$, and $(\mathbb{L}, K) \in Sub^{\mathcal{L},\mathcal{L}}_{\alpha_{\mathcal{L}}}$ such that $L_1 \xRightarrow{(\mathbb{L},K)}_{\mathcal{L}} L_2$.

The other two reductions, i.e. $\xRightarrow{Sub^{\mathcal{T},\mathcal{L}}_{\alpha_{\mathcal{L}}}}_{\mathcal{L}}$ and $\circ\!\xrightarrow{Sub^{\mathcal{T},\mathcal{L}}_{\alpha_{\mathcal{L}}}}_{\mathcal{L}\backslash\alpha}$, are defined analogously. Recall that all redexes in $Sub^{\mathcal{L},\mathcal{L}}_{\alpha_{\mathcal{L}}}$ are evaluation redexes, so there is no reduction $\circ\!\xrightarrow{Sub^{\mathcal{L},\mathcal{L}}_{\alpha_{\mathcal{L}}}}_{\mathcal{L}\backslash\alpha}$. $\square$

Recall that $\rightarrow_{\mathcal{T}\backslash\alpha}$ and $\rightarrow_{\mathcal{C}\backslash\alpha}$ are defined for all representatives of an α-equivalence class if they are defined for one. However, $\rightarrow_{\mathcal{L}\backslash\alpha}$ may be defined on some, but not all reprentatives of an $\alpha_{\mathcal{L}}$-equivalence class.

Example 2.107. Let L and L' be the linking expressions from example 2.83. Then

$$\begin{array}{lclcl} L & = & [A \mapsto \lambda x.h, h \mapsto 5] \oplus [B \mapsto A \mathbin{@} h', h' \mapsto 7] & \Rightarrow_{\mathcal{L}} & [A \mapsto \lambda x.h, h \mapsto 5, B \mapsto A \mathbin{@} h', h' \mapsto 7] \\ L' & = & [A \mapsto \lambda x.h, h \mapsto 5] \oplus [B \mapsto A \mathbin{@} h, h \mapsto 7] & \not\Rightarrow_{\mathcal{L}} & (h \text{ defined in both modules}) \end{array}$$

Then, despite the conflict of the hidden labels in L', $\Rightarrow_{\mathcal{L}\backslash\alpha}$ is defined on the $\alpha_{\mathcal{L}}$-equivalence class of L', and the result of the reduction is the same $\alpha_{\mathcal{L}}$-equivalence class as for the L:

$$_{\alpha_{\mathcal{L}}}\langle [A \mapsto \lambda x.h, h \mapsto 5] \oplus [B \mapsto A \mathbin{@} h, h \mapsto 7] \rangle \quad \Rightarrow_{\mathcal{L}\backslash\alpha} \quad {}_{\alpha_{\mathcal{L}}}\langle [A \mapsto \lambda x.h, h \mapsto 5, B \mapsto A \mathbin{@} h', h' \mapsto 7] \rangle,$$

□

Remark 2.108. From the point of view of implementation this version of (link) is analogous to a two-stage linking operation that at the first stage resolves name conflicts between the hiddens of the two modules (by performing appropriate renaming of hiddens), and at the second stage combines the bindings of the two modules. Since $\Rightarrow_{\mathcal{L}\backslash\alpha}$ is defined on $\alpha_{\mathcal{L}}$-equivalence classes, i.e. such $\alpha_{\mathcal{L}}$-renaming is performed automatically, there is no need to introduce a separate user-level operator $\overline{\oplus}$ as we did in [MT00]. This renaming is similar to α-renaming required in other module calculi linking operations (e.g., in [FF98] when rewriting the **compound** linking form to the **unit** module form). □

The theorem below is the key property of α-renaming at the linking level. It states that to consider reductions on linking expressions, it is sufficient to consider these reductions on canonical representatives of $\alpha_{\mathcal{L}}$-equivalence classes. The theorem generalizes lemma 2.93 from $\alpha_{\mathcal{C}}$-equivalence classes to $\alpha_{\mathcal{L}}$-equivalence classes defined above.

Theorem 2.109. *For any $\alpha_{\mathcal{L}}$-equivalence class $L_{\alpha_{\mathcal{L}}}$ $L_{\alpha_{\mathcal{L}}} \xRightarrow{Sub^{\mathcal{L},\mathcal{L}}_{\alpha_{\mathcal{L}}}}_{\mathcal{L}\backslash\alpha} L'_{\alpha_{\mathcal{L}}}$ (respectively $L_{\alpha_{\mathcal{L}}} \xRightarrow{Sub^{\mathcal{T},\mathcal{L}}_{\alpha_{\mathcal{L}}}}_{\mathcal{L}\backslash\alpha} L'_{\alpha_{\mathcal{L}}}$ or $L_{\alpha_{\mathcal{L}}} \circ\!\xrightarrow{Sub^{\mathcal{T},\mathcal{L}}_{\alpha_{\mathcal{L}}}}_{\mathcal{L}\backslash\alpha} L'_{\alpha_{\mathcal{L}}}$) if and only if for every canonical expression $\hat{L} \in L_{\alpha_{\mathcal{L}}}$ there exist $\hat{L}' \in L'_{\alpha_{\mathcal{C}}}$ and $(\mathbb{L}, K) \in Sub^{\mathcal{L},\mathcal{L}}_{\alpha_{\mathcal{L}}}$ (respectively $(\mathbb{F}, M) \in Sub^{\mathcal{T},\mathcal{L}}_{\alpha_{\mathcal{L}}}$) such that $\hat{L} \underset{\sim}{\xRightarrow{(\mathbb{L},K)}}_{\mathcal{L}} \hat{L}'$ (respectively $\hat{L} \underset{\sim}{\xRightarrow{(\mathbb{F},M)}}_{\mathcal{L}} \hat{L}'$ or $\hat{L} \circ\!\underset{\sim}{\xrightarrow{(\mathbb{F},M)}}_{\mathcal{L}} \hat{L}'$).* □

Proof. The "if" part follows from definition 2.106.

"Only if": suppose $L_{\alpha_{\mathcal{L}}} \xRightarrow{Sub^{\mathcal{L},\mathcal{L}}_{\alpha_{\mathcal{L}}}}_{\mathcal{L}\backslash\alpha} L'_{\alpha_{\mathcal{L}}}$, then by definition 2.106 there exists $L \in L_{\alpha_{\mathcal{L}}}$, $L' \in L'_{\alpha_{\mathcal{L}}}$, and $(\mathbb{L}, K) \in Sub^{\mathcal{L},\mathcal{L}}_{\alpha_{\mathcal{L}}}$ such that $L \xRightarrow{(\mathbb{L},K)}_{\mathcal{L}} L'$. Let $\hat{L}$ be a canonical expression in $L_{\alpha_{\mathcal{L}}}$. By lemma 2.105 there exists a sequence of $\alpha_{\mathcal{L}}$-renamings S and $\hat{L}' \in L'_{\alpha_{\mathcal{L}}}$ such that $L \xrightarrow{S}{}^{*}_{\alpha_{\mathcal{L}}} \hat{L}$ and $\hat{L} \underset{\sim}{\xRightarrow{(\mathbb{L}',K')}}_{\mathcal{L}} \hat{L}'$, where $(\mathbb{L}, K) \xrightarrow{S}{}^{*}_{\alpha_{\mathcal{L}}} (\mathbb{L}', K')$, i.e. $(\mathbb{L}', K') \in Sub^{\mathcal{L},\mathcal{L}}_{\alpha_{\mathcal{L}}}$. The proof is analogous for the other two reductions. □

Theorem 2.109, together with part 4 of lemma 2.88 (the existence of a canonical representative in every $\alpha_{\mathcal{C}}$-equivalence class), allow us to restrict proofs to canonical reductions on canonical representatives, and to generalize the results to the entire $\alpha_{\mathcal{L}}$-equivalence classes. For instance, suppose we want to show confluence of $\Rightarrow_{\mathcal{L}\backslash\alpha}$: if $L_{1\alpha_{\mathcal{L}}} \Rightarrow^{*}_{\mathcal{L}\backslash\alpha} L_{2\alpha_{\mathcal{L}}}$ and $L_{1\alpha_{\mathcal{L}}} \Rightarrow^{*}_{\mathcal{L}\backslash\alpha} L_{3\alpha_{\mathcal{L}}}$, where $L_{2\alpha_{\mathcal{L}}} \neq L_{3\alpha_{\mathcal{L}}}$, then there exists $L_{4\alpha_{\mathcal{L}}}$ such that $L_{2\alpha_{\mathcal{L}}} \Rightarrow^{*}_{\mathcal{L}\backslash\alpha} L_{4\alpha_{\mathcal{L}}}$ and $L_{3\alpha_{\mathcal{L}}} \Rightarrow^{*}_{\mathcal{L}\backslash\alpha} L_{4\alpha_{\mathcal{L}}}$. Then it is sufficient to choose a canonical $\hat{L}_1 \in L_{1\alpha_{\mathcal{L}}}$ and to show that for every pair of sequences of canonical reductions $\hat{L}_1 \underset{\sim}{\Rightarrow}{}^{*}_{\mathcal{L}} \hat{L}_2$ and $\hat{L}_1 \underset{\sim}{\Rightarrow}{}^{*}_{\mathcal{L}} \hat{L}_3$ such that ${}_{\alpha_{\mathcal{L}}}\langle \hat{L}_2 \rangle \neq {}_{\alpha_{\mathcal{L}}}\langle \hat{L}_3 \rangle$ there exists $\hat{L}_4$ such that $\hat{L}_2 \underset{\sim}{\Rightarrow}{}^{*}_{\mathcal{L}} \hat{L}_4$ and $\hat{L}_3 \underset{\sim}{\Rightarrow}{}^{*}_{\mathcal{L}} \hat{L}_4$. Since $\underset{\sim}{\Rightarrow}_{\mathcal{L}}$ involves $\alpha_{\mathcal{C}}$-renaming only after a reduction by the rule (let), the proof of the latter statement is much easier than of the former.

Having defined $\Rightarrow_{\mathcal{L}\backslash\alpha}$, we define classification of $\alpha_{\mathcal{L}}$-equivalence classes of linking expressions as follows:

$$Cl_{\mathcal{L}\backslash\alpha}(L_{\alpha_{\mathcal{L}}}) = \begin{cases} \mathbf{evaluatable}_{\mathcal{L}} & \text{if there exists} \quad L'_{\alpha_{\mathcal{L}}} \quad \text{such that} \quad L_{\alpha_{\mathcal{L}}} \Rightarrow_{\mathcal{L}\backslash\alpha} L'_{\alpha_{\mathcal{L}}}, \\ Cl_{\mathcal{C}\backslash\alpha}(H) & \text{if there exists} \quad H \in L_{\alpha_{\mathcal{L}}} \quad \text{such that} \quad H \quad \text{is an} \Rightarrow_{\mathcal{C}} \text{normal form}, \\ \mathbf{error}_{\mathcal{L}} & \text{otherwise.} \end{cases}$$

We define $Cl_{\mathcal{L}}(L) = Cl_{\mathcal{L}\backslash\alpha}(\alpha_{\mathcal{L}}\langle L\rangle)$. The reason why we define classification first on $\alpha_{\mathcal{L}}$-equivalence classes, and only then extend this definition to concrete linking expressions, is that a linking expression may be not evaluatable even if its $\alpha_{\mathcal{L}}$-equivalence class is. For instance, consider L and L' in example 2.107. Since we work with $\alpha_{\mathcal{L}}$-equivalence classes of linking expressions, we expect that $Cl_{\mathcal{L}}(L') = Cl_{\mathcal{L}}(L) = \mathbf{evaluatable}_{\mathcal{L}}$, which is the case when we adopt the definition above. If we were to define classification on concrete linking expressions directly, the classification of L' would not have been $\mathbf{evaluatable}_{\mathcal{L}}$, because there is no linking expression that L' evaluates to.

The proviso "H is an $\Rightarrow_{\mathcal{C}}$ normal form" in the second case of the definition is necessary to guarantee that the classes are disjoint. Note that if there is one $H \in L_{\alpha_{\mathcal{L}}}$ that satisfies the condition, then every representative of $L_{\alpha_{\mathcal{L}}}$ satisfies it by parts 3 and 6 of lemma 2.55.

The class $\mathbf{error}_{\mathcal{L}}$ captures the case when a linking expression never evaluates to a module because it contains a $\oplus$ operation whose arguments export the same names, for instance:

$$Cl_{\mathcal{L}}([A \mapsto 2+3] \oplus [A \mapsto \lambda x.x]) = \mathbf{error}_{\mathcal{L}}.$$

Another example of a linking error is an attempt to rename a bound label in a module to another bound label, e.g.

$$Cl_{\mathcal{L}}([A \mapsto \lambda x.x, B \mapsto 2][A \stackrel{\text{ren}}{\leftarrow} B]) = \mathbf{error}_{\mathcal{L}}.$$

Note that $D \notin \mathbf{HTerm}_{\mathcal{C}}$ is also considered a linking error. However, we show that the result of evaluating a linking expression cannot be of this form, since all modules that form a linking expression are h-closed.

$Outcome_{\mathcal{L}\backslash\alpha}$ in the usual way, i.e. $Outcome_{\mathcal{L}\backslash\alpha}(L_{\alpha_{\mathcal{L}}}) = Cl_{\mathcal{L}\backslash\alpha}(Eval_{\mathcal{L}\backslash\alpha}(L_{\alpha_{\mathcal{L}}}))$ if $Eval_{\mathcal{L}\backslash\alpha}(L_{\alpha_{\mathcal{L}}})$ exists, otherwise $Outcome_{\mathcal{L}\backslash\alpha}(L) = \perp_{\mathcal{L}}$, and $\mathbf{Value}_{\mathcal{L}} = \mathbf{Value}_{\mathcal{C}}$.

Remark 2.110 (Changes from [MT00] in Definition of Classification). The main changes are due introduction of $\alpha_{\mathcal{L}}$-renaming of linking expressions (in particular, renaming of hidden labels). The other changes are for the purpose of consistency with classifications of $\mathcal{T}$ and $\mathcal{C}$. Below is the list of changes:

- The classification is now defined on $\alpha_{\mathcal{L}}$-equivalence classes rather than on concrete linking expressions,
- Since at the core module level we no longer require that modules do not import hiddens, we explicitly state specify that all modules which are part of linking expressions do not import hiddens (i.e. are h-closed), and explicitly state that module values are h-closed. However, this change is only notational, since the set of modules over which module expressions are formed is exactly the same as in [MT00], i.e., in our current terminology, the set $\mathbf{HTerm}_{\mathcal{C}}$ of h-closed modules.
- We have introduced the notion of **evaluatable** for each level of the calculus separately, in particular $\mathbf{evaluatable}_{\mathcal{L}}$ at the linking level. In the old classification we used the term **linkable** for linking expressions that are yet to be evaluated to a single module, and once they have been reduced to a module, their classification would be that of the module. In the new classification the class of a single module considered as a linking expression is $\mathbf{evaluatable}_{\mathcal{L}}$. After a linking expression is evaluated to a $\Rightarrow_{\mathcal{C}}$ normal form, its class is the class of the normal form, as in the presentation in [MT00] (but see section 2.4 for changes in the classification of modules). The new classification makes the definition of **evaluatable** the same in all three calculi: a term is evaluatable if it can be evaluated further.
- We have introduced a new class $\mathbf{error}_{\mathcal{L}}$ for errors at the linking level (see example above). Linking errors are different from module errors (see section 2.4). The new classification allows us to distinguish

between the two: for instance, if a conflict between visible labels prevents linking of two modules, then the outcome of the linking expression is $\mathbf{error}_{\mathcal{L}}$, but if a linking expression evaluates to a module, which in turn evaluates to a normal form other than a value, then the outcome of the expression is $\mathbf{error}_{\mathcal{C}}$. The introduction of $\mathbf{error}_{\mathcal{L}}$ corrects the classification of [MT00], where a non-module linking expression is considered **linkable** even if no link-level evaluation step is possible from it.

□

Many properties of $\rightarrow_{\mathcal{L}\backslash\alpha}$ are similar to those of $\rightarrow_{\mathcal{C}\backslash\alpha}$. In the discussion below we adopt the convention of the calculi $\mathcal{T}\backslash\alpha$ and $\mathcal{C}\backslash\alpha$ of using concrete terms instead of their α-equivalence classes, e.g. convention 2.40.

Example 2.111 (Non-confluence of $\rightarrow_{\mathcal{C}\backslash\alpha}$). The lack of confluence of $\rightarrow_{\mathcal{C}\backslash\alpha}$ is inherited by $\rightarrow_{\mathcal{L}\backslash\alpha}$, as illustrated by the following example:

$$\begin{array}{lll} [A \mapsto \lambda x.B, B \mapsto \lambda x.A] \oplus [C \mapsto 2] & \circ\!\!\rightarrow_{\mathcal{L}\backslash\alpha} & [A \mapsto \lambda x.\lambda x.A, B \mapsto \lambda x.A] \oplus [C \mapsto 2], \\ [A \mapsto \lambda x.B, B \mapsto \lambda x.A] \oplus [C \mapsto 2] & \circ\!\!\rightarrow_{\mathcal{L}\backslash\alpha} & [A \mapsto \lambda x.B, B \mapsto \lambda x.\lambda x.B] \oplus [C \mapsto 2]. \end{array}$$

As in the calculus $\mathcal{C}\backslash\alpha$, there is no term both of these expressions reduce to. Note that, as in $\mathcal{C}\backslash\alpha$, both reductions are non-evaluation steps. □

Despite the lack of confluence of $\rightarrow_{\mathcal{L}\backslash\alpha}$, we are still able to show the following:

Theorem 2.112. *$\Rightarrow_{\mathcal{L}\backslash\alpha}$ is confluent.* □

As for $\Rightarrow_{\mathcal{C}\backslash\alpha}$, we show that it cannot be the case that $\Rightarrow_{\mathcal{L}\backslash\alpha}$ diverges on one path and leads to a normal form on another.

Lemma 2.113. *If $L \Rightarrow^*_{\mathcal{L}\backslash\alpha} \mathit{Eval}_{\mathcal{L}}(L)$, then there is no infinite sequence of $\Rightarrow_{\mathcal{L}\backslash\alpha}$ steps originating at L.* □

Theorem 2.114. *$\mathcal{L}\backslash\alpha$ has the standardization property.* □

We show that every linking expression evaluates to a module, unless the evaluation encounters a linking error:

Theorem 2.115 (Non-error Linking Expressions Evaluate to a Module). *If $\mathit{Outcome}_{\mathcal{L}\backslash\alpha}(L) \neq \mathbf{error}_{\mathcal{L}}$, then there exists H such that $L \Rightarrow^*_{\mathcal{L}\backslash\alpha} H$.* □

Since module evaluation steps are evaluation steps at the linking level only if they are performed in an empty linking context, any evaluation sequence in $\mathcal{L}\backslash\alpha$ performs all link-level steps first, followed by module level evaluation. We call this property *staging*.

Theorem 2.116 (Staging). *Given a sequence $L_1 \stackrel{S}{\Rightarrow}{}^*_{\mathcal{L}\backslash\alpha} L_2$, there exists L' such that $L_1 \stackrel{S_1}{\Rightarrow}{}^*_{\mathcal{L}\backslash\alpha} L' \stackrel{S_2}{\Rightarrow}{}^*_{\mathcal{L}\backslash\alpha} L_2$, where $S = S_1; S_2$, $S_1 = (\mathbb{L}_1, K_1); \ldots; (\mathbb{L}_n, K_n)$, $n \geq 0$, and $S_2 = (\mathbb{F}_1, M_1); \ldots; (\mathbb{F}_m, M_m)$, $m \geq 0$.* □

Staging models a typical behavior of programs which consist of several modules: first all the modules are linked into one, and then this one module gets evaluated.

Proofs of these properties are given in appendix D.

2.5.7 Change in the rule (let) from [MT00]

In addition to changes in classification, a significant change from [MT00] is in the evaluation rule (let). In [MT00] we have considered a "non-deterministic" rule (let) which allows (but does not force) substitution of the definition term L_1 into the body L_2 at any point of evaluation:

$$\mathbf{let}\ I = L_1\ \mathbf{in}\ L_2 \quad \Rightarrow_{\mathcal{L}} \quad L_2[I := L_1] \qquad \text{([MT00]-let)}$$

The new rule introduced in this presentation

$$\mathbf{let}\ I = H\ \mathbf{in}\ L \quad \Rightarrow_{\mathcal{L}} \quad L[I := H] \qquad \text{(let)}$$

requires the definition term to be first evaluated to a module H. After that the result is substituted into L_2. The latter rule is actually a particular case of the former: the structure of linking contexts in $\mathcal{L}$ allows evaluation of any linking redex at any time, so in particular one evaluation strategy may be to evaluate L_1 to a module H first, and then to perform the substitution.

The reason for the change in the rule (let) is that the new rule is consistent with the call-by-value nature of other reductions in our calculus: the call-by-value term calculus and the substitution at the core module level.

Note, however, that even though the definition term in the new (let) rule must be evaluated to a module before it gets substituted, it cannot be evaluated to a *module value*, because a module evaluation is considered evaluation at the linking level only when the module appears in an empty context.

EXAMPLE 2.117 ("CALL-BY-VALUE" (LET)).

$$\begin{array}{ll}
\mathbf{let}\ I = [A \mapsto B+1] \oplus [B \mapsto 2]\ \mathbf{in}\ I \oplus [C \mapsto A*3] & \Longrightarrow_{\mathcal{L}} \\
\mathbf{let}\ I = [A \mapsto B+1, B \mapsto 2]\ \mathbf{in}\ I \oplus [C \mapsto A*3] & \Longrightarrow_{\mathcal{L}} \\
[A \mapsto B+1, B \mapsto 2] \oplus [C \mapsto A*3] & \Longrightarrow_{\mathcal{L}} \\
[A \mapsto B+1, B \mapsto 2, C \mapsto A*3] & \Longrightarrow^*_{\mathcal{L}} \\
[A \mapsto 3, B \mapsto 2, C \mapsto 9]. &
\end{array}$$

The module $[A \mapsto B+1, B \mapsto 2]$ is not a module value, since it has a substitution redex B. However, the substitution cannot be performed as a part of an evaluation sequence until the entire linking expression is reduced to a single module. □

REMARK 2.118. As the example shows, the evaluation at the linking level with the new rule (let) is "call-by-value" in a slightly different sense than calculi $\mathcal{T}$ and $\mathcal{C}$. To make the rule completely analogous to these calculi, we could have defined it as follows:

$$\mathbf{let}\ I = H\ \mathbf{in}\ L \quad \Longrightarrow_{\mathcal{L}} \quad L[I := H] \quad \text{if } H \in \mathbf{Value}_{\mathcal{C}} \qquad \text{(let*)}$$

This rule requires the **let**-bound expression to evaluate to a module value, rather than just a module, before reducing the **let**.

However, introducing the rule (let*) instead of (let) causes problems: suppose we adopt the rule (let*) instead of (let), then the reduction sequence

$$\begin{array}{lll}
\mathbf{let}\ I = [A \mapsto 2+3]\ \mathbf{in}\ I & \circ\!\!\longrightarrow_{\mathcal{L}} & \text{(mod-nev)} \\
\mathbf{let}\ I = [A \mapsto 5]\ \mathbf{in}\ I & \Longrightarrow_{\mathcal{L}} & \text{(let*)} \\
[A \mapsto 5] & &
\end{array}$$

violates standardization. To fix the problem, we need to relax the condition for (mod-ev) rule to allow evaluation of a module in a non-empty context to be an evaluation step at the linking level. The new rule may be formulated as follows:

$$\mathbb{L}\{H\} \quad \Longrightarrow_{\mathcal{L}} \quad \mathbb{L}\{H\} \quad \text{if } H \Longrightarrow_{\mathcal{C}} H'. \qquad \text{(mod-ev*)}$$

Then the reduction sequence above becomes:

$$\begin{array}{lll}
\mathbf{let}\ I = [A \mapsto 2+3]\ \mathbf{in}\ I & \Longrightarrow_{\mathcal{L}} & \text{(mod-ev*)} \\
\mathbf{let}\ I = [A \mapsto 5]\ \mathbf{in}\ I & \Longrightarrow_{\mathcal{L}} & \text{(let*)} \\
[A \mapsto 5] & &
\end{array}$$

However, this sequence violates the staging property (see theorem 2.116), because a module evaluation step is performed before a linking step. □

Note that if the module identifier I of an expression $\mathbf{let}\ I = L_1\ \mathbf{in}\ L_2$ is used in the body of L_2, then evaluation with our current (let) rule gives the same outcome of the expression as evaluation with the rule ([MT00]-let):

Example 2.119 (The Same Outcome of (let) and ([MT00]-let)). The expression

$$\textbf{let } I = [A \mapsto 6] \oplus [B \mapsto 7] \textbf{ in } I \oplus I[A \overset{\text{ren}}{\leftarrow} C][B \overset{\text{ren}}{\leftarrow} D]$$

evaluates to the module value $[A \mapsto 6, B \mapsto 7, C \mapsto 6, D \mapsto 7]$ with both rules.

The outcome of the expression

$$\textbf{let } I = [A \mapsto 6] \oplus [A \mapsto 7] \textbf{ in } I \oplus I[A \overset{\text{ren}}{\leftarrow} C][B \overset{\text{ren}}{\leftarrow} D]$$

is $\mathbf{error}_{\mathcal{L}}$ in both cases: if we adopt the current rule (let), then the expression is already a linking error, and if we use ([MT00]-let), then the expression evaluates to

$$[A \mapsto 6] \oplus [A \mapsto 7] \oplus [A \mapsto 6] \oplus [A \mapsto 7][A \overset{\text{ren}}{\leftarrow} C][B \overset{\text{ren}}{\leftarrow} D],$$

which is also a linking error. □

However, if I does not appear in L_2, then the two outcomes may differ:

Example 2.120 (Different Outcomes of (let) and ([MT00]-let)). Consider the expression

$$\textbf{let } I = [A \mapsto 6] \oplus [A \mapsto 7] \textbf{ in } [C \mapsto \lambda x.x].$$

According to the rule (let), it is an error, but it evaluates to a module value $[C \mapsto \lambda x.x]$ by ([MT00]-let). □

The property that every linking expression evaluates to a module, unless the evaluation encounters a linking error (theorem 2.115), as well as staging (theorem 2.116), are shown in appendix D for the more general (let) rule of [MT00]. The fact that the calculus of [MT00] has these properties implies that these properties also hold for the calculus with the more restricted "call-by-value" (let) in our current presentation.

2.6 Core Module Calculus and Linking Calculus with GC Rule

2.6.1 Motivation for (GC) rule

We would like to augment the reduction rules for the core module calculus $\mathcal{C}$ with a rule for garbage collection of groups of (possibly mutually recursive) hidden values, whose labels are not referenced in the rest of the module. Since the labels are hidden, these components cannot be used in another module, and therefore do not contribute to evaluation of the module or of any linking expression that the module may appear in. Note that we only consider garbage collection of values, since removing an evaluatable or a stuck component may change the behavior of the module. Consider, for instance, $[A \mapsto 2, w \mapsto \lambda x.xx \mathbin{@} \lambda x.xx]$. By removing the hidden component bound to w we are changing the outcome of the module from divergence to $[A \mapsto \mathbf{const}(2)]$. Similar problem occurs when a hidden component is stuck on an imported label, so that substituting for the label may cause the module to diverge. Errors could be considered garbage-collectable, but for simplicity we choose to collect only values.

Let $\longrightarrow_{\mathrm{GC}}$ denote the reduction step that performs garbage collection. We would like it to work as in the following example:

$$\begin{aligned}&[P \mapsto \lambda w.g \mathbin{@} (w+1), f \mapsto \lambda x.h, g \mapsto \lambda y.y * 2, h \mapsto \lambda z.f]\\ &\quad \longrightarrow_{\mathrm{GC}} [P \mapsto \lambda w.g \mathbin{@} (w+1), g \mapsto \lambda y.y * 2]\end{aligned}$$

The mutually recursive bindings for f and h can be removed because all references to these hidden labels occur inside of the values named by these labels. However, g cannot be removed, since an exported term references it. We do not require that a GC step removes the maximal set of non-referenced hidden labels, but mutually recursive hidden components must be removed all at once.

The following example shows why a GC reduction is desirable in our calculus:

$$\begin{array}{ll}
\textbf{let} & A = [F \mapsto \lambda x.x + Y] \\
 & B = [F \mapsto \lambda x.x * Y] \\
 & C = [Y \mapsto 3] \\
 & D = [Z \mapsto F \mathbin{@} 2] \\
\textbf{in} & ((A \oplus C \oplus D)\{\text{hide } F, Y\}[Z \stackrel{\text{ren}}{\leftarrow} Z_1]) \oplus ((B \oplus C \oplus D)\{\text{hide } F, Y\}[Z \stackrel{\text{ren}}{\leftarrow} Z_2]) \\
\Longrightarrow^*_{\mathcal{L}} & [h_1 \mapsto \lambda x.x + h_2, h_2 \mapsto 3, Z_1 \mapsto h_1 \mathbin{@} 2, h_3 \mapsto \lambda x.x * h_4, h_4 \mapsto 3, Z_2 \mapsto h_3 \mathbin{@} 2] \\
\Longrightarrow^*_{\mathcal{L}} & [h_1 \mapsto \lambda x.x + 3, h_2 \mapsto 3, Z_1 \mapsto 5, h_3 \mapsto \lambda x.x * 3, h_4 \mapsto 3, Z_2 \mapsto 6]
\end{array}$$

Reusing the modules and renaming leads to duplicating components of modules C and B, so the resulting module has 6 components. However, because of hiding, only two of these components (with labels Z_1 and Z_2) are exported. The rest of the module consists of hidden components that have no effect on Z_1 and Z_2 once their values have been calculated, and therefore on any module to which Z_1 or Z_2 may be exported. Garbage collection would allow us to reduce the module to $[Z_1 \mapsto 5, Z_2 \mapsto 6]$, thus reducing memory needed to store it. Without such a rule we are forced to keep all 6 components.

Having a GC rule becomes crucial for justifying program transformations: not only does it allow removing non-referenced hidden components, as in the example above (a transformation which is a form of dead-code elimination), but also it enables introduction of a new hidden component in a module by a backward GC step. By computational soundness any two terms equivalent in the calculus, i.e. connected by a sequence of backward and forward reduction steps, have the same meaning, so adding such a component does not change the meaning. Recall the example of cross-module lambda-splitting given in the introduction. If the calculus has a GC rule, then the following sequence of forward and backward reduction steps justifies the transformation:

$$\begin{array}{lcc}
[U \mapsto \lambda x.\mathbb{C}\{\lambda y.M'\}] \oplus [X \mapsto \mathbb{A}\{U \mathbin{@} N\}] & \Longrightarrow_{\mathcal{L}} & \text{(link)} \\
[U \mapsto \lambda x.\mathbb{C}\{\lambda y.M'\}, X \mapsto \mathbb{A}\{U \mathbin{@} N\}] & ?\leftarrow_{\text{GC}}? & ?\ \text{(GC)}\ ? \\
[U \mapsto \lambda x.\mathbb{C}\{\lambda y.M'\}, h \mapsto \lambda y.M', X \mapsto \mathbb{A}\{U \mathbin{@} N\}] & \hookleftarrow\!\circ_{\mathcal{L}} & \text{(mod-nev)} \\
[U \mapsto \lambda x.\mathbb{C}\{h\}, h \mapsto \lambda y.M', X \mapsto \mathbb{A}\{U \mathbin{@} N\}] & \Longrightarrow_{\mathcal{L}} & \text{(mod-ev)} \\
[U \mapsto \lambda x.\mathbb{C}\{h\}, h \mapsto \lambda y.M', X \mapsto \mathbb{A}\{\lambda x.\mathbb{C}\{h\} \mathbin{@} N\}] & \Longleftarrow_{\mathcal{L}} & \text{(hide)} \\
([U \mapsto \lambda x.\mathbb{C}\{U_e\}, U_e \mapsto \lambda y.M', X \mapsto \mathbb{A}\{\lambda x.\mathbb{C}\{U_e\} \mathbin{@} N\}])\{\text{hide } U_e\} & \Longleftarrow_{\mathcal{L}} & \text{(link)} \\
([U \mapsto \lambda x.\mathbb{C}\{U_e\}, U_e \mapsto \lambda y.M'] \oplus [X \mapsto \mathbb{A}\{\lambda x.\mathbb{C}\{U_e\} \mathbin{@} N\}])\{\text{hide } U_e\}. & &
\end{array}$$

In the above example it does not matter if the new rule (GC) is an evaluation or a non-evaluation step, in either case it is a calculus step, which is sufficient to justify the transformation. Note that (GC) is a core module reduction. The example above is at the level of linking expressions, so the GC reduction is, at the linking level, either $\Longrightarrow_{\mathcal{L}}$ by (mod-ev), (if GC is an evaluation step) or $\circ\!\!\longrightarrow_{\mathcal{L}}$ by (mod-nev) (otherwise).

2.6.2 Error in [MT00]: GC as an evaluation step

In [MT00] we introduced GC as a $\mathcal{C}$ evaluation step with the rule[11]

$$\mathbb{M}\{[h_j \overset{m}{\underset{j=1}{\mapsto}} V_j]\} \quad \Longrightarrow_{\mathcal{C}} \quad \mathbb{M}\{[]\} \text{ if } FL(\mathbb{M}) \cap \{h_j \mid 1 \le j \le m\} = \emptyset \quad \text{(GC) in [MT00]}$$

where by definition $FL(\mathbb{M}) = \bigcup_{i=1}^{n} FL(M_i) \setminus \{l_i \mid 1 \le i \le n\}$ if $\mathbb{M} = [l_i \overset{n}{\underset{i=1}{\mapsto}} M_i, \Box]$. Unfortunately, we later discovered that the resulting calculus does not have standardization property 2.7 as we claimed it did. Standardization is a key property in proving computational soundness, and indeed the calculus introduced in [MT00] is not computationally sound, as shown below. The following counterexample to standardization illustrates the problem:

$$[l \mapsto \lambda y.(\lambda x.2 \mathbin{@} \lambda z.h), h \mapsto 3] \overset{(comp-ev)}{\circ\!\!\longrightarrow}_{\mathcal{C}} [l \mapsto \lambda y.2, h \mapsto 3] \overset{(GC)}{\Longrightarrow}_{\mathcal{C}} [l \mapsto \lambda y.2]$$

[11]Since we have not introduced contexts $\mathbb{M} \in \textbf{Context}_{\mathcal{C},\mathcal{C}}$ in [MT00], the GC rule there is formulated in a different, but equivalent, way.

There is no standard sequence corresponding to the above non-standard one, since the label h cannot be garbage collected in $[l \mapsto \lambda y.\lambda x.2 \mathbin{@} \lambda z.h, h \mapsto 3]$ before the non-evaluation step removes the reference to it. In fact, this module is a normal form w.r.t. evaluation.

Remark 2.121. In [MT00] we worked explicitly with hidden labels (for instance, renaming of hiddens was an explicit linking operation). In the current presentation we identify modules up to consistent renaming of hidden labels. We have discovered that certain results, such as standardization, hold only at the level of α-equivalence classes of modules, but not at the concrete level. Note, however, that the counterexample to standardization above and the counterexample to computational soundness below are still valid at the level of $\alpha_{\mathcal{C}}$-equivalence classes of modules. $\square$

According to the classification of modules defined in [MT00], the sequence also violates computational soundness. The classification in [MT00] is defined as:

$$Cl_{\mathcal{C}}(D) = [l_i \overset{n}{\underset{i=1}{\mapsto}} Cl_{\mathcal{T}}(M_i)], \text{ where } D = [l_i \overset{n}{\underset{i=1}{\mapsto}} M_i]. \quad \text{([MT00] definition)}$$

Outcome is defined as the class of the evaluation normal form if it exists (here we ignore the issue of non-confluence of $\Longrightarrow_{\mathcal{L}}$ on concrete modules due to the choice of names of λ-bound variables during beta-reduction, since names of λ-bound variables do not affect the counterexample below). Since in the example above the module before the $\circ\!\xrightarrow{(comp-ev)}_{\mathcal{C}}$ step is an evaluation normal form, according to the old classification we have:

$$\begin{array}{lcl}
Outcome_{\mathcal{C}}([l \mapsto \lambda y.(\lambda x.2 \mathbin{@} \lambda z.h), h \mapsto 3]) & = & \text{(as in [MT00])} \\
Cl_{\mathcal{C}}([l \mapsto \lambda y.(\lambda x.2 \mathbin{@} \lambda z.h), h \mapsto 3]) & = & \\
[l \mapsto \mathbf{abs}, h \mapsto \mathbf{const}(3)], & & \\
Outcome_{\mathcal{C}}([l \mapsto \lambda y.2, h \mapsto 3]) & = & \\
Cl_{\mathcal{C}}(Eval_{\mathcal{C}}([l \mapsto \lambda y.2, h \mapsto 3])) & = & \\
Cl_{\mathcal{C}}([l \mapsto \lambda y.2]) & = & \\
[l \mapsto \mathbf{abs}]. & &
\end{array}$$

This example shows that a $\circ\!\!\longrightarrow_{\mathcal{C}}$ step does not preserve the outcome, and therefore the calculus is not computationally sound.

2.6.3 Correction of the error: GC as a non-evaluation step

In this presentation we have corrected the error of [MT00] by introducing GC as a non-evaluation step and changing the classification of core modules.

The new definition of GC rule is the same as before, only now this reduction is considered to be a non-evaluation step:

$$\mathbb{M}\{[h_j \overset{m}{\underset{j=1}{\mapsto}} V_j]\} \quad \circ\!\!\longrightarrow_{\mathcal{C}} \quad \mathbb{M}\{[]\} \text{ if } FL(\mathbb{M}) \cap \{h_j \mid 1 \leq j \leq m\} = \emptyset \quad \text{(GC-nev)}$$

We say that the subterm occurrence $(\mathbb{M}, [h_j \overset{m}{\underset{j=1}{\mapsto}} V_j])$ is the redex of the GC reduction, and write D $\circ\!\xrightarrow{(\mathbb{M}, [h_j \overset{m}{\underset{j=1}{\mapsto}} V_j])}_{\mathcal{C}} D'$. Whenever unambiguous, we refer to the new rule as just (GC).

If GC is a non-evaluation step, then the sequence above that violated standardization and soundness when GC was an evaluation step is no longer a problem:

$$[l \mapsto \lambda y.(\lambda x.2 \mathbin{@} \lambda z.h), h \mapsto 3] \circ\!\xrightarrow{(comp-ev)}_{\mathcal{C}} [l \mapsto \lambda y.2, h \mapsto 3] \circ\!\xrightarrow{(GC)}_{\mathcal{C}} [l \mapsto \lambda y.2]$$

Both reduction steps are non-evaluation steps, and therefore the sequence is standard (recall that a sequence is standard if it is of the form $\Longrightarrow^{*}\circ\!\!\longrightarrow^{*}$). We show (see appendix E) that the calculus with the new GC rule has standardization.

However, the change in the GC rule caused a change in classification. Observe that, according to the classification of [MT00], in the example above the GC step changes classification:

$$\begin{array}{lcll} Cl_{\mathcal{C}}([l \mapsto \lambda y.2, h \mapsto 3]) & = & [l \mapsto \mathbf{abs}, h \mapsto \mathbf{const}(3)], & \text{(as in [MT00])} \\ Cl_{\mathcal{C}}([l \mapsto \lambda y.2]) & = & [l \mapsto \mathbf{abs}]. & \end{array}$$

One of the requirements of our framework is that a non-evaluation step does not change classification, i.e. if $D \circ\!\!\longrightarrow_{\mathcal{C}} D'$, then $Cl_{\mathcal{C}}(D) = Cl_{\mathcal{C}}(D')$. The new classification of modules fixes the problem:

$$Cl_{\mathcal{C}}(D) = \begin{cases} \mathbf{evaluatable}_{\mathcal{C}} & \text{if there exists } D' \text{ s.t. } D \Longrightarrow_{\mathcal{C}} D' \\ [v_i \overset{n}{\underset{i=1}{\mapsto}} Cl_{\mathcal{T}}(V_i)] & \text{if } D = [v_i \overset{n}{\underset{i=1}{\mapsto}} V_i, h_j \overset{m}{\underset{j=1}{\mapsto}} V'_j], \\ \mathbf{error}_{\mathcal{C}} & \text{otherwise} \end{cases}$$

Hidden labels are not exposed in the new classification, and we have

$$Cl_{\mathcal{C}}([l \mapsto \lambda y.2, h \mapsto 3]) \quad = \quad Cl_{\mathcal{C}}([l \mapsto \lambda y.2]) \quad = \quad [l \mapsto \mathbf{abs}]$$

Since GC step removes hidden labels bound to values, it clearly does not change classification of the module.

2.6.4 Calculi $\mathcal{C}_{GC}$ and $\mathcal{C}_{GC}\backslash\alpha$

Let $\mathcal{C}_{GC}$ be the calculus $\mathcal{C}$ (as defined on Figure 1 and in section 2.4) with the rule (GC-nev) as above added to the rules (comp-nev) and (subst-nev) for a non-evaluation step. We use notations $\longrightarrow_{\mathcal{C}_{GC}}$, $\Longrightarrow_{\mathcal{C}_{GC}}$, etc. for the reductions of the new calculus. Since terms, contexts, and classification of the new calculus are the same as in $\mathcal{C}$, we use the $\mathcal{C}$ notations for those. We also keep the same definitions of $\alpha_{\mathcal{C}}$-equivalence classes of modules.

We extend one step $\alpha_{\mathcal{C}}$-renaming to subterm occurrences of the form $(\mathbb{M}, D)$ as follows:

Definition 2.122 ($\alpha_{\mathcal{C}}$-Renaming of Subterm Occurrences $(\mathbb{M}, D)$). Let $\mathbb{M} = [l_i \overset{n}{\underset{i=1}{\mapsto}} M_i, \Box]$, $D = [k_j \overset{m}{\underset{j=1}{\mapsto}} N_j]$, $D' = [k_j \overset{m}{\underset{j=1}{\mapsto}} N'_j]$, $\mathbb{M}' = [l'_i \overset{n}{\underset{i=1}{\mapsto}} M'_i, \Box]$.

We say that $(\mathbb{M}, D)$ reduces to $(\mathbb{M}', D')$ by a one-step $\alpha_{\mathcal{C}}$-renaming with the $\alpha_{\mathcal{C}}$-redex (h, h'), written $(\mathbb{M}, D) \xrightarrow{(h,h')}_{\alpha_{\mathcal{C}}} (\mathbb{M}', D')$, if $h \in BL(\mathbb{M}\{D\}) \cap \mathbf{Hidden}$, $h' \notin Hid(\mathbb{M}\{D\})$, $M'_i = M_i[h := h']$ for all $1 \leq i \leq n$, $N'_j = N_j[h := h']$ for all $1 \leq j \leq m$, and

- either $l_{i_0} = h$ for some $1 \leq i_0 \leq n$, $l'_{i_0} = h'$, $l'_i = l_i$ for $i \neq i_0$, $1 \leq i \leq n$, and $k'_j = k_j$ for all $1 \leq j \leq m$,
- or $k_{j_0} = h$ for some $1 \leq j_0 \leq m$, $k'_{j_0} = h'$, $k'_j = k_j$ for $j \neq j_0$, $1 \leq j \leq m$, and $l'_i = l_i$ for all $1 \leq i \leq n$.

We say that $(\mathbb{M}, D)$ reduces to $(\mathbb{M}', D')$ by a one-step $\alpha_{\mathcal{C}}$-renaming with the $\alpha_{\mathcal{C}}$-redex $(\mathbb{D}, x, y)$, written $(\mathbb{M}, D) \xrightarrow{(\mathbb{D},x,y)}_{\alpha_{\mathcal{C}}} (\mathbb{M}', D')$ if $\mathbb{D} = \mathbb{M}_1\{[l \mapsto \mathbb{C}]\}$, and

- either $l = l_{i_0}$ for some $1 \leq i_0 \leq n$, $M_{i_0} \xrightarrow{(\mathbb{C},x,y)}_{\alpha_{\mathcal{T}}} M'_{i_0}$, and $M'_i = M_i$ for all $i \neq i_0$, $1 \leq i \leq n$, and $N'_j = N_j$ for all $1 \leq j \leq m$,
- or $l = k_{j_0}$ for some $1 \leq j_0 \leq m$, $N_{j_0} \xrightarrow{(\mathbb{C},x,y)}_{\alpha_{\mathcal{T}}} N'_{j_0}$, and $N'_j = N_j$ for all $j \neq j_0$, $1 \leq j \leq m$, and $M'_i = M_i$ for all $1 \leq i \leq n$.

We define $\alpha_{\mathcal{C}}$-equivalence classes of such subterm occurrences in a traditional way, and use a meta-variable $Sub^{\mathcal{C},\mathcal{C}}_{\alpha_{\mathcal{C}}}$ to range over these classes. □

Note that the definition implies that $\mathbb{M}\{D\} \xrightarrow{(h,h')}_{\alpha_{\mathcal{C}}} \mathbb{M}'\{D'\}$ in the first case and $\mathbb{M}\{D\} \xrightarrow{(\mathbb{D},x,y)}_{\alpha_{\mathcal{C}}} \mathbb{M}'\{D'\}$ in the second.

Recall that $=$ on subterm occurrences denotes componentwise equality. We can show the following:

Lemma 2.123 ($\alpha_{\mathcal{C}}$-Renaming Preserves Distinctness of Subterm Occurrences). *Suppose* $\mathbb{M}_1\{D_1\} = \mathbb{M}_2\{D_2\}$, $(\mathbb{M}_1, D_1) \neq (\mathbb{M}_2, D_2)$, $(\mathbb{M}_1, D_1) \xrightarrow{(h,h')}_{\alpha_{\mathcal{C}}} (\mathbb{M}'_1, D'_1)$, *and* $(\mathbb{M}_2, D_2) \xrightarrow{(h,h')}_{\alpha_{\mathcal{C}}} (\mathbb{M}'_2, D'_2)$ *(respectively* $(\mathbb{M}_1, D_1) \xrightarrow{(\mathbb{D},x,y)}_{\alpha_{\mathcal{C}}} (\mathbb{M}'_1, D'_1)$ *and* $(\mathbb{M}_2, D_2) \xrightarrow{(\mathbb{D},x,y)}_{\alpha_{\mathcal{C}}} (\mathbb{M}'_2, D'_2)$*), then* $(\mathbb{M}'_1, D'_1) \neq (\mathbb{M}'_2, D'_2)$. □

Proof. Given a concrete module D, its subterm occurrence $(\mathbb{M}_1, D_1)$ is uniquely determined by the set of labels of D_1. Since $(\mathbb{M}_1, D_1) \neq (\mathbb{M}_2, D_2)$, their sets of labels differ from each other. In the case of $\alpha_{\mathcal{C}}$-renaming $\xrightarrow{(\mathbb{D},x,y)}_{\alpha_{\mathcal{C}}}$ these sets of labels do not change, therefore they still differ after the $\alpha_{\mathcal{C}}$-renaming. In the case of $\alpha_{\mathcal{C}}$-renaming $\xrightarrow{(h,h')}_{\alpha_{\mathcal{C}}}$ the sets of labels may change, but clearly they will still differ from each other. □

We also show the following:

Lemma 2.124 ($\alpha_{\mathcal{C}}$-Renaming Preserves (GC) Redexes). *If* $D_1 \circ\!\!\xrightarrow{(\mathbb{M},D)}_{\mathcal{C}} D_2$ *and* $D_1 \xrightarrow{(h,h')}_{\alpha_{\mathcal{C}}} D'_1$ *(respectively* $D_1 \xrightarrow{(\mathbb{D},x,y)}_{\alpha_{\mathcal{C}}} D'_1$*), then there exists* $D'_2 \in {}_{\alpha_{\mathcal{C}}}\langle D_2\rangle$ *such that* $D'_1 \circ\!\!\xrightarrow{(\mathbb{M}',D')}_{\mathcal{C}} D'_2$*, where* $(\mathbb{M}, D) \xrightarrow{(h,h')}_{\alpha_{\mathcal{C}}} (\mathbb{M}', D')$ *(respectively* $(\mathbb{M}, D) \xrightarrow{(\mathbb{D},x,y)}_{\alpha_{\mathcal{C}}} (\mathbb{M}', D')$*).* □

Proof. $(\mathbb{M}, D)$ is a (GC) redex, hence $D = [h_j \overset{m}{\underset{j=1}{\mapsto}} V_j]$, and $BL(D) \cap FL(\mathbb{M}) = \emptyset$.

If $D_1 \xrightarrow{(\mathbb{D},x,y)}_{\alpha_{\mathcal{C}}} D'_1$, then $BL(D) = BL(D')$, and by part 1 of lemma 2.35 $FL(\mathbb{M}') = FL(\mathbb{M})$. Let $\mathbb{D} = \mathbb{M}_1\{l \mapsto \mathbb{C}\}$, where $\mathbb{M}_1 = [l_i \overset{n}{\underset{i=1}{\mapsto}} M_i]$. If $l \in BL(D)$, then $\mathbb{M}' = \mathbb{M}$, and therefore $D'_2 = \mathbb{M}'\{[]\} = \mathbb{M}\{[]\} = D_2$. Otherwise l is bound in $\mathbb{M}$. In this case let $\mathbb{M}'_1 = [l'_j \overset{m}{\underset{j=1}{\mapsto}} M'_j]$, where $l'_i \mapsto M'_i \in l'_j \overset{m}{\underset{j=1}{\mapsto}} M'_j$ if and only if $l'_i \mapsto M'_i \in l_i \overset{n}{\underset{i=1}{\mapsto}} M_i$ and $l'_i \notin BL(D)$, i.e. $\mathbb{M}'_1$ includes all bindings of $\mathbb{M}_1$ except for those in D. Then $D_2 = \mathbb{M}'_1\{[l \mapsto M']\} \xrightarrow{(\mathbb{M}'_1\{\mathbb{C}\},x,y)}_{\alpha_{\mathcal{C}}} \mathbb{M}'_1\{[l \mapsto M]\} = D'_2$, where $M \xrightarrow{(\mathbb{C},x,y)}_{\alpha_{\mathcal{T}}} M'$.

Suppose $D_1 \xrightarrow{(h,h')}_{\alpha_{\mathcal{C}}} D'_1$. Let $(\mathbb{M}, D) \xrightarrow{(h,h')}_{\alpha_{\mathcal{C}}} (\mathbb{M}', D')$. If $h \in BL(D)$, then $BL(D') = BL(D)\backslash\{h\}\cup\{h'\}$, and since $h \notin FL(\mathbb{M})$ and h is not bound in $\mathbb{M}$, we have $\mathbb{M}' = \mathbb{M}$, and $D_2 = \mathbb{M}\{[]\} = \mathbb{M}'\{[]\} = D'_2$. If $h \notin BL(D)$, then $\mathbb{M} = [h \mapsto M_0, l_i \overset{n}{\underset{i=1}{\mapsto}} M_i]$. Then $D_2 = [h \mapsto M_0, l_i \overset{n}{\underset{i=1}{\mapsto}} M_i] \xrightarrow{(h,h')}_{\alpha_{\mathcal{C}}} [h' \mapsto M'_0, l'_i \overset{n}{\underset{i=1}{\mapsto}} M'_i] = D'_2$, where $M'_j = M_j[h := h']$, $0 \leq j \leq n$. □

We extend the definition of the reduction $\longrightarrow_{\mathcal{C}\backslash\alpha}$ on $\alpha_{\mathcal{C}}$-equivalence classes of modules to the reduction $\longrightarrow_{\mathcal{C}_{GC}\backslash\alpha}$ which includes a (GC) redex.

Definition 2.125 (Relation $\longrightarrow_{\mathcal{C}_{GC}\backslash\alpha}$). $D_{\alpha_{\mathcal{C}}} \circ\!\!\xrightarrow{Sub^{\mathcal{C},\mathcal{C}}_{\alpha_{\mathcal{C}}}}_{\mathcal{C}_{GC}\backslash\alpha} D'_{\alpha_{\mathcal{C}}}$ if there exists $D \in D_{\alpha_{\mathcal{C}}}$, $D' \in D'_{\alpha_{\mathcal{C}}}$, and $(\mathbb{M}_1, D_1) \in Sub^{\mathcal{C},\mathcal{C}}_{\alpha_{\mathcal{C}}}$ such that $D \circ\!\!\xrightarrow{(\mathbb{M}_1,D_1)}_{\mathcal{C}_{GC}} D'$.

We also lift definition of $\longrightarrow_{\mathcal{C}\backslash\alpha}$ to the calculus $\mathcal{C}_{GC}$:

- $D_{\alpha_{\mathcal{C}}} \xRightarrow{Sub_{\alpha_{\mathcal{C}}}}_{\mathcal{C}_{GC}\backslash\alpha} D'_{\alpha_{\mathcal{C}}}$ if $D_{\alpha_{\mathcal{C}}} \xRightarrow{Sub_{\alpha_{\mathcal{C}}}}_{\mathcal{C}\backslash\alpha} D'_{\alpha_{\mathcal{C}}}$,
- $D_{\alpha_{\mathcal{C}}} \circ\!\!\xrightarrow{Sub_{\alpha_{\mathcal{C}}}}_{\mathcal{C}_{GC}\backslash\alpha} D'_{\alpha_{\mathcal{C}}}$ if $D_{\alpha_{\mathcal{C}}} \circ\!\!\xrightarrow{Sub_{\alpha_{\mathcal{C}}}}_{\mathcal{C}\backslash\alpha} D'_{\alpha_{\mathcal{C}}}$.

□

As in the calculus $\mathcal{C}\backslash\alpha$, $\longrightarrow_{\mathcal{C}_{GC}\backslash\alpha}$ is not confluent, but $\Longrightarrow_{\mathcal{C}_{GC}\backslash\alpha}$ is. It is also the case that, as in $\mathcal{C}\backslash\alpha$, if a module in $\mathcal{C}_{GC}\backslash\alpha$ has an evaluation normal form, then it cannot diverge w.r.t. evaluation. These properties of $\Longrightarrow_{\mathcal{C}_{GC}\backslash\alpha}$ are inherited from $\Longrightarrow_{\mathcal{C}\backslash\alpha}$ (in fact, the two relations are the same relation by definition 2.125 above).

We show in appendix E that $\mathcal{C}_{GC}\backslash\alpha$ has standardization.

2.6.5 Calculi $\mathcal{L}_{GC}$ and $\mathcal{L}_{GC}\backslash\alpha$

Similarly to $\mathcal{C}_{GC}$, we define the calculus $\mathcal{L}_{GC}$ of linking expressions that supports module-level (GC). We extend the (mod-nev) rule of $\mathcal{L}$ as follows:

$$\mathbb{L}\{H\} \circ\!\!\longrightarrow_{\mathcal{L}_{GC}} \mathbb{L}\{H'\}, \quad \begin{array}{l}\text{where } H \longrightarrow_{\mathcal{C}_{GC}} H' \\ \text{and } \mathbb{L} \neq \Box \text{ or } H \circ\!\!\longrightarrow_{\mathcal{C}_{GC}} H'\end{array} \qquad \text{(mod-nev-GC)}$$

Note that $H \circ\!\!\longrightarrow_{\mathcal{C}_{GC}} H'$ includes the GC reduction.

Recall that GC is a non-evaluation step, therefore we do not need to change (mod-ev) rule of $\mathcal{L}$, so by definition in all other cases $\Longrightarrow_{\mathcal{L}_{GC}} = \Longrightarrow_{\mathcal{L}}$ and $\circ\!\!\longrightarrow_{\mathcal{L}_{GC}} = \circ\!\!\longrightarrow_{\mathcal{L}}$.

To lift GC redexes to the linking level, we introduce the following set of contexts: contexts which are filled with modules so that the result of the filling is a linking expression.

Definition 2.126. Let $\mathbf{Context}_{\mathcal{C},\mathcal{L}}$ denote the set of contexts $\{\mathbb{L}\{\mathbb{M}\} \mid \mathbb{L} \in \mathbf{Context}_{\mathcal{L},\mathcal{L}}, \mathbb{M} \in \mathbf{Context}_{\mathcal{C},\mathcal{C}}\}$, and let $\mathbb{J}$ range over $\mathbf{Context}_{\mathcal{C},\mathcal{L}}$. The result of filling a context $\mathbb{J} = \mathbb{L}\{\mathbb{M}\}$ with a module $D \in \mathbf{Term}_{\mathcal{C}}$ is defined as $\mathbb{L}\{\mathbb{M}\{D\}\}$ if $\mathbb{M}\{D\} \in \mathbf{HTerm}_{\mathcal{C}}$, otherwise it is undefined. Note that, similarly to contexts $\mathbb{F} = \mathbb{L}\{\mathbb{D}\} \in \mathbf{Context}_{\mathcal{T},\mathcal{L}}$, for every $\mathbb{J} \in \mathbf{Context}_{\mathcal{C},\mathcal{L}}$ there exists a unique pair $\mathbb{L}, \mathbb{M}$ such that $\mathbb{J} = \mathbb{L}\{\mathbb{M}\}$. $\Box$

Using the newly introduced contexts $\mathbb{J}$, we can represent GC module redexes at the linking level as $(\mathbb{J}, D)$, where $\mathbb{J} = \mathbb{L}\{\mathbb{M}\}$ and $(\mathbb{M}, D)$ is a GC redex in $\mathcal{C}_{GC}$.

We extend the notion of $\alpha_{\mathcal{L}}$-renaming to subterm occurrences corresponding to GC redexes, and show that such redexes are preserved by $\alpha_{\mathcal{L}}$-renaming.

Definition 2.127 ($\alpha_{\mathcal{L}}$-Renaming of subterm occurrences $(\mathbb{L}, D)$). Let $\mathbb{J} = \mathbb{L}\{\mathbb{M}\}$, $\mathbb{J}' = \mathbb{L}'\{\mathbb{M}'\}$.

- A subterm occurrence $(\mathbb{J}, D)$ reduces to a subterm occurrence $(\mathbb{J}', D')$ by a one-step $\alpha_{\mathcal{L}}$-renaming with the $\alpha_{\mathcal{L}}$-redex $(\mathbb{F}, x, y)$, where $\mathbb{F} = \mathbb{L}_1\{\mathbb{D}\}$ if
  - either $\mathbb{L} = \mathbb{L}'$ and $(\mathbb{M}, D) \xrightarrow{(\mathbb{D},x,y)}_{\alpha_{\mathcal{C}}} (\mathbb{M}', D')$,
  - or $(\mathbb{M}, D) = (\mathbb{M}', D')$ and there exists ${}^2\mathbb{L} \in \mathbf{Context}_{\mathcal{L}\times\mathcal{L},\mathcal{L}}$ such that $\mathbb{L} \xrightarrow{({}^2\mathbb{L}\{\mathbb{D}\},x,y)}_{\alpha_{\mathcal{C}}} \mathbb{L}'$ and $\mathbb{L}_1 = {}^2\mathbb{L}\{\Box, \mathbb{M}\{D\}\}$ or $\mathbb{L}_1 = {}^2\mathbb{L}\{\mathbb{M}\{D\}, \Box\}$.
- A subterm occurrence $(\mathbb{J}, D)$ reduces to a subterm occurrence $(\mathbb{J}', D')$ by a one-step $\alpha_{\mathcal{L}}$-renaming with the $\alpha_{\mathcal{L}}$-redex $(\mathbb{L}_1, h, h')$ if
  - either $\mathbb{L} = \mathbb{L}' = \mathbb{L}_1$ and $(\mathbb{M}, D) \xrightarrow{(h,h')}_{\alpha_{\mathcal{C}}} (\mathbb{M}', D')$,
  - or $(\mathbb{M}, D) = (\mathbb{M}', D')$ and there exists ${}^2\mathbb{L} \in \mathbf{Context}_{\mathcal{L}\times\mathcal{L},\mathcal{L}}$ such that $\mathbb{L} \xrightarrow{({}^2\mathbb{L},h,h')}_{\alpha_{\mathcal{C}}} \mathbb{L}'$ and $\mathbb{L}_1 = {}^2\mathbb{L}\{\Box, \mathbb{M}\{D\}\}$ or $\mathbb{L}_1 = {}^2\mathbb{L}\{\mathbb{M}\{D\}, \Box\}$.
- A subterm occurrence $(\mathbb{J}, D)$ reduces to a subterm occurrence $(\mathbb{J}', D')$ by a one-step $\alpha_{\mathcal{L}}$-renaming with the $\alpha_{\mathcal{L}}$-redex $(\mathbb{L}_1, I, I')$ if $\mathbb{J}\{D\} \xrightarrow{(\mathbb{L}_1,I,I')}_{\alpha_I} \mathbb{J}'\{D'\}$. Note that in this case $(\mathbb{M}, D) = (\mathbb{M}', D')$, since renaming a module identifier does not change any modules in the linking expression.

$\Box$

We use a meta-variable $Sub^{\mathcal{C},\mathcal{L}}_{\alpha_{\mathcal{L}}}$ to range over $\alpha_{\mathcal{L}}$-equivalence classes of subterm occurrences of the form $(\mathbb{J}, D)$.

Similarly to $\mathcal{C}_{GC}$, we can show that $\alpha_{\mathcal{L}}$-renaming of subterm occurrences of the form $(\mathbb{J}, D)$ preserves distinctness of subterm occurrences, as well as GC redexes (see lemmas 2.128 and 2.129 below). We omit the proofs of the lemmas, which are similar to those of lemmas 2.123 and 2.124.

Lemma 2.128 ($\alpha_{\mathcal{L}}$-Renaming Preserves Distinctness of Subterm Occurrences). *Suppose* $\mathbb{J}_1\{D_1\} = \mathbb{J}_2\{D_2\}$, $(\mathbb{J}_1, D_1) \neq (\mathbb{J}_2, D_2)$, *and*

- $(\mathbb{J}_1, D_1) \xrightarrow{(\mathbb{L},h,h')}_{\alpha_\mathcal{L}} (\mathbb{J}'_1, D'_1)$, *and* $(\mathbb{J}_2, D_2) \xrightarrow{(\mathbb{L},h,h')}_{\alpha_\mathcal{L}} (\mathbb{J}'_2, D'_2)$, *or*
- $(\mathbb{J}_1, D_1) \xrightarrow{(\mathbb{F},x,y)}_{\alpha_\mathcal{L}} (\mathbb{J}'_1, D'_1)$ *and* $(\mathbb{J}_2, D_2) \xrightarrow{(\mathbb{F},x,y)}_{\alpha_\mathcal{L}} (\mathbb{J}'_2, D'_2)$, *or*
- $(\mathbb{J}_1, D_1) \xrightarrow{(\mathbb{L},I,I')}_{\alpha_\mathcal{L}} (\mathbb{J}'_1, D'_1)$ *and* $(\mathbb{J}_2, D_2) \xrightarrow{(\mathbb{L},I,I')}_{\alpha_\mathcal{L}} (\mathbb{J}'_2, D'_2)$,

then $(\mathbb{J}'_1, D'_1) \neq (\mathbb{J}'_2, D'_2)$. □

Lemma 2.129 ($\alpha_\mathcal{L}$-Renaming Preserves GC Redexes). *If* $L_1 \circ\!\xrightarrow{(\mathbb{J},D)}_\mathcal{L} L_2$ *and* $L_1 \xrightarrow{(\mathbb{L},h,h')}_{\alpha_\mathcal{L}} L'_1$ *(respectively* $L_1 \xrightarrow{(\mathbb{F},x,y)}_{\alpha_\mathcal{L}} L'_1$ *or* $L_1 \xrightarrow{(\mathbb{L},I,I')}_{\alpha_\mathcal{L}} L'_1$*), then there exists* $L'_2 \in {}_{\alpha_\mathcal{L}}\langle L_2\rangle$ *such that* $L'_1 \circ\!\xrightarrow{(\mathbb{J}',D')}_\mathcal{L} L'_2$*, where* $(\mathbb{J}, D) \xrightarrow{(\mathbb{L},h,h')}_{\alpha_\mathcal{L}} (\mathbb{J}', D')$ *(respectively* $(\mathbb{J}, D) \xrightarrow{(\mathbb{F},x,y)}_{\alpha_\mathcal{L}} (\mathbb{J}', D')$ *or* $(\mathbb{J}, D) \xrightarrow{(\mathbb{L},I,I')}_{\alpha_\mathcal{L}} (\mathbb{J}', D')$*).* □

Similarly to the other calculi, we extend $\longrightarrow_{\mathcal{L}_{GC}}$ to $\alpha_\mathcal{L}$-equivalence classes of linking expressions. As for the calculus $\mathcal{C}_{GC}$, $\longrightarrow_{\mathcal{L}_{GC}\backslash\alpha}$ consists of the new GC reduction and all the reductions of $\longrightarrow_{\mathcal{L}\backslash\alpha}$.

Definition 2.130 (Relation $\longrightarrow_{\mathcal{L}_{GC}\backslash\alpha}$). $L_{\alpha_\mathcal{L}} \circ\!\xrightarrow{Sub^{\mathcal{C},\mathcal{L}}_{\alpha_\mathcal{L}}}_{\mathcal{L}_{GC}\backslash\alpha} L'_{\alpha_\mathcal{L}}$ if there exists $L \in L_{\alpha_\mathcal{L}}$, $D' \in L'_{\alpha_\mathcal{L}}$, and $(\mathbb{J}_1, D_1) \in Sub^{\mathcal{C},\mathcal{L}}_{\alpha_\mathcal{L}}$ such that $L \circ\!\xrightarrow{(\mathbb{J}_1,D_1)}_{\mathcal{L}_{GC}} L'$.

We also lift definition of $\longrightarrow_{\mathcal{L}\backslash\alpha}$ to the calculus $\mathcal{L}_{GC}$:

- $L_{\alpha_\mathcal{L}} \circ\!\xrightarrow{Sub^{\mathcal{T},\mathcal{L}}_{\alpha_\mathcal{C}}}_{\mathcal{L}_{GC}\backslash\alpha} L'_{\alpha_\mathcal{L}}$ if $L_{\alpha_\mathcal{L}} \circ\!\xrightarrow{Sub^{\mathcal{T},\mathcal{L}}_{\alpha_\mathcal{C}}}_{\mathcal{L}\backslash\alpha} L'_{\alpha_\mathcal{L}}$,
- $L_{\alpha_\mathcal{L}} \xRightarrow{Sub^{\mathcal{C},\mathcal{L}}_{\alpha_\mathcal{C}}}_{\mathcal{L}_{GC}\backslash\alpha} L'_{\alpha_\mathcal{L}}$ if $L_{\alpha_\mathcal{L}} \xRightarrow{Sub^{\mathcal{C},\mathcal{L}}_{\alpha_\mathcal{C}}}_{\mathcal{L}\backslash\alpha} L'_{\alpha_\mathcal{L}}$,
- $L_{\alpha_\mathcal{L}} \xRightarrow{Sub^{\mathcal{T},\mathcal{L}}_{\alpha_\mathcal{C}}}_{\mathcal{L}_{GC}\backslash\alpha} L'_{\alpha_\mathcal{L}}$ if $L_{\alpha_\mathcal{L}} \xRightarrow{Sub^{\mathcal{T},\mathcal{L}}_{\alpha_\mathcal{C}}}_{\mathcal{L}\backslash\alpha} L'_{\alpha_\mathcal{L}}$.

□

Recall that in section 2.5.4 we have introduced the notion a canonical expression, i.e. a linking expression in which all hidden labels of all modules are distinct, and a canonical reduction $\underset{\sim}{\longrightarrow}_\mathcal{L}$: a subset of $\longrightarrow_\mathcal{L}$ such that $L \underset{\sim}{\longrightarrow}_\mathcal{L} L'$ implies that both L and L' are canonical. The definition of a canonical reduction trivially extends to GC reductions, due to the fact that if $L \circ\!\!\longrightarrow_{\mathcal{L}_{GC}} L'$ by the (GC) rule and L is canonical, then L' is canonical, because the reduction has removed some hidden labels, but clearly has not created new conflicts between hiddens. Therefore we define canonical GC as follows:

Definition 2.131 (Canonical GC). $\hat{L} \circ\!\!\underset{\sim}{\longrightarrow}_{\mathcal{L}_{GC}} L$ by the rule (GC) iff $\hat{L} \circ\!\!\longrightarrow_{\mathcal{L}_{GC}} L$ and $\hat{L}$ is canonical. □

We augment theorem 2.109 with the following case for GC reduction:

Lemma 2.132. *For any* $\alpha_\mathcal{L}$*-equivalence class* $L_{\alpha_\mathcal{L}}$ $L_{\alpha_\mathcal{L}} \circ\!\xrightarrow{Sub^{\mathcal{C},\mathcal{L}}_{\alpha_\mathcal{L}}}_{\mathcal{L}_{GC}\backslash\alpha} L'_{\alpha_\mathcal{L}}$ *if and only if for every canonical expression* $\hat{L} \in L_{\alpha_\mathcal{L}}$ *there exist* $\hat{L}' \in L'_{\alpha_\mathcal{L}}$ *and* $(\mathbb{J}, D) \in Sub^{\mathcal{C},\mathcal{L}}_{\alpha_\mathcal{L}}$ *such that* $\hat{L} \circ\!\underset{\sim}{\xrightarrow{(\mathbb{J},D)}}_{\mathcal{L}_{GC}} \hat{L}'$. □

The lemma together with theorem 2.109 imply that if there exists a $\mathcal{L}_{GC}\backslash\alpha$ reduction of one $\alpha_\mathcal{L}$-equivalence class to another, then there exist a canonical reduction between canonical representatives of the two classes.

As the corresponding relations in $\mathcal{L}\backslash\alpha$, $\longrightarrow_{\mathcal{L}_{GC}\backslash\alpha}$ is not confluent, but $\Longrightarrow_{\mathcal{L}_{GC}\backslash\alpha}$ is, and we are able to show that if a linking expression has an evaluation normal form, then there is no diverging evaluation path starting from that expression.

We also show that if $Outcome_{\mathcal{C}_{GC}\backslash\alpha}(L) \neq \mathbf{error}_\mathcal{L}$, then there exists H such that $L \Longrightarrow^*_{\mathcal{L}_{GC}\backslash\alpha} H$.

The following theorem, which we prove in appendix E, states the staging property (see theorem 2.116) extended to $\mathcal{L}_{GC}\backslash\alpha$. The theorem below takes into account GC redexes introduced in this section.

Theorem 2.133 (Staging in $\mathcal{L}_{GC}\backslash\alpha$). *Given a sequence* $L_1 \overset{S}{\Longrightarrow}^*_{\mathcal{L}_{GC}\backslash\alpha} L_2$*, there exists* L' *such that* $L_1 \overset{S_1}{\Longrightarrow}^*_{\mathcal{L}_{GC}\backslash\alpha} L' \overset{S_2}{\Longrightarrow}^*_{\mathcal{L}\backslash\alpha} L_2$*, where* $S = S_1; S_2$*,* S_1 *consists of linking redexes of the form* $(\mathbb{L}, K)$*, and* S_2 *consists of module redexes of the form* $(\mathbb{F}, M)$ *and* $(\mathbb{J}, D)$*.* □

We also show in appendix E that $\mathcal{L}_{GC}\backslash\alpha$ has the standardization property.

3 Computational Soundness and Meaning Preservation

3.1 The Meaning of Programs

The meaning of a program in a language is often defined by an *operational semantics* for that language. An operational semantics can be expressed in many ways, including: abstract machines [Lan64], small-step transition rules [Plo81], and big-step transition rules [Kah87]. In this study, we follow [FF86], and specify for each level $\mathcal{X}$ of our module calculus a small-step evaluation relation $\Longrightarrow_{\mathcal{X}}$ defined by restricting the contexts for the calculus relation $\longrightarrow_{\mathcal{X}}$ to a restricted set of evaluation contexts (see section 2). Traditionally, such small-step evaluation relations are functions, but for two levels of our calculus ($\mathcal{C}$ and $\mathcal{L}$), the evaluation relation is *not a function* (i.e. a term may evaluate to more than one term), even at the level of α-equivalence classes. Nevertheless, both $\Longrightarrow_{\mathcal{C}\backslash\alpha}$ and $\Longrightarrow_{\mathcal{L}\backslash\alpha}$ are confluent, so if a term has a normal form w.r.t. evaluation, then such a normal form is unique (up to α-renaming). As defined in section 2.2, a normal form w.r.t. evaluation of a term X in a calculus $\mathcal{X}$ is denoted by $Eval_{\mathcal{X}}(X)$.

A classification function $Cl_{\mathcal{X}}$ characterizes a term X w.r.t. an evaluation relation $\Longrightarrow_{\mathcal{X}}$. There may be two situations: either there exists a term Y such that $X \Longrightarrow_{\mathcal{X}} Y$, in which case X is called *evaluatable* and, by definition, $Cl_{\mathcal{X}}(X) = \mathbf{evaluatable}_{\mathcal{X}}$, or $X \in NF_{\Longrightarrow_{\mathcal{X}}}$. In the latter case X belongs to one of the classifications of normal forms of $\mathcal{X}$. Intuitively, these classifications correspond to observable properties of results of evaluation. For instance, in the term calculus $\mathcal{T}$, the constants 3 and 4 are observably distinct. Our classification reflects this fact by mapping 3 to a classification token $\mathbf{const}(3)$ and 4 to a different classification token $\mathbf{const}(4)$. Another kind of result in $\mathcal{T}$ is a function (i.e., a λ-abstraction), whose definition is not observable. The only observable property of a function is its input/output behavior, so two functions cannot be distinguished without applying them. Accordingly, the classification of every function in $\mathcal{T}$ is **abs**. Both constants and λ-abstractions are considered values.

Not all normal forms are values. For instance, a $\mathcal{T}$ eval normal form may have the classification $\mathbf{stuck}(l)$, which designates a term *stuck on a label* l. In any of our calculi, an observable result of a term may have the classification **error**. For example, a term with this classification is $2 \mathbin{@} 3 \in \mathbf{Term}_{\mathcal{T}}$; it cannot be evaluated further, yet it is not a value and not a stuck term[12]. Stuck terms in $\mathcal{T}$ are classified separately from errors because in a module context, a term stuck on a label l may be evaluated further if a substitution step substitutes a value for l. For instance, the term $A + 3$ in the module expression $[B \mapsto A + 3, A \mapsto 4]$ is stuck on A, but after a substitution step, this term rewrites to $4 + 3$, which is evaluatable. In contrast, no progress can be made on the term $2 \mathbin{@} 3$ in any module context.

The classification of values and separation of normal forms w.r.t. evaluation from evaluatables introduced above follows the approach of Plotkin in [Plo75]. Classification of normal forms other than values have been introduced (without using the term "classification" or "classes") by Z. Ariola and M. Felleisen in [AF97] for a call-by-need calculus. Their classification distinguishes between answers (a class of normal form that includes values), terms of the form $\mathbb{E}\{x\}$, where $\mathbb{E}$ is an evaluation context and x is a variable, and terms that can be evaluated further (such terms correspond to our **evaluatable** classification). An example of their third class of term (using our notation) is $(\lambda x.\mathbb{E}\{x\}) \mathbin{@} V$, which rewrites in one step to $(\lambda x.\mathbb{E}\{V\}) \mathbin{@} V$. Intuitively, the subterm $\mathbb{E}\{x\}$ is "stuck on x" in the same way that a $\mathcal{T}$ term can be stuck on a label. An important difference is that $\mathcal{T}$ terms can never be stuck on variables, only labels.

[12] One may notice that the term $2 \mathbin{@} 3$ has a *type* error (2 is in the function position, but does not have a function type), and therefore this term is illegal in a typed calculus. However, adding a type system to the calculus would not completely get rid of the class **error**: consider the term $1/0$.

The meaning of a term X in a calculus $\mathcal{X}$ is given by its *outcome*, $\mathit{Outcome}_{\mathcal{X}}(X)$, which specifies its behavior w.r.t. $\Rightarrow_{\mathcal{X}}$. Unlike, $\mathit{Cl}_{\mathcal{X}}(X)$ which gives the status of a term at the present moment, the outcome characterizes the result of repeatedly applying $\Rightarrow_{\mathcal{X}}$ to X as long as possible. There are two possibilities: X may diverge w.r.t. evaluation or it may reach a normal form. Recall that $\mathit{Outcome}_{\mathcal{X}}(X) = \perp$ in the first case, and $\mathit{Outcome}_{\mathcal{X}}(X) = \mathit{Cl}_{\mathcal{X}}(\mathit{Eval}_{\mathcal{X}}(X))$ in the second. While it is theoretically possible for a term to diverge on one evaluation path and to reach a normal form on another (without violating confluence of the reduction), we have shown that this does not happen for $\Rightarrow_{\mathcal{C}\backslash\alpha}$, $\Rightarrow_{\mathcal{C}_{GC}\backslash\alpha}$, $\Rightarrow_{\mathcal{L}\backslash\alpha}$, or $\Rightarrow_{\mathcal{L}_{GC}\backslash\alpha}$ (see Lemmas 2.62 and 2.113 and properties of $\mathcal{C}_{GC}$ in the end of section 2.6), so the two possibilities are mutually exclusive for the calculi considered here.

Remark 3.1. Note that the same fact does not hold true for general reduction (as opposed to evaluation) paths. For instance, in the term calculus $\mathcal{T}$, suppose that the diverging term $(\lambda x.(x \mathbin{@} x)) \mathbin{@} (\lambda x.(x \mathbin{@} x))$ is denoted by Ω. Then the term $M_{3.1} = (\lambda x.2) \mathbin{@} (\lambda y.\Omega)$ is the source of the terminating reduction sequence $M_{3.1} \longrightarrow_{\mathcal{T}} 2$ and the diverging reduction sequence $M_{3.1} \longrightarrow_{\mathcal{T}} M_{3.1} \longrightarrow_{\mathcal{T}} \ldots$. Note that from every term in the diverging sequence there is a path (in this case by one reduction step) to the normal form 2. $\square$

3.2 Observational Equivalence

The main goal of this work is to show that any two terms related by a sequence of backward and forward reduction steps (including non-evaluation steps) are *observationally equivalent*. That is, when one term is replaced by the other in some context, the meaning (i.e. the outcome) of the context filled with the term is preserved. The definition of observational (or operational, or contextual) equivalence of two terms in a calculus was introduced by Plotkin in [Plo75]. Plotkin presented λ-calculi in which two terms M and N are observationally equivalent if for any context $\mathbb{C}$, $\mathbb{C}\{M\}$ evaluates to a value if and only if $\mathbb{C}\{N\}$ does, and if one of these values is a basic constant c, then so is the other.

We have extended the notion of observational equivalence to the case when the two terms may belong to a calculus other than that of the context. If $\mathcal{X}$ and $\mathcal{X}'$ are the same calculus, then the definition is analogous to the one in [Plo75].

Definition 3.2 (Observational Equivalence). Two terms Y and Z of a calculus $\mathcal{X}'$ are *observationally equivalent* w.r.t. a calculus $\mathcal{X}$, written $Y \cong_{\mathcal{X}} Z$, iff $\mathit{Outcome}_{\mathcal{X}}(\mathbb{X}\{Y\}) = \mathit{Outcome}_{\mathcal{X}}(\mathbb{X}\{Z\})$ for all contexts $\mathbb{X} \in \mathbf{Context}_{\mathcal{X}',\mathcal{X}}$ such that $\mathbb{X}\{Y\}$ and $\mathbb{X}\{Z\}$ are well-formed terms of $\mathcal{X}$. $\square$

Example 3.3. Consider the following module expressions:

$$D_{3.3a} = [F \mapsto \lambda x.(x * a), a \mapsto 2]$$
$$D_{3.3b} = [F \mapsto \lambda y.(y + y)]$$

$D_{3.3a} \cong_{\mathcal{L}} D_{3.3b}$ because the exported F behaves like a doubling function in any linking context, and no linking context can detect the hidden label a. However, $D_{3.3a} \ncong_{\mathcal{C}} D_{3.3b}$, because there are module contexts $\mathbb{M} \in \mathbf{Context}_{\mathcal{C},\mathcal{C}}$ that can distinguish the two modules by the presence of the hidden label a, e.g., $\mathbb{M}_{3.3} = [B \mapsto a + 1, \square]$:

$$\mathit{Outcome}_{\mathcal{C}}(\mathbb{M}_{3.3}\{D_{3.3a}\}) = [B \mapsto \mathbf{const}(3), F \mapsto \mathbf{abs}]$$
$$\mathit{Outcome}_{\mathcal{C}}(\mathbb{M}_{3.3}\{D_{3.3b}\}) = \mathbf{error}_{\mathcal{L}}$$

$\square$

Example 3.4. Suppose that M and N are arbitrary terms in $\mathcal{T}$, and assume that $+$ is commutative on the numbers used in the module language. Is $M + N$ observationally equivalent to $N + M$ w.r.t. $\mathcal{T}$? The answer is no. The sequential left-to-right order of evaluation of operands in primitive applications can detect an observable difference between an error and an infinite loop. As a concrete example, define $M_{3.4} = (2 \mathbin{@} 3)$ and $N_{3.4} = \Omega$, and compare the meaning of $M_{3.4} + N_{3.4}$ and $N_{3.4} + M_{3.4}$ in an empty context in $\mathbf{Context}_{\mathcal{T},\mathcal{T}}$:

$$\mathit{Outcome}_{\mathcal{T}}((2 \mathbin{@} 3) + \Omega) = \mathbf{error}_{\mathcal{L}}$$
$$\mathit{Outcome}_{\mathcal{T}}(\Omega + (2 \mathbin{@} 3)) = \perp$$

So $M_{3.4} + N_{3.4} \ncong_{\mathcal{T}} N_{3.4} + M_{3.4}$. This example can be lifted to the core module calculus and linking calculus to show that $M_{3.4} + N_{3.4} \ncong_{\mathcal{C}} N_{3.4} + M_{3.4}$ and $M_{3.4} + N_{3.4} \ncong_{\mathcal{L}} N_{3.4} + M_{3.4}$. □

EXAMPLE 3.5. Suppose that k and l are arbitrary labels in $\mathcal{T}$, and assume that $+$ is commutative on numbers. Is the $\mathcal{T}\backslash\alpha$ term $l+k$ observationally equivalent to $k+l$ w.r.t. any calculus? For $\mathcal{T}\backslash\alpha$, the answer is no, due to the fact that the classification $\mathbf{stuck}(l)$ is parameterized with respect to the label l:

$$\begin{array}{c} Outcome_{\mathcal{T}\backslash\alpha}(l+k) = \mathbf{stuck}(l) \\ Outcome_{\mathcal{T}\backslash\alpha}(k+l) = \mathbf{stuck}(k) \end{array}$$

Interestingly, the answer is yes for $\mathcal{C}\backslash\alpha$ and $\mathcal{L}\backslash\alpha$. Below, we sketch why $l+k \cong_{\mathcal{C}\backslash\alpha} k+l$. A similar argument shows that $l+k \cong_{\mathcal{L}\backslash\alpha} k+l$.

Suppose that q ranges over $\mathbf{Value}_{\mathcal{T}} \cup \mathbf{Label}$. Given two terms M and N in $\mathbf{Term}_{\mathcal{T}}$, we say that M *rewrites via a swap* to N, written $M \rightharpoonup N$, if $M = \mathbb{C}\{q_1 + q_2\}$ and $M = \mathbb{C}\{q_2 + q_1\}$. Define $\rightleftharpoons$ as the reflexive, transitive closure of $\rightharpoonup$; since $\rightharpoonup$ is clearly symmetric, so is $\rightleftharpoons$. If $M \rightleftharpoons N$, we say that M and N are *swap equivalent*. For example, the following four (nonsensical) terms are pairwise swap equivalent:

$$\begin{array}{ll} k + (\lambda x.(x+2)) & \rightleftharpoons \\ k + (\lambda x.(2+x)) & \rightleftharpoons \\ (\lambda x.(2+x)) + k & \rightleftharpoons \\ (\lambda x.(x+2)) + k. & \end{array}$$

Recall the calculus $\underline{\mathcal{T}}$ that supports marked subterm occurrences (see definition 2.27). We naturally extend $\rightharpoonup$ and $\rightleftharpoons$ to terms in $\mathbf{Term}_{\underline{\mathcal{T}}}$ so that "swapping" preserves the markings of subterms. We say that $(\mathbb{C}, M) \rightleftharpoons (\mathbb{C}', M')$ if $\mathbb{C}\{M\} \rightleftharpoons \mathbb{C}'\{M'\}$ in $\mathcal{T}$, and there exist $(\widetilde{\mathbb{C}}, \widetilde{M})$ and $(\widetilde{\mathbb{C}}', \widetilde{M}')$ in $\underline{\mathcal{T}}$ such that:

- $\widetilde{M}$ and $\widetilde{M}'$ are the only marked subterms in $\widetilde{\mathbb{C}}\{\widetilde{M}\}$ and $\widetilde{\mathbb{C}}'\{\widetilde{M}'\}$, respectively,
- $\widetilde{\mathbb{C}}\{\widetilde{M}\} \rightleftharpoons \widetilde{\mathbb{C}}'\{\widetilde{M}'\}$ in $\underline{\mathcal{T}}$,
- $(\mathbb{C}, M) = |(\widetilde{\mathbb{C}}, \widetilde{M})|$, $(\mathbb{C}', M') = |(\widetilde{\mathbb{C}}', \widetilde{M}')|$ (where $|\ |$ is defined in definition 2.28).

The notion of swap equivalence on terms and subterm occurrences can also be extended to $\mathbf{Term}_{\mathcal{T}\backslash\alpha}$ and $\mathbf{Term}_{\mathcal{C}\backslash\alpha}$.

Given these definitions, we can show the following:

1. If $M_{\alpha_{\mathcal{T}}} \rightleftharpoons M'_{\alpha_{\mathcal{T}}}$ then:
   (a) $M_{\alpha_{\mathcal{T}}} \in \mathbf{Value}_{\mathcal{T}\backslash\alpha}$ iff $M'_{\alpha_{\mathcal{T}}} \in \mathbf{Value}_{\mathcal{T}\backslash\alpha}$;
   (b) if $M_{\alpha_{\mathcal{T}}} \in \mathbf{Value}_{\mathcal{T}\backslash\alpha}$, then $Cl_{\mathcal{T}\backslash\alpha}(M_{\alpha_{\mathcal{T}}}) = Cl_{\mathcal{T}\backslash\alpha}(M'_{\alpha_{\mathcal{T}}})$.

2. If $D_{1\alpha_{\mathcal{C}}} \rightleftharpoons D_{2\alpha_{\mathcal{C}}}$, then $(D_{1\alpha_{\mathcal{C}}} \xrightarrow{Sub_{\alpha_{\mathcal{C}}}}_{\mathcal{C}\backslash\alpha} D_1{}'_{\alpha_{\mathcal{C}}})$ implies $(D_{2\alpha_{\mathcal{C}}} \xrightarrow{Sub'_{\alpha_{\mathcal{C}}}}_{\mathcal{C}\backslash\alpha} D_2{}'_{\alpha_{\mathcal{C}}})$, where $Sub_{\alpha_{\mathcal{C}}} \rightleftharpoons Sub'_{\alpha_{\mathcal{C}}}$ and $D_1{}'_{\alpha_{\mathcal{C}}} \rightleftharpoons D_2{}'_{\alpha_{\mathcal{C}}}$. This can be shown by case analysis on the form of the redex $Sub_{\alpha_{\mathcal{C}}}$. The fact that the module calculus is call-by-value is critical in several spots. For instance, if $Sub_{\alpha_{\mathcal{C}}}$ is a substitution redex ${}_{\alpha_{\mathcal{C}}}\langle(\mathbb{G}, l)\rangle$, then $Sub'_{\alpha_{\mathcal{C}}} = {}_{\alpha_{\mathcal{C}}}\langle(\mathbb{G}', l')\rangle$, $\mathbb{G}\{l\} \downarrow l = V \in \mathbf{Value}_{\mathcal{T}}$, $\mathbb{G}'\{l'\} \downarrow l' = V' \in \mathbf{Value}_{\mathcal{T}}$, and ${}_{\alpha_{\mathcal{C}}}\langle\mathbb{G}\{V\}\rangle \rightleftharpoons {}_{\alpha_{\mathcal{C}}}\langle\mathbb{G}'\{V'\}\rangle$. The final swap equivalence would not generally hold if the values V, V' were replaced by arbitrary terms N, N'. Here is a counterexample:

$$\begin{array}{c} \mathbb{G} = [A \mapsto \Box + 1, B \mapsto 2 * 3] \\ \mathbb{G}' = [A \mapsto 1 + \Box, B \mapsto 2 * 3] \\ \mathbb{G}\{B\} = [A \mapsto B + 1, B \mapsto 2 * 3] \rightleftharpoons [A \mapsto 1 + B, B \mapsto 2 * 3] = \mathbb{G}'\{B\} \\ \mathbb{G}\{2 * 3\} = [A \mapsto (2 * 3) + 1, B \mapsto 2 * 3] \not\rightleftharpoons [A \mapsto 1 + (2 * 3), B \mapsto 2 * 3] = \mathbb{G}'\{2 * 3\} \end{array}$$

   The final swap equivalence fails to hold because $(2 * 3) \notin \mathbf{Value}_{\mathcal{T}\backslash\alpha} \cup \mathbf{Label}$. Note that the swap equivalence is restored if $(2 * 3)$ is replaced by the value 6.

3. If $D_{1\alpha_{\mathcal{C}}} \rightleftharpoons D_{2\alpha_{\mathcal{C}}}$ then $Cl_{\mathcal{C}\setminus\alpha}(D_{1\alpha_{\mathcal{C}}}) = Cl_{\mathcal{C}\setminus\alpha}(D_{2\alpha_{\mathcal{C}}})$. This follows from (1) and the fact that $D_{1\alpha_{\mathcal{C}}}$ is evaluatable iff $D_{2\alpha_{\mathcal{C}}}$ is evaluatable (a consequence of (2)). The key fact here is that module classification hides any distinctions about the label on which a term component may be stuck via the catch-all classification $\mathbf{error}_{\mathcal{C}}$. For example, $Cl_{\mathcal{C}\setminus\alpha}([A \mapsto l + k]) = Cl_{\mathcal{C}\setminus\alpha}([A \mapsto k + l]) = \mathbf{error}_{\mathcal{C}}$.

4. For any module context $\mathbb{D}$, clearly ${}_{\alpha_{\mathcal{C}}}\langle\mathbb{D}\{l + k\}\rangle \rightleftharpoons {}_{\alpha_{\mathcal{C}}}\langle\mathbb{D}\{k + l\}\rangle$. By (2) and (3), $Outcome_{\mathcal{C}\setminus\alpha}({}_{\alpha_{\mathcal{C}}}\langle\mathbb{D}\{l + k\}\rangle) = Outcome_{\mathcal{C}\setminus\alpha}({}_{\alpha_{\mathcal{C}}}\langle\mathbb{D}\{k + l\}\rangle)$,

□

Remark 3.6. Whether or not two terms are considered observationally equivalent depends on the the granularity of the observables $\mathbf{Obs}_{\mathcal{X}}$ and the classification function $Cl_{\mathcal{X}}$. For instance, consider an alternative classification function for terms, $Cl'_{\mathcal{T}}(M)$, defined as follows:

$$\begin{array}{llllll} \mathbf{const}(c) & \text{if } M = c & \mathbf{abs} & \text{if } M = \lambda x.N & \mathbf{evaluatable}_{\mathcal{T}} & \text{if } M = \mathbb{E}\{R\} \\ \mathbf{var} & \text{if } M = x & \perp_{\mathcal{T}} & \text{if } M = \mathbb{E}\{l\} & \perp_{\mathcal{T}} & \text{otherwise} \end{array}$$

This alternative classification function maps all non-value $\mathcal{T}$ eval normal forms, including those stuck on a particular label, to the same token $\perp_{\mathcal{T}}$ used for denoting divergence of outcome. Using this classification function, the terms $(2 @ 3) + \Omega$ and $\Omega + (2 @ 3)$ are observationally equivalent because there is no observable distinction between divergence and errors. It is possible to define a partial order on classification functions indicating that one classification function refines another, but to keep things simple we consider only a single classification function for each calculus.

□

Remark 3.7. The ability to place terms into arbitrary contexts in order to distinguish them allows us to use a classification that does not completely characterize the term. For instance, even though $Cl_{\mathcal{T}}(\lambda x.x) = Cl_{\mathcal{T}}(\lambda y.5) = \mathbf{abs}$, we can distinguish the two λ-abstractions by placing them in a context $\square @ 3$: $\lambda x.x @ 3 \Rightarrow_{\mathcal{T}} 3$, $\lambda y.5 @ 3 \Rightarrow_{\mathcal{T}} 5$, and $Cl_{\mathcal{T}}(3) = \mathbf{const}(3) \neq \mathbf{const}(5) = Cl_{\mathcal{T}}(5)$. Therefore, $\lambda x.x \not\cong_{\mathcal{C}} \lambda y.5$. Indeed, in some systems, the only two observable outcomes are termination and divergence and appropriate contexts are used to distinguish what would normally be though of as different values (e.g., [Abr90, PS98]). . □

Remark 3.8. Using definition 3.2, we have been able to simplify the classification of modules from that of [MT00], because placing modules into arbitrary link-level contexts is sufficient to be able to distinguish between any two modules that "behave differently". In [MT00] the classification of a module is a class token that incorporates the classifications of all module components, i.e. by definition $Cl_{\mathcal{C}}([l_i \overset{n}{\underset{i=1}{\mapsto}} M_i]) = [l_i \overset{n}{\underset{i=1}{\mapsto}} Cl_{\mathcal{T}}(M_i)]$. The simplified classification (see the definition of $Cl_{\mathcal{C}}$ in section 2.4) has classifications $\mathbf{evaluatable}_{\mathcal{C}}$ and $\mathbf{error}_{\mathcal{C}}$ for evaluatable modules and errors respectively, and exposes only visible components of a module value. (However, for a module value hidden components are required to be bound to values.) If hidden components contribute to the behavior of the module, we can find a context that exposes this difference. For instance, two module values $[A \mapsto \lambda x.h, h \mapsto \lambda y.2]$ and $[A \mapsto \lambda x.h, h \mapsto \lambda y.y]$, both with a classification $[A \mapsto \mathbf{abs}]$, may be distinguished by placing them in the context $\square \oplus [B \mapsto A @ 3 @ 5]$. The classification $\mathbf{error}_{\mathcal{C}}$ refers to all modules that are not evaluatable and not values. The name $\mathbf{error}_{\mathcal{C}}$ does not imply that all modules in this category are errors, but only that obtaining such a module as a result of evaluation (i.e. at the point where one would expect a module value) would be an error. For instance, the module $[A \mapsto B]$ is not a reasonable final result of evaluating a module or a linking expression (because it depends on an external label B), but using it as a part of a linking expression is completely legitimate: $[A \mapsto B] \oplus [B \mapsto 5]$. By placing an "error" module into a linking context we can expose its behavior.

The classification of linking expressions distinguishes between three categories: evaluatables, errors, and module normal forms. In the linking calculus, evaluation of an expression leads to a module, unless it reaches a linking error (by the staging property; see Theorem 2.116). Therefore evaluation of a linking expression may result in one of the following: a link-level error (such as a conflict between visible labels of the arguments of $\oplus$), or a module, which may, in turn, diverge, reach a module value, or reach a module error. The observational equivalence of two modules or two linking expressions means that, being placed in an arbitrary link-level context, they cause the same behavior of the linking expression w.r.t. evaluation. If the result of evaluation is a module value, then the two values must have the same classification in $\mathcal{C}$. □

3.3 Program Transformations and Meaning Preservation

It is often desirable to transform one program into another one that has the same meaning as the original but which is "better" in some dimension. For example, the transformed program might be smaller than the original, or it might consume less time or space when executed, or it might be in a form that allows certain properties to be verified more easily.

For the purposes of the present paper, we shall introduce a very simple notion of program transformation:

Definition 3.9 (Transformation). A *transformation* T in a calculus $\mathcal{X}$ is a relation $T : \mathcal{X} \times \mathcal{X}$. Even though T in general is not a function, we sometimes write $Z = T(Y)$ if $(Y, Z) \in T$. □

A transformation is meaning preserving if the original and the transformed terms are observationally equivalent (see definition 3.2):

Definition 3.10 (Meaning Preservation). A transformation T in a calculus X' is *meaning preserving* w.r.t. a calculus $\mathcal{X}$ if $(Y, Z) \in T$ implies $Y \cong_{\mathcal{X}} Z$. □

Example 3.11. Consider the following core module expressions:

$$D_{3.11a} = [F \mapsto \lambda x.x + a, a \mapsto 1 + 2], D_{3.11b} = [F \mapsto \lambda x.x + 3, a \mapsto 3]$$

$D_{3.11b}$ is the result of performing the constant folding and propagation transformation CFP on $D_{3.11a}$. This transformation on core module expressions is meaning preserving w.r.t. the linking calculus: i.e., $D_{3.11a} \cong_{\mathcal{L}} D_{3.11b}$. □

Example 3.12. Based on example 3.4, a transformation that swaps two operands of $+$ is not generally meaning preserving. But example 3.5 shows that the special case where the two operands of $+$ are labels is meaning preserving w.r.t. $\mathcal{C}\backslash\alpha$ and $\mathcal{L}\backslash\alpha$, but not w.r.t. $\mathcal{T}\backslash\alpha$. □

The notion of transformation given above is too simple to include many common program transformations. For instance, consider closure conversion, which transforms a program that may contain open functions (functions that have free variables) into an equivalent program in which every function is closed (has no free variables) [Han95, MMH96, SW97, Sis99, DWM$^+$01]. This is traditionally achieved by transforming an open function into a data structure that pairs a closed function with the values of the open function's free variables. Any call site to which the open function flows must be transformed in a way that is consistent with the new representation. In this sort of global representation transformation, we cannot expect that the transformation of an arbitrary term will be observationally equivalent to the original. After all, a pair of a closed function and an environment of free variables is certainly not observationally equivalent to an open function! For such a transformation, we might need a notion of meaning preservation that somehow transforms the context of a term along with the term. Alternatively, meaning preservation might be shown by a bisimulation in which execution states of the transformed program are shown to be related to execution states of the original program (e.g., [Pit97]).

Despite the above caveats, for the present work we shall stick with the simple notion of meaning preserving transformation defined above. Such a notion is adequate for characterizing many classical local transformations such as constant folding, function inlining, algebraic simplification, etc. We leave alternative approaches of defining meaning preserving transformations on our calculi for future research.

3.4 Observational and Computational Soundness

Proving that a transformation T is meaning preserving is generally rather challenging. However, meaning preservation follows automatically if T can be expressed as a *calculus-based transformation* for a calculus with *observational soundness*.

Definition 3.13 (Calculus-based Transformation). A transformation T is a *calculus-based transformation for $\mathcal{X}$ ($\longrightarrow_{\mathcal{X}}$-based transformation)* if $Y \leftrightarrow_{\mathcal{X}} Z$ for all $(Y, Z) \in T$. □

Definition 3.14 (Observational Soundness). A calculus $\mathcal{X}'$ is *observationally sound* w.r.t. $\mathcal{X}$ if $Y \leftrightarrow_{\mathcal{X}'} Z$ implies $Y \cong_{\mathcal{X}} Z$. □

Lemma 3.15 (Observational Soundness Implies Meaning Preserving Transformations). *If $\mathcal{X}'$ is observationally sound w.r.t. $\mathcal{X}$, then a $\longrightarrow_{\mathcal{X}'}$-based transformation is meaning preserving w.r.t. $\mathcal{X}$.* □

Proof. Follows directly from definitions 3.10, 3.13, and 3.14. □

One of the main results of [Plo75] is the observational soundness[13] of both the call-by-name and call-by-value λ-calculus: if two terms are provably equivalent in the calculus, then they are observationally equivalent. (In Plotkin's case, $\mathcal{X}$ and $\mathcal{X}'$ are the same.) By lemma 3.15, any transformation expressible via a sequence of λ-calculus steps preserves the meaning of a term.

Observational soundness is closely related to a notion that we call *computational soundness*.

Definition 3.16 (Computational Soundness). A calculus $\mathcal{X}$ is *computationally sound* if $Y \leftrightarrow_{\mathcal{X}} Z$ implies $\mathit{Outcome}_{\mathcal{X}}(Y) = \mathit{Outcome}_{\mathcal{X}}(Z)$. □

Computational soundness states that performing calculus steps (in any direction) on a term is safe in the sense that it cannot possibly change the observable outcome of a term. This is a key property for a calculus, since it can be used to justify that calculus-based transformations are safe. From the point of view of designing a calculus, any proposed calculus step that violates the preservation of outcome should be viewed with extreme suspicion. For instance, it is unwise to extend $\mathcal{T}$ with the notion of reduction $(M + N) \leadsto_{\mathcal{T}} (N + M)$, since example 3.4 shows that this would violate computational soundness.

To show *observational equivalence* w.r.t. $\mathcal{X}$ of two terms that are calculus equivalent in $\mathcal{X}'$ (where $\mathcal{X}$ may or may not be the same as $\mathcal{X}'$), we need to show that these terms have the same observable behavior in arbitrary contexts of $\mathcal{X}$. This follows if the relation $\longrightarrow_{\mathcal{X}'}$ of the calculus $\mathcal{X}'$ is *embedded* in the relation $\longrightarrow_{\mathcal{X}}$ of $\mathcal{X}$ – i.e., by wrapping arbitrary contexts of $\mathcal{X}$ around two terms connected by $\longrightarrow_{\mathcal{X}'}$ we get a subrelation of $\longrightarrow_{\mathcal{X}}$.

Definition 3.17 (Embedding). A relation $\longrightarrow_{\mathcal{X}'}$ is embedded in a relation $\longrightarrow_{\mathcal{X}}$ (written $\longrightarrow_{\mathcal{X}'} \preceq \longrightarrow_{\mathcal{X}}$) if $Y \longrightarrow_{\mathcal{X}'} Z$ implies that $\mathbb{X}\{Y\} \longrightarrow_{\mathcal{X}} \mathbb{X}\{Z\}$ for any context $\mathbb{X} \in \mathbf{Context}_{\mathcal{X}',\mathcal{X}}$ such that $\mathbb{X}\{Y\}$ and $\mathbb{X}\{Z\}$ are well-formed terms of $\mathcal{X}$. □

The self-embedding $\longrightarrow_{\mathcal{X}} \preceq \longrightarrow_{\mathcal{X}}$ means that the relation $\longrightarrow_{\mathcal{X}}$ is a *congruence*[14] relative to the set of one-hole contexts $\mathbf{Context}_{\mathcal{X},\mathcal{X}}$. The relation generated from a relation $\longrightarrow_{\mathcal{X}}$ by wrapping arbitrary contexts $\mathbb{X} \in \mathbf{Context}_{\mathcal{X},\mathcal{X}}$ around the two terms connected by $\longrightarrow_{\mathcal{X}}$ is the *contextual closure*[15] of $\longrightarrow_{\mathcal{X}}$. Self-embedding means that the contextual closure of $\longrightarrow_{\mathcal{X}}$ is a subset[16] of $\longrightarrow_{\mathcal{X}}$.

Lemma 3.18 (Module Calculus Embeddings for Concrete Terms). *The following are embeddings in our module calculus:*

1. $\longrightarrow_{\mathcal{T}} \preceq \longrightarrow_{\mathcal{T}}$ *($\mathcal{T}$ is a congruence);*

2. $\longrightarrow_{\mathcal{T}} \preceq \longrightarrow_{\mathcal{C}}$ *(term reductions can be performed in the bindings of a module);*

3. $\longrightarrow_{\mathcal{C}} \preceq \longrightarrow_{\mathcal{C}}$ *($\mathcal{C}$ is a congruence);*

[13] "Observational soundness" is our term; Plotkin referred to this property as "consistency".

[14] Such relations are called *compatible* in [Bar84].

[15] This notion is called *compatible closure* in [Bar84].

[16] In the case when the calculus has an empty context, as is the case with all our calculi, self-embedding means that the contextual closure of $\longrightarrow_{\mathcal{X}}$ equals $\longrightarrow_{\mathcal{X}}$. However, in general it may be the case that, even though every two terms related by the contextual closure of $\longrightarrow_{\mathcal{X}}$ are also related by $\longrightarrow_{\mathcal{X}}$, the converse may not be true, i.e. two terms related by $\longrightarrow_{\mathcal{X}}$ may not be represented as a context wrapped around two terms related by $\longrightarrow_{\mathcal{X}}$. Therefore we only require that the contextual closure of $\longrightarrow_{\mathcal{X}}$ is a subset of $\longrightarrow_{\mathcal{X}}$.

4. $\rightarrow_{\mathcal{T}} \preceq \rightarrow_{\mathcal{L}}$ *(term reductions can be performed in the bindings of a module in a linking expression);*

5. $\rightarrow_{\mathcal{C}} \preceq \rightarrow_{\mathcal{L}}$ *(core module reductions can be performed within a linking term);*

6. $\rightarrow_{\mathcal{L}} \preceq \rightarrow_{\mathcal{L}}$ *($\mathcal{L}$ is a congruence);*

7. $\rightarrow_{\mathcal{T}} \preceq \rightarrow_{\mathcal{C}_{GC}}$*;*

8. $\rightarrow_{\mathcal{C}_{GC}} \preceq \rightarrow_{\mathcal{L}_{GC}}$*;*

9. $\rightarrow_{\mathcal{T}} \preceq \rightarrow_{\mathcal{L}_{GC}}$*;*

10. $\rightarrow_{\mathcal{L}_{GC}} \preceq \rightarrow_{\mathcal{L}_{GC}}$ *($\mathcal{L}_{GC}$ is a congruence).*

□

Proof. Follows from the definitions of the relations $\rightarrow_{\mathcal{T}}$, $\rightarrow_{\mathcal{C}}$, $\rightarrow_{\mathcal{L}}$, $\rightarrow_{\mathcal{C}_{GC}}$, and $\rightarrow_{\mathcal{L}_{GC}}$ □

Remark 3.19. Note that while the core module calculus is a congruence ($\rightarrow_{\mathcal{C}} \preceq \rightarrow_{\mathcal{C}}$), the extension of this calculus with the GC rule is *not* a congruence ($\rightarrow_{\mathcal{C}_{GC}} \not\preceq \rightarrow_{\mathcal{C}_{GC}}$). The reduction $D_1 \rightarrow_{\mathcal{C}_{GC}} D_2$ does not imply that for all contexts $\mathbb{M} \in \mathbf{Context}_{\mathcal{C},\mathcal{C}}$, $\mathbb{M}\{D_1\} \rightarrow_{\mathcal{C}_{GC}} \mathbb{M}\{D_1\}$, since $\mathbb{M}$ may contain a reference to the hidden components being removed by the (GC) step. For instance, suppose $\mathbb{M}_{3.19} = [l' \mapsto l \mathbin{@} h, \square]$. Then:

$$\begin{array}{rclcccl} D_{3.19a} & = & [l \mapsto \lambda x.x, h \mapsto 2] & \rightarrow_{\mathcal{C}_{GC}} & [l \mapsto \lambda x.x] & = & D_{3.19b} \\ \mathbb{M}_{3.19}\{D_{3.19a}\} & = & [l' \mapsto l \mathbin{@} h, l \mapsto \lambda x.x, h \mapsto 2] & \not\rightarrow_{\mathcal{C}_{GC}} & [l' \mapsto l \mathbin{@} h, l \mapsto \lambda x.x] & = & \mathbb{M}_{3.19}\{D_{3.19b}\} \end{array}$$

All the other embedding relations involving $\mathcal{C}$ and $\mathcal{L}$ are preserved by their versions extended with (GC). □

We care not so much about the calculi with concrete terms, but the calculi with α-equivalence classes of terms. For this reason, we need to develop a notion of embedding for calculi at the α-equivalence level.

Definition 3.20 (Filling a Context with an α-Equivalence Class). If $\mathbb{X}$ is a (concrete) context in $\mathbf{Context}_{\mathcal{X}',\mathcal{X}}$ and $Y_{\alpha_{\mathcal{X}'}}$ is an α-equivalence class of terms in $\mathcal{X}'$, then

$$\mathbb{X}\{Y_{\alpha_{\mathcal{X}'}}\} = \begin{cases} {}_{\alpha_{\mathcal{X}}}\langle \mathbb{X}\{Z\}\rangle, & \text{where } Z \in Y_{\alpha_{\mathcal{X}'}} \text{ and } \mathbb{X}\{Z\} \text{ is defined, if } {}_{\alpha_{\mathcal{X}}}\langle \mathbb{X}\{Z'\}\rangle = {}_{\alpha_{\mathcal{X}}}\langle \mathbb{X}\{Z''\}\rangle \\ & \text{for all } Z', Z'' \in Y_{\alpha_{\mathcal{X}'}} \text{ such that } \mathbb{X}\{Z'\} \text{ and } \mathbb{X}\{Z''\} \text{ are defined} \\ \text{undefined}, & \text{otherwise} \end{cases}$$

□

In the above definition we only consider Z such that the filling of the context $\mathbb{X}\{Z\}$ is defined, i.e. in a situation when $\mathbb{X}\{Z\}$ is not defined on some $Z \in Y_{\alpha_{\mathcal{X}'}}$, such elements do not automatically make the result of filling the context undefined. The reason for this is the following: suppose we consider filling a module context $\mathbb{M}$ that exports a hidden, such as $[A \mapsto B, a \mapsto 4, \square]$, with the $\alpha_{\mathcal{C}}$-equivalence class of a module that also exports a hidden and does not import the hidden exported by the context. For instance, let us take $[B \mapsto 3, b \mapsto 7]$. If we had required that filling of the context is defined for all elements in $Y_{\alpha_{\mathcal{X}'}}$, then filling such a context with such an α-equivalence class would have been undefined, because there exists an instance of the $\alpha_{\mathcal{C}}$-equivalence class of the module that exports the label already defined in the context. In our example such an instance is $[B \mapsto 3, a \mapsto 7] \in {}_{\alpha_{\mathcal{C}}}\langle [B \mapsto 3, b \mapsto 7]\rangle$.

Note, however, that the definition implies that for the result of filling a context to be defined there must exist at least one $Z \in Y_{\alpha_{\mathcal{X}'}}$ such that $\mathbb{X}\{Z\}$ is defined.

Example 3.21. Consider the context $\mathbb{C}_{3.21} = \lambda x.\square \in \mathbf{Context}_{\mathcal{T},\mathcal{T}}$. Then $\mathbb{C}_{3.21}\{{}_{\alpha_{\mathcal{T}}}\langle \lambda y.x\rangle\} = {}_{\alpha_{\mathcal{T}}}\langle \lambda x.\lambda y.x\rangle$ and $\mathbb{C}_{3.21}\{{}_{\alpha_{\mathcal{T}}}\langle \lambda y.y\rangle\} = {}_{\alpha_{\mathcal{T}}}\langle \lambda x.\lambda y.y\rangle$. Indeed, for any $\mathbb{C} \in \mathbf{Context}_{\mathcal{T},\mathcal{T}}$ and $M \in \mathbf{Term}_{\mathcal{T}}$, $\mathbb{C}\{{}_{\alpha_{\mathcal{T}}}\langle M\rangle\} = {}_{\alpha_{\mathcal{T}}}\langle \mathbb{C}\{M\}\rangle$. □

Example 3.22. Consider the context $\mathbb{M}_{3.22} = [A \mapsto d, \square] \in \mathbf{Context}_{\mathcal{C},\mathcal{C}}$. It makes sense to fill $\mathbb{M}_{3.22}$ with some α-equivalence classes, as in $\mathbb{M}_{3.22}\{_{\alpha_\mathcal{C}}\langle[B \mapsto \lambda x.x]\rangle\} = {}_{\alpha_\mathcal{C}}\langle[A \mapsto d, B \mapsto \lambda x.x]\rangle$. However, filling $\mathbb{M}_{3.22}$ with other classes is undefined. For example, consider

$$D_{3.22a} = [B \mapsto c, c \mapsto 2] =_\alpha^\mathcal{C} [B \mapsto d, d \mapsto 2] = D_{3.22b}$$

and observe that

$$\mathbb{M}_{3.22}\{D_{3.22a}\} = [A \mapsto d, B \mapsto c, c \mapsto 2] \neq_\alpha^\mathcal{C} [A \mapsto d, B \mapsto d, d \mapsto 2] = \mathbb{M}_{3.22}\{D_{3.22b}\}.$$

In this case, α-equivalence fails because the free hidden label d in $\mathbb{M}_{3.22}$ is bound for one representative module filling the hole but not for the other representatives. □

Definition 3.23 (α-Compatibility). A calculus $\mathcal{X}'$ is *α-compatible* in $\mathcal{X}$, written $\mathcal{X}' \sqsubseteq_\alpha \mathcal{X}$, iff for all $\mathbb{X} \in \mathbf{Context}_{\mathcal{X}',\mathcal{X}}$ and all $Y_{\alpha_{\mathcal{X}'}} \in \mathbf{Term}_{\mathcal{X}'\backslash\alpha}$, $\mathbb{X}\{Y_{\alpha_{\mathcal{X}'}}\}$ is defined in $\mathbf{Term}_{\mathcal{X}\backslash\alpha}$. By definition 3.20, $\mathcal{X}' \sqsubseteq_\alpha \mathcal{X}$ implies that if $Z \in \mathbf{Term}_{\mathcal{X}'}$, then $\mathbb{X}\{_{\alpha_{\mathcal{X}'}}\langle Z\rangle\} = {}_{\alpha_\mathcal{X}}\langle\mathbb{X}\{Z\}\rangle$. □

Lemma 3.24 (α-Compatibility in the Module Calculus). *The following α-compatibility relations hold in the module calculus:*

1. $\mathcal{T} \sqsubseteq_\alpha \mathcal{T}$
2. $\mathcal{T} \sqsubseteq_\alpha \mathcal{C}$
3. $\mathcal{T} \sqsubseteq_\alpha \mathcal{L}$
4. $\mathcal{C} \sqsubseteq_\alpha \mathcal{L}$
5. $\mathcal{L} \sqsubseteq_\alpha \mathcal{L}$

□

Remark 3.25. Example 3.22 shows that $\mathcal{C} \not\sqsubseteq_\alpha \mathcal{C}$. This is the only case in the module calculus where a free α-renamable entity in the context can be affected by declarations in terms filling the hole. □

Given the definition of filling a context with an α-equivalence class (definition 3.20), the definition of embedding (definition 3.17) makes sense when considering calculi at the α-equivalence level.

Lemma 3.26 (Module Calculus Embeddings for α-Equivalence Classes of Terms). *The following are embeddings at the α-equivalence level of our module calculus:*

1. $\mathcal{T}\backslash\alpha \preceq \mathcal{T}\backslash\alpha$
2. $\mathcal{T}\backslash\alpha \preceq \mathcal{C}\backslash\alpha$
3. $\mathcal{T}\backslash\alpha \preceq \mathcal{L}\backslash\alpha$
4. $\mathcal{C}\backslash\alpha \preceq \mathcal{L}\backslash\alpha$
5. $\mathcal{L}\backslash\alpha \preceq \mathcal{L}\backslash\alpha$

□

Proof. The proofs for 1 – 5 are all similar; we give only the proof for 2. Suppose $M_{\alpha_\mathcal{T}} \longrightarrow_{\mathcal{T}\backslash\alpha} N_{\alpha_\mathcal{T}}$; we need to show that for all $\mathbb{D} \in \mathbf{Context}_{\mathcal{T},\mathcal{C}}$, $\mathbb{D}\{M_{\alpha_\mathcal{T}}\} \longrightarrow_{\mathcal{C}\backslash\alpha} \mathbb{D}\{N_{\alpha_\mathcal{T}}\}$. By definition 2.37, $M_{\alpha_\mathcal{T}} \longrightarrow_{\mathcal{T}\backslash\alpha} N_{\alpha_\mathcal{T}}$ implies the existence of $M' \in M_{\alpha_\mathcal{T}}$ and $N' \in N_{\alpha_\mathcal{T}}$ such that $M' \longrightarrow_\mathcal{T} N'$. The embedding $\longrightarrow_\mathcal{T} \preceq \longrightarrow_\mathcal{C}$ implies that for all $\mathbb{D} \in \mathbf{Context}_{\mathcal{T},\mathcal{C}}$, $\mathbb{D}\{M'\} \longrightarrow_\mathcal{C} \mathbb{D}\{N'\}$. By definition 2.57, ${}_{\alpha_\mathcal{C}}\langle\mathbb{D}\{M'\}\rangle \longrightarrow_{\mathcal{C}\backslash\alpha} {}_{\alpha_\mathcal{C}}\langle\mathbb{D}\{N'\}\rangle$. Because $\mathcal{T} \sqsubseteq_\alpha \mathcal{C}$, ${}_{\alpha_\mathcal{C}}\langle\mathbb{D}\{P\}\rangle = \mathbb{D}\{_{\alpha_\mathcal{T}}\langle P\rangle\}$ for any $P \in \mathbf{Term}_\mathcal{T}$. So $\mathbb{D}\{_{\alpha_\mathcal{T}}\langle M'\rangle\} \longrightarrow_{\mathcal{C}\backslash\alpha} \mathbb{D}\{_{\alpha_\mathcal{T}}\langle N'\rangle\}$, which can be rewritten $\mathbb{D}\{M_{\alpha_\mathcal{T}}\} \longrightarrow_{\mathcal{T}\backslash\alpha} \mathbb{D}\{N_{\alpha_\mathcal{T}}\}$. □

Together, the computational soundness of a calculus $\mathcal{X}$ and the embedding of $\rightarrow_{\mathcal{X}'}$ into $\rightarrow_{\mathcal{X}}$ imply that calculus-based transformations of $\mathcal{X}'$ are meaning preserving w.r.t. $\mathcal{X}$:

Lemma 3.27 (Comp. Soundness + Embedding = Observational Soundness). *If a calculus $\mathcal{X}$ is computationally sound and $\rightarrow_{\mathcal{X}'} \preceq \rightarrow_{\mathcal{X}}$, then $\mathcal{X}'$ is observationally sound w.r.t. $\mathcal{X}$.* □

Proof. Suppose $Y \leftrightarrow_{\mathcal{X}'} Z$. By the embedding $\rightarrow_{\mathcal{X}'} \preceq \rightarrow_{\mathcal{X}}$ (definition 3.17), $\mathbb{X}\{Y\} \leftrightarrow_{\mathcal{X}} \mathbb{X}\{Z\}$ for any context $\mathbb{X}$, By the computational soundness of $\mathcal{X}$, $\mathit{Outcome}_{\mathcal{X}}(\mathbb{X}\{Y\}) = \mathit{Outcome}_{\mathcal{X}}(\mathbb{X}\{Z\})$. Thus, $\mathcal{X}'$ is observationally sound w.r.t. $\mathcal{X}$. □

Corollary 3.28 (Comp. Soundness + Embedding = Meaning Preserving Transformations). *If a calculus $\mathcal{X}$ is computationally sound and $\rightarrow_{\mathcal{X}'} \preceq \rightarrow_{\mathcal{X}}$, then any calculus-based transformation T in $\mathcal{X}'$ is meaning preserving in $\mathcal{X}$.* □

A main result of this work is that $\mathcal{T}\backslash\alpha$, $\mathcal{C}\backslash\alpha$, and $\mathcal{L}\backslash\alpha$ (as well as $\mathcal{C}_{GC}\backslash\alpha$ and $\mathcal{L}_{GC}\backslash\alpha$) are all computationally sound. Given the embeddings for these calculi listed in Lemma 3.26, corollary 3.28 implies that calculus-based transformations are meaning preserving in all of these calculi, *except* in the cases where $\mathcal{C}\backslash\alpha$ (resp. $\mathcal{C}_{GC}\backslash\alpha$) transformations are performed in non-empty $\mathcal{C}\backslash\alpha$ (resp. $\mathcal{C}_{GC}\backslash\alpha$) contexts. See section 3.7 for examples of meaning preserving transformations for some of this calculi.

Remark 3.29. The fact that calculus-based transformations are not necessarily meaning preserving for modules in non-empty module contexts underscores that the meaning preservation of calculus-based transformations is not assured. This serves as a justification for the formal development presented here. Similarly, the fact that $\rightarrow_{\mathcal{C}}$ is embedded in $\rightarrow_{\mathcal{C}}$ but $\rightarrow_{\mathcal{C}\backslash\alpha}$ is not embedded in $\rightarrow_{\mathcal{C}\backslash\alpha}$ underscores that the results involving α-equivalence classes, while seeming perhaps tedious, are important and not entirely straightforward. □

Example 3.30. While two terms connected by calculus steps are guaranteed to be observationally equivalent in the cases noted above, the converse is not true. Many observationally equivalent terms cannot be connected by calculus steps. As a concrete example, consider $l + k$ and $k + l$, which are observationally equivalent in $\mathcal{C}\backslash\alpha$ (see example 3.5), but are not connected by any sequence of calculus steps in $\mathcal{T}\backslash\alpha$. To see this, we shall assume otherwise and derive a contradiction. Suppose that $l + k \leftrightarrow_{\mathcal{T}\backslash\alpha} k + l$. Because $\rightarrow_{\mathcal{T}}$ is confluent, this implies that there is an $M_{\alpha_{\mathcal{T}}} \in \mathbf{Term}_{\mathcal{T}\backslash\alpha}$ such that $l + k \rightarrow^*_{\mathcal{T}\backslash\alpha} M$ and $k + l \rightarrow^*_{\mathcal{T}\backslash\alpha} M$. But $l + k$ and $k + l$ are both $\rightarrow_{\mathcal{T}\backslash\alpha}$ normal forms and are not equal to each other, so no such $M_{\alpha_{\mathcal{T}}}$ exists. Thus, the original assumption $l + k \leftrightarrow_{\mathcal{T}\backslash\alpha} k + l$ must be incorrect. A similar technique can be used to show that $\lambda x.x + x$ and $\lambda x.x * 2$ are not equivalent in $\mathcal{T}\backslash\alpha$ even though they are observationally equivalent w.r.t. $\mathcal{T}\backslash\alpha$. □

3.5 A Classical Technique for Proving Computational Soundness

In this section we review the traditional "ingredients" and "recipe" for proving computational soundness. It turns out that the traditional recipe fails for calculi $\mathcal{C}\backslash\alpha$ and $\mathcal{L}\backslash\alpha$. In the next section, we present a new technique for proving computational soundness that works for these calculi because it has weaker preconditions.

The traditional recipe for proving the computational soundness of a calculus $\mathcal{X}$ has three ingredients:

1. Confluence of $\rightarrow_{\mathcal{X}}$ and $\Rightarrow_{\mathcal{X}}$ (definition 2.6);
2. Standardization of $\mathcal{X}$ (definition 2.7);
3. The *class preservation* property defined below.

Definition 3.31 (Class Preservation). Calculus $\mathcal{X}$ has the class preservation property if $M \circ\!\!\rightarrow_{\mathcal{X}} N$ implies $Cl_{\mathcal{X}}(M) = Cl_{\mathcal{X}}(N)$. □

We discuss these ingredients in more detail in the following subsections, followed by a description of how to combine them into a proof of computational soundness.

3.5.1 Confluence

Confluence is a classical property that holds for many calculi. The important fact with relation to our work is that $\rightarrow_{\mathcal{C}\backslash\alpha}$ is not confluent, as shown in section 2.4.3. This fact implies that we cannot apply the traditional technique for proving computational soundness to $\mathcal{C}\backslash\alpha$ (and also of $\mathcal{L}\backslash\alpha$). The lack of confluence of $\rightarrow_{\mathcal{C}\backslash\alpha}$ motivated our development of a new technique for proving computation soundness (see section 3.6).

Note that confluence of $\Rightarrow_{\mathcal{X}}$ is independent from that of $\rightarrow_{\mathcal{X}}$. For many calculi, $\Rightarrow_{\mathcal{X}}$ is a function, and so is trivially confluent. For calculi like $\mathcal{C}\backslash\alpha$ and $\mathcal{L}\backslash\alpha$, where $\Rightarrow_{\mathcal{X}}$ is not a function, the condition that $\Rightarrow_{\mathcal{X}}$ is confluent is necessary in order for $Eval_{\mathcal{X}}$ to be well-defined.

Our proofs of computational soundness use the following standard lemma related to confluence:

Lemma 3.32. *If $\rightarrow$ is confluent and $Y \leftrightarrow Z$, then there is a W such that $Y \rightarrow^* W$ and $Z \rightarrow^* W$.* □

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

3.5.2 Standardization

Standardization traditionally refers to the property of the calculus that every reduction sequence $Y \rightarrow^*_{\mathcal{X}} Z$ can be rewritten as a sequence from Y to Z in which the reduction steps are performed in a "standard" order. Definition of a standard order depends on a particular calculus. However, a standard sequence usually has the following two properties (as applicable to a particular calculus):

- if a term reaches a value by some reduction sequence, then it reaches a value by a standard sequence;
- for a calculus where the meaning of a term is defined via a small-step operational semantics, a standard reduction sequence performs all the "standard", i.e. those corresponding to the operational semantics, steps first, possibly followed by some "non-standard" steps, i.e. those that do not change the observable behavior of a term (for instance, steps performed under a λ in a λ-calculus).

The intuition behind a standard sequence is as follows: at every step of such a sequence the reduced redex is the one which is "needed" to make progress on the term towards reaching a normal form, if it exists, which usually corresponds to the strategy employed by operational semantics of the calculus. The notion of such *needed* redex has been formalized in [HL91] for TRS.

Intuitively, "standard" and "non-standard" reduction steps correspond, respectively, to the evaluation and non-evaluation steps presented in section 2.2. However, definitions of a standard sequence in the literature often depend on the details of the calculi in question, and are sometimes quite complex. In particular, while our evaluation/non-evaluation classification of steps depends only on the current redex, the standard/non-standard classification of steps can depend on the reduction sequence leading up to a step. For instance, the inductive definition of a standard reduction sequence in [Plo75] is such that reduction of an outermost application cannot be performed if the operand or the operator has been reduced under a λ in the same sequence. However, such an application may be reduced as a "standard" step if the preceding part of the sequence has not performed a reduction under a λ.

The definition in [Bar84] is given in terms of residuals of redexes reduced earlier in the reduction sequence. Barendregt also relates standard reduction sequences for the call-by-name λ-calculus to the notion of a *head* redex: a standard sequence reduces all head redexes first, followed by *internal* (i.e. non-head) redexes. [GLM92] point out that a standard redex has exactly one residual w.r.t. any other redex, and takes this property as the basis of the definition of a "standard" step. Using an approach based on combinatory reduction systems, Wells and Muller give a general definition of a standard sequence which may be instantiated for a particular calculus in such a way that the calculus is guaranteed to have the standardization property [WM00]. Ariola and Felleisen employ a different approach, defining a standard sequence to be a sequence that performs all the evaluation steps first, and proving what they refer to as "the important part of Standardization theorem", namely that if a closed term reaches an answer (a special class of normal forms w.r.t. evaluation) by a reduction sequence, then the answer may be reached by a standard sequence [AF97]. This is indeed sufficient to show computational soundness of their calculus, as well as to demonstrate that standard reductions correspond to a normalizing strategy. Our approach is close to that of [AF97], with the difference that we consider arbitrary standard reduction sequences, not only those that end in a normal form.

As illustrated in the proof of Theorem 3.35 below, an important use of the standardization property is to show that a calculus is computationally sound. It turns out that for this purpose it is sufficient to define a standard sequence in a calculus $\mathcal{X}$ as a sequence of the form $X \Longrightarrow^*_{\mathcal{X}} X_1 \circ\!\!\longrightarrow^*_{\mathcal{X}} X'$ and to show that any reduction sequence $X \longrightarrow^*_{\mathcal{X}} X'$ has a corresponding standard reduction sequence. This is the property that we refer to as *standardization* in definition 2.7. Even though it is formulated somewhat differently from definitions of standardization in the literature mentioned above, it is essentially the same property, because it captures the idea of reordering reduction sequences in such a way that all evaluation steps (i.e. reductions of the "needed" redex) are performed first. This definition also abstracts over details of the calculus. Note that in our case $\Longrightarrow_{\mathcal{X}}$ is not a function for $\mathcal{C}\backslash\alpha$, $\mathcal{C}_{GC}\backslash\alpha$, and $\mathcal{L}\backslash\alpha$, so that, in contrast to traditional notions of standardization, the evaluation part $X \Longrightarrow^*_{\mathcal{X}} X_1$ of a standard sequence is not unique. The non-evaluation part $X_1 \circ\!\!\longrightarrow^*_{\mathcal{X}} X'$ is also not required to be unique in our calculi; this is the case in some other calculi as well. For instance, in the calculus of [AF97] the two reduction sequences below[17] are both standard (we use our notations for evaluation and non-evaluation steps, but omitting the symbol @ , following the notations in [AF97]):

$$\begin{array}{lcll}
(\lambda f.f)\lambda w.(II)w & \Longrightarrow & (\lambda f.\lambda w.(II)w)\lambda w.(II)w & \circ\!\!\longrightarrow \\
(\lambda f.\lambda w.((\lambda z.I)I)w)\lambda w.(II)w & \circ\!\!\longrightarrow & (\lambda f.\lambda w.((\lambda z.I)I)w)\lambda w.((\lambda z.I)I)w, & \\
(\lambda f.f)\lambda w.(II)w & \Longrightarrow & (\lambda f.\lambda w.(II)w)\lambda w.(II)w & \circ\!\!\longrightarrow \\
(\lambda f.\lambda w.(II)w)\lambda w((\lambda z.I)I)w & \circ\!\!\longrightarrow & (\lambda f.\lambda w.((\lambda z.I)I)w)\lambda w.((\lambda z.I)I)w. &
\end{array}$$

Some other authors require that the entire standard sequence, including the non-evaluation part, is uniquely defined, given the original and the resulting term (see, for instance, theorem 12.3.14 in [Bar84]).

3.5.3 Class Preservation

The class preservation property has two important implications:

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

2. A non-evaluation reduction sequence cannot change an evaluatable term to an eval normal form, and vice versa. (In fact, one usually shows a stronger property: a non-evaluation step can not create, remove, or duplicate an evaluation redex. [Plo75] and [Bar84] show this property, while [GLM92] take this property as the basis for axiomatic definition of a "standard" reduction step. We show this stronger property via elementary lift and project diagrams, see appendix A for details).

These implications are formalized via the following lemmas:

Lemma 3.33. *If $\mathcal{X}$ has class preservation, $Y \in NF_{\Longrightarrow_{\mathcal{X}}}$, and $Y \longrightarrow^*_{\mathcal{X}} Z$, then each step in $Y \longrightarrow^*_{\mathcal{X}} Z$ is a non-evaluation step and $Cl_{\mathcal{X}}(Y) = Cl_{\mathcal{X}}(Z)$.* □

Proof. By induction on the number of steps n in $Y \longrightarrow^*_{\mathcal{X}} Z$. If $n = 0$, the result is trivially true. For $n > 0$, $Y \longrightarrow_{\mathcal{X}} Y' \longrightarrow^*_{\mathcal{X}} Z$. By the induction hypothesis, each step in $Y' \longrightarrow^*_{\mathcal{X}} Z$ is a non-evaluation step and $Cl_{\mathcal{X}}(Y') = Cl_{\mathcal{X}}(Z)$. Since $Y \in NF_{\Longrightarrow_{\mathcal{X}}}$, the step $Y \longrightarrow_{\mathcal{X}} Y'$ must be a non-evaluation step, and by class preservation, $Cl_{\mathcal{X}}(Y) = Cl_{\mathcal{X}}(Y')$. □

Lemma 3.34. *If $\mathcal{X}$ has class preservation and $Y \circ\!\!\longrightarrow^*_{\mathcal{X}} Z$, then $Y \in NF_{\Longrightarrow_{\mathcal{X}}}$ iff $Z \in NF_{\Longrightarrow_{\mathcal{X}}}$.* □

[17] the rule used in the reductions is the call-by-need version of β-reduction: $(\lambda x.\mathbb{E}\{x\})V \longrightarrow (\lambda x.\mathbb{E}\{V\})V$. The reduction step is an evaluation step if the redex occurs in an evaluation context, otherwise it is a non-evaluation step. See [AF97] for details.

Proof. It is easy to show that $Cl_{\mathcal{X}}(Y) = Cl_{\mathcal{X}}(Z)$ by induction on the number of steps $Y \circ\!\!\longrightarrow^*_{\mathcal{X}} Z$. Suppose that $Y \notin NF_{\Longrightarrow_{\mathcal{X}}}$; then by property 2.4 and class preservation, $\mathbf{evaluatable}_{\mathcal{X}} = Cl_{\mathcal{X}}(Y) = Cl_{\mathcal{X}}(Z) = \mathbf{evaluatable}_{\mathcal{X}}$, so $Z \notin NF_{\Longrightarrow_{\mathcal{X}}}$. A similar argument shows that $Y \in NF_{\Longrightarrow_{\mathcal{X}}}$ implies $Z \in NF_{\Longrightarrow_{\mathcal{X}}}$. □

Both of these consequences of class preservation are implicitly used in computational soundness proofs (see [Plo75]). The second consequence is used even more widely in standardization proofs ([Plo75, Bar84, AF97]). The property that a non-standard reduction step does not change the classification of a term is explicitly stated and proven in [AF97] for a call-by-need calculus. In [Plo75] the property is used implicitly, separated into several lemmas. For instance, Lemma 6 in the discussion of the call-by-value calculus states that if an application reduces to a constant or a λ-abstraction, then the reduction step is a "standard" (i.e. an evaluation) step. Barendregt shows that a head redex cannot be created, duplicated, or removed by reduction of an internal redex (lemma 11.4.3 in [Bar84]).

When designing an operational semantics for a calculus, the class preservation property can be used as a guide for deciding which $\longrightarrow$ steps are evaluation steps and which are non-evaluation steps and for choosing a sensible classification. For instance, the change of the GC rule from an evaluation step (as in [MT00]) to a non-evaluation step in this presentation has prompted the change in classification of modules (see section 2.6 for details).

3.5.4 Proving Computational Soundness

Here we present a traditional proof of computational soundness that generalizes the approach in [Plo75].

Theorem 3.35 (Computational Soundness via Confluence). *If $\longrightarrow_{\mathcal{X}}$ and $\Longrightarrow_{\mathcal{X}}$ are confluent and $\mathcal{X}$ has standardization and class preservation, then $\mathcal{X}$ is computationally sound.* □

Proof. Assume that $Y \leftrightarrow_{\mathcal{X}} Z$. We show that $Outcome_{\mathcal{X}}(Y) = Outcome_{\mathcal{X}}(Z)$ by the following two cases:

1. $Outcome_{\mathcal{X}}(Y) \neq \bot$: As shown in figure 2[18], $Y \Longrightarrow^*_{\mathcal{X}} Y' = Eval_{\mathcal{X}}(Y)$; $Eval_{\mathcal{X}}$ is well-defined by the confluence of $\Longrightarrow_{\mathcal{X}}$. By confluence of $\longrightarrow_{\mathcal{X}}$ and lemma 3.32, there is an W such that $Y' \longrightarrow^*_{\mathcal{X}} W$ and $Z \longrightarrow^*_{\mathcal{X}} W$. Since $Y' \in NF_{\Longrightarrow_{\mathcal{X}}}$, by class preservation and lemma 3.33, $Y' \circ\!\!\longrightarrow^*_{\mathcal{X}} W$. By standardization, $Z \longrightarrow^*_{\mathcal{X}} W$ implies that there is an Z' such that $Z \Longrightarrow^*_{\mathcal{X}} Z' \circ\!\!\longrightarrow^*_{\mathcal{X}} W$. Since Y', W, and Z' are connected only by $\circ\!\!\longrightarrow_{\mathcal{X}}$ steps, class preservation implies $Cl_{\mathcal{X}}(Y') = Cl_{\mathcal{X}}(W) = Cl_{\mathcal{X}}(Z')$, and property 2.4 implies that $Z' \in NF_{\Longrightarrow_{\mathcal{X}}}$ since $Y' \in NF_{\Longrightarrow_{\mathcal{X}}}$. So $Z' = Eval_{\mathcal{X}}(Z)$, and $Cl_{\mathcal{X}}(Z') = Cl_{\mathcal{X}}(Y')$.

2. $Outcome_{\mathcal{X}}(Y) = \bot$: If $Eval_{\mathcal{X}}(Z)$ exists, then by the above argument we can show that $Eval_{\mathcal{X}}(Y)$ also exists. So $Outcome_{\mathcal{X}}(Z) = \bot$.

□

3.6 A Novel Technique for Proving Computational Soundness

Traditionally, proofs of computational soundness (like the one in the previous section) state confluence of $\longrightarrow_{\mathcal{X}}$ as a requirement. But what if $\longrightarrow_{\mathcal{X}}$ is not confluent, as in $\mathcal{C}\backslash\alpha$ and $\mathcal{L}\backslash\alpha$? An inspection of the proof of theorem 3.35 reveals that confluence of $\longrightarrow_{\mathcal{X}}$ is stronger than what is actually required for the proof to go through. The fact that one side of the commutative square in figure 2 consists purely of evaluation steps rather than arbitrary calculus reduction steps suggests that a weaker form of confluence might suffice. We have developed a novel technique for proving computational soundness that replaces the requirement that $\longrightarrow_{\mathcal{C}\backslash\alpha}$ be confluent by a a weaker condition that we call *confluence w.r.t. evaluation*.

[18] In figures 2–3, double-headed arrows denote reflexive, transitive closures of the respective relations, and a line with arrows on both ends denotes the reflexive, symmetric, transitive closure of the respective relation. Solid lines denote given relations, while dashed lines denote relations whose existence is implied by the the proofs.

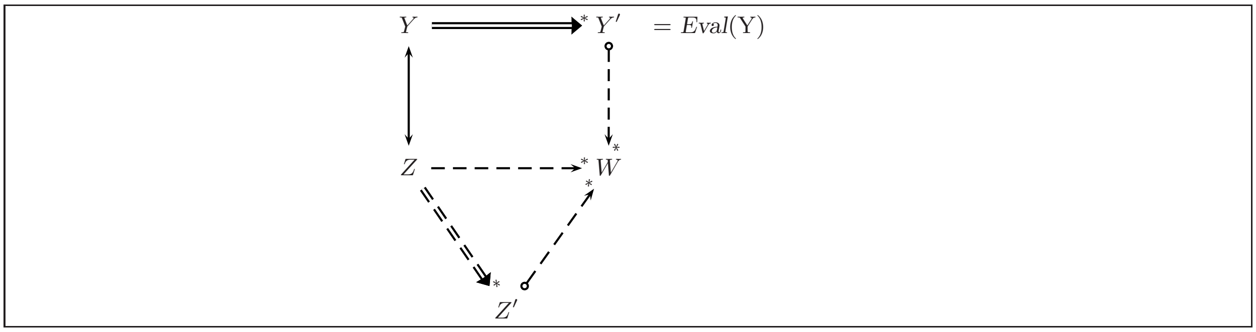

Figure 2: Sketch of the traditional proof of computational soundness. Note that $\mathit{Cl}(Y') = \mathit{Cl}(W) = \mathit{Cl}(Z')$, from which it can be deduced that $Z' = \mathit{Eval}(Z)$ and $\mathit{Outcome}(Y) = \mathit{Outcome}(Z)$.

Definition 3.36 (Confluence w.r.t. evaluation). A calculus reduction relation $\rightarrow_{\mathcal{X}}$ is *confluent w.r.t.* an evaluation relation $\Rightarrow_{\mathcal{X}}$ iff $Y \Rightarrow^*_{\mathcal{X}} Y'$ and $Y \rightarrow^*_{\mathcal{X}} Z$ implies the existence of a Z' such that $Z \Rightarrow^*_{\mathcal{X}} Z'$ and $Y' \rightarrow^*_{\mathcal{X}} Z'$. □

It turns out that while $\rightarrow_{\mathcal{C}\backslash\alpha}$ and $\rightarrow_{\mathcal{L}\backslash\alpha}$ are not confluent, they are confluent w.r.t. evaluation, and so $\mathcal{C}\backslash\alpha$ and $\mathcal{L}\backslash\alpha$ are computationally sound by our technique (see theorem 3.42).

The technique for proving computational soundness based on confluence w.r.t. evaluation has a different flavor than that used in theorem 3.35. To gain insight into the nature of this new proof, we first introduce an alternative technique for proving computational soundness based on a pair of related properties that we call *lift* and *project*. These are defined below and depicted in figure 3. It turns out that the lift property is equivalent to standardization and that the project property, in conjunction with the confluence of $\Rightarrow_{\mathcal{X}}$, implies confluence w.r.t. evaluation. Together, lift, project, the confluence of $\Rightarrow_{\mathcal{X}}$, and class preservation imply computational soundness (see theorem 3.41). A small tweak to the proof based on lift and project yields a proof based on standardization and confluence w.r.t. evaluation (see theorem 3.42).

Definition 3.37 (Lift). A calculus $\mathcal{X}$ has the *lift property* if for any reduction sequence $Y \circ\!\!\rightarrow_{\mathcal{X}} Z \Rightarrow^*_{\mathcal{X}} Z'$ there exists a sequence $Y \Rightarrow^*_{\mathcal{X}} Y' \circ\!\!\rightarrow^*_{\mathcal{X}} Z'$. □

Definition 3.38 (Project). A calculus $\mathcal{X}$ has the *project property* if $Y \circ\!\!\rightarrow_{\mathcal{X}} Z$, $Y \Rightarrow^*_{\mathcal{X}} Y'$ implies that there exist terms Y'', Z' such that $Y' \Rightarrow^*_{\mathcal{X}} Y''$, $Z \Rightarrow^*_{\mathcal{X}} Z'$, and $Y'' \circ\!\!\rightarrow^*_{\mathcal{X}} Z'$. □

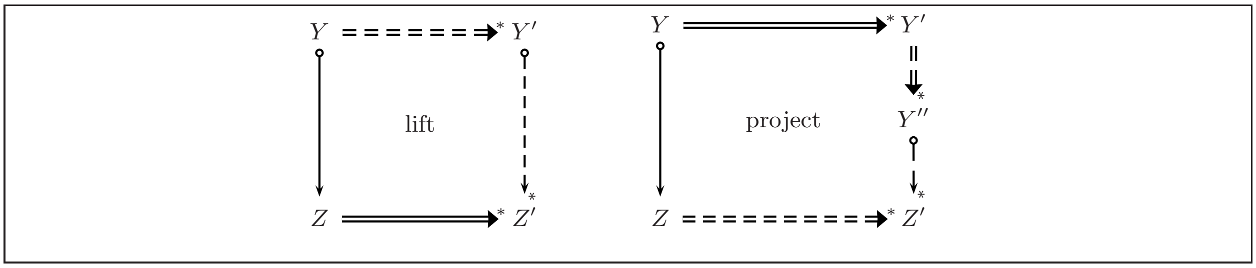

Figure 3: The lift and project properties.

When proving computational soundness for a calculus that lack confluence (such as $\mathcal{C}\backslash\alpha$ or $\mathcal{L}\backslash\alpha$), there is a benefit in proving standardization via the lift property rather than directly via the definition of standardization: proofs of both the lift and project properties use the same mechanism (certain properties of residuals and finite developments [Bar84]) and therefore share several intermediate results. See appendix A for more details.

The formal proofs of the lift and project properties for the calculi $\mathcal{C}\backslash\alpha$ and $\mathcal{L}\backslash\alpha$ are given in appendices C and D, respectively. The proofs for $\mathcal{C}_{GC}\backslash\alpha$ and $\mathcal{L}_{GC}\backslash\alpha$ are given in appendix E. Here we illustrate the properties by examples.

Consider the following two modules connected by a non-evaluation step (the substitution is performed under a λ):

$$D = [A \mapsto \lambda x.B, B \mapsto 2, C \mapsto A] \circ\!\!\longrightarrow_{\mathcal{C}\backslash\alpha} [A \mapsto \lambda x.2, B \mapsto 2, C \mapsto A] = D_1.$$

The first of the two modules evaluates in a single step as follows:

$$D = [A \mapsto \lambda x.B, B \mapsto 2, C \mapsto A] \Longrightarrow_{\mathcal{C}\backslash\alpha} [A \mapsto \lambda x.B, B \mapsto 2, C \mapsto \lambda x.B] = D'.$$

The project property implies the existence of D'', D'_1 such that $D_1 \Longrightarrow^*_{\mathcal{C}\backslash\alpha} D'_1$ and $D' \Longrightarrow^*_{\mathcal{C}\backslash\alpha} D'' \circ\!\!\longrightarrow^*_{\mathcal{C}\backslash\alpha} D'_1$. Indeed, if we take $D'' = D'$ and $D'_1 = [A \mapsto \lambda x.2, B \mapsto 2, C \mapsto \lambda x.2]$, then:

$$\begin{array}{lll} [A \mapsto \lambda x.2, B \mapsto 2, C \mapsto A] & \Longrightarrow_{\mathcal{C}\backslash\alpha} & [A \mapsto \lambda x.2, B \mapsto 2, C \mapsto \lambda x.2]; \\ [A \mapsto \lambda x.B, B \mapsto 2, C \mapsto \lambda x.B] & \circ\!\!\longrightarrow_{\mathcal{C}\backslash\alpha} & [A \mapsto \lambda x.2, B \mapsto 2, C \mapsto \lambda x.B] \quad \circ\!\!\longrightarrow_{\mathcal{C}\backslash\alpha} \\ [A \mapsto \lambda x.2, B \mapsto 2, C \mapsto \lambda x.2]. & & \end{array}$$

To illustrate lift, let us assume that D, D_1, and D'_1 are given: then D' is the module whose existence is guaranteed by the lift property.

In both the lift and project diagrams in Figure 3, suppose that we "mark" the original non-evaluation redex (the vertical non-evaluation step on the left hand sides of the diagrams). Then the resulting $\circ\!\!\longrightarrow^*_{\mathcal{C}\backslash\alpha}$ sequence (the vertical multi-step sequence on the right hand sides of the diagrams) reduces copies of the marked redex that are created by the evaluation sequence across the top of the diagram. The copies of the marked redex are known as *residuals* of the redex. In the above example, the non-evaluation redex (the label B that gets replaced by 2) is copied by the evaluation step $D \Longrightarrow_{\mathcal{C}\backslash\alpha} D'$ to the component bound to C, so a two-step reduction sequence $D_1 \circ\!\!\longrightarrow^*_{\mathcal{C}\backslash\alpha} D'_1$ is necessary to reduce both residuals. We formalize the notion of a residual in the appendix A.

In some cases, some or all of the residuals of the original non-evaluation redex may become evaluation redexes (in both lift and project). Consider the following simple example in the term calculus $\mathcal{T}\backslash\alpha$.[19]

$$\begin{array}{lllllll} M & = & (\lambda x.((\lambda y.y) @ (\lambda z.z)) @ x) @ 5 & \circ\!\!\longrightarrow_{\mathcal{T}\backslash\alpha} & (\lambda x.(\lambda z.z) @ x) @ 5 & = & N, \\ M & = & (\lambda x.((\lambda y.y) @ (\lambda z.z)) @ x) @ 5 & \Longrightarrow_{\mathcal{T}\backslash\alpha} & ((\lambda y.y) @ (\lambda z.z)) @ 5 & = & M'. \end{array}$$

To complete the project diagram, we set $M'' = \lambda z.z @ 5$ and $N' = M''$. Then:

$$\begin{array}{lllllll} N & = & (\lambda x.(\lambda z.z) @ x) @ 5 & \Longrightarrow_{\mathcal{T}\backslash\alpha} & \lambda z.z @ 5 & = & N', \\ M' & = & ((\lambda y.y) @ (\lambda z.z)) @ 5 & \Longrightarrow_{\mathcal{T}\backslash\alpha} & \lambda z.z @ 5 & = & M'', \end{array}$$

and $M'' \circ\!\!\longrightarrow^*_{\mathcal{T}\backslash\alpha} N'$ in 0 steps. On the other hand, if we are given M, N, and N', and we want to complete the lift diagram, then $M \Longrightarrow^*_{\mathcal{T}\backslash\alpha} M''$ in two steps (via M'), and $M'' \circ\!\!\longrightarrow^*_{\mathcal{T}\backslash\alpha} N'$ in 0 steps, so M'' is the term that completes the diagram. Note that the reduction $M' \Longrightarrow_{\mathcal{T}\backslash\alpha} M''$ reduces the residual of the original non-evaluation redex $(\lambda y.y) @ (\lambda z.z)$, but the redex has become an evaluation redex after the evaluation step $M \Longrightarrow_{\mathcal{T}\backslash\alpha} M'$. This example explains why the diagrams for lift and project are not completely symmetric. Examples of evaluation redexes which are residuals of a non-evaluation redex exist in $\mathcal{C}\backslash\alpha$ and are inherited by $\mathcal{L}\backslash\alpha$.

It is easy to see that the lift property (3.37) is implied by standardization (property 2.7). As shown below, the converse is also true, so lift is in fact *equivalent* to standardization.

[19] Since $\longrightarrow_{\mathcal{T}\backslash\alpha}$ is confluent, the computational soundness proof for $\mathcal{T}\backslash\alpha$ does not require the project property, only lift, which is equivalent to standardization, but the project property also holds; see appendix B.

Lemma 3.39 (Lift Implies Standardization). *If $\mathcal{X}$ has the lift property (i.e., for any reduction sequence $Y \circ\!\!\longrightarrow Z \Longrightarrow^* Z'$ there exists a sequence $Y \Longrightarrow^* Y' \circ\!\!\longrightarrow^* Z'$) then it has the standardization property (i.e., for any sequence $Y_1 \longrightarrow^* Y_2$ there exists Y_3 such that $Y_1 \Longrightarrow^* Y_3 \circ\!\!\longrightarrow^* Y_2$).* □

Proof. A non-standard sequence S from Y_1 to Y_2 must have the form:

$$Y_1 \quad \longrightarrow^* \quad Y_1' \quad \circ\!\!\longrightarrow \quad Y_2' \quad \Longrightarrow^+ \quad Y_3' \quad \circ\!\!\longrightarrow^* \quad Y_2.$$

By the lift property there exists Y_2'' such that $Y_1' \Longrightarrow^* Y_2'' \circ\!\!\longrightarrow^* Y_3'$. Define $\Phi(S)$ as the sequence[20]:

$$Y_1 \quad \longrightarrow^* \quad Y_1' \quad \Longrightarrow^* \quad Y_2'' \quad \circ\!\!\longrightarrow^* \quad Y_3' \quad \circ\!\!\longrightarrow^* \quad Y_2.$$

Note that each application of Φ on a reduction sequence from Y_1 to Y_2 yields a new reduction sequence from Y_1 to Y_2 that has fewer non-evaluation steps to the left of the rightmost evaluation step in the sequence. Thus, iterating Φ starting with an arbitrary reduction sequence from Y_1 to Y_2 will eventually terminate with a reduction sequence from Y_1 to Y_2 in which there are no non-evaluation steps to the left of any evaluation step – i.e., when a standard sequence from Y_1 to Y_2 has been obtained. □

The project property is closely related to the notion of confluence w.r.t. evaluation:

Lemma 3.40 (Project Implies Confluence w.r.t. Evaluation). *If $\Longrightarrow_{\mathcal{X}}$ is confluent and $\mathcal{X}$ has the project property, then $\longrightarrow_{\mathcal{X}}$ is confluent w.r.t. $\Longrightarrow_{\mathcal{X}}$.* □

Proof. Suppose that $Y \Longrightarrow_{\mathcal{X}} Y'$ and $Y \longrightarrow^*_{\mathcal{X}} Z$. We wish to show that there is a Z' such that $Z \Longrightarrow^*_{\mathcal{X}} Z'$ and $Y' \longrightarrow^*_{\mathcal{X}} Z'$. We proceed by induction on the length n of the sequence $Y \longrightarrow^*_{\mathcal{X}} Z$. If $n = 0$, then $Z = Y$ and $Z' = Y'$. If $n > 0$, then $Y \longrightarrow_{\mathcal{X}} W \longrightarrow^*_{\mathcal{X}} Z$, and there are two cases:

- If $Y \longrightarrow_{\mathcal{X}} W$ via $\Longrightarrow_{\mathcal{X}}$, then confluence of $\Longrightarrow_{\mathcal{X}}$ implies the existence of W' such that $W \Longrightarrow^*_{\mathcal{X}} W'$ and $Y' \longrightarrow^*_{\mathcal{X}} W'$. By the induction hypothesis, there is a Z' such that $Z \Longrightarrow^*_{\mathcal{X}} Z'$ and $W' \longrightarrow^*_{\mathcal{X}} Z'$. By transitivity of $\longrightarrow_{\mathcal{X}}$, $Y' \longrightarrow^*_{\mathcal{X}} Z'$.
- If $Y \longrightarrow_{\mathcal{X}} W$ via $\circ\!\!\longrightarrow_{\mathcal{X}}$, then the project property implies the existence of W' such that $W \Longrightarrow^*_{\mathcal{X}} W'$ and $Y' \longrightarrow^*_{\mathcal{X}} W'$. Again, applying the inductive hypothesis completes the case.

□

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

Theorem 3.41 (Computational Soundness via Lift and Project). *If $\Longrightarrow_{\mathcal{X}}$ is confluent and $\mathcal{X}$ has the lift, project, and class preservation properties, then $\mathcal{X}$ is computationally sound.* □

Proof. Assume that $Y \leftrightarrow_{\mathcal{X}} Z$. We show that $\mathit{Outcome}_{\mathcal{X}}(Y) = \mathit{Outcome}_{\mathcal{X}}(Z)$ by the following two cases:

1. $\mathit{Outcome}_{\mathcal{X}}(Y) \neq \bot$: We show the result for the case where Y and Z are connected by a single step. Using the single-step result, it is easy to prove the multi-step result via an easy induction on the length of the sequence $Y \leftrightarrow_{\mathcal{X}} Z$.

   Let $Y' = \mathit{Eval}_{\mathcal{X}}(Y)$. We show that $\mathit{Outcome}_{\mathcal{X}}(Z) = \mathit{Cl}_{\mathcal{X}}(Y') = \mathit{Outcome}_{\mathcal{X}}(Y)$ in all four cases relating Y and Z by a single reduction step (see figure 4):

   (a) $Y \circ\!\!\longrightarrow_{\mathcal{X}} Z$. By the project property, $Y \Longrightarrow^*_{\mathcal{X}} Y'$ implies that there exist Y'', Z' such that $Y' \Longrightarrow^*_{\mathcal{X}} Y''$, $Z \Longrightarrow^*_{\mathcal{X}} Z'$, and $Y'' \circ\!\!\longrightarrow^*_{\mathcal{X}} Z'$. But $Y' \in NF_{\Longrightarrow_{\mathcal{X}}}$, so $Y' = Y''$. By the class preservation property, $\mathit{Cl}_{\mathcal{X}}(Y') = \mathit{Cl}_{\mathcal{X}}(Z')$, and property 2.4 implies that $Z' \in NF_{\Longrightarrow_{\mathcal{X}}}$. Hence, $Z' = \mathit{Eval}_{\mathcal{X}}(Z)$, and $\mathit{Outcome}_{\mathcal{X}}(Z) = \mathit{Cl}_{\mathcal{X}}(Z') = \mathit{Cl}_{\mathcal{X}}(Y') = \mathit{Outcome}_{\mathcal{X}}(Y)$.

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

(b) $Z \circ\!\!\rightarrow_{\mathcal{X}} Y$. By the lift property, $Y \Rightarrow^*_{\mathcal{X}} Y'$ implies that there exists Z' such that $Z \Rightarrow^*_{\mathcal{X}} Z'$ and $Z' \circ\!\!\rightarrow^*_{\mathcal{X}} Y'$. Class preservation implies $Cl_{\mathcal{X}}(Z') = Cl_{\mathcal{X}}(Y')$, and since $Y' \in NF_{\Rightarrow_{\mathcal{X}}}$, property 2.4 implies that $Z' \in NF_{\Rightarrow_{\mathcal{X}}}$, and, as above, $Outcome_{\mathcal{X}}(Z) = Outcome_{\mathcal{X}}(Y)$.

(c) $Y \Rightarrow_{\mathcal{X}} Z$. By confluence of $\Rightarrow_{\mathcal{X}}$ there exists Z' such that $Z \Rightarrow^*_{\mathcal{X}} Z'$, $Y' \Rightarrow^*_{\mathcal{X}} Z'$. But Y' is a normal form, so $Y' = Z' = Eval_{\mathcal{X}}(Z)$.

(d) $Z \Rightarrow_{\mathcal{X}} Y$. By transitivity of $\Rightarrow^*_{\mathcal{X}}$, $Z \Rightarrow^*_{\mathcal{X}} Y'$, and since Y' is a normal form, $Y' = Eval_{\mathcal{X}}(Z)$.

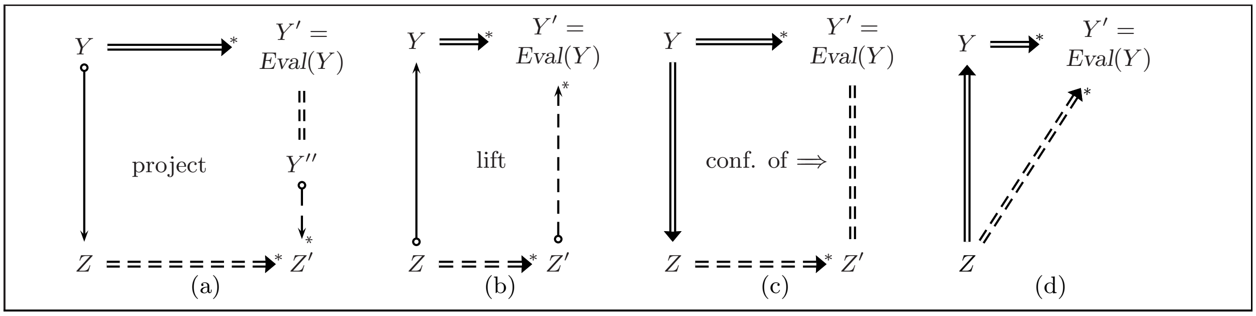

Figure 4: Proof of soundness via lift and project.

2. $Outcome_{\mathcal{X}}(Y) = \perp$: If $Outcome_{\mathcal{X}}(Z) \neq \perp$, then by the above argument $Outcome_{\mathcal{X}}(Y) = Outcome_{\mathcal{X}}(Z) \neq \perp$, and we get a contradiction. Therefore, if $Outcome_{\mathcal{X}}(Y) = \perp$, then $Outcome_{\mathcal{X}}(Z) = \perp$.

□

The calculi $\mathcal{C}\backslash\alpha$, $\mathcal{C}_{GC}\backslash\alpha$, $\mathcal{L}\backslash\alpha$, and $\mathcal{L}_{GC}\backslash\alpha$ have the lift, project, and class preservation properties (the respective proofs are given in appendicies), so they enjoy the computational soundness property.

Now we are ready to state a computational soundness theorem in terms of standardization and confluence w.r.t. evaluation rather than in terms of lift and project:

Theorem 3.42 (Computational Soundness via Confluence w.r.t. Evaluation). *If $\Rightarrow_{\mathcal{X}}$ is confluent, $\rightarrow_{\mathcal{X}}$ is confluent w.r.t. evaluation, and $\mathcal{X}$ has the standardization and class preservation properties, then $\mathcal{X}$ is computationally sound.* □

Proof. The proof is similar to that for theorem 3.41. Case 2 and Case 1(d) are exactly the same. Case 1(b) follows from standardization, which is equivalent to lift by lemma 3.39. Cases 1(a) and 1(c) follow from the fact that $\rightarrow_{\mathcal{X}}$ is confluent w.r.t. $\Rightarrow_{\mathcal{X}}$. □

3.7 Examples of Meaning Preserving Transformations.

Since all levels of our module calculus have the computational soundness property, all transformations provable in the calculus are meaning preserving. Many classical program transformations (both at the term and at the module level) fall into this category: e.g., constant folding and propagation, function inlining, and simple forms of dead-code elimination that eliminate unused value bindings. All of these (and any combination thereof) can easily be shown meaning preserving because all are justified by simple calculus steps.

We consider the following "one-step" transformations T_1: constant folding (CF_1), constant propagation (CP_1), function inlining (FI_1), and a simple form of dead-code elimination (DCE_1).

T_1	**On Terms**	**On Modules**
CF_1	$c_1\ op\ c_2 \overset{\mathrm{CF}}{\rightarrow} c$, where c is the result of $c_1\ op\ c_2$.	$\mathbb{M}\{[l \mapsto M]\} \overset{\mathrm{CF}}{\rightarrow} \mathbb{M}\{[l \mapsto M']\}$ where $M \overset{\mathrm{CP}}{\rightarrow} M'$;
CP_1	$(\lambda x.M)\ @\ c \overset{\mathrm{CP}}{\rightarrow} M[x := c]$	$\mathbb{M}\{[l \mapsto M]\} \overset{\mathrm{CP}}{\rightarrow} \mathbb{M}\{[l \mapsto M']\}$, where $M \overset{\mathrm{CP}}{\rightarrow} M'$; $\mathbb{M}\{[l_1 \mapsto c, l_2 \mapsto \mathbb{C}\{l_1\}]\} \overset{\mathrm{CP}}{\rightarrow} \mathbb{M}\{[l_1 \mapsto c, l_2 \mapsto \mathbb{C}\{c\}]\}$
FI_1:	$(\lambda x.M)\ @\ (\lambda y.N) \overset{\mathrm{FI}}{\rightarrow} M[x := \lambda y.N]$	$\mathbb{M}\{[l \mapsto M]\} \overset{\mathrm{FI}}{\rightarrow} \mathbb{M}\{[l \mapsto M']\}$, where $M \overset{\mathrm{FI}}{\rightarrow} M'$; $\mathbb{M}\{[l_1 \mapsto \lambda y.N, l_2 \mapsto \mathbb{C}\{l_1\}]\} \overset{\mathrm{FI}}{\rightarrow} \mathbb{M}\{[l_1 \mapsto \lambda y.N, l_2 \mapsto \mathbb{C}\{\lambda y.N\}]\}$
DCE_1	$(\lambda x.M)\ @\ V \overset{\mathrm{DCE}}{\rightarrow} \mathrm{M}$, where $x \notin FV(M)$	$\mathbb{M}\{[l \mapsto M]\} \overset{\mathrm{DCE}}{\rightarrow} \mathbb{M}\{[l \mapsto M']\}$, where $M \overset{\mathrm{DCE}}{\rightarrow} M'$; $[l_i \overset{n}{\underset{i=1}{\mapsto}} M_i, h_j \overset{M}{\underset{j=1}{\mapsto}} V_j] \overset{\mathrm{DCE}}{\rightarrow} [l_i \overset{n}{\underset{i=1}{\mapsto}} M_i]$, where $\forall_{1 \leq j \leq M}.h \notin \cup_{i=1}^{n} FL(M_i)$

All the above transformations are specified by a calculus step, and therefore are meaning preserving. Moreover, any combination of these transformations is also provable in the calculus, and therefore meaning preserving. For instance, the common combination of constant folding and constant propagation is also meaning preserving at both the term and module levels.

The following is an important property of certain transformations:

Definition 3.43 (Strong Normalization). A transformation T in $\mathcal{X} \times \mathcal{X}$ is *strongly normalizing* if for every term M in $\mathbf{Term}_{\mathcal{X}}$, there is no diverging path of Ts starting at M. □

For example, CP_1, CF_1, and DCE_1 are all strongly normalizing, since each of these steps monotonically reduces the number of redexes in a given term. On the other hand, FI_1 is not strongly normalizing, since it can be applied forever in cases like $(\lambda x.(x\ @\ x))\ @\ (\lambda y.(y\ @\ y))$.

Consider the reflexive, transitive closure T_1^* of a "one-step" transformation T_1. If T_1 is a calculus-based transformation, then so is T_1^*, so T_1^* is meaning-preserving in any computationally sound calculus. If T_1 is strongly normalizing, then clearly T_1^* is strongly normalizing, too.

If T_1 is both confluent and strongly normalizing, we define the *completion* of T_1 as the relation $\{(M, N) \mid M \xrightarrow{T_1}{}^{*} N$ and $N \in NF_{\xrightarrow{T_1}}\}$. We will write CP, CF and DCE for the completions of CP_1, CF_1 and DCE_1, respectively. We will also write CFP for the completion of $\mathrm{CP}_1 \cup \mathrm{CF}_1$. Since FI_1 is not strongly normalizing, its completion is not well-defined. However, we will consider a restricted form of function inlining in the core module calculus, FIM, that is the completion of a relation FIM_1 that only performs acyclic module substitutions. That is, the substitution

$$\mathbb{M}\{[l_1 \mapsto \lambda y.N, l_2 \mapsto \mathbb{C}\{l_1\}]\} \overset{\mathrm{FI}}{\rightarrow} \mathbb{M}\{[l_1 \mapsto \lambda y.N, l_2 \mapsto \mathbb{C}\{\lambda y.N\}]\}$$

is only in FIM_1 if N does not directly or indirectly refer to l_2. The FIM_1 relation is both confluent and strongly normalizing, so its completion is defined.

Our calculus is powerful enough to justify some cross-module transformations, such as the cross-module lambda-splitting transformation presented in section 1.1. Figure 5 presents a sequence of steps justifying lambda-splitting in the calculus $\mathcal{L}_{GC}$ defined in section 2.6 (i.e. a linking calculus over the core module calculus augmented with the rule (GC-nev)). In the figure, we assume that $\lambda y.M'$ is a closed abstraction. Since $\mathcal{L}_{GC}$ is computationally sound, the sequence is a proof that cross-module lambda-splitting is meaning preserving in $\mathcal{L}_{GC}$.

We emphasize that there are numerous common transformations that are *not* calculus-based and so their meaning preservation cannot be shown via this technique. For instance, example 3.30 shows that swapping the label arguments of + in a module expression is not-calculus based even though it is meaning preserving. As discussed in section 3.3, many global transformations are not handled by our framework.

$$
\begin{array}{lll}
[F \mapsto \lambda x.\mathbb{C}\{\lambda y.M'\}] \oplus [X \mapsto \mathbb{A}\{F \text{ @ } N\}] & \Rightarrow_{\mathcal{L}_{GC}} & \text{(link)} \\
[F \mapsto \lambda x.\mathbb{C}\{\lambda y.M'\}, X \mapsto \mathbb{A}\{F \text{ @ } N\}] & \hookleftarrow_{\mathcal{L}_{GC}} & \text{(GC-nev)} \\
[F \mapsto \lambda x.\mathbb{C}\{\lambda y.M'\}, h \mapsto \lambda y.M', X \mapsto \mathbb{A}\{F \text{ @ } N\}] & \hookleftarrow_{\mathcal{L}_{GC}} & \text{(mod-nev)} \\
[F \mapsto \lambda x.\mathbb{C}\{h\}, h \mapsto \lambda y.M', X \mapsto \mathbb{A}\{F \text{ @ } N\}] & \Rightarrow_{\mathcal{L}_{GC}} & \text{(mod-ev)} \\
[F \mapsto \lambda x.\mathbb{C}\{h\}, h \mapsto \lambda y.M', X \mapsto \mathbb{A}\{\lambda x.\mathbb{C}\{h\} \text{ @ } N\}] & \Leftarrow_{\mathcal{L}_{GC}} & \text{(hide)} \\
([F \mapsto \lambda x.\mathbb{C}\{F_{exp}\}, F_{exp} \mapsto \lambda y.M', X \mapsto \mathbb{A}\{\lambda x.\mathbb{C}\{F_{exp}\} \text{ @ } N\}])\{\text{hide } F_{exp}\} & \Leftarrow_{\mathcal{L}_{GC}} & \text{(link)} \\
([F \mapsto \lambda x.\mathbb{C}\{F_{exp}\}, F_{exp} \mapsto \lambda y.M'] \oplus [X \mapsto \mathbb{A}\{\lambda x.\mathbb{C}\{F_{exp}\} \text{ @ } N\}])\{\text{hide } F_{exp}\}. & &
\end{array}
$$

Figure 5: A sequence of calculus steps proving that lambda-splitting is meaning preserving in $\mathcal{L}_{GC}$.

For example, closure conversion [Han95, MMH96, SW97, Sis99]. assignment conversion [WS97], uncurrying [HH98], and useless variable elimination [WS99], are examples of non-calculus-based transformations. An interesting avenue for future research is to augment the calculus with rules that would justify some of these transformations (for instance, the label swapping transformation) in such a way that the resulting calculus is still computationally sound, similar to the way we have added the (GC) rule to $\mathcal{C}$.

4 Weak Distributivity

We say that a module transformation T is weakly distributive if and only if

$$T(D_1 \oplus D_2) = T(T(D_1) \oplus T(D_2)),$$

where $=$ is syntactic equality (modulo $\alpha_{\mathcal{C}}$-renaming).

Let T_{link} be a single module transformation performing all link-time optimizations. Suppose that the translator from source modules to intermediate modules is given by $s2i(D) = T_{\text{link}}(D)$[21]. Also suppose that the linking operator on intermediate modules is defined as $D_1 \oplus_{\text{link}} D_2 = T_{\text{link}}(D_1 \oplus D_2)$. Then if T_{link} is weakly distributive, we have that

$$
\begin{array}{rcl}
s2i(D_1) \oplus_{\text{link}} s2i(D_2) & = & T_{\text{link}}(T_{\text{link}}(D_1) \oplus T_{\text{link}}(D_2)) \\
& = & T_{\text{link}}(D_1 \oplus D_2) \\
& = & s2i(D_1 \oplus D_2).
\end{array}
$$

Thus, compiling a "link tree" of modules in the link-time compilation model gives exactly the same code as compilation in whole-program model. This is the sense in which weakly distributive transformations are promising candidates for link-time optimizations.

Here we discuss some conditions that imply that a module transformations T is weakly distributive. A simple class of weakly distributive transformations are those satisfying two conditions: (1) idempotence: $T(T(D)) = T(D)$; and (2) (strong) distributivity over $\oplus$: $T(D_1 \oplus D_2) = T(D_1) \oplus T(D_2)$. Suppose T satisfies these conditions. Then

$$T(D_1 \oplus D_2) = T(T(D_1 \oplus D_2)) = T(T(D_1) \oplus T(D_2))$$

Simple examples of such a transformation are the term-level transformations CF, CP, DCE defined in section 3.7. . Note that the second condition implies that the transformation independently transforms the components of a module; i.e., the transformation cannot use the (subst) or (GC) rule. For instance, the module-level versions of CF and CP are *not* strongly distributive, as shown in section 1.1.

For a second class of weakly distributive transformations, we consider transformations satisfying the following locality property:

[21] For simplicity, we assume the source and intermediate languages are the same.

Definition 4.1 (Locality). A module transformation T is *local* iff it is the completion of a confluent and strongly normalizing transformation T_1, and for all module contexts $\mathbb{M}$, $D \xrightarrow{T_1} D'$ implies $\mathbb{M}\{D\} \xrightarrow{T_1} \mathbb{M}\{D'\}$. □

Locality says that a module transformation that works on a set of bindings cannot be invalidated by adding more bindings. CF, CP, and CFP are all local transformations at the module level. However, DCE is non-local at the module level since adding more bindings can introduce a reference to a hidden label that can block the (GC) rule. The FIM transformation is also non-local. To see why, observe that

$$[X \mapsto \lambda w.Y, Z \mapsto \lambda x.X] \overset{\text{FIM}}{\to} [X \mapsto \lambda w.Y, Z \mapsto \lambda x.\lambda w.Y]$$

(the substitution is acyclic), but that

$$\mathbb{M}\{[X \mapsto \lambda w.Y, Z \mapsto \lambda x.X]\} \overset{\text{FIM}}{\not\to} \mathbb{M}\{[X \mapsto \lambda w.Y, Z \mapsto \lambda x.\lambda w.Y]\}$$

where $\mathbb{M} = [Y \mapsto \lambda y.Z, \square]$ (because there are no acyclic substitutions in the filled context).

Non-local transformations can block weak distributivity. For example, let $D_1 = [X \mapsto \lambda w.Y, Z \mapsto \lambda x.X]$ and $D_2 = [Y \mapsto \lambda y.Z]$. Then FIM(FIM($D_1$) $\oplus$ FIM(D_2)) yields the following result:

$$\begin{array}{ll} & [X \mapsto \lambda w.Y, Z \mapsto \lambda x.X] \oplus [Y \mapsto \lambda y.Z] \\ \overset{\text{FI}}{\to} & [X \mapsto \lambda w.Y, Z \mapsto \lambda x.\lambda w.Y] \oplus [Y \mapsto \lambda y.Z] \\ \to_{\mathcal{L}} & [X \mapsto \lambda w.Y, Z \mapsto \lambda x.\lambda w.Y, Y \mapsto \lambda y.Z] \\ \overset{\text{FI}}{\to} & [X \mapsto \lambda w.\lambda y.Z, Z \mapsto \lambda x.\lambda w.Y, Y \mapsto \lambda y.Z] \end{array}$$

But since the result of linking D_1 and D_2 has only cyclic substitutions, $\text{FIM}(D_1 \oplus D_2) = D_1 \oplus D_2$.

We have the following result for a restricted class of local transformations:

Lemma 4.2. *Suppose that T is a local transformation that is the completion of a confluent and strongly normalizing transformation T_1. Then T is weakly distributive.* □

Proof. Let $T(D_1) = D'_1$ and $T(D_2) = D'_2$, and D be the result of linking D_1 and D_2. Let $\mathbb{M}_1$ be a module context that contains all bindings of D_2 and a hole; i.e., $D = \mathbb{M}_1\{D_1\}$. By locality $\mathbb{M}_1\{D_1\} \xrightarrow{T_1}{}^* \mathbb{M}_1\{D'_1\}$. Let $\mathbb{M}_2$ be a module context that contains all bindings of of D'_1 and a hole; i.e., $\mathbb{M}_1\{D'_1\} = \mathbb{M}_2\{D_2\}$. By locality $\mathbb{M}_2\{D_2\} \xrightarrow{T_1}{}^* \mathbb{M}_2\{D'_2\}$. Note that $\mathbb{M}_2\{D'_2\} = D'$, where D' is the result of linking D'_1 and D'_2. Since $T(D)$ is a normal form of D with respect to T_1, and since T_1 is confluent by assumption, it must be the case that $T(D) = T(D')$, establishing that $T(D_1 \oplus D_2) = T(T(D_1) \oplus (D_2))$. □

5 Future Work

There are several directions in which we plan to extend the work presented here:

- *Types*: We are exploring several type systems for our module calculus, especially ones which express polymorphism via intersection and union types. These have intriguing properties for modular analysis and link-time compilation [Jim96, Ban97, KW99]. How to interface intersection and union types, which we view primarily as implementation types hidden from the user, with universal and exisential types specified by users is an especially important area for research.

- *Non-local Transformations*: So far, we have only considered meaning preservation and weak distributivity in the context of simple local transformations. We are investigating global transformations like closure conversion, uncurrying, and useless variable elimination in the context of link-time compilation. Inspired by results like those in [Ban97, FF97], we hypothesize that in many cases (1) the analyses upon which such transformations are based can be effectively modularized and (2) the results of such analyses can be stored with the module and combined at link-time without reanalyzing the module that results from linking.

- *Other Applications of Lift and Project*: Thus far we have only applied the lift/project technique of proving computational soundness to our module calculus. We are searching for other non-confluent calculi which may benefit from the application of our technique.
- *Weakening Weak Distributivity*: Weak distributivity requires the rather strong condition of synactic equality between $T(D_1 \oplus D_2)$ and $T(T(D_1) \oplus T(D_2))$. Weaker notions of equality may also be suitable. Note that "has the same meaning as" is *too* weak, since it does not capture the pragmatic relationship between the two sides; they should have "about the same efficiency".
- *Additional Language Features*: We plan to investigate extending our calculus with additional term and module language features, such as: term recursion, call-by-need evaluation, sharing, side effects, and nested modules. We will explore the impact of each extension on computational soundness.
- *Abstracting over the Base Language*: Our framework assumes that the module calculus is built upon a particular base calculus. Ideally we would like to define a module calculus over any base calculus that satisfies certain assumptions, but we have not yet found an effective way to abstract over the base calculus in such a way that we maintain our results. Ideas presented in [AZ99] for parameterizing modules over base languages may prove helpful in this regard.
- *Pragmatics*: We plan to empirically evaluate if link-time compilation can give reasonable "bang for the buck" in the context of a simple prototype compiler. In particular, a key question is whether any non-trivial compilation can be done *before* link-time. If not, then link-time compilation degenerates into a kind of modular approach to whole program compilation. But even this would be a point in the compiler design space worth exploring.

References

[AB97] Z. M. Ariola and S. Blom. Cyclic lambda calculi. In *TACS 97, Sendai, Japan*, 1997.

[Abr90] S. Abramsky. The lazy lambda calculus. In D. Turner, ed., *Research Topics in Functional Programming*, pp. 65–116. Addison Wesley, 1990.

[AF97] Z. M. Ariola and M. Felleisen. The call-by-need lambda calculus. *J. Funct. Programming*, 3(7), May 1997.

[AFM+95] Z. M. Ariola, M. Felleisen, J. Maraist, M. Odersky, and P. Wadler. The call-by-need lambda calculus. In *Conf. Rec. 22nd Ann. ACM Symp. Princ. of Prog. Langs.*, pp. 233–246. ACM, 1995.

[AK97] Z. M. Ariola and J. W. Klop. Lambda calculus with explicit recursion. *Inform. & Comput.*, 139(2):154–233, 15 Dec. 1997.

[AZ99] D. Ancona and E. Zucca. A primitive calculus for module systems. In Nadathur [PPDP99], pp. 62–79.

[BA97] M. Blume and A. Appel. Lambda-splitting: A higher order approach to cross-module optimization. In ICFP '97 [ICFP97], pp. 112–124.

[Ban97] A. Banerjee. A modular, polyvariant, and type-based closure analysis. In ICFP '97 [ICFP97].

[Bar84] H. P. Barendregt. *The Lambda Calculus: Its Syntax and Semantics*. North-Holland, revised edition, 1984.

[Car97] L. Cardelli. Program fragments, linking, and modularization. In POPL '97 [POPL97], pp. 266–277.

[CF58] H. B. Curry and R. Feys. *Combinatory Logic*, vol. 1. North-Holland, 1958.

[CHP99] K. Crary, R. Harper, and S. Puri. What is a recursive module? In *Proc. ACM SIGPLAN '99 Conf. Prog. Lang. Design & Impl.*, 1999.

[DEW99] S. Drossopoulou, S. Eisenbach, and D. Wragg. A fragment calculus — towards a model of separate compilation, linking and binary compatibility. In *Proc. 14th Ann. IEEE Symp. Logic in Computer Sci.*, July 1999.

[DS96] D. Duggan and C. Sourelis. Mixin modules. In *Proc. 1996 Int'l Conf. Functional Programming*, pp. 262–273. ACM Press, 1996.

[DS98] D. Duggan and C. Sourelis. Parameterized modules, recursive modules, and mixin modules. In *ACM SIGPLAN Workshop on ML*, pp. 87–96, 1998.

[Dug01] D. Duggan. Sharing in typed module assembly language. In TIC '00 [TIC01]. To be published in the LNCS series.

[DWM+01] A. Dimock, I. Westmacott, R. Muller, F. Turbak, and J. B. Wells. Functioning without closure: Type-safe customized function representations for Standard ML. In *Proc. 2001 Int'l Conf. Functional Programming*. ACM Press, 2001.

[ESOP00] *Programming Languages & Systems, 9th European Symp. Programming*, vol. 1782 of *LNCS*. Springer-Verlag, 2000.

[Fer95] M. F. Fernandez. Simple and effective link-time optimization of Modula-3 programs. In *Proc. ACM SIGPLAN '95 Conf. Prog. Lang. Design & Impl.*, pp. 103–115, 1995.

[FF86] M. Felleisen and D. Friedman. Control operators, the SECD-machine, and the λ-calculus. In M. Wirsing, ed., *Formal Description of Programming Concepts — III*, pp. 193–219. North-Holland, 1986.

[FF97] C. Flanagan and M. Felleisen. Componential set-based analysis. In PLDI '97 [PLDI97].

[FF98] M. Flatt and M. Felleisen. Units: Cool modules for HOT languages. In *Proc. ACM SIGPLAN '98 Conf. Prog. Lang. Design & Impl.*, pp. 236–248, 1998.

[FH92] M. Felleisen and R. Hieb. The revised report on the syntactic theories of sequential control and state. *Theoret. Comput. Sci.*, 102:235–271, 1992.

[FRR00] K. Fisher, J. Reppy, and J. G. Riecke. A calculus for compiling and linking classes. In ESOP '00 [ESOP00], pp. 135–149.

[GLM92] G. Gonthier, J.-J. Lévy, and P.-A. Melliès. An abstract standardisation theorem. In *Proc. 7th Ann. IEEE Symp. Logic in Computer Sci.*, 1992.

[GM99] N. Glew and G. Morrisett. Type-safe linking and modular assembly language. In POPL '99 [POPL99], pp. 250–261.

[Han95] J. Hannan. Type systems for closure conversion. In *Workshop on Types for Program Analysis*, pp. 48–62, 1995. The TPA '95 proceedings are DAIMI PB-493.

[HH98] J. Hannan and P. Hicks. Higher-order uncurrying. In *Conf. Rec. POPL '98: 25th ACM Symp. Princ. of Prog. Langs.*, pp. 1–11, 1998.

[HL91] G. Huet and J.-J. Lévy. Computations in orthogonal rewriting systems, I. In J.-L. Lassez and G. Plotkin, eds., *Computational Logic: Essays in Honor of Alan Robinson*, pp. 395–414. MIT Press, 1991.

[HL94] R. Harper and M. Lillibridge. A type-theoretic approach to higher-order modules with sharing. In POPL '94 [POPL94], pp. 123–137.

[HWC01] M. Hicks, S. Weirich, and K. Crary. Safe and flexible dynamic linking of native code. In TIC '00 [TIC01]. To be published in the LNCS series.

[ICFP97] *Proc. 1997 Int'l Conf. Functional Programming*. ACM Press, 1997.

[Jim96] T. Jim. What are principal typings and what are they good for? In POPL '96 [POPL96].

[Kah87] G. Kahn. Natural semantics. In *Proceedings of STACS '87, 4th Annual Symposium on Theoretical Aspects of Computer Science*, vol. 247 of *LNCS*, pp. 22–39. Springer-Verlag, 1987.

[KW99] A. J. Kfoury and J. B. Wells. Principality and decidable type inference for finite-rank intersection types. In POPL '99 [POPL99], pp. 161–174.

[Lan64] P. J. Landin. The mechanical evaluation of expressions. *The Computer Journal*, pp. 308–320, Jan. 1964.

[Ler94] X. Leroy. Manifest types, modules, and separate compilation. In POPL '94 [POPL94], pp. 109–122.

[Ler96] X. Leroy. A modular module system. Tech. Rep. 2866, INRIA, Apr. 1996.

[MMH96] Y. Minamide, G. Morrisett, and R. Harper. Typed closure conversion. In POPL '96 [POPL96].

[MT00] E. Machkasova and F. A. Turbak. A calculus for link-time compilation. In ESOP '00 [ESOP00], pp. 260–274.

[PC97] M. P. Plezbert and R. K. Cytron. Is "just in time" = "better late than never"? In POPL '97 [POPL97], pp. 120–131.

[Pit97] A. M. Pitts. Operationally-based theories of program equivalence. In P. Dybjer and A. M. Pitts, eds., *Semantics and Logics of Computation*, Publications of the Newton Institute, pp. 241–298. Cambridge University Press, 1997.

[PLDI97] *Proc. ACM SIGPLAN '97 Conf. Prog. Lang. Design & Impl.*, 1997.

[Plo75] G. D. Plotkin. Call-by-name, call-by-value and the lambda calculus. *Theoret. Comput. Sci.*, 1:125–159, 1975.

[Plo81] G. D. Plotkin. A structural approach to operational semantics. Technical Report DAIMI FN-19, Aarhus University Computer Science Department, Sept. 1981.

[POPL94] ACM. *Conf. Rec. 21st Ann. ACM Symp. Princ. of Prog. Langs.*, 1994.

[POPL96] ACM. *Conf. Rec. POPL '96: 23rd ACM Symp. Princ. of Prog. Langs.*, 1996.

[POPL97] *Conf. Rec. POPL '97: 24th ACM Symp. Princ. of Prog. Langs.*, 1997.

[POPL99] *Conf. Rec. POPL '99: 26th ACM Symp. Princ. of Prog. Langs.*, 1999.

[PPDP99] G. Nadathur, ed. *Proc. Int'l Conf. on Principles and Practice of Declarative Programming*, vol. 1702 of *LNCS*, Paris, France, 29 Sept. – 1 Oct. 1999. Springer-Verlag.

[PS98] A. M. Pitts and I. D. B. Stark. Operational reasoning for functions with local state. In A. D. Gordon and A. M. Pitts, eds., *Higher Order Operational Techniques in Semantics*, Publications of the Newton Institute, pp. 227–273. Cambridge University Press, 1998.

[Rus99] C. Russo. Non-dependent types for Standard ML modules. In Nadathur [PPDP99], pp. 80–97.

[SA93] Z. Shao and A. Appel. Smartest recompilation. In *Conf. Rec. 20th Ann. ACM Symp. Princ. of Prog. Langs.*, 1993.

[Sha98] Z. Shao. Typed cross-module compilation. In *Proc. 1998 Int'l Conf. Functional Programming*, pp. 141–152. ACM Press, 1998.

[Sha99] Z. Shao. Transparent modules with fully syntactic signatures. In *Proc. 1999 Int'l Conf. Functional Programming*, pp. 220–232. ACM Press, 1999.

[Sis99] J. M. Siskind. Flow-directed lightweight closure conversion. Technical Report 99-190R, NEC Research Institute, Inc., Dec. 1999.

[SW97] P. Steckler and M. Wand. Lightweight closure conversion. *ACM Trans. on Prog. Langs. & Systs.*, 19(1):48–86, Jan. 1997.

[TIC01] *Proceedings of the Third Workshop on Types in Compilation (TIC 2000)*, vol. 2071 of *LNCS*. Springer-Verlag, 2001. To be published in the LNCS series.

[WM00] J. B. Wells and R. Muller. Standardization and evaluation in Combinatory Reduction Systems. Unpublished draft to be submitted, 2000.

[WS97] M. Wand and G. T. Sullivan. Denotational semantics using an operationally-based term model. In POPL '97 [POPL97], pp. 386–399.

[WS99] M. Wand and I. Siveroni. Constraint systems for useless variable elimination. In POPL '99 [POPL99], pp. 291–302.

[WV99] J. B. Wells and R. Vestergaard. Confluent equational reasoning for linking with first-class primitive modules (long version). A short version is [WV00]. Full paper with three appendices for proofs, Aug. 1999.

[WV00] J. B. Wells and R. Vestergaard. Equational reasoning for linking with first-class primitive modules. In ESOP '00 [ESOP00], pp. 412–428. A long version is [WV99].

A A General Proof of Lift and Project

A.1 What to Call This Subsection?

In section 3 we have discussed two techniques of proving computational soundness of a calculus (property 3.16): one by showing that the calculus has confluence 2.6 and standardization 2.7, the other by showing the properties lift 3.37 and project 3.38. In both cases the calculus has to have the class preservation property 3.31. Additionally, in the second case we require that the calculus evaluation relation is confluent (if it is non-deterministic). In fact, as we see later in this section, we prove lift and project using properties of evaluation related to confluence (see properties A.22 and A.23). Even though these properties are, strictly speaking, independent from confluence, it is likely to be the case that proofs of these properties and of confluence have a lot in common, as it is the case with our calculi.

In this section we give the next (after section 3), a more detailed layer of the "top-down" presentation of the soundness proof - a general (i.e. independent of details of a particular calculus) proof of lift and project via certain properties of developments. Note that, since standardization is equivalent to lift (see lemma 3.39), some of the same properties of developments can be used for a confluent calculus to show standardization (we use this approach for the term calculus $\mathcal{T}$ - see section B) The subsequent sections provide calculus-specific proofs of the properties postulated in this section for each of the calculi.

Since in this section we are concerned only with a general proof, we omit calculus-specific definitions of some of the notions used in the proofs, for instance the notion of a residual or a development. Instead, we give axiomatic definitions of these notions, i.e. lists of requirements that a particular definition has to satisfy. However, in all the cases where a definition can be given precisely via notions that have already been defined, we give a precise definition, rather than an axiomatic one.

Convention A.1. Even though the discussion in this section is concerned only with general properties of a calculus, we use more traditional notations used for the term calculus (for instance, M, N for terms, $\mathbb{A}, \mathbb{B}, \mathbb{C}$ for contexts) We omit a calculus subscript for relations denoted by arrows in this section, unless we want to stress that we are dealing with a particular calculus. □

Formalizing notions of a redex, residuals, and a development. The notions of a redex, a residual, and a development are widely used in literature. Formal definitions of these notions are given, in particular, in [Bar84] for a call-by-name lambda-calculus. Our definitions are given from a slightly different point of view than those by Barendregt. In particular, we work explicitly with contexts containing redexes, and rather than using annotated ("marked") terms for defining a development, we use explicit sets of "marked" redexes. This way of defining redexes and developments is better suited for the proof techniques that we use (for instance, working with multi-hole contexts). For some of our calculi (in particular, for $\mathcal{C}$) we also have to define a development in a way that's different from the traditional definition to guarantee that developments are finite and confluent.

Redex. The notion of a *redex* is fairly intuitive. However, a formal definition of a redex presents some challenges. Consider the term $(\lambda x.2+3)$ @ $(\lambda y.2+3)$ in the calculus $\mathcal{T}$. If we say that $2+3$ is a redex in this term, then we get the following ambiguity: both $(\lambda x.5)$ @ $(\lambda y.2+3)$ and $(\lambda x.2+3)$ @ $(\lambda y.5)$ are obtained by reducing a redex $2+3$ in the original term. In this presentation we resolve this ambiguity by specifying not only the redex itself, but also the context that contains it. Let us write such context/redex pairs corresponding to each of the two redexes: the first redex is written as a pair $((\lambda x.\square)$ @ $(\lambda y.2+3), 2+3)$, and the second one as $((\lambda x.2+3)$ @ $(\lambda y.\square), 2+3)$. We use this notation to denote that a term M gets reduced to N by the context/redex pair $(\mathbb{C}, R)$: $M \xrightarrow{(\mathbb{C},R)} N$.

One may argue that specifying the term N (the result of the reduction) resolves the ambiguity on its own, without the need for context/redex pairs. Indeed, given M and N s.t. $M \longrightarrow N$, it seems that one can uniquely identify the context/redex pair in M that got reduced in the reduction step (for instance, knowing the resulting term in the reduction of the example above certainly allows us to determine which of the two redexes $2+3$ has been reduced). However, consider the following term from the call-by-name λ-calculus: $(\lambda x.x)$ @ $((\lambda y.y)$ @ $(\lambda z.z))$. Reducing the first application results in the term $(\lambda y.y)$ @ $(\lambda z.z)$, reducing the second one gives $(\lambda x.x)$ @ $(\lambda z.z)$, but the latter two terms are the same up to α-renaming! Moreover, if we had chosen to use abbreviation I for $\lambda x.x$, the original term would've been written (omitting the symbol @ for application) as $I(II)$, and the results of *both* reductions as II (see also a similar example 3.1.19 due to Levy in [Bar84]). To avoid dealing with these ambiguities, we consider a redex to be a context/redex pair, as illustrated above.

Definition A.2. A pair $(\mathbb{C}, R)$ is called a *redex* of a term M in a calculus $\mathcal{X}$ with reduction $\longrightarrow_{\mathcal{X}}$ defined by rules of the form $\mathbb{C}\{R\} \longrightarrow_{\mathcal{X}} \mathbb{C}\{Q\}$ if $\mathbb{C}\{R\}$ matches the left-hand side of one of the rules and satisfies all the restrictions of the rule. This implies that there exists $N = \mathbb{C}\{Q\}$ s.t. $M \longrightarrow_{\mathcal{X}} N$, and we write $M \overset{(\mathbb{C},R)}{\longrightarrow}_{\mathcal{X}} N$.

If the reduction is an evaluation step in $\mathcal{X}$, then we also write $M \overset{(\mathbb{C},R)}{\Longrightarrow}_{\mathcal{X}} N$, and similarly $M \overset{(\mathbb{C},R)}{\circ\!\!\longrightarrow}_{\mathcal{X}} N$ for a non-evaluation step. □

The condition that $\mathbb{C}\{R\}$ matches the left-hand side of one of reduction rules and satisfies the restrictions of the rule guarantees that $(\mathbb{C}, R)$ is indeed a redex. Note that the restrictions may affect both the context and a redex. For instance, $\mathbb{D}\{l\}$ is a redex in $\mathcal{C}$ only if l is bound to a value in the context $\mathbb{D}$. Another example of restrictions on the context is the rule (mod-ev) in the calculus $\mathcal{L}$: the rule is applied only in an empty context, i.e. in this case $D \xRightarrow{(\square,D)}_{\mathcal{L}} D'$. The definition does not require that R and Q are terms in the calculus $\mathcal{X}$ and $\mathbb{C}$ is a context in $\mathcal{X}$. This is because in some cases reductions of a calculus are defined via embedding of the relation of one calculus into the relation of another one (see definition 3.17). For instance, in the calculus $\mathcal{C}$ (without GC) $R, Q \in \mathbf{Term}_{\mathcal{T}}$.

Specifying the context together with the subterm being reduced allows us to resolve ambiguities like the one mentioned above: given the term $(\lambda x.x)$ @ $((\lambda y.y)$ @ $(\lambda z.z))$, we can distinguish the two reductions, because the occur in different contexts: one reduces the redex $(\square, (\lambda x.x)$ @ $((\lambda y.y)$ @ $(\lambda z.z))$, and the other reduces $((\lambda x.x)$ @ $\square, (\lambda y.y)$ @ $(\lambda z.z))$.

Residual. Definition of a residual is calculus-specific. Below we give an axiomatic definition, i.e. requirements that such a definition has to satisfy. As an example of such a definition see B.5.

Definition A.3 (Axiomatic Definition of a Residual). Let $(\mathbb{C}_1, R_1)$ and $(\mathbb{C}_2, R_2)$ be two redexes in a term M, and let $M \xrightarrow{(\mathbb{C}_1,R_1)} N$ for some calculus $\mathcal{X}$. A *set of residuals of* $(\mathbb{C}_2, R_2)$ *w.r.t.* $(\mathbb{C}_1, R_1)$ denoted by $(\mathbb{C}_2, R_2)/(\mathbb{C}_1, R_1)$ is a set of *redexes* in N which satisfies the following property:

1. Let $(\mathbb{C}_1, R_1), (\mathbb{C}_2, R_2)$, and $(\mathbb{C}_3, R_3)$ be 3 redexes in M s.t. $(\mathbb{C}_2, R_2) \neq (\mathbb{C}_3, R_3)$, and suppose $M \xrightarrow{(\mathbb{C}_1,R_1)} N$. Then $(\mathbb{C}_2, R_2)/(\mathbb{C}_1, R_1) \cap (\mathbb{C}_3, R_3)/(\mathbb{C}_1, R_1) = \emptyset$.

$\square$

Note that by definition a residual of a redex is itself a redex. The requirement states that two distinct redexes in a term can not have the same residual after a reduction, i.e. any residual originates from only one redex in the term being reduced. Note that we do not require that $(\mathbb{C}_2, R_2)$ and $(\mathbb{C}_3, R_3)$ are different from $(\mathbb{C}_1, R_1)$. This is because in some calculi (in particular, in $\mathcal{C}$) $(\mathbb{C}_1, R_1)/(\mathbb{C}_1, R_1)$, i.e. a set of residuals of the redex being reduced, may be non-empty. Also note that a residual set is always finite, since any term can be parsed as a context/redex pair only in a finite number of ways.

Residuals of Sets. We extend the notion of residuals to reduction sequences in a straightforward manner.

Definition A.4. Let $M \xrightarrow{(\mathbb{C},R)} M'$, and let F and F' be sets of redexes of M and M' respectively. Then F' is the *set of residuals of* F *w.r.t. the redex* $(\mathbb{C}, R)$, denoted as $F/(\mathbb{C}, R)$, if $F' = \bigcup\limits_{(\mathbb{C}',R') \in F} (\mathbb{C}', R')/(\mathbb{C}, R)$. $\square$

Let S be an ordered sequence of redexes which is either empty (denoted by ϵ), or a single redex $(\mathbb{C}, R)$, or is of the form $S'; (\mathbb{C}, R)$, where S' is another sequence. We write $M \xrightarrow{S}^* M'$ to indicate that the reduction sequence reduces all redexes in S in the left-to-right order. For instance, $M \xrightarrow{(\mathbb{C}_1,R_1);(\mathbb{C}_2,R_2)}^* M'$ denotes the following reduction sequence: $M \xrightarrow{(\mathbb{C}_1,R_1)} M_1 \xrightarrow{(\mathbb{C}_2,R_2)} M'$. Note that $(\mathbb{C}_2, R_2)$ is a redex in M_1. If $S_1 = \epsilon$, then by convention both $S_1; S_2$ and $S_2; S_1$ denote the sequence S_2.

Definition A.5. Let $(\mathbb{C}, R)$ be a redex in a term M, and suppose $M \xrightarrow{S}^* M'$. Then the *set of residuals of* $(\mathbb{C}, R)$ *w.r.t. the sequence* S, denoted as $(\mathbb{C}, R)/S$, is a set of redexes of M' defined as follows:

- If $S = \epsilon$, then $(\mathbb{C}, R)/S = (\mathbb{C}, R)$,
- If $S = (\mathbb{C}', R')$, then $(\mathbb{C}, R)/S = (\mathbb{C}, R)/(\mathbb{C}', R')$,
- If $S = S'; (\mathbb{C}', R')$, where S' is non-empty, then $(\mathbb{C}, R)/S = F/(\mathbb{C}', R')$, where $F = (\mathbb{C}, R)/S'$.

$\square$

Note that in the last case of the definition $(\mathbb{C}, R)/S'$ is a set of redexes in the term M'' s.t. $M \xrightarrow{S'}^* M'' \xrightarrow{(\mathbb{C}',R')} M'$, and the residuals of such sets are defined by definition A.4.

We also introduce a notation F/S, where F is a set of redexes, and S is a sequence, in the obvious way.

Extended calculus relation. To be able to reason about developments, we need to extend our calculus relation to terms in which some of the redexes are marked. Rather than defining a relation on annotated terms, we specify a *set* of "marked" redexes together with the term (note that this corresponds to the situation when all "marked" redexes are annotated with the same symbol), and define a new calculus relation on pairs of a term and a set of its "marked" redexes.

Definition A.6. Let M be a term and let F be a set of redexes. (M, F) is called a *well-formed* pair if for all $(\mathbb{C}, R) \in F$ $\mathbb{C}\{R\} = M$, i.e. all elements of F are redexes of M. □

Definition A.7. Let (M, F) be a well-formed pair. We say that a (M, F) reduces to the pair (M', F') by contracting a redex $(\mathbb{C}, R)$ (written as $(M, F) \xrightarrow{(\mathbb{C},R)} (M', F')$) if $M \xrightarrow{(\mathbb{C},R)} M'$, where $(\mathbb{C}, R)$ is a redex of M, not necessarily in F, and $F' = \{(\mathbb{C}', R') \mid (\mathbb{C}', R') \in (\mathbb{C}'', R'')/(\mathbb{C}, R),\ \text{where } (\mathbb{C}'', R'') \in F\}$. We call this reduction the *extended calculus reduction* (or just the *extended reduction*), and denote it by the same symbol as the calculus reduction on terms. □

Convention A.8. The notation (M, F) assumes that the pair is well-formed, unless specified otherwise. □

Lemma A.9. *If* $(M, \emptyset) \longrightarrow (N, F)$*, then* $F = \emptyset$. □

Proof. By definition A.7 F is the union of the sets of residuals of redexes in the set of "marked" redexes of the first pair. Since the set of such redexes is empty, $F = \emptyset$. □

We write $(M, F) \xrightarrow{S}{}^{*} (M', F')$, to denote a reduction sequence that reduces redexes in S in the left-to-right order, completely analogous to $M \xrightarrow{S}{}^{*} M'$.

Given a calculus $\mathcal{X}$ with a calculus relation $\longrightarrow_{\mathcal{X}}$ and definitions of a redex and a residual in this calculus, the extended reduction is uniquely defined.

We write $(M, F) \xrightarrow{S}{}^{*} (M', F')$, to denote a reduction sequence that reduces redexes in S in the left-to-right order, completely analogous to $M \xrightarrow{S}{}^{*} M'$.

Developments and γ-developments . The extended reduction is closely connected to the notion of a *development*, which is one of the most important notions in this presentation. Informally, a development is a reduction that reduces only "marked" redexes, i.e., in our presentation, redexes in F. The two fundamental properties that we would like developments to have are finiteness and standardization. Finiteness of developments (see property A.37) means that for a given pair (M, F) there is no infinite reduction that reduces only marked redexes (i.e. residuals of those in F). Standardization of developments means that for any development sequence $(M_1, F_1) \underset{dev}{\rightarrow}^{*} (M_2, F_2)$ there exists a standard development $(M_1, F_1) \underset{dev}{\Longrightarrow}^{*} (M', F') \underset{dev}{\circ\!\!\longrightarrow}^{*} (M_2, F_2)$ (see property A.29). Note that standardization of developments is different from standardization of the calculus (property 2.7) which claims that for every calculus reduction (not necessarily a development) there exists a standard reduction with the same starting and ending terms.

These properties indeed hold for the term calculus $\mathcal{T}$ for developments defined in a traditional way (see definition A.10). However, it turns out (see examples below) that for the calculus $\mathcal{C}$ (and consequently for $\mathcal{L}$) with developments defined in a traditional way none of the above two properties holds. Therefore we introduce another reduction on pairs (M, F), which we call a γ-development and denote $\underset{\gamma}{\rightarrow}$. Similarly to developments, it reduces only marked redexes. However, it is defined only on a subset of well-formed pairs (M, F) and only a subset of residuals of a marked redex is marked after the reduction. As with the definition of residuals (definition A.3), in this section we give an axiomatic definition of a γ-development (definition A.11), postponing the calculus-specific details until the sections on the respective calculi. With the generalized definition of γ-developments in the core module calculus $\mathcal{C}$ we are able to show finiteness of developments. The γ-developments of the module calculus still do not have general standardization of developments A.29, but we are able to show a weaker property A.31 (standardization of a certain kind of γ-developments) which is sufficient for our proofs. Below we give details and examples.

A traditional and straightforward way to define a development reduction step is as follows:

Definition A.10 (Definition of a Development via Extended Reduction). We say that (M, F) is reduced to (M', F') by a *development step*, and write $(M, F) \underset{dev}{\overset{(\mathbb{C},R)}{\rightarrow}} (M', F')$, if $(M, F) \xrightarrow{(\mathbb{C},R)} (M', F')$ and $(\mathbb{C}, R) \in F$. □

The following example illustrates this approach for the term calculus $\mathcal{T}$.

Example 1. In the term calculus $\mathcal{T}$, consider a term $(\lambda x.xx) @ (\lambda y.2+3)$, and suppose the redex in the second subterm is marked. We denote this term (with the marked redex) as the pair $((\lambda x.xx) @ (\lambda y.2+3), \{((\lambda x.xx) @ (\lambda y.\Box) 3)\})$, where the second component is the set consisting of a single context/redex pair corresponding to the marked redex. Consider a reduction on this pair that reduces the redex in the marked redex:

$$((\lambda x.x @ x) @ (\lambda y.2+3), \{((\lambda x.x @ x) @ (\lambda y.\Box), 2+3)\}) \underset{\gamma}{\rightarrow} ((\lambda x.x @ x) @ (\lambda y.5), \emptyset).$$

This reduction can be considered a development because it reduces a marked redex. According to the rules of reductions on pairs (M, F) the resulting pair has an empty set as the second component, i.e. it has no marked redexes. Now let us consider a different reduction starting from the same pair:

$$\begin{array}{r} ((\lambda x.x @ x) @ (\lambda y.2+3), \{((\lambda x.x @ x) @ (\lambda y.\Box), 2+3)\}) \quad \longrightarrow \\ ((\lambda y.2+3) @ (\lambda y.2+3), \{((\lambda y.\Box) @ (\lambda y.2+3), 2+3), ((\lambda y.2+3) @ (\lambda y.\Box), 2+3)\}). \end{array}$$

This reduction also satisfies the definition of the extended reduction, but it is not a development, since the redex reduced in this reduction, i.e. $(\Box, (\lambda x.x @ x) @ (\lambda y.2+3))$, is not in the set of marked redexes $\{((\lambda x.x @ x) @ (\lambda y.\Box), 2+3)\}$. Performing a reduction of any one of the two redexes in the above set would be a development step. If we reduce both redexes, then the resulting pair has $\emptyset$ as its second component, i.e. it does not have any marked redexes:

$$\begin{array}{rl} ((\lambda y.2+3) @ (\lambda y.2+3), \{((\lambda y.\Box) @ (\lambda y.2+3), 2+3), ((\lambda y.2+3) @ (\lambda y.\Box), 2+3)\}) & \underset{\gamma}{\rightarrow} \\ ((\lambda y.5) @ (\lambda y.2+3), \{((\lambda y.5) @ (\lambda y.\Box), 2+3)\}) & \underset{\gamma}{\rightarrow} \\ ((\lambda y.5) @ (\lambda y.5), \emptyset). & \end{array}$$

It is easy to check that reducing the two redexes in the other order gives the same result. We also see that all the "development sequences" in this example lead to a pair with $\emptyset$ as the second component, i.e. they are finite. It is straightforward to show (see section B) that in the case of $\mathcal{T}$ developments defined this way are both finite and confluent.

The following examples, however, show that this approach does not produce finite and confluent developments for all calculi. For instance, the example below shows a non-finite reduction in the core module calculus $\mathcal{C}$ which reduces only marked redexes, i.e. satisfies the definition of a development A.10.

Example 2. Consider the calculus $\mathcal{C}$, where a context is defined on figure 1, a redex is defined by A.2 and a residual is defined in definition C.7. Consider the following reduction:

$$\begin{array}{rl} ([A \mapsto \lambda x.A], \{([A \mapsto \lambda x.\Box], A)\}) & \longrightarrow_{\mathcal{C}} \\ ([A \mapsto \lambda x.\lambda x.A], \{([A \mapsto \lambda x.\lambda x.\Box], A)\}) & \longrightarrow_{\mathcal{C}} \quad \ldots \end{array}$$

We see that the use of the extended reduction in this case produces an infinite sequence s.t. at every step the contracted redex is in F in the pair being reduced. Therefore, if we define a development step in $\mathcal{C}$ as in definition A.10, we get a non-finite development.

Example 3. This example illustrates that not only finiteness of developments, but also their confluence fails if we define developments in $\mathcal{C}$ by definition A.10. This example adopts the non-confluence example given in section 2.4 to reductions on pairs: suppose in the module $[A \mapsto \lambda x.B, B \mapsto \lambda y.A]$ both substitution redexes are "marked", i.e. included in F, then we get:

$$\begin{array}{r} ([A \mapsto \lambda x.B, B \mapsto \lambda y.A], \{([A \mapsto \lambda x.\Box, B \mapsto \lambda y.A], B), ([A \mapsto \lambda x.B, B \mapsto \lambda y.\Box], A)\}) \quad \longrightarrow_{\mathcal{C}} \\ ([A \mapsto \lambda x.\lambda y.A, B \mapsto \lambda y.A], \{([A \mapsto \lambda x.\lambda y.\Box, B \mapsto \lambda y.A], A), ([A \mapsto \lambda x.\lambda y.A, B \mapsto \lambda y.\Box], A)\}), \\ ([A \mapsto \lambda x.B, B \mapsto \lambda y.A], \{([A \mapsto \lambda x.\Box, B \mapsto \lambda y.A], B), ([A \mapsto \lambda x.B, B \mapsto \lambda y.\Box], A)\}) \quad \longrightarrow_{\mathcal{C}} \\ ([A \mapsto \lambda x.B, B \mapsto \lambda y.\lambda x.B], \{([A \mapsto \lambda x.\Box, B \mapsto \lambda y.\lambda x.B], B), ([A \mapsto \lambda x.B, B \mapsto \lambda y.\lambda x.\Box], B)\}). \end{array}$$

As we have seen above, the two resulting modules do not reduce to the same module by the calculus reduction, and therefore by the extended calculus reduction. We avoid situations like this by restricting the domain of a

γ-development step. Even though we do not get general confluence of developments even with the restriction, we get a reduction that's sufficiently "well-behaved" so that γ-developments are standartizible.

Below we give definition of γ-developments and some related definitions. After that we state the properties of γ-developments that allow us to prove lift and project.

Definition A.11 (Axiomatic Definition of a γ-development Step). A γ-*development* is defined by a set $dom(\gamma)$ of well-formed pairs (M, F) s.t. $(M, \{(\mathbb{C}, R)\}) \in dom(\gamma))$ for all well-formed pairs $(M, \{(\mathbb{C}, R)\})$ s.t. $(\mathbb{C}, R)$ is a non-evaluation redex, and by a relation $\underset{\gamma}{\rightarrow}$ s.t. $(M, F) \underset{\gamma}{\rightarrow} (M', F'))$ if $(M, F) \in dom(\gamma)$ and

- either $M = M'$, $F' \subset F$, and $(M', F') \in dom(\gamma)$ if $F' \neq \emptyset$,
- or there exists $(\mathbb{C}, R)$ s.t. the 5 conditions below are satisfied (in this case we write $(M, F) \underset{\gamma}{\overset{(\mathbb{C},R)}{\rightarrow}} (M', F')$):
  1. $M \xrightarrow{(\mathbb{C},R)} M'$, i.e. a γ-development projected to terms is a calculus reduction on terms,
  2. $(\mathbb{C}, R) \in F$ (the contracted redex is "marked"),
  3. If $(M, F) \xrightarrow{(\mathbb{C},R)} (M', F'')$, then $F' \subseteq F''$. Here $\xrightarrow{(\mathbb{C},R)}$ is the extended calculus reduction that reduces the same redex. The requirement says that F' can not have any "extra" redexes in addition to those in F'', i.e. in a γ-development some of the redexes that would've been marked by the extended reduction are not marked. By excluding from F' some of the redexes that are included in F'' we are able to guarantee finiteness of γ-developments for $\mathcal{C}$.
  4. If $(M, F) \underset{\gamma}{\rightarrow} (M', F')$ s.t. $F' \neq \emptyset$, then there exists (M'', F'') s.t. $(M', F') \underset{\gamma}{\overset{(\mathbb{C}',R')}{\rightarrow}} (M'', F'')$ for every $(\mathbb{C}', R') \in F'$. This requirement guarantees that if a γ-development is defined for a pair (M, F), then it is defined for all descendants of this pair in a sequence of γ-development steps.
  5. For all $(\mathbb{C}, R) \in F$ $(\mathbb{C}, R)/(\mathbb{C}, R) \cap F' = \emptyset$. This guarantees finiteness of developments.

If $(M, F) \underset{\gamma}{\rightarrow} (M', F')$, then we say that (M, F) is related to (M', F') by a γ-development step. □

The last condition is necessary to prevent infinite reductions, s.a. the one in example 2.

Note that the reduction defined in definition A.10 satisfies all of the above requirements. In fact, it can be obtained from the axiomatic definition by setting $F' = F''$ in part 3 of the requirements and defining the domain of $\underset{\gamma}{\rightarrow}$ to be the same as the domain of the extended relation (which consists of all pairs (M, F) s.t. all elements of F are redexes in M, see convention A.8). It is easy to check that in this case all the other requirements are satisfied.

In this section we give a proof of lift and project assuming the axiomatic definition of γ-developments (definition A.11). In the next section we'll show how some of the properties that the proof is based on can be deduced assuming the particular case of γ-developments given in definition A.10. For the rest of this section a γ-*development step* is assumed to be as specified in A.11.

We introduce the notation $(M, F) \underset{\gamma}{\rightarrow} (M', F')$, omitting the redex. We use the notation $(M_1, F_1) \underset{\gamma}{\overset{S}{\rightarrow}}{}^{*} (M_2, F_2)$ to denote a reduction sequence S on pairs of the form (M, F), where every reduction step is a γ-development step; we call such a sequence a γ-*development* of (M, F). Analogously to $\xRightarrow{(\mathbb{E},R)}$ and $\circ\!\xrightarrow{(\mathbb{C},R)}$, we introduce notations $\underset{\gamma}{\xRightarrow{(\mathbb{E},R)}}$ for a γ-development step that reduces an evaluation redex, and $\circ\!\underset{\gamma}{\xrightarrow{(\mathbb{C},R)}}$ for a γ-development step with a non-evaluation redex. Note that if $(M_1, F_1) \underset{\gamma}{\overset{S}{\rightarrow}}{}^{*} (M_2, F_2)$, then $M_1 \xrightarrow{S}{}^{*} M_2$.

Definition A.12. A *combined evaluation relation* denoted by $\underset{\cup}{\Rightarrow}$ is defined as follows: $(M, F) \underset{\cup}{\overset{(\mathbb{C},R)}{\Longrightarrow}} (M', F')$ if either $(M, F) \overset{(\mathbb{C},R)}{\Longrightarrow} (M', F')$ or $(M, F) \underset{\gamma}{\overset{(\mathbb{C},R)}{\Longrightarrow}} (M', F')$. The reflexive transitive closure of $\underset{\cup}{\Rightarrow}$ is denoted by $\underset{\cup}{\Rightarrow}^*$. □

A complete γ-development is defined in a traditional manner:

Definition A.13. A γ-development $(M, F) \underset{\gamma}{\rightarrow}^* (M', \emptyset)$ is called a *complete γ-development of* (M, F). We denote it by $(M, F) \underset{c-\gamma}{\twoheadrightarrow} (M', \emptyset)$. □

It easily follows from lemma A.9 and definition of γ-development that for any M there is no N, F s.t. $(M, \emptyset) \underset{\gamma}{\rightarrow} (N, F)$. Therefore a complete γ-development can not be extended further as a γ-development .

Sometimes (see, for instance, properties A.25 and A.26) we need to reason about complete γ-developments of subsets of marked redexes, i.e. γ-developments that reduce all residuals of some of marked redexes, but not of all such redexes. The following definition defines such γ-developments . Note that when $F_1 = F$, the definition is equivalent to definition A.13.

Definition A.14. A sequence $(M, F) \underset{\gamma}{\overset{S}{\rightarrow}}^* (M', F')$ is a *complete γ-development* of a set $F_1 \subseteq F$ if:

1. For all $(\mathbb{C}, R)$ s.t. $S = S_1; (\mathbb{C}, R); S_2$ there exists $(\mathbb{C}_1, R_1) \in F_1$ s.t. $(\mathbb{C}, R) \in (\mathbb{C}_1, R_1)/S_1$, i.e. all redexes reduced in the sequences are residuals of redexes in F_1,
2. There is no $(\mathbb{C}, R) \in F_1$ and $(\mathbb{C}', R') \in F'$ s.t. $(\mathbb{C}', R') \in (\mathbb{C}, R)/S$, i.e. there are no residuals of F_1 in F'.

□

Note that if $F_1 = \emptyset$, then part 1 of the definition implies that $S = \epsilon$.

Property A.15 (Composition). *If* $(M_1, F_1) \underset{\gamma}{\overset{S_1}{\rightarrow}}^* (M_2, F_2) \underset{\gamma}{\overset{S_2}{\rightarrow}}^* (M_3, F_3)$, *then* $(M_1, F_1) \underset{\gamma}{\overset{S_1;S_2}{\rightarrow}}^* (M_3, F_3)$. □

Lemma A.16. *γ-developments have the composition property A.15.* □

Proof. By condition 4 of the axiomatic definition A.11. □

Lift and project on pairs (M, F)**.** The version of lift and project that we are going to prove in this section is formulated for the extended reduction on pairs of the form (M, F) (see below). The proprefTwopro:liftpro:project given in section 3 are obtained from the properties below by erasing the second component of pairs (M, F), i.e. by "unmarking" all the marked redexes. See lemma A.20 for details.

The properties below are illustrated on figure 6.

Property A.17 (Lift). *If* $(M_1, \{(\mathbb{C}, R)\}) \underset{\gamma}{\overset{(\mathbb{C},R)}{\hookrightarrow}} (M_2, \emptyset) \Longrightarrow^* (N_2, \emptyset)$ *, then there exists* (N_1, F_1) *s.t.* $(M_1, \{(\mathbb{C}, R)\}) \underset{\cup}{\Rightarrow}^* (N_1, F_1) \underset{\gamma}{\hookrightarrow}^* (N_2, \emptyset)$. □

Property A.18 (Project). *If* $(M_1, \{(\mathbb{C}, R)\}) \underset{\gamma}{\overset{(\mathbb{C},R)}{\hookrightarrow}} (M_2, \emptyset)$ *and* $(M_1, \{(\mathbb{C}, R)\}) \Longrightarrow^* (N_1, F_1)$, *then there exist* $(N_2, F_2), N_3$ *s.t.* $(N_1, F_1) \underset{\gamma}{\Rightarrow}^* (N_2, F_2) \underset{\gamma}{\hookrightarrow}^* (N_3, \emptyset)$ *and* $(M_2, \emptyset) \Longrightarrow^* (N_3, \emptyset)$. □

Lift and project for pairs indeed imply the respective properties for terms without marked redexes due to the following lemma:

$$
\begin{tikzcd}
(M_1, \{(\mathbb{C}, R)\}) \arrow[r, Rightarrow, two heads, dashed, "\cup"'] \arrow[d, "(\mathbb{C},R)"', "\gamma"] & (N_1, F_1) \arrow[d, dashed, "\gamma"] \\
(M_2, \emptyset) \arrow[r, Rightarrow, two heads] & (N_2, \emptyset)
\end{tikzcd}
\qquad
\begin{tikzcd}
(M_1, \{(\mathbb{C}, R)\}) \arrow[r, Rightarrow, two heads] \arrow[d, "(\mathbb{C},R)"', "\gamma"] & (N_1, F_1) \arrow[r, Rightarrow, two heads, dashed, "\gamma"'] & (N_2, F_2) \arrow[d, dashed, "\gamma"] \\
(M_2, \emptyset) \arrow[rr, Rightarrow, two heads, dashed] & & (N_3, \emptyset)
\end{tikzcd}
$$

lift A.17 project A.18

Figure 6: Lift and project for the extended reduction

Lemma A.19. *If* $(M_1, F_1) \underset{\gamma}{\Rightarrow}^* (M_2, F_2)$ *(respectively* $(M_1, F_1) \underset{\gamma}{\circ\!\!\longrightarrow}^* (M_2, F_2)$*), then* $M_1 \Longrightarrow^* M_2$ *(respectively* $M_1 \circ\!\!\longrightarrow^* M_2$*).* □

Proof. Each of the $(M_1, F_1) \underset{\gamma}{\Rightarrow} (M_2, F_2)$ is either s.t. $M_1 = M_2$ or it is a step $(M_1, F_1) \xRightarrow[\gamma]{(\mathbb{E},R)} (M_2, F_2)$, where $(\mathbb{E}, R)$ is an evaluation redex, and by definition A.11 $M_1 \xRightarrow{(\mathbb{E},R)} M_2$. Therefore $(M_1, F_1) \underset{\gamma}{\Rightarrow}^* (M_2, F_2)$ implies $M_1 \Longrightarrow^* M_2$. Similarly for $\underset{\gamma}{\circ\!\!\longrightarrow}^*$. □

Lemma A.20. *Lift property for pairs (property A.17) implies lift property for terms (property 3.37). Similarly project property for pairs (property A.18) implies project property for terms (property 3.38).* □

Proof. We show the claim of the lemma for lift property, the proof for project is completely analogous. Suppose $M_1 \circ\!\!\longrightarrow M_2 \Longrightarrow^* N_2$. Then there exists a redex $(\mathbb{C}, R)$ s.t. $M_1 \overset{(\mathbb{C},R)}{\circ\!\!\longrightarrow} M_2$. Then $(M_1, \{(\mathbb{C}, R)\}) \in dom(\gamma)$ by definition A.11, and by condition 5 of the definition $(M_1, \{(\mathbb{C}, R)\}) \underset{\gamma}{\overset{(\mathbb{C},R)}{\circ\!\!\longrightarrow}} (M_2, \emptyset)$. By definition of extended reduction $(M_2, \emptyset) \Longrightarrow^* (N_2, \emptyset)$. Then by lift property for pairs there exists (N_1, F_1) s.t. $(M_1, \{(\mathbb{C}, R)\}) \underset{\cup}{\Rightarrow}^* (N_1, F_1) \underset{\gamma}{\circ\!\!\longrightarrow}^* (N_2, \emptyset)$, and by lemma A.19 $M_1 \Longrightarrow^* N_1 \circ\!\!\longrightarrow^* N_2$. □

Properties sufficient to prove lift and project. Showing the following properties of γ-developments and residuals allows us to prove lift and project.

Confluence of $\Longrightarrow$ *and related properties.* The proof of computational soundness given in section 3 requires $\Longrightarrow$ to be confluent (see definition 2.6): if $M_1 \Longrightarrow^* M_2$ and $M_1 \Longrightarrow^* M_3$, then there exists M_4 s.t. $M_2 \Longrightarrow^* M_4$ and $M_3 \Longrightarrow^* M_4$. Note that we do not formulate this property for pairs (M, F), since such pairs are only a tool in the proof of lift and project (recall that the goal of this section is to show lift and project as formulated in 3.37 and 3.38), and confluence is applied in the proof of computational soundness which assumes that lift and project have been already shown.

However, the proof of project given in this section requires a property of $\Longrightarrow$ which is similar to confluence. We give two versions of this property, one implied by the other, because the stronger version is simpler, but holds only in some of the calculi we consider. In the cases when it does not hold (for instance, see "erasing" γ-development steps in the calculus $\mathcal{C}$, defined in C.9) we are able to show the weaker version. Note that since these properties are used in the proof of project based on reductions on pairs (M, F), they are formulated for such pairs. We call these properties γ-confluence of $\Longrightarrow$. Note that there may be the case when one or both of the resulting reductions are 0-step. This happens when the two given reduction steps reduce the same redex. It is possible (for instance, in $\mathcal{C}$) that the resulting γ-development sequence is non-empty, whereas the resulting evaluation step is empty. See the proof of lemma C.28 for an example of such situation.

Notation A.21. For a one-step relation $\longrightarrow$, let $\xrightarrow{0/1}$ denote its reflexive closure. □

$$
\begin{tikzcd}
(M_1,F_1) \arrow[r, Rightarrow] \arrow[d, Rightarrow, "\gamma"] & (M_3,F_3) \arrow[d, Rightarrow, dashed, "0/1"', "\gamma"] \\
(M_2,F_2) \arrow[r, Rightarrow, dashed, "0/1"] & (M_4,F_4)
\end{tikzcd}
\qquad
\begin{tikzcd}
(M_1,F_1) \arrow[r, Rightarrow] \arrow[d, Rightarrow, "\gamma"] & (M_3,F_3) \arrow[d, Rightarrow, twoheadrightarrow, "\gamma"] \\
(M_2,F_2) \arrow[r, Rightarrow, dashed, "0/1"] & (M_4,F_4)
\end{tikzcd}
$$

A.22 A.23

Figure 7: Properties of evaluation

Property A.22 (Strong γ-confluence of $\Rightarrow$.). *If $(M_1, F_1) \underset{\gamma}{\Rightarrow} (M_2, F_2)$ and $(M_1, F_1) \Rightarrow (M_3, F_3)$, then there exists (M_4, F_4) s.t. $(M_2, F_2) \overset{0/1}{\Longrightarrow} (M_4, F_4)$ and $(M_3, F_3) \underset{\gamma}{\overset{0/1}{\Longrightarrow}} (M_4, F_4)$.* □

Property A.23 (Weak γ-confluence of $\Rightarrow$). *If $(M_1, F_1) \underset{\gamma}{\Rightarrow} (M_2, F_2)$ and $(M_1, F_1) \Rightarrow (M_3, F_3)$, then there exists (M_4, F_4) s.t. $(M_2, F_2) \overset{0/1}{\Longrightarrow} (M_4, F_4)$ and $(M_3, F_3) \underset{\gamma}{\Rightarrow}^* (M_4, F_4)$.* □

It is clear that property A.22 implies A.23. Note that both properties are independent from confluence of $\Rightarrow$: even though $(M, F) \underset{\gamma}{\Rightarrow} (M', F')$ implies $M \Rightarrow M'$, the converse is not true, i.e. it is possible that for some F there is no F' s.t. $(M, F) \underset{\gamma}{\Rightarrow} (M', F')$ (for instance, if $(M, F) \notin dom(\gamma)$). However, in the case when a γ-development step is in fact an extended reduction step (definition A.10), both confluence of $\Rightarrow$ and strong γ-confluence of $\Rightarrow$ are implied by the following property:

Property A.24. *If $M_1 \overset{(\mathbb{E},R)}{\Longrightarrow} M_2$ and $M_1 \overset{(\mathbb{E}',R')}{\Longrightarrow} M_3$, then there exists M_4 s.t. $M_2 \overset{(\mathbb{E}',R')/(\mathbb{E},R)}{\Longrightarrow} M_4$, $M_3 \overset{(\mathbb{E},R)/(\mathbb{E}',R')}{\Longrightarrow} M_4$, provided $(\mathbb{E}, R) \neq (\mathbb{E}', R')$.* □

Elementary lift and project diagrams. The next group of properties deals with interactions between evaluation and non-evaluation redexes.

$$
\begin{tikzcd}
(M_1,F_1) \arrow[r, Rightarrow, "{(\mathbb{E},R')}"] \arrow[d, "{(\mathbb{C},R)}"', "\gamma"] & (M_3,F_3) \arrow[d, dashed, "{c-\gamma \ \text{ of } (\mathbb{C},R)/(\mathbb{E},R')}"] \\
(M_2,F_2) \arrow[r, Rightarrow, dashed] & (M_4,F_4)
\end{tikzcd}
\qquad
\begin{tikzcd}
(M_1,F_1) \arrow[r, Rightarrow, dashed, "{(\mathbb{E},R')}"] \arrow[d, "{(\mathbb{C},R)}"', "\gamma"] & (M_3,F_3) \arrow[d, dashed, "{c-\gamma \ \text{ of } (\mathbb{C},R)/(\mathbb{E},R')}"] \\
(M_2,F_2) \arrow[r, Rightarrow] & (M_3,F_3)
\end{tikzcd}
$$

Elem. project
diagram A.25

Elem. lift
diagram A.26

Figure 8: Elementary project and lift diagrams

Property A.25 (Elementary project diagram). *If $(M_1, F_1) \underset{\gamma}{\overset{(\mathbb{C},R)}{\circ\!\!\longrightarrow}} (M_2, F_2)$, and $(M_1, F_1) \overset{(\mathbb{E},R')}{\Longrightarrow} (M_3, F_3)$, then there exists (M_4, F_4) s.t. $(M_2, F_2) \Rightarrow (M_4, F_4)$, and $(M_3, F_3) \underset{\gamma}{\rightarrow}^* (M_4, F_4)$, where the latter sequence is a complete γ-development of $(\mathbb{C}, R)/(\mathbb{E}, R')$ (see figure 8).* □

Property A.26 (Elementary lift diagram). *If* $(M_1, F_1) \overset{(\mathbb{C},R)}{\underset{\gamma}{\circ\!\!\longrightarrow}} (M_2, F_2) \Longrightarrow (M_4, F_4)$, *then there exists* (M_3, F_3) *s.t.* $(M_1, F_1) \overset{(\mathbb{E},R')}{\Longrightarrow} (M_3, F_3) \underset{\gamma}{\longrightarrow^*} (M_4, F_4)$, *where the latter* γ*-development is a complete* γ*-development of* $(\mathbb{C}, R)/(\mathbb{E}, R')$ *(see figure 8).* ☐

Lemma A.27. *Properties A.25 and A.15 imply the following: if* $(M_1, F_1) \overset{S}{\underset{\gamma}{\circ\!\!\longrightarrow}}{}^* (M_2, F_2)$ *and* $(M_1, F_1) \overset{(\mathbb{E},R)}{\Longrightarrow} (M_3, F_3)$, *then there exists* (M_4, F_4) *s.t.* $(M_2, F_2) \Longrightarrow (M_4, F_4)$, *and* $(M_3, F_3) \underset{\gamma}{\longrightarrow^*} (M_4, F_4)$. ☐

Proof. The proof is by induction on n, where n is the number of steps in the non-evaluation sequence S. The case when $n = 0$ is trivial. The *base case* $n = 1$ is straightforward by property A.25.

$$\begin{tikzcd} (M_1,F_1) \arrow[r, Rightarrow, "(\mathbb{E},R)"] \arrow[d, "S"', "\gamma"] \arrow[dr, phantom, "\mathrm{IH}"] & (M_3,F_3) \arrow[d, dashed, "\gamma"] \\ (M_2',F_2') \arrow[r, Rightarrow, dashed, "(\mathbb{E}',R')"] \arrow[d, "(\mathbb{C},\tilde{R})"', "\gamma"] \arrow[dr, phantom, "A.25"] & (M_4',F_4') \arrow[d, dashed, "\gamma"] \\ (M_2,F_2) \arrow[r, Rightarrow, dashed] & (M_4,F_4) \end{tikzcd} \quad \underset{A.15}{\longrightarrow} \quad \begin{tikzcd} (M_1,F_1) \arrow[r, Rightarrow, "(\mathbb{E},R)"] \arrow[d, "S;(\mathbb{C},\tilde{R})"', "\gamma"] & (M_3,F_3) \arrow[d, dashed, "\gamma"] \\ (M_2,F_2) \arrow[r, Rightarrow, dashed] & (M_4,F_4) \end{tikzcd}$$

Figure 9: Inductive step of proof of lemma A.27

Induction step. The induction step is illustrated on figure 9. ☐

Lemma A.28. *Properties A.26 and A.15 imply the following: if* $(M_1, F_1) \overset{S}{\underset{\gamma}{\circ\!\!\longrightarrow}}{}^* (M_2, F_2)$ *and* $(M_2, F_2) \overset{(\mathbb{E},R)}{\Longrightarrow} (M_4, F_4)$, *then there exists* (M_3, F_3) *s.t.* $(M_1, F_1) \overset{(\mathbb{E}_1,R_1)}{\Longrightarrow} (M_3, F_3)$, *and* $(M_3, F_3) \underset{\gamma}{\longrightarrow^*} (M_4, F_4)$, *where* $(\mathbb{E}, R) = (\mathbb{E}_1, R_1)/S$. ☐

Proof. Similarly to the proof of lemma A.27, the proof is by induction on n - the number of steps in S. The base case is by property A.26. The induction step is illustrated on figure 10. ☐

Standardization of Developments. Standardization of (complete) γ-developments is a crucial property in proving lift and project, and therefore computational soundness. We give two versions of this property, where version 1 is stronger than version 2: version 1 does not require the γ-development to be complete, and does not put any restriction on the initial pair (M, F), whereas version 2 requires this pair to be obtained from a pair $(N, \{(\mathbb{C}, R)\})$, where $(\mathbb{C}, R)$ is a non-evaluation redex, by an evaluation sequence. Moreover, we also require that the resulting pair $(M', \emptyset)$ of the development does not have any marked redexes (i.e. the development is complete), and we also require the evaluation sequences leading to (M, F) and to $(M', \emptyset)$ to be related by elementary diagrams (see definition A.30 below). The reason for such requirements is that

$$
\begin{tikzcd}
(M_1,F_1) \arrow[r, Rightarrow, dashed, "(\mathbb{E}_1,R_1)"] \arrow[d, "(\mathbb{C},\tilde{R})"', "\gamma"] \arrow[dr, phantom, "A.26"] & (M_3,F_3) \arrow[d, dashed, "\gamma"] \\
{} \arrow[r, Rightarrow, dashed, "(\mathbb{E}',R')"] \arrow[d, "S"', "\gamma"] \arrow[dr, phantom, "\mathrm{IH}"] & (M_4',F_4') \arrow[d, dashed, "\gamma"] \\
(M_2,F_2) \arrow[r, Rightarrow, "(\mathbb{E},R)"] & (M_4,F_4)
\end{tikzcd}
\quad \xrightarrow[A.15]{} \quad
\begin{tikzcd}
(M_1,F_1) \arrow[r, Rightarrow, dashed, "(\mathbb{E}_1,R_1)"] \arrow[d, "(\mathbb{C},\tilde{R});S"', "\gamma"] & (M_3,F_3) \arrow[d, dashed, "\gamma"] \\
(M_2,F_2) \arrow[r, Rightarrow, "(\mathbb{E},R)"] & (M_4,F_4)
\end{tikzcd}
$$

Figure 10: Inductive step of proof of lemma A.28

core module calculus has complete developments that are not standartizible. The restriction on the starting and ending modules rules out such complete developments.

Property A.29 (Standardization of γ-developments (stronger version)). *If $(M,F) \underset{\gamma}{\longrightarrow}^* (M',F')$, then there exists (M'',F'') s.t. $(M,F) \underset{\gamma}{\Longrightarrow}^* (M'',F'') \underset{\gamma}{\circ\!\!\longrightarrow}^* (M',F')$.* $\square$

Definition A.30. Let $(N,\{(\mathbb{C},R)\}) \underset{\gamma}{\overset{(\mathbb{C},R)}{\circ\!\!\longrightarrow}} (N',\emptyset)$. We say that the evaluation sequences $(N,\{(\mathbb{C},R)\}) \Longrightarrow^* (M,F)$ and $(N',\emptyset) \Longrightarrow^* (M',\emptyset)$ are *related by elementary diagrams* if for every step $(N_i,F_i) \Longrightarrow (N_{i+1},F_{i+1})$ s.t. $(N,\{(\mathbb{C},R)\}) \Longrightarrow^* (N_i,F_i) \Longrightarrow (N_{i+1},F_{i+1}) \Longrightarrow^* (M,F)$ and

- there exists $(N_i',\emptyset)$ s.t. $(N',\emptyset) \Longrightarrow^* (N_i',\emptyset) \Longrightarrow^* (M',\emptyset)$, $(N_i,F_i) \underset{c-\gamma}{\twoheadrightarrow} (N_i',\emptyset)$,
- there exist $(\tilde{N}_i,\tilde{F}_i)$ and $(\tilde{N}_{i+1},\tilde{F}_{i+1})$ s.t. $(N_i,F_i) \underset{\gamma}{\Longrightarrow}^* (\tilde{N}_i,\tilde{F}_i) \overset{0/1}{\Longrightarrow} (\tilde{N}_{i+1},\tilde{F}_{i+1})$ and $(N_{i+1},F_{i+1}) \underset{\gamma}{\Longrightarrow}^* (\tilde{N}_{i+1},\tilde{F}_{i+1})$, where the sequences are constructed of the diagrams as in properties A.22 and A.23 (see figure 7), and one of the following takes place:
  - the step $(\tilde{N}_i,\tilde{F}_i) \overset{0/1}{\Longrightarrow} (\tilde{N}_{i+1},\tilde{F}_{i+1})$ is a 0-step,
  - the step $(\tilde{N}_i,\tilde{F}_i) \overset{0/1}{\Longrightarrow} (\tilde{N}_{i+1},\tilde{F}_{i+1})$ is a non-zero step, and there exist sequences $(\tilde{N}_i,\tilde{F}_i) \underset{\gamma}{\circ\!\!\longrightarrow}^* (N_i',\emptyset)$, $(\tilde{N}_{i+1},\tilde{F}_{i+1}) \underset{\gamma}{\longrightarrow}^* (N_{i+1}',\emptyset)$, and $(N_i',\emptyset) \Longrightarrow (N_{i+1}',\emptyset)$, which, together with the step $(\tilde{N}_i,\tilde{F}_i) \Longrightarrow (\tilde{N}_{i+1},\tilde{F}_{i+1})$ are constructed from the elementary diagrams A.25 as in lemma A.27 (see figure 9).

$\square$

Property A.31 (Standardization of γ-developments (weaker version)). *Suppose $(N,\{(\mathbb{C},R)\}) \Longrightarrow^* (M,F) \underset{\gamma}{\longrightarrow}^* (M',\emptyset)$ and $(N,\{(\mathbb{C},R)\}) \underset{\gamma}{\overset{(\mathbb{C},R)}{\circ\!\!\longrightarrow}} (N',\emptyset) \Longrightarrow^* (M',\emptyset)$, where the sequences $(N,\{(\mathbb{C},R)\}) \Longrightarrow^*$*

(M, F) and $(N', \emptyset) \Rightarrow^* (M', \emptyset)$ are related by elementary diagrams, then there exists (M'', F'') s.t. $(M, F) \underset{\gamma}{\Rightarrow}^* (M'', F'') \underset{\gamma}{\circ\!\!\longrightarrow}^* (M', \emptyset)$. $\square$

Theorem A.32. *Suppose a calculus has the following properties:*

- *Property A.26,*
- *Standardization of γ-developments (property A.29 or a weaker property A.31),*
- *Composition of γ-developments (property A.15).*

Then the calculus has lift property A.17. $\square$

Proof. The proof is by induction on the number of steps in the sequence $(M_2, \emptyset) \Rightarrow^* (N_2, \emptyset)$. Let n denote this number. The proof is illustrated on figure 11.

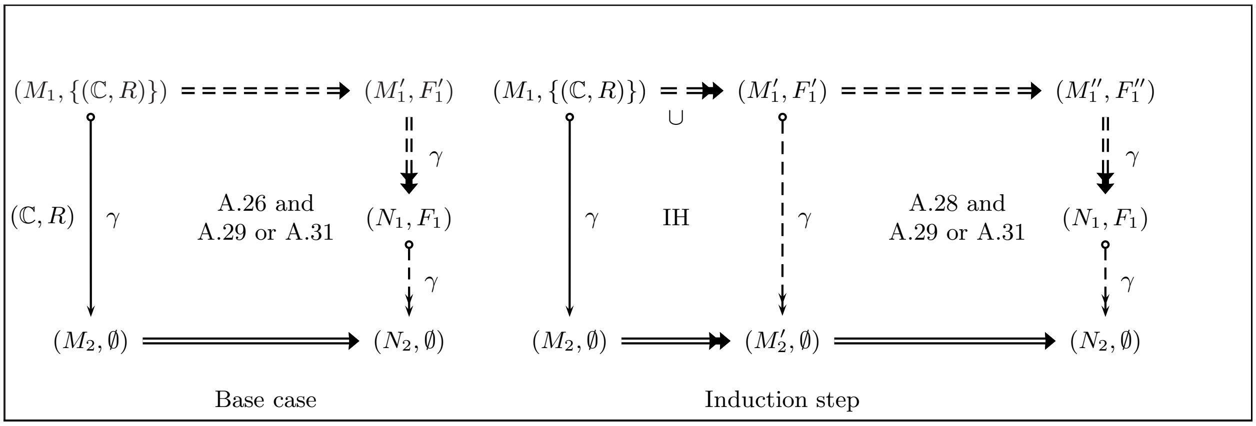

Figure 11: Proof of lift property

Base case ($n = 1$). Let $(M_1, \{(\mathbb{C}, R)\}) \underset{\gamma}{\circ\!\!\longrightarrow} (M_2, \emptyset) \Rightarrow (N_2, \emptyset)$. By property A.26 there exists (M_1', F_1') s.t. $(M_1, \{(\mathbb{C}, R)\}) \Rightarrow (M_1', F_1') \underset{\gamma}{\longrightarrow}^* (N_2, \emptyset)$. By property A.29 or A.31 there exists (N_1, F_1) s.t. $(M_1', F_1') \underset{\gamma}{\Rightarrow}^* (N_1, F_1) \underset{\gamma}{\circ\!\!\longrightarrow}^* (N_2, \emptyset)$. Therefore $(M_1, \{(\mathbb{C}, R)\}) \underset{\cup}{\Rightarrow}^* (N_1, F_1) \underset{\gamma}{\circ\!\!\longrightarrow}^* (N_2, \emptyset)$.

Induction step. Inductive hypothesis: let $(M_1, \{(\mathbb{C}, R)\}) \underset{\gamma}{\circ\!\!\longrightarrow} (M_2, \emptyset)$, and let $(M_2, \emptyset) \Rightarrow^* (M_2', \emptyset)$ in $n-1$ steps. Then we suppose that there exists (M_1', F_1') s.t. $(M_1, \{(\mathbb{C}, R)\}) \underset{\cup}{\Rightarrow}^* (M_1', F_1') \underset{\gamma}{\circ\!\!\longrightarrow}^* (M_2', \emptyset)$.

Now let $(M_2', \emptyset) \Rightarrow (N_2, \emptyset)$. By lemma A.28 properties A.26 and A.15 imply that there exists (M_1'', F_1'') s.t. $(M_1', F_1') \Rightarrow (M_1'', F_1'') \underset{\gamma}{\longrightarrow}^* (N_2, \emptyset)$. Then by standardization of γ-developments (property A.29 or A.31) there exists (N_1, F_1) s.t. $(M_1'', F_1'') \underset{\gamma}{\Rightarrow}^* (N_1, F_1) \underset{\gamma}{\circ\!\!\longrightarrow}^* (N_2, \emptyset)$. Therefore $(M_1, \{(\mathbb{C}, R)\}) \underset{\cup}{\Rightarrow}^* (N_1, F_1)$ (since $(M_1, \{(\mathbb{C}, R)\}) \underset{\cup}{\Rightarrow}^* (M_1', F_1') \Rightarrow (M_1'', F_1'') \underset{\gamma}{\Rightarrow}^* (N_1, F_1)$) and $(N_1, F_1) \underset{\gamma}{\circ\!\!\longrightarrow}^* (N_2, \emptyset)$. $\square$

Theorem A.33. *Suppose a calculus has the following properties:*

- *Property A.25,*
- *Standardization of γ-developments (property A.29 or a weaker property A.31),*
- *γ-diamond property of $\Rightarrow$ (property A.22) or its weaker analog A.23,*
- *Composition of γ-developments (property A.15),*

Then the calculus has project property A.18. □

Proof. The proof is by induction on the number of steps in the given evaluation sequence $(M_1, \{(\mathbb{C}, R)\}) \Longrightarrow^* (N_1, F_1)$, which we denote by n. See figure 12 for the illustration of the proof.

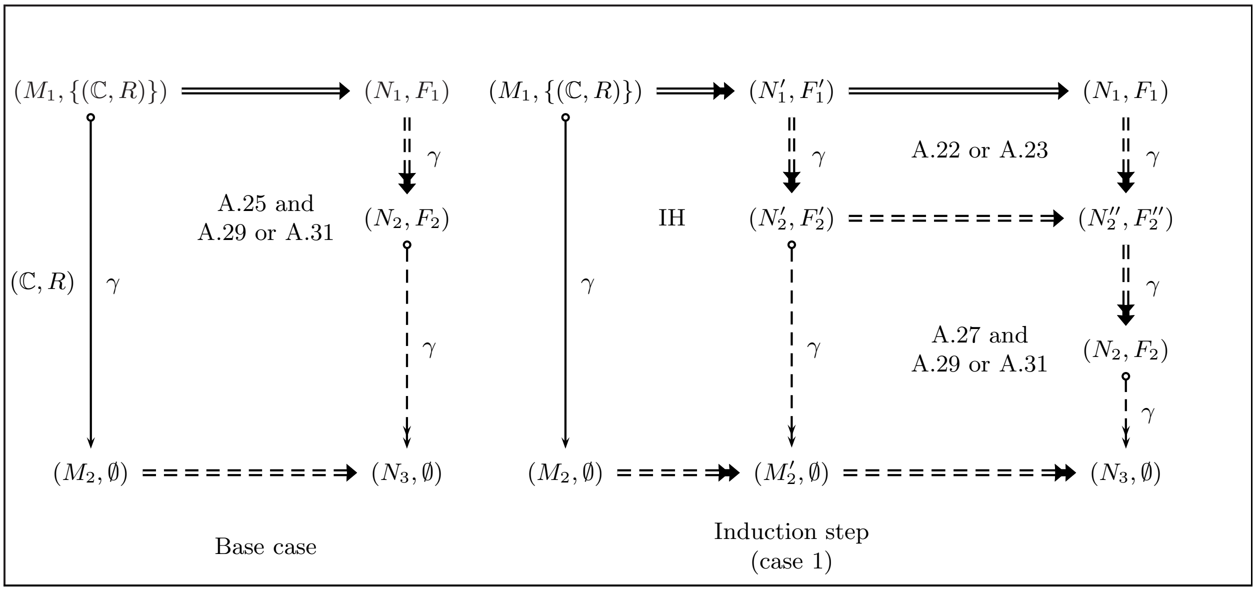

Figure 12: Proof of project property

Base case ($n = 1$). Let $(M_1, \{(\mathbb{C}, R)\}) \xrightarrow[\gamma]{(\mathbb{C},R)} (M_2, \emptyset)$, $(M_1, \{(\mathbb{C}, R)\}) \Longrightarrow (N_1, F_1)$. Then by property A.25 there exists (N_3, F_3) s.t. $(N_1, F_1) \underset{\gamma}{\longrightarrow}^* (N_3, F_3)$ and $(M_2, \emptyset) \Longrightarrow (N_3, F_3)$. The latter reduction implies that $F_3 = \emptyset$ (since $\emptyset/(\mathbb{A}, R) = \emptyset$ for any $(\mathbb{A}, R)$). Then by property A.29 or A.31 there exists (N_2, F_2) s.t. $(N_1, F_1) \underset{\gamma}{\Rrightarrow}^* (N_2, F_2) \underset{\gamma}{\circ\!\!\longrightarrow}^* (N_3, \emptyset)$ (note that we are able to use property A.31 because, as we have shown, $F_3 = \emptyset$, so $(N_1, F_1) \underset{\gamma}{\longrightarrow}^* (N_3, \emptyset)$ is a complete development).

Induction step. As the induction hypothesis suppose the following: if $(M_1, \{(\mathbb{C}, R)\}) \Longrightarrow^* (N_1', F_1')$ in $n-1$ steps and $(M_1, \{(\mathbb{C}, R)\}) \xrightarrow[\gamma]{(\mathbb{C},R)} (M_2, \emptyset)$, then there exist (N_2', F_2') and $(M_2', \emptyset)$ s.t. $(N_1', F_1') \underset{\gamma}{\Rrightarrow}^* (N_2', F_2') \underset{\gamma}{\circ\!\!\longrightarrow}^* (M_2', \emptyset)$ and $(M_2, \emptyset) \Longrightarrow^* (M_2', \emptyset)$.

Let $(N_1', F_1') \Longrightarrow (N_1, F_1)$. Each of the properties A.22 and A.23 implies that, given $(N_1', F_1') \underset{\gamma}{\Rrightarrow}^* (N_2', F_2')$ and $(N_1', F_1') \Longrightarrow (N_1, F_1)$, there exists (N_2'', F_2'') s.t. $(N_1, F_1) \underset{\gamma}{\Rrightarrow}^* (N_2'', F_2'')$ and $(N_2', F_2') \overset{0/1}{\Longrightarrow} (N_2'', F_2'')$.

Case 1. Suppose $(N_2', F_2') \Longrightarrow (N_2'', F_2'')$. Then the existence of the sequence $(N_2', F_2') \underset{\gamma}{\circ\!\!\longrightarrow}^* (M_2', \emptyset)$ implies that there exists N_3 s.t. $(M_2', \emptyset) \Longrightarrow (N_3, \emptyset)$, $(N_2'', F_2'') \underset{\gamma}{\longrightarrow}^* (N_3, \emptyset)$ (from properties A.25 and A.15 by lemma A.27). As in the base case, the set of redexes marked in N_3 is empty, since the set of marked redexes in M_2' is empty. Then by A.29 or A.31 there exists (N_2, F_2) s.t. $(N_2'', F_2'') \underset{\gamma}{\Rrightarrow}^* (N_2, F_2) \underset{\gamma}{\circ\!\!\longrightarrow}^* (N_3, \emptyset)$. We have shown that there exist (N_2, F_2) and N_3 s.t. $(N_1, F_1) \underset{\gamma}{\Rrightarrow}^* (N_2, F_2) \underset{\gamma}{\circ\!\!\longrightarrow}^* (N_3, \emptyset)$.

Case 2. Suppose $(N_2', F_2') = (N_2'', F_2'')$. By the inductive hypothesis $(N_2'', F_2'') \underset{\gamma}{\circ\!\!\longrightarrow}^* (M_2', \emptyset)$, and the claim of the theorem holds. □

We extend the definition of standardization property to reductions on pairs of a term and a set of its marked redexes, similarly to the extensions for lift and project (properties A.17 and A.18):

Property A.34 (Standardization). *If $(M, F) \rightarrow^* (M', F')$, then there exists (M'', F'') s.t. $(M, F) \Longrightarrow^* (M'', F'') \circ\!\!\longrightarrow^* (M', F')$.* □

As shown in section 3.6 (lemma 3.39), the lift property implies standardization. The proof of lemma 3.39 works for reductions on pairs of the form (M, F) as well.

A.2 Properties of γ-developments in more detail.

The previous section presents proofs of lift and project via certain properties of γ-developments , s.a. standardization of γ-developments (A.29 or A.31). Continuing moving from more general properties to more specific ones, we discuss the ways of proving the properties defined in the previous section.

Property A.35. *For any (M, F) there exists a complete γ-development of n steps s.t. any other γ-development of (M, F) has no more than n steps.* □

Notation A.36. Let $\mathrm{MAX}[(M, F)]$ denote the number of reduction steps in a γ-development of (M, F) of the maximal length (see property A.35 above). □

Property A.37 (Finiteness of γ-developments). *There is no infinite reduction sequence $(M_1, F_1) \underset{\gamma}{\rightarrow} (M_2, F_2) \underset{\gamma}{\rightarrow} \dots$.* □

Property A.37 trivially follows from property A.35.

Property A.38. *If $(M_1, F_1) \underset{\gamma}{\circ\!\!\longrightarrow} (M_2, F_2) \underset{\gamma}{\Longrightarrow} (M_3, F_3)$, then there exists (M', F') s.t. $(M_1, F_1) \underset{\gamma}{\Longrightarrow} (M', F') \underset{\gamma}{\rightarrow}^* (M_3, F_3)$.* □

This property seems similar to property A.26 (see also figure 8). However, the two properties are in fact independent (i.e. one neither implies, nor is implied by the other), because A.38 restricts the given evaluation step to be a γ-development step, and requires the resulting evaluation step to be a γ-development step as well. The calculus $\mathcal{C}$ is an example of a calculus that has property A.26 but not A.38.

Lemma A.39. *If a calculus has properties A.35 and A.38, then it has strong standardization of γ-developments (property A.29 i.e. for any sequence $(M, F) \underset{\gamma}{\rightarrow}^* (M', F')$ there exists (M'', F'') s.t. $(M, F) \underset{\gamma}{\Longrightarrow}^* (M'', F'') \underset{\gamma}{\circ\!\!\longrightarrow}^* (M', F')$.* □

Proof. Let $(M, F) \underset{\gamma}{\rightarrow}^* (M', F')$ be a γ-development sequence. The following construction repeatedly applied to this sequence will in a finite number of applications produce a standard sequence $(M, F) \underset{\gamma}{\Longrightarrow}^* (M'', F'') \underset{\gamma}{\circ\!\!\longrightarrow}^* (M', F')$:

Construction of a standard γ-development .

- *Step 1.* Check if the given sequence $(M, F) \underset{\gamma}{\rightarrow}^* (M', F')$ is standard. If yes, then the construction is finished. If no, go to step 2.
- *Step 2.* Since the given sequence is not standard, it can be parsed as follows:

$$(M, F) \underset{\gamma}{\Longrightarrow}^* (M_0, F_0) \underset{\gamma}{\circ\!\!\longrightarrow}^* (M_1, F_1) \underset{\gamma}{\circ\!\!\longrightarrow} (M_2, F_2) \underset{\gamma}{\Longrightarrow} (M_3, F_3) \underset{\gamma}{\rightarrow}^* (M', F').$$

Then by property A.38 there exists (M_2', F_2') s.t. $(M, F) \underset{\gamma}{\Rightarrow} (M_2', F_2') \underset{\gamma}{\rightarrow}^* (M_3, F_3)$. Then we replace the initial reduction sequence by the sequence

$$(M, F) \underset{\gamma}{\Rightarrow}^* (M_0, F_0) \underset{\gamma}{\hookrightarrow}^* (M_1, F_1) \underset{\gamma}{\Rightarrow} (M_2', F_2') \underset{\gamma}{\rightarrow}^* (M_3, F_3) \underset{\gamma}{\rightarrow}^* (M', F').$$

and repeat step 1.

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

$$(M, F) \overset{S_1}{\underset{\gamma}{\Longrightarrow}}{}^* (M_0, F_0) \overset{S_2}{\underset{\gamma}{\hookrightarrow}}{}^* (M_1, F_1) \underset{\gamma}{\hookrightarrow} (M_2, F_2) \underset{\gamma}{\Rightarrow} (M_3, F_3) \underset{\gamma}{\rightarrow}^* (M', F').$$

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

$$(M, F) \overset{S_1}{\underset{\gamma}{\Longrightarrow}}{}^* (M_0, F_0) \overset{S_2}{\underset{\gamma}{\hookrightarrow}}{}^* (M', F'),$$

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

We consider pairs (n_1, n_2) to be ordered lexicographically, i.e. $(n_1, n_2) < (n_1', n_2')$ if $n_1 < n_1'$ or $n_1 = n_1'$ and $n_2 < n_2'$. It is easy to observe that if (n_1, n_2) corresponds to a sequence S before an iteration of the construction, and (n_1', n_2') corresponds to S' which is the result of the iteration, then $(n_1, n_2) < (n_1', n_2')$: either the subsequence $(M_0, F_0) \overset{S_2}{\underset{\gamma}{\hookrightarrow}}{}^* (M_1, F_1)$ is non-empty, in which case n_1 does not change, and n_2 increases by 1, or the subsequence is empty, and then the sequence $(M, F) \overset{S_1}{\underset{\gamma}{\Longrightarrow}}{}^* (M_0, F_0)$ becomes followed by at least one more evaluation step, so n_1 increases.

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

Special case of γ-developments : traditional developments. A special case of a γ-development step is a development step defined in definition A.10: a development step $(M, F) \overset{(\mathbb{C}, R)}{\underset{dev}{\rightarrow}} (M', F')$ is an extended reduction step $(M, F) \overset{(\mathbb{C}, R)}{\underset{dev}{\rightarrow}} (M', F')$, where $(\mathbb{C}, R) \in F$. In this case we may simplify some of the proofs. In particular, the following corollary A.42 allows us to prove properties A.25 and A.26 only for the case when the initial pair (M_1, F_1) (see figure 8) has only one marked redex in F_1, the general case of an arbitrary F_1 follows automatically. Note that we use notation $\underset{dev}{\rightarrow}$ to distinguish this particular case from the general case of $\underset{\gamma}{\rightarrow}$.

Lemma A.40. *If* $(M, \{(\mathbb{C}_1, R_1)\}) \xrightarrow{(\mathbb{A}, R)} (M', F_1')$ *and* $(M, \{(\mathbb{C}_2, R_2)\}) \xrightarrow{(\mathbb{A}, R)} (M', F_2')$*, then* $(M, \{(\mathbb{C}_1, R_1), (\mathbb{C}_2, R_2)\}) \xrightarrow{(\mathbb{A}, R)} (M', F_1' \cup F_2')$. $\square$

Proof. The lemma follows directly from definition A.7 of the extended reduction (i.e. reduction on pairs (M, F)). $\square$

Note that since $\underset{dev}{\rightarrow}$ is a particular case of the extended reduction $\longrightarrow$, the lemma above is true when one of the two given reductions is a development step. For instance, if $(\mathbb{A}, R) = (\mathbb{C}_1, R_1)$ and $(\mathbb{C}_1, R_1) \neq (\mathbb{C}_2, R_2)$, then the first of the two given reductions is a $\underset{dev}{\rightarrow}$ step, and the reduction $(M, \{(\mathbb{C}_1, R_1), (\mathbb{C}_2, R_2)\}) \xrightarrow{(\mathbb{A},R)}$ $(M', F_1' \cup F_2')$ is also a $\underset{dev}{\rightarrow}$ step.

Lemma A.41. *Let S be an arbitrary sequence of $\rightarrow$ steps. If $(M, \{(\mathbb{C}_1, R_1)\}) \xrightarrow{S}^* (M', F_1')$ and $(M, \{(\mathbb{C}_2, R_2)\}) \xrightarrow{S}^* (M', F_2')$, then $(M, \{(\mathbb{C}_1, R_1), (\mathbb{C}_2, R_2)\}) \xrightarrow{S}^* (M', F_1' \cup F_2')$.* □

Proof. Follows from lemma A.40 by induction on the number of steps in S. □

Finally, we generalize the result to arbitrary sets of "marked" redexes in M.

Corollary A.42. *Let S be an arbitrary sequence of $\longrightarrow$ steps. If $(M, F_1) \xrightarrow{S}^* (M', F_1')$ and $(M, F_2) \xrightarrow{S}^*$ (M', F_2'), then $(M, F_1 \cup F_2) \xrightarrow{S}^* (M', F_1' \cup F_2')$.* □

B Soundness of the Term Calculus.

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

Recall that the general definition of a redex in section A (definition A.2) defines a redex for all the calculi. However, the definition of residuals in section A is axiomatic, i.e. it specifies the properties of a residual, but the details are left to the calculus-specific definition. Before we define residuals in $\mathcal{T}$, let us introduce some important notations.

Definition B.1 (Multi-hole contexts in $\mathcal{T}$). A multi-hole context in $\mathcal{T}$ is defined as follows:

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

$H(\mathbb{C})$ denotes the number of holes in a context $\mathbb{C}$ (defined in the obvious way). $\mathbf{Context}^n_{\mathcal{T}}$ denotes the set $\{\mathbb{C} \mid H(\mathbb{C}) = n\}$. Note that $\mathbf{Context}^0_{\mathcal{T}} = \mathbf{Term}_{\mathcal{T}}$, $\mathbf{Context}^1_{\mathcal{T}} = \mathbf{Context}_{\mathcal{T}}$ as defined in section 2.3.

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

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

Definition B.2 (glb of contexts). The greatest lower bound of two contexts $\mathrm{glb}(\mathbb{C}_1, \mathbb{C}_2)$ is a context defined the following way:

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

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

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

□

We also define free variables of a multi-hole context and substitution in a context:

Definition B.3. The set of free variables of a multi-hole context $\mathbb{C}$, denoted $FV(\mathbb{C})$, is defined as follows:

$$
\begin{array}{rcll}
FV(\mathbb{C}) & = & FV(M) & \text{if } \ \mathbb{C} = M, \\
& & \emptyset & \text{if } \ \mathbb{C} = \Box, \\
& & FV(\mathbb{C}_1) \cap FV(\mathbb{C}_2) & \text{if } \ \mathbb{C} = \mathbb{C}_1 \mathbin{@} \mathbb{C}_2 \quad \text{or} \quad \mathbb{C} = \mathbb{C}_1 \mathbin{op} \mathbb{C}_2, \\
& & FV(\mathbb{C}_1) \setminus \{x\} & \text{if } \ \mathbb{C} = \lambda x.\mathbb{C}_1.
\end{array}
$$

We say that a variable x is *not bound in* $\mathbb{C}$ if either $x \in FV(\mathbb{C})$ or x does not appear in $\mathbb{C}$. □

Definition B.4. If x is not bound in $\mathbb{C}$, then substitution of a term M for x in $\mathbb{C}$, denoted $\mathbb{C}[x := M]$, is defined as follows:

$$
\begin{array}{rcll}
\mathbb{C}[x := M] & = & N[x := M] & \text{if } \ \mathbb{C} = N, \\
& & \Box & \text{if } \ \mathbb{C} = \Box, \\
& & \mathbb{C}_1[x := M] \mathbin{@} \mathbb{C}_2[x := M] & \text{if } \ \mathbb{C} = \mathbb{C}_1 \mathbin{@} \mathbb{C}_2, \\
& & \mathbb{C}_1[x := M] \mathbin{op} \mathbb{C}_2[x := M] & \text{if } \ \mathbb{C} = \mathbb{C}_1 \mathbin{op} \mathbb{C}_2, \\
& & \lambda y.(\mathbb{C}_1[x := M]) & \text{if } \ \mathbb{C} = \lambda y.\mathbb{C}_1.
\end{array}
$$

Note that in the last clause $x \neq y$ and $y \notin FV(M)$ by the distinct variable convention. □

Now we give definition of a set of residuals of a redex in $\mathcal{T}$. It is easy to check that the set of residuals defined below satisfies the properties postulated in the axiomatic definition A.3.

Definition B.5. Let $(\mathbb{C}_1, R_1)$ be a redex in a term M, and suppose $M \xrightarrow{(\mathbb{C}_2,R_2)}_{\mathcal{T}} N$. A *set of residuals* of $(\mathbb{C}_1, R_1)$ w.r.t. the reduction $\xrightarrow{(\mathbb{C}_2,R_2)}_{\mathcal{T}}$ (denoted $(\mathbb{C}_1, R_1)/(\mathbb{C}_2, R_2)$) is defined as follows: let $\mathbb{A} = \text{glb}(\mathbb{C}_1, \mathbb{C}_2)$, and let $R_2 \leadsto Q_2$, then

1. if $\mathbb{C}_1 = \mathbb{C}_2$ (and therefore $R_1 = R_2$), then $(\mathbb{C}_1, R_1)/(\mathbb{C}_2, R_2) = \emptyset$,
2. if $\mathbb{A} \in \mathbf{Context}^2_{\mathcal{T}}$, then $(\mathbb{C}_1, R_1)/(\mathbb{C}_2, R_2) = \{(\mathbb{A}\{\Box, Q_2\}, R_1)\}$ (assuming w.l.o.g. that R_1 fills the first hole in $\mathbb{A}$).
3. if $M = \mathbb{A}\{\lambda x.\mathbb{B}^n\{x, \ldots, x\} \mathbin{@} V\}$, and $R_2 = \lambda x.\mathbb{B}^n\{x, \ldots, x\} \mathbin{@} V$, where $\mathbb{B}^n$ contains all occurrences of x in the operand, and $V = \mathbb{C}\{R_1\}$, then $(\mathbb{C}_1, R_1)/(\mathbb{C}_2, R_2) = \{(\mathbb{A}\{\mathbb{B}^n\{V, \ldots, V, \mathbb{C}_i, V, \ldots V\}\}, R_1) \mid 1 \leq i \leq n\}$, where $\mathbb{C}_i$ is the context $\mathbb{C}$ filling the i-th hole of $\mathbb{B}^n$.
4. if $M = \mathbb{A}\{\lambda x.\mathbb{B}\{R_1\} \mathbin{@} V\}$, where $R_2 = \lambda x.\mathbb{B}\{R_1\} \mathbin{@} V$, then $(\mathbb{C}_1, R_1)/(\mathbb{C}_2, R_2) = \{(\mathbb{A}\{\mathbb{B}\}, R_1[x := V])\}$.
5. if $M = \mathbb{A}\{R_1\}$, where $R_1 = \mathbb{B}\{R_2\}$, then $(\mathbb{C}_1, R_1)/(\mathbb{C}_2, R_2) = \{(\mathbb{A}, \mathbb{B}\{Q_2\})\}$.

□

Having defined a set of residuals of a redex w.r.t. a reduction step, we can extend it to a set of residuals of a *set of redexes* w.r.t. a reduction step and to a set of residuals of a redex or a set of redexes *w.r.t. a reduction sequence* by definitions A.4 and A.5. We also use the notion of *extended reduction* given in definition A.7.

For the term calculus $\mathcal{T}$ we define a development step via the extended reduction, i.e. by definition A.10. According to this definition $(M_1, F_1) \overset{(\mathbb{C},R)}{\underset{dev}{\rightarrow}} (M_2, F_2)$ if $(M_1, F_1) \xrightarrow{(\mathbb{C},R)} (M_2, F_2)$ and $(\mathbb{C}, R) \in F_1$.

Below we prove an important property of the term calculus: finiteness of developments, where a development step $\underset{dev}{\rightarrow}$ is as defined above. Even though the proof is traditional (see Chapter 11 of [Bar84]), we give it in some detail, since some notions defined for the proof will be used later in the proof of finiteness of γ-developments of the core module calculus $\mathcal{C}$. In particular the notion of a weighting introduced below will be used in both this and the next section.

Definition B.6. A calculus of weighted terms $\mathcal{T}'$ is defined as follows: let n range over positive integers, then

$$\begin{array}{rcll} x^n & \in & \mathcal{T}', & \text{where } x \in \textbf{Variable}, \\ c^n & \in & \mathcal{T}', & \text{where } c \in \textbf{Const} \\ l^n & \in & \mathcal{T}' & \text{where } l \in \textbf{Label} \\ \lambda x.M & \in & \mathcal{T}' & \text{if } M \in \mathcal{T}' \\ M \mathbin{@} N & \in & \mathcal{T}' & \text{if } M, N \in \mathcal{T}' \\ M \; op \; N & \in & \mathcal{T}' & \text{if } M, N \in \mathcal{T}' \end{array}$$

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

If $M \in \mathcal{T}'$, the *measure* of M (denoted $|| M ||$) is by definition the sum of all weights (of variables, constants, and labels) occurring in M. Note that $|| M ||> 0$.

Assuming that all occurrences of variables (except for those immediately preceded by a lambda), constants, and labels in a term $M' \in \mathcal{T}'$ are ordered by their position from left to right in M', we can consider M as a pair (M, I), where $M \in \mathcal{T}$ is a term obtained from M' by erasing all weights, and I is a list of weights of variables, labels, and constants in M' from left to right. I is called a *weighting* of M. Note that there is a one-to-one correspondence between terms $M' \in \mathcal{T}'$ and pairs (M, I).

We write $I(x)$, where x is a particular occurrence of x in M, to denote the weight of the occurrence of x in the corresponding $M' \in \mathcal{T}'$, and similarly for $I(c)$ and $I(l)$. □

Now we extend the reduction on pairs (M, F) to include weightings.

Definition B.7. Let $M, N \in \mathcal{T}'$. Then $M \longrightarrow N$ if and only if $M = \mathbb{C}\{R\}$, $N = \mathbb{C}\{Q\}$, and one of the following takes place:

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

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

  and the rest of the rules as usual.

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

□

Note: in the above definition $\mathbb{C}$ is a context over terms of $\mathcal{T}'$. We omit a straightforward definition of such contexts.

Remark B.8. Since terms of $\mathcal{T}'$ are in one-to-one correspondence with pairs (M, I), definition B.7 also defines a reduction on such pairs. □

Definition B.9. A weighting I is called a *decreasing weighting* of a pair (M, F) ($M \in \textbf{Term}_{\mathcal{T}}$) if for any $(\mathbb{C}, R) \in F$ s.t. $R = (\lambda x.N) \mathbin{@} V$ for all occurrences of x in N $|| x ||>|| V ||$. □

Lemma B.10. *For any pair (M, F) there exists a decreasing weighting.* □

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

Definition B.11. We say that a triple (M_1, F_1, I_1) reduces to a triple (M_2, F_2, I_2) by a redex $(\mathbb{C}, R)$ (denoted $(M_1, F_1, I_1) \xrightarrow{(\mathbb{C},R)} (M_2, F_2, I_2)$) if both conditions below hold:

- $(M_1, I_1) \xrightarrow{(\mathbb{C},R)} (M_2, I_2)$, as defined in B.7, see also remark B.8.
- $(M_1, F_1) \xrightarrow{(\mathbb{C},R)} (M_2, F_2)$ by the extended calculus reduction.

□

Convention: the notation (M, F, I) assumes that the triple is well-formed, i.e. I is a weighting of M, and the pair (M, F) is well-formed. Trivially generalizing definition of developments to the triples (M, F, I), we say that a *development step* of (M, F, I) is a reduction step that reduces a redex in F, and a *development* is a sequence of such steps.

Note: in the lemma below we consider measure $|| \cdot ||$ on pairs (M, I), since this measure is defined on terms of $\mathcal{T}'$ which are in one-to-one correspondence with such pairs.

Lemma B.12. *If* $(M_1, F_1, I_1) \xrightarrow{(\mathbb{C},R)} (M_2, F_2, I_2)$*, and* I_1 *is a decreasing weighting of* (M_1, F_1)*, then* I_2 *is a decreasing weighting of* (M_2, F_2)*, and* $|| (M_1, I_1) || > || (M_2, I_2) ||$. □

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

Lemma B.13. *Any development of a pair* (M, F) *is finite.* □

Proof. By lemma B.10 there exists a decreasing weighting of (M, F), let I denote such a weighting. Consider a development of (M, F, I). By lemma B.12 for every development step $(M, F, I) \underset{dev}{\rightarrow} (M', F', I')$, where I is decreasing, we have: I' is also decreasing, and $|| (M, I) || < || (M', I') ||$. Since $|| (M, I) || > 0$ for any well-defined pair (M, I), there can not be an infinite development. □

In order to prove confluence of $\mathcal{T}$, we prove a parallel move lemma (lemma B.19 below) for pairs (M, F), i.e. terms with sets of marked redexes. To do this, we need to show not only weak confluence for unmarked terms, but also that for any marked redex in the original term the set of its residuals on both reduction paths given by the parallel move lemma is the same. Note that in general (i.e. for arbitrary reduction paths, not necessarily those given by the parallel move lemma) the former does not imply the latter: it is possible that a term M reduces to a term N by two different reduction paths, but a marked redex in M has different residual sets in M. For more on this issue see discussion of strongly equivalent reductions in [Bar84].

We prove parallel move lemma in two steps: first we show the property for one marked redex (lemma B.18), and then generalize it to an arbitrary set of marked redexes. Before we prove the lemmas, we need to define terminology for mutual positions of two subterms in a term.

Definition B.14.

1. A *subterm* of a term $M \in \mathbf{Term}_{\mathcal{T}}$ is a pair $(\mathbb{C}, N)$ s.t. $M = \mathbb{C}\{N\}$ (specifying $\mathbb{C}$ allows us to avoid ambiguity if N occurs in M more than once). If the position of N in M is unambiguous or irrelevant, we say that N is a subterm of M.
2. 2 subterms $(\mathbb{C}_1, N_1)$ and $(\mathbb{C}_2, N_2)$ of a term M are *independent* if there exists a two-hole context $\mathbb{A}$ s.t. $\mathbb{C}_1 = \mathbb{A}\{\square, N_2\}$ and $\mathbb{C}_2 = \mathbb{A}\{N_1, \square\}$ or, alternatively, $\mathbb{C}_1 = \mathbb{A}\{N_2, \square\}$ and $\mathbb{C}_2 = \mathbb{A}\{\square, N_1\}$. Note that this implies $M = \mathbb{A}\{N_1, N_2\}$ or $M = \mathbb{A}\{N_2, N_1\}$.
3. If $(\mathbb{C}_1, N_1)$ and $(\mathbb{C}_2, N_2)$ are subterms of the same term M and are not independent, then it must be the case that either N_2 is a subterm of N_1 (in which case we say that $(\mathbb{C}_1, N_1)$ *contains* $(\mathbb{C}_2, N_2)$) or vice versa (i.e. $(\mathbb{C}_2, N_2)$ *contains* $(\mathbb{C}_1, N_1)$). As in case 1, sometimes we will omit the contexts, e.g. say that N_1 contains N_2.
4. We say that two redexes $(\mathbb{C}_1, R_1)$ and $(\mathbb{C}_2, R_2)$ of a term M are *independent* (respectively $(\mathbb{C}_1, R_1)$ contains $(\mathbb{C}_2, R_2)$) if they are independent as subterms (respectively $(\mathbb{C}_1, R_1)$ contains $(\mathbb{C}_2, R_2)$ as subterms).

□

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

Lemma B.15. *If R is a redex and x is not bound in R, then $R[x := V]$ is a redex.*
If $\mathbb{E} \in \mathbf{EvalContext}_{\mathcal{T}}$ and x is not bound in $\mathbb{E}$, then $\mathbb{E}[x := V] \in \mathbf{EvalContext}_{\mathcal{T}}$ $\square$

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

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

Lemma B.16. *If $R = \mathbb{A}\{l\}$ is a redex, then $R' = \mathbb{A}\{V\}$ is also a redex.*
If $\mathbb{E} = \mathbb{A}\{\square, l\} \in \mathbf{EvalContext}_{\mathcal{T}}$ (respectively $\mathbb{E} = \mathbb{A}\{l, \square\} \in \mathbf{EvalContext}_{\mathcal{T}}$), then $\mathbb{E}' = \mathbb{A}\{\square, V\} \in \mathbf{EvalContext}_{\mathcal{T}}$ (respectively $\mathbb{E}' = \mathbb{A}\{V, \square\} \in \mathbf{EvalContext}_{\mathcal{T}}$). $\square$

Proof. Similar to the proof of lemma B.15. Note that labels in a term are not bound, therefore the occurrence of l is free in a redex or in an evaluation context. $\square$

Lemma B.17. *If $\mathbb{E} = \mathbb{A}\{\square, R\} \in \mathbf{EvalContext}_{\mathcal{T}}$ (or $\mathbb{E} = \mathbb{A}\{R, \square\} \in \mathbf{EvalContext}_{\mathcal{T}}$) and $R \rightsquigarrow_{\mathcal{T}} Q$, then $\mathbb{E}' = \mathbb{A}\{\square, Q\} \in \mathbf{EvalContext}_{\mathcal{T}}$ (respectively $\mathbb{E}' = \mathbb{A}\{Q, \square\} \in \mathbf{EvalContext}_{\mathcal{T}}$).* $\square$

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

The following is a key lemma for many subsequent proofs. In addition to weak confluence of the calculus reduction $\longrightarrow_{\mathcal{T}}$, it shows that for any marked redex in M_1 its residuals are the same on both reduction paths. We also show confluence of a complete development of $(\mathbb{C}_1, R_1)/(\mathbb{C}_2, R_2)$ for any two redexes. Note that we do not show and do not use a general confluence of complete developments, i.e. confluence of a complete development of an arbitrary F/S, even though such confluence holds in $\mathcal{T}$. This is because our goal is to show standardization of developments, and for this purpose it is sufficient to show that a non-evaluation step followed by an evaluation step in a development can be replaced by a development sequence which starts with an evaluation step (corollary B.26). This property, in addition to finiteness of developments that we have already shown, implies standardization of developments.

Lemma B.18. *Part 1. Let $(M_1, \{(\mathbb{A}, \tilde{R})\}) \xrightarrow{(\mathbb{C}_1, R_1)} (M_2, F_2)$, $(M_1, \{(\mathbb{A}, \tilde{R})\}) \xrightarrow{(\mathbb{C}_2, R_2)} (M_3, F_3)$ s.t. $(\mathbb{C}_1, R_1) \neq (\mathbb{C}_2, R_2)$. Then there exists (M_4, F_4) s.t. $(M_2, F_2) \overset{(\mathbb{C}_2,R_2)/(\mathbb{C}_1,R_1)}{\underset{dev}{\rightarrow^*}} (M_4, F_4)$, $(M_3, F_3) \overset{(\mathbb{C}_1,R_1)/(\mathbb{C}_2,R_2)}{\underset{dev}{\rightarrow^*}} (M_4, F_4)$.*

Part 2. If $(M_2, F_2) \overset{(\mathbb{C}_2,R_2)/(\mathbb{C}_1,R_1)}{\underset{dev}{\rightarrow^*}} (M_4, F_4)$ for some complete development sequence $\overset{(\mathbb{C}_2,R_2)/(\mathbb{C}_1,R_1)}{\underset{dev}{\rightarrow^*}}$, then $(M_2, F_2) \overset{(\mathbb{C}_2,R_2)/(\mathbb{C}_1,R_1)}{\underset{dev}{\rightarrow^*}} (M_4, F_4)$ for any complete development sequence $\overset{(\mathbb{C}_2,R_2)/(\mathbb{C}_1,R_1)}{\underset{dev}{\rightarrow^*}}$, and the same for a complete development $\overset{(\mathbb{C}_1,R_1)/(\mathbb{C}_2,R_2)}{\underset{dev}{\rightarrow^*}}$. $\square$

Proof. The proof is by cases on mutual positions (and the kinds) of the three redexes in M_1. We show enough cases to demonstrate the proof technique. The other cases are similar to the ones shown. We don't list cases symmetric to the ones given (i.e. those obtained by switching $(\mathbb{C}_1, R_1)$ and $(\mathbb{C}_2, R_2)$). We suppose that $R_1 \rightsquigarrow Q_1$, $R_2 \rightsquigarrow Q_2$, $\tilde{R} \rightsquigarrow \tilde{Q}$.

1. All three redexes are independent - trivial.
2. $(\mathbb{A}, \tilde{R}) = (\mathbb{C}_1, R_1)$, $(\mathbb{C}_2, R_2)$ is independent from $(\mathbb{C}_1, R_1)$ - trivial.
3. $(\mathbb{A}, \tilde{R}) = (\mathbb{C}_1, R_1)$, $(\mathbb{C}_2, R_2)$ is contained in $(\mathbb{C}_1, R_1)$.

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

- $N_1 = \mathbb{B}\{R_2, x, \ldots, x\}$, i.e. N_1 has several occurrences of x and an occurrence[22] of R_2. Then

$$\begin{array}{llll}(\lambda x.\mathbb{B}\{R_2, x, \ldots, x\}) @ V_1 & \to_{\mathcal{T}} \mathbb{B}\{R_2, V_1, \ldots, V_1\} & \to_{\mathcal{T}} & \mathbb{B}\{Q_2, V_1, \ldots, V_1\}, \\ (\lambda x.\mathbb{B}\{R_2, x, \ldots, x\}) @ V_1 & \to_{\mathcal{T}} (\lambda x.\mathbb{B}\{Q_2, x, \ldots, x\}) @ V_1 & \to_{\mathcal{T}} & \mathbb{B}\{Q_2, V_1, \ldots, V_1\}.\end{array}$$

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

$$\begin{array}{llll}(\lambda x.\mathbb{B}\{x, \ldots, x\}) @ \lambda y.\tilde{\mathbb{B}}\{R_2\} & \to_{\mathcal{T}} \mathbb{B}\{\lambda y.\tilde{\mathbb{B}}\{R_2\}, \ldots, \lambda y.\tilde{\mathbb{B}}\{R_2\}\} & \to^*_{\mathcal{T}} & \mathbb{B}\{\lambda y.\tilde{\mathbb{B}}\{Q_2\}, \ldots, \lambda y.\tilde{\mathbb{B}}\{Q_2\}\}, \\ (\lambda x.\mathbb{B}\{x, \ldots, x\}) @ \lambda y.\tilde{\mathbb{B}}\{R_2\} & \to_{\mathcal{T}} (\lambda x.\mathbb{B}\{x, \ldots, x\}) @ \lambda y.\tilde{\mathbb{B}}\{Q_2\} & \to_{\mathcal{T}} & \mathbb{B}\{\lambda y.\tilde{\mathbb{B}}\{Q_2\}, \ldots, \lambda y.\tilde{\mathbb{B}}\{Q_2\}\}.\end{array}$$

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

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

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

   In this case $M_1 = \mathbb{B}\{R_1, \tilde{R}\}$. By case 3 we know that since $(\mathbb{C}_1, R_1)$ contains $(\mathbb{C}_2, R_2)$, the two redexes can be performed in any order. $(\mathbb{A}, \tilde{R})$ is independent from these redexes, so clearly it has the same residual on both reduction paths. As in case 3, the order in which copies of R_2 are reduced in the multi-step reduction does not matter.

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

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

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

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

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

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

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

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

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

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

8. $(\mathbb{C}_1, R_1)$ contains $(\mathbb{C}_2, R_2)$, $(\mathbb{A}, \tilde{R})$ is contained in $(\mathbb{C}_2, R_2)$. Similar to previous cases.
9. $(\mathbb{C}_1, R_1)$ contains $(\mathbb{C}_2, R_2)$, $(\mathbb{A}, \tilde{R})$ is contained in $(\mathbb{C}_1, R_1)$, $(\mathbb{A}, \tilde{R})$ contains $(\mathbb{C}_2, R_2)$. Similar to previous cases.
10. $(\mathbb{C}_1, R_1)$ and $(\mathbb{C}_2, R_2)$ are independent, $(\mathbb{A}, \tilde{R})$ contains both $(\mathbb{C}_1, R_1)$ and $(\mathbb{C}_2, R_2)$. Similar to previous cases.
11. $(\mathbb{C}_1, R_1)$ and $(\mathbb{C}_2, R_2)$ are independent, $(\mathbb{A}, \tilde{R})$ contains $(\mathbb{C}_1, R_1)$, but not $(\mathbb{C}_2, R_2)$. Similar to previous cases.
12. $(\mathbb{C}_1, R_1)$ and $(\mathbb{C}_2, R_2)$ are independent, $(\mathbb{A}, \tilde{R})$ is contained in $(\mathbb{C}_1, R_1)$. Similar to previous cases.

□

Lemma B.19 (Parallel Moves Lemma). *Let* $(M_1, F_1) \xrightarrow{(\mathbb{C}_1, R_1)} (M_2, F_2)$, $(M_1, F_1) \xrightarrow{(\mathbb{C}_2, R_2)} (M_3, F_3)$. *Then there exists* (M_4, F_4) *s.t.* $(M_2, F_2) \xrightarrow{(\mathbb{C}_2, R_2)/(\mathbb{C}_1, R_1)}{}^* (M_4, F_4)$, $(M_3, F_3) \xrightarrow{(\mathbb{C}_1, R_1)/(\mathbb{C}_2, R_2)}{}^* (M_4, F_4)$. □

Proof. By induction on the number of redexes in F_1. The base case is given by Part 1 of lemma B.18. Suppose F_1' contains n redexes, and $(M_1, F_1') \xrightarrow{(\mathbb{C}_1, R_1)} (M_2, F_2')$, $(M_1, F_1') \xrightarrow{(\mathbb{C}_2, R_2)} (M_3, F_3')$ imply that there exists (M_4, F_4') s.t. $(M_2, F_2') \xrightarrow{(\mathbb{C}_2, R_2)/(\mathbb{C}_1, R_1)}{}^* (M_4, F_4')$, $(M_3, F_3') \xrightarrow{(\mathbb{C}_1, R_1)/(\mathbb{C}_2, R_2)}{}^* (M_4, F_4')$. Let $F = F_1' \cup \{(\mathbb{A}, \tilde{R})\}$, where $(\mathbb{A}, \tilde{R}) \notin F_1'$. Let $\tilde{F}_2, \tilde{F}_3$ be s.t. $(M_1, \{(\mathbb{A}, \tilde{R})\}) \xrightarrow{(\mathbb{C}_1, R_1)} (M_2, \tilde{F}_2)$ and $(M_1, \{(\mathbb{A}, \tilde{R})\}) \xrightarrow{(\mathbb{C}_2, R_2)} (M_3, \tilde{F}_3)$. By Part 1 of lemma B.18 there exist $(\tilde{M}_4, \tilde{F}_4)$ s.t. $(M_2, \tilde{F}_2) \xrightarrow{(\mathbb{C}_2, R_2)/(\mathbb{C}_1, R_1)}{}^* (\tilde{M}_4, \tilde{F}_4)$ and $(M_3, \tilde{F}_3) \xrightarrow{(\mathbb{C}_1, R_1)/(\mathbb{C}_2, R_2)}{}^* (\tilde{M}_4, \tilde{F}_4)$. By Part 2 of lemma B.18 all complete developments of the same sets of marked redexes end at the same pair $(\tilde{M}_4, \tilde{F}_4)$. Therefore we can assume that the sequences $\xrightarrow{(\mathbb{C}_2, R_2)/(\mathbb{C}_1, R_1)}{}^*$ and $\xrightarrow{(\mathbb{C}_1, R_1)/(\mathbb{C}_2, R_2)}{}^*$ constructed for the initial pair $(M_1, \{(\mathbb{A}, \tilde{R})\})$ are the same as the respective sequences in the inductive hypothesis. Then $\tilde{M}_4 = M_4$, since by the inductive hypothesis (and by definition of extended reduction) $M_3 \xrightarrow{(\mathbb{C}_2, R_2)/(\mathbb{C}_1, R_1)}{}^* M_4$, where the reduction sequence is the same as the sequence in $(M_3, \tilde{F}_3) \xrightarrow{(\mathbb{C}_1, R_1)/(\mathbb{C}_2, R_2)}{}^* (\tilde{M}_4, \tilde{F}_4)$. Finally, by corollary A.42 $(M_1, F_1' \cup \{(\mathbb{A}, \tilde{R})\}) \xrightarrow{(\mathbb{C}_1, R_1)} (M_2, F_2' \cup \tilde{F}_2)$ and $(M_1, F_1' \cup \{(\mathbb{A}, \tilde{R})\}) \xrightarrow{(\mathbb{C}_2, R_2)} (M_3, F_3' \cup \tilde{F}_3)$ imply that $(M_2, F_2' \cup \tilde{F}_2) \xrightarrow{(\mathbb{C}_2, R_2)/(\mathbb{C}_1, R_1)}{}^* (M_4, F_4' \cup \tilde{F}_4)$ and $(M_3, F_3' \cup \tilde{F}_3) \xrightarrow{(\mathbb{C}_1, R_1)/(\mathbb{C}_2, R_2)}{}^* (M_4, F_4' \cup \tilde{F}_4)$. □

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

Lemma B.20 (Strip Lemma). *Let* $(M_1, F_1) \longrightarrow (M_2, F_2)$, $(M_1, F_1) \longrightarrow^* (M_3, F_3)$. *Then there exist* M_4, F_4 *s.t.* $(M_2, F_2) \longrightarrow^* (M_4, F_4)$, $(M_3, F_3) \longrightarrow^* (M_4, F_4)$. □

Proof. Let $(\mathbb{C}, R)$ be the redex reduced in the reduction $(M_1, F_1) \longrightarrow (M_2, F_2)$, and let S be the reduction sequence $(M_1, F_1) \longrightarrow^* (M_3, F_3)$.

Claim 1. Given $(M_1, F_1) \xrightarrow{(\mathbb{C}, R)} (M_2, F_2)$ and $(M_1, F_1) \xrightarrow{S}{}^* (M_3, F_3)$, there exist M_4, F_4 s.t. $(M_2, F_2) \xrightarrow{S'}{}^* (M_4, F_4)$ and $(M_3, F_3) \xrightarrow{(\mathbb{C}, R)/S}{}^* (M_4, F_4)$.

We prove the claim by induction on the number of steps in S. Base case follows immediately from lemma B.19. Induction step: suppose Claim 1 holds for a sequence S of n steps. Let us consider a sequence $S; (\mathbb{A}, \tilde{R})$ of $n+1$ steps s.t. $(M_1, F_1) \xrightarrow{S}{}^* (M_3', F_3') \xrightarrow{(\mathbb{A}, \tilde{R})} (M_3, F_3)$. By the inductive hypothesis there exist M_4', F_4' s.t. $(M_2, F_2) \xrightarrow{S'}{}^* (M_4', F_4')$, $(M_3', F_3') \xrightarrow{(\mathbb{C}, R)/S}{}^* (M_4', F_4')$.

Let $\hat{F}$ be the set of residuals of $(\mathbb{C}, R)$ in M_3 w.r.t. $S; (\mathbb{A}, \tilde{R})$, and let $\tilde{F}$ be the set of residuals of $(\mathbb{A}, \tilde{R})$ in M'_4 w.r.t. the reduction sequence $(M'_3, F'_3) \xrightarrow{(\mathbb{C},R)/S}{}^{*} (M'_4, F'_4)$. Let $(\mathbb{C}', R')$ be the first redex reduced in the sequence $\xrightarrow{(\mathbb{C},R)/S}{}^{*}$. By lemma B.19 there exist M', F' s.t. $(M_3, F_3) \xrightarrow{(\mathbb{C}',R')/(\mathbb{A},\tilde{R})}{}^{*} (M', F')$ and $(M'_3, F'_3) \xrightarrow{(\mathbb{C}',R')} (M'', F'') \xrightarrow{(\mathbb{A},\tilde{R})/(\mathbb{C}',R')}{}^{*} (M', F')$. Now we can apply lemma B.19 to the first redex in the sequence $\xrightarrow{(\mathbb{A},\tilde{R})/(\mathbb{C}',R')}{}^{*}$ and the first redex in $\xrightarrow{(\mathbb{C}',R')/(\mathbb{A},\tilde{R})}{}^{*}$, and so on. This process is guaranteed to terminate, because all redexes reduced in all the constructed sequences are residuals of redexes in the sets $\hat{F}$ or $\tilde{F}$, therefore by lemma B.13 on finiteness of developments these reductions must be finite.

We have shown Claim 1. The claim of the lemma follows by omitting the redex annotations from the reduction sequences. $\square$

Confluence of the (extended) reduction of term calculus immediately follows from the strip lemma B.20 (see [Bar84], Chapter 11 for the proof).

Theorem B.21 (Confluence of $\mathcal{T}$). *If $(M_1, F_1) \longrightarrow^*_{\mathcal{T}} (M_2, F_2)$ and $(M_1, F_1) \longrightarrow^*_{\mathcal{T}} (M_3, F_3)$, then there exist M_4, F_4 s.t. $(M_2, F_2) \longrightarrow^*_{\mathcal{T}} (M_4, F_4)$, $(M_3, F_3) \longrightarrow^*_{\mathcal{T}} (M_4, F_4)$.* $\square$

Now when we have proven confluence, it remains to show class preservation and standardization in order to show computational soundness. Before we prove these two properties, we show some results needed for the proofs.

Lemma B.22. *Let $M \overset{(\mathbb{C},R_1)}{\circ\!\!\longrightarrow}_{\mathcal{T}} N$, where $N = R_2$ is a redex in $\mathcal{T}$. Then $N = (\lambda x.N_1) \mathbin{@} V$ and $M = (\lambda x.\mathbb{A}\{R_1\}) \mathbin{@} V$ or $M = (\lambda x.N_1) \mathbin{@} \lambda y.\mathbb{A}\{R_1\}$.* $\square$

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

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

Lemma B.23. *Let $M \overset{(\mathbb{C},R_1)}{\circ\!\!\longrightarrow}_{\mathcal{T}} N$. Then $M = \mathbb{E}\{R\}$ if and only if $N = \mathbb{E}'\{R'\}$. If $M = \mathbb{E}\{R\}, N = \mathbb{E}'\{R'\}$, then $(\mathbb{E}', R') = (\mathbb{E}, R)/(\mathbb{C}, R_1)$ (recall that the notation implies that $(\mathbb{E}', R')$ is the only residual of $(\mathbb{E}, R)$).* $\square$

Proof. The proof is by induction on the structure of an evaluation context. We want to show that $M = \mathbb{E}\{R\}$ if and only if $N = \mathbb{E}'\{R'\}$ and $\mathbb{E}$ and $\mathbb{E}'$ are of the same shape (see 5 cases of definition of an evaluation context in $\mathcal{T}$ on figure 1).

Base case. Suppose $M = \mathbb{E}\{R\}$ and $\mathbb{E} = \square$. Then $M = c_1 \; op \; c_2$ or $M = \lambda x.M_1 \mathbin{@} V$. However, since $M \circ\!\!\longrightarrow_{\mathcal{T}} N$, it can not be the case that $M = c_1 \; op \; c_2$. Therefore $M = (\lambda x.M_1) \mathbin{@} V$. The redex R_1 reduced by the non-evaluation step may occur either in M_1 or in V, i.e. there are the following two cases:

$$\begin{array}{lcllcl} M & = & (\lambda x.\mathbb{A}\{R_1, x, \ldots, x\}) \mathbin{@} V & \overset{(\mathbb{C},R_1)}{\circ\!\!\longrightarrow}_{\mathcal{T}} & (\lambda x.\mathbb{A}\{Q_1, x, \ldots, x\}) \mathbin{@} V & = & N \\ M & = & (\lambda x.\mathbb{A}\{x, \ldots, x\}) \mathbin{@} \lambda y.\mathbb{B}\{R_1\} & \overset{(\mathbb{C},R_1)}{\circ\!\!\longrightarrow}_{\mathcal{T}} & (\lambda x.\mathbb{A}\{x, \ldots, x\}) \mathbin{@} \lambda y.\mathbb{B}\{Q_1\} & = & N \end{array}$$

In both cases $N = \mathbb{E}'\{R'_1\}$, where $\mathbb{E}' = \square$.

Now suppose that $N = \mathbb{E}'\{R'_1\}, \mathbb{E}' = \square$. By lemma B.22 $N = \lambda x.N_1 \mathbin{@} V$ and either $M = (\lambda x.\mathbb{A}\{R_1\}) \mathbin{@} V$ or $M = (\lambda x.N_1) \mathbin{@} \lambda y.\mathbb{A}\{R_1\}$. In both cases M is a redex, i.e. $M = \mathbb{E}\{R\}$, where $\mathbb{E} = \square$.

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

1. $M = \mathbb{E}\{R_2\}$, where $\mathbb{E} = \mathbb{E}_1 \ @\ M_1$. Since $M \circ\!\!\xrightarrow{(\mathbb{C},R_1)}_{\mathcal{T}} N$, we have two possibilities: either $\mathbb{C} = \mathbb{C}_1 \ @\ M_1$, in which case $\mathbb{C}_1$ is a non-evaluation context, or $\mathbb{C} = (\mathbb{E}_1\{R_2\}) \ @\ \mathbb{C}_1$. In the former case $\mathbb{E}_1\{R_2\} = \mathbb{C}_1\{R_1\} \circ\!\!\xrightarrow{(\mathbb{C}_1,R_1)}_{\mathcal{T}} \mathbb{C}_1\{Q_1\}$, and by inductive hypothesis (since $\mathbb{E}_1$ is a subcontext of $\mathbb{E}$) we have $\mathbb{C}_1\{Q_1\} = \mathbb{E}'_1\{R'_2\}$, and hence $N = \mathbb{E}'\{R'_2\}$, where $\mathbb{E}' = \mathbb{E}'_1 \ @\ M_1$. In the latter case $(\mathbb{E}_1\{R_2\}) \ @\ \mathbb{C}_1\{R_1\} \circ\!\!\xrightarrow{(\mathbb{C},R_1)}_{\mathcal{T}} (\mathbb{E}_1\{R_2\}) \ @\ \mathbb{C}_1\{Q_1\}$, i.e. $N = \mathbb{E}'\{R_2\}$, where $\mathbb{E}' = \mathbb{E}_1 \ @\ \mathbb{C}_1\{Q_1\}$.

   Now suppose that $N = \mathbb{E}'\{R'_2\}$, where $\mathbb{E}' = \mathbb{E}'_1 \ @\ N_1$. Since $M \circ\!\!\xrightarrow{(\mathbb{C},R_1)}_{\mathcal{T}} N$, we have two cases:

   - $M = (\mathbb{C}_1\{R_1\}) \ @\ N_1$, where $\mathbb{C}_1$ is a non-evaluation context. Then $\mathbb{C}_1\{R_1\} \circ\!\!\xrightarrow{(\mathbb{C}_1,R_1)}_{\mathcal{T}} \mathbb{C}_1\{Q_1\} = \mathbb{E}'_1\{R'_2\}$, and by the inductive hypothesis $\mathbb{C}_1\{R_1\} = \mathbb{E}_1\{R_2\}$, so $M = (\mathbb{E}_1\{R_2\}) \ @\ N_1$.
   - $M = (\mathbb{E}'_1\{R'_2\}) \ @\ \mathbb{C}_1\{R_1\}$, where $\mathbb{C}_1$ may be an evaluation or a non-evaluation context. In this case $M = \mathbb{E}\{R'_2\}$, where $\mathbb{E} = \mathbb{E}'_1 \ @\ \mathbb{C}_1\{R_1\}$, i.e. $\mathbb{E}$ is of the same shape as $\mathbb{E}'$.

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

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

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

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

□

Lemma B.24 (Class Preservation). *If $M \circ\!\!\xrightarrow{(\mathbb{C},R_1)}_{\mathcal{T}} N$, then $Cl_{\mathcal{T}}(M) = Cl_{\mathcal{T}}(N)$.* □

Proof. It is straightforward to show that if $Cl_{\mathcal{T}}(M)$ is one of the following: $\mathbf{const}(c), \mathbf{var}, \mathbf{abs}$, or $\mathbf{stuck}(l)$, then $Cl_{\mathcal{T}}(N) = Cl_{\mathcal{T}}(M)$. Similarly if we assume that $Cl_{\mathcal{T}}(N)$ is one of the above 4 classes, then $Cl_{\mathcal{T}}(M) = Cl_{\mathcal{T}}(N)$. The cases of classes $\mathbf{const}(c), \mathbf{var}$, and $\mathbf{abs}$ are easy. Below we show the case $\mathbf{stuck}(l)$:

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

Base case: the case when $M = l$ is impossible, since there is no N s.t. $M \circ\!\!\longrightarrow_{\mathcal{T}} N$. Instead we use the following as base cases for M: $l \ @\ M_1$, $(\lambda x.M_1) \ @\ l$, $l\ op\ M_1$, $c\ op\ l$. If $M = l \ @\ M_1$, then $M \circ\!\!\longrightarrow_{\mathcal{T}} N = l \ @\ N_1$, where $M_1 \circ\!\!\longrightarrow_{\mathcal{T}} N_1$, i.e. $Cl_{\mathcal{T}}(N) = \mathbf{stuck}(l)$. Similarly, $N = (\lambda x.N_1) \ @\ l$, where $M_1 \longrightarrow_{\mathcal{T}} N_1$ (i.e. $\lambda x.M_1 \circ\!\!\longrightarrow_{\mathcal{T}} \lambda x.N_1$) if $M = (\lambda x.M_1) \ @\ l$, $N = l\ op\ N_1$, where $M_1 \circ\!\!\longrightarrow_{\mathcal{T}} N_1$ if $M = l\ op\ M_1$, and it may not be the case that $M \circ\!\!\longrightarrow_{\mathcal{T}} N$ if $M = c\ op\ l$.

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

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

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

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

It follows from lemma B.23 that $Cl_{\mathcal{T}}(M) = \mathbf{evaluatable}$ if and only if $Cl_{\mathcal{T}}(N) = \mathbf{evaluatable}$. This concludes the proof since the remaining class **error** is defined as the class of terms that do not belong to any of the above classes. □

Our proof of standardization of developments follows the approach described in section A.2, i.e. we show property A.38, and since we have shown the finiteness of developments, by lemma A.39 we prove standardization of developments. The following lemma implies property A.38 of $\mathcal{T}$.

Lemma B.25. *If* $M_1 \circ\!\!\xrightarrow{(\mathbb{C},R_1)} M_2 \xRightarrow{(\mathbb{E}',R_2')} M_3$. *Then there exists* M_4 *s.t.* $M_1 \xRightarrow{(\mathbb{E},R_2)} M_4 \xrightarrow{(\mathbb{C},R_1)/(\mathbb{E},R_2)}{}^{*} M_3$, *where* $(\mathbb{E}', R_2') = (\mathbb{E}, R_2)/(\mathbb{C}, R_1)$. □

Proof. The proof is by induction on the structure of the evaluation context $\mathbb{E}'$, similarly to the proof of lemma B.23 above.

Base case. Suppose $M_2 = \mathbb{E}'\{R_2'\}$ and $\mathbb{E}' = \Box$. By lemma B.22 $M_2 = (\lambda x.N) \mathbin{@} V$, and either $M_1 = (\lambda x.\mathbb{A}\{R_1, x, \ldots, x\}) \mathbin{@} V$ (where $\mathbb{A}$ is an $n+1$-hole context if x occurs in N n times) or $M_1 = (\lambda x.N) \mathbin{@} \lambda y.\mathbb{B}\{R_1\}$. Suppose $R_1 \rightsquigarrow Q_1$. In the first case, we have:

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

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

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

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

Induction Step. As in the proof of lemma B.22, we have 4 cases. We only show one case, the rest is similar.

- Suppose $M_2 = (\mathbb{E}_1'\{R_2'\}) \mathbin{@} N \Rightarrow_{\mathcal{T}} (\mathbb{E}_1'\{Q_2'\}) \mathbin{@} N = M_3$. Since $M_1 \circ\!\!\xrightarrow{(\mathbb{C},R_1)}_{\mathcal{T}} M_2$, we have one of the following: either $M_1 = (\mathbb{C}_1\{R_1\}) \mathbin{@} N$, or $M_1 = (\mathbb{E}_1'\{R_2'\}) \mathbin{@} \mathbb{C}_1\{R_1\}$.

  In the former case $\mathbb{C}_1\{R_1\} \circ\!\!\xrightarrow{(\mathbb{C}_1,R_1)}_{\mathcal{T}} \mathbb{E}_1'\{R_2'\}$, hence by lemma B.22 $\mathbb{C}_1\{R_1\} = \mathbb{E}_1\{R_2\}$, and by the inductive hypothesis $(\mathbb{E}_1', R_2') = (\mathbb{E}_1, R_2)/(\mathbb{C}_1, R_1)$, and there exists M_4' s.t. $\mathbb{E}_1\{R_2\} \xRightarrow{(\mathbb{E}_1,R_2)}_{\mathcal{T}} M_4' \xrightarrow{(\mathbb{C},R_1)/(\mathbb{E}_1,R_2)}{}^{*}_{\mathcal{T}} \mathbb{E}_1'\{Q_2'\}$. Therefore if we take $M_4 = M_4' \mathbin{@} N$, then the claim of the lemma holds.

  In the latter case $M_1 = (\mathbb{E}_1'\{R_2'\}) \mathbin{@} \mathbb{C}_1\{R_1\} \Rightarrow_{\mathcal{T}} (\mathbb{E}_1'\{Q_2'\}) \mathbin{@} \mathbb{C}_1\{R_1\} \rightarrow_{\mathcal{T}} (\mathbb{E}_1'\{Q_2'\}) \mathbin{@} \mathbb{C}_1\{Q_1\} = M_3$ (assuming $R_1 \rightsquigarrow Q_1$), and the claim holds.

Cases when $M_2 = (\lambda x.N) \mathbin{@} \mathbb{E}_1'\{R_2'\}$, $M_2 = \mathbb{E}_1'\{R_2'\}\ op\ N$, and $M_2 = c_1\ op\ \mathbb{E}_1'\{R_2'\}$, are similar.

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

In the case when both given redexes in M_1 are marked, lemma B.25 immediately implies the following:

Corollary B.26. *If* $(M_1, F_1) \underset{dev}{\circ\!\!\rightarrow} (M_2, F_2) \underset{dev}{\Rightarrow} (M_3, F_3)$, *then there exists* (M_4, F_4) *s.t.* $(M_1, F_1) \underset{dev}{\Rightarrow} (M_4, F_4) \underset{dev}{\rightarrow^{*}} (M_3, F_3)$. □

Proof. Let $(\mathbb{C}, R_1)$ be the redex reduced in the step $(M_1, F_1) \underset{dev}{\circ\!\!\rightarrow} (M_2, F_2)$, and $(\mathbb{E}', R_2')$ be the redex reduced in $(M_2, F_2) \underset{dev}{\Rightarrow} (M_3, F_3)$. Then by lemma B.25 there exists a redex $(\mathbb{E}, R_2)$ in M_1 s.t. for some M_4 $M_1 \xRightarrow{(\mathbb{E},R_2)} M_4 \xrightarrow{(\mathbb{C},R_1)/(\mathbb{E},R_2)}{}^{*} M_3$. Since both given steps are developments, i.e. they reduced marked redexes, it must be the case that $\{(\mathbb{C}, R_1), (\mathbb{E}, R_2)\} \subseteq F_1$. Then $M_1 \xRightarrow{(\mathbb{E},R_2)} M_4$ is a development step, and the sequence $M_4 \xrightarrow{(\mathbb{C},R_1)/(\mathbb{E},R_2)}{}^{*} M_3$ is a development sequence (since all the redexes reduced in this sequence are marked).

If $F_1 = \{(\mathbb{C}, R_1), (\mathbb{E}, R_2)\}$, then $F_3 = \emptyset$ and we are done. Otherwise let $(\mathbb{A}, R) \notin \{(\mathbb{C}, R_1), (\mathbb{E}, R_2)\}$, $(\mathbb{A}, R) \in F_1$. Let us consider an initial pair $(M_1, \{(\mathbb{A}, R)\})$. By lemma B.25 $M_1 \underset{dev}{\circ\!\xrightarrow{(\mathbb{C},R_1)}} M_2 \underset{dev}{\xRightarrow{(\mathbb{E}',R_2')}} M_3$ implies that there exists M_4' s.t. $M_1 \xRightarrow{(\mathbb{E},R_2)} M_4' \xrightarrow{(\mathbb{C},R_1)/(\mathbb{E},R_2)}{}^{*} M_3$, where $(\mathbb{E}', R_2') = (\mathbb{E}, R_2)/(\mathbb{C}, R_1)$. Since the reduction $\xRightarrow{(\mathbb{E},R_2)}$ from M_1 is uniquely defined, $M_4' = M_4$. By part 1 of lemma B.18 $(M_1, \{(\mathbb{A}, R)\}) \underset{dev}{\circ\!\xrightarrow{(\mathbb{C},R_1)}} (M_2, F_2')$ and $(M_1, \{(\mathbb{A}, R)\}) \xRightarrow{(\mathbb{E},R_2)} (M_3, F_3')$ imply that there exists (M_4'', F_4') s.t. $(M_2, F_2') \xrightarrow{(\mathbb{E},R_2)/(\mathbb{C},R_1)}{}^{*} (M_4'', F_4')$ and $(M_4'', F_4') \xrightarrow{(\mathbb{C},R_1)/(\mathbb{E},R_2)}{}^{*} (M_3', F_3')$. Since $(\mathbb{E}', R_2') = (\mathbb{E}, R_2)/(\mathbb{C}, R_1)$, we get $M_4'' = M_4$. By part 2 of lemma B.18 all reductions $\xrightarrow{(\mathbb{C},R_1)/(\mathbb{E},R_2)}{}^{*}$ lead to the same result, therefore we may assume that the reductions $M_4 \xrightarrow{(\mathbb{C},R_1)/(\mathbb{E},R_2)}{}^{*} M_3$ above and $(M_4'', F_4') \xrightarrow{(\mathbb{C},R_1)/(\mathbb{E},R_2)}{}^{*} (M_3', F_3')$ reduce the same sequence of redexes.

Now, when we have shown the claim for one redex, we can show the general claim by induction on the number of redexes in F_1, with the initial pair $(M_1, \{(\mathbb{C}, R_1), (\mathbb{E}, R_2)\})$ as the base case. The induction is similar to that in the proof of the parallel moves lemma B.19 and uses corollary A.42 to join sets of residuals. □

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

Lemma B.27 (Standardization of Developments). *Given* $(M_1, F_1) \underset{dev}{\rightarrow^*} (M_2, F_2)$*, there exists* (M', F') *s.t.* $(M_1, F_1) \underset{dev}{\Rightarrow^*} (M', F') \underset{dev}{\circ\!\!\longrightarrow^*} (M_2, F_2)$. □

Proof. We have shown that the calculus $\mathcal{T}$ has finiteness of developments property A.37 (lemma B.13). In fact, lemma B.10 gives a maximal value of $||\ (M, F)\ ||$, so $\mathcal{T}$ also has property A.35. Corollary B.26 above states that $\mathcal{T}$ has property A.38. Therefore by lemma A.39 it has standardization of developments. □

Another property implied by lemma B.25 is the elementary lift diagram (property A.26).

Lemma B.28 (Elementary Lift Diagram). *If* $(M_1, F_1) \underset{dev}{\circ\!\xrightarrow{(\mathbb{C},R)}} (M_2, F_2) \Rightarrow (M_4, F_4)$*, then there exists* (M_3, F_3) *s.t.* $(M_1, F_1) \xRightarrow{(\mathbb{E},R')} (M_3, F_3) \underset{dev}{\rightarrow^*} (M_4, F_4)$*, where the latter development is a complete development of* $(\mathbb{C}, R)/(\mathbb{E}, R')$. □

Proof. Follows from lemma B.25 in the case when only the redex $(\mathbb{C}, R)$ is marked. □

Since developments in $\mathcal{T}$ are just extended calculus reductions, the calculus has property A.15, and we can show that $\mathcal{T}$ has the standardization property A.34:

Theorem B.29 (Standardization). *If* $(M, F) \rightarrow^*_{\mathcal{T}} (M', F')$*, then there exists* (M'', F'') *s.t.* $(M, F) \Rightarrow^*_{\mathcal{T}} (M'', F'') \circ\!\!\longrightarrow^*_{\mathcal{T}} (M', F')$. □

Proof. $\mathcal{T}$ the following properties: property A.26 by lemma B.28, standardization of developments (property A.29 by lemma B.27), and composition of developments (property A.15) since developments are defined via extended reduction. Therefore by theorem A.32 $\mathcal{T}$ has the lift property A.17. By lemma 3.39 $\mathcal{T}$ has standardization property. □

Theorem B.30 (Computational Soundness of $\mathcal{T}$). *If* $M \leftrightarrow_{\mathcal{T}} N$*, then* $Outcome_{\mathcal{T}}(M) = Outcome_{\mathcal{T}}(N)$. □

Proof. The calculus with the extended reduction (i.e. reductions on pairs (M, F)) has confluence (theorem B.21), class preservation (lemma B.24), and standardization (theorem B.29). By lemma A.20 standardization (i.e. lift) property holds for terms of $\mathcal{T}$ without marked redexes. Similarly confluence for marked terms implies confluence for terms with no marked redexes. Therefore by theorem 3.35 $\mathcal{T}$ is computationally sound. □

C Soundness of the Core Module Calculus.

In this section we show soundness of the core module calculus without a GC reduction, as defined in section 2.4. The next section proves soundness of the calculus $\mathcal{C}_{GC}$ with a GC reduction as defined in section 2.6.

Definitions. Section A gives general proofs of lift and project properties based on axiomatic definitions of a residual and a γ-development step. In this section we fill in calculus-specific details of these definitions and show that they satisfy the requirements given in section A.

Similarly to definition of residuals in the term calculus (Definition B.5) we define a set of residuals of a redex using multi-hole contexts and the greatest lower bound of contexts. To be able to define filling of a module context, we assume that labels in **Label** are ordered according to some total order $<$ (for instance, label names are ordered lexicographically). This allows us to specify the order in which holes of a module context are filled. Note that in this section we are dealing with particular modules, not with α-equivalence classes of modules, and therefore the order of hidden labels in a module is fixed.

Definition C.1. A multi-hole module context is defined as follows:

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

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

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

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

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

Definition C.2. The greatest lower bound of two module contexts is defined as

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

provided $\text{glb}(\mathbb{C}_i, \mathbb{C}'_i)$ is defined for all i s.t. $1 \leq i \leq n$, otherwise it is undefined. Note that for glb to be defined the two module contexts must have the same labels of all components.

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

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

□

Definition C.3 (Projection). If $D = [l_1 \mapsto M_1, \ldots, l_n \mapsto M_n]$, then M_i is called a *projection* of D on a label l_i, denoted by $D \downarrow l_i$.

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

Definition C.4. Let $(\mathbb{D}, R)$, where $\mathbb{D} = [l_i \overset{k}{\underset{i=1}{\mapsto}} M_i, l_k \mapsto \mathbb{C}, l_i \overset{n}{\underset{i=k+1}{\mapsto}} M_i]$, be a module redex. Then l_k is called *the binding label* of $(\mathbb{D}, R)$, and $(\mathbb{C}, R)$ is called *the term projection of the redex* $(\mathbb{D}, R)$ (denoted $(\mathbb{D}, R) \downarrow$). Note that $\mathbb{C} = \mathbb{D} \downarrow l_k$.

If $F = \{(\mathbb{D}_1, R_1), \ldots, (\mathbb{D}_n, R_n)\}$, then $F \downarrow l = \{(\mathbb{D}_i, R_i) \mid (\mathbb{D}_i, R_i) \in F,$ and l is the binding label of $(\mathbb{D}_i, R_i)\}$. □

The following definition of independent subterms and redexes in the module calculus is analogous to definition B.14 of independent subterms and redexes in the term calculus. Note that the "term" part of a module subterm is a term of the calculus $\mathcal{T}$, not $\mathcal{C}$.

Definition C.5. 1. A $\mathcal{T}$-subterm of a module D is a pair $(\mathbb{D}, M)$, where $\mathbb{D} \in \mathbf{Context}_{\mathcal{C}}$, $M \in \mathbf{Term}_{\mathcal{T}}$, and $\mathbb{D}\{M\} = D$.

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

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

$\square$

Definition C.6. A redex $(\mathbb{D}, l)$ is called a *self-referential redex* if l is its binding label. In this case $\mathbb{D} \downarrow l$ is called the *self-referential context* of the redex. $\square$

Definition C.7. Let $D = \mathbb{D}_1\{R_1\}$, where $(\mathbb{D}_1, R_1)$ is a redex, and assume that $D \xrightarrow{(\mathbb{D}_2,R_2)} D'$. A *set of residuals* of $(\mathbb{D}_1, R_1)$ w.r.t. the reduction $\xrightarrow{(\mathbb{D}_2,R_2)}$ denoted $(\mathbb{D}_1, R_1)/(\mathbb{D}_2, R_2)$ is defined as follows: let $\mathbb{D}_3 = \mathrm{glb}(\mathbb{D}_1, \mathbb{D}_2)$ and assume (without loss of generality) that $D = \mathbb{D}_3\{R_1, R_2\}$ in the case when $\mathbb{D}_3 \in \mathbf{Context}^2_{\mathcal{C}}$, i.e. that R_1, R_2 fill the two holes of $\mathbb{D}_3$ in this order, and $R_2 \rightsquigarrow Q_2$ if $R_2 \in \mathbf{TermRedex}$, then

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

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

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

4. In this case a given redex is contained in a term redex being reduced. Note that a term redex of the form c_1 *op* c_2 can not contain another redex, so the term redex can only be an application. If $\mathbb{D}_3 \in \mathbf{Context}^1_{\mathcal{C}}$ and $R_2 = \mathbb{A}\{R_1\}$, then $R_2 \in \mathbf{TermRedex}$, and:
   - If $R_2 = \lambda x.\mathbb{C}^n\{R_1, x, \ldots, x\}$ @ V, then $(\mathbb{D}_1, R_1)/(\mathbb{D}_2, R_2) = \{(\mathbb{D}_2\{\mathbb{C}^n\{\square, V, \ldots, V\}\}, R_1[x := V])\}$. This case defines the residual of a redex inside the abstraction when an application is reduced.
   - If $R_2 = \lambda x.\mathbb{C}^n\{x, \ldots, x\}$ @ V, $V = \mathbb{A}\{R_1\}$, then $(\mathbb{D}_1, R_1)/(\mathbb{D}_2, R_2) = \{(\mathbb{D}_2\{\mathbb{B}^1_i\}, R_1) \mid 1 \leq i \leq n\}$, where $\mathbb{B}^1_i = \mathbb{C}^n\{V, \ldots, V, \square_i, V \ldots, V\}$, so that $\square$ fills the i-th hole of $\mathbb{C}^n$. In this case a given redex is contained in the operand part of an application.

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

$\square$

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

Lemma C.8. *A set of residuals of a redex $(\mathbb{D}_1, R_1)$ w.r.t. a redex $(\mathbb{D}_2, R_2)$ defined in C.7 satisfies the axiomatic definition A.3.* $\square$

Proof. We need to check, firstly, that every element of a set $(\mathbb{D}_1, R_1)/(\mathbb{D}_2, R_2)$ defined in C.7 is a module redex, and, secondly, that the set it satisfies the requirements of axiomatic definition A.3. Both conditions can be easily checked by considering all possible cases. $\square$

We generalize the notion of a set of residuals to a set of redexes (rather than a single redex) and to a sequence of reduction steps. The precise definitions and notations are given in section A.

According to the framework described in section A we define γ-developments for $\mathcal{C}$. First we define a non-restricted γ-development step, and then restrict it to particular kinds of modules with marked redex - those that originate from a module with a single marked non-evaluation redex. In this case the γ-development reduction has the desired properties, s.a. finiteness and standardization.

Definition C.9. • A *non-restricted γ-development step* of (D, F) is a reduction $(D, F) \overset{(\mathbb{D},R)}{\underset{n-\gamma}{\to}} (D', F')$ s.t. $D \xrightarrow{(\mathbb{D},R)} D'$, $(\mathbb{D}, R) \in F$, and

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

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

- A domain of γ-development $\mathrm{dom}(\gamma)$ is defined as follows: $(D, F) \in \mathrm{dom}(\gamma)$ if
  - either there exists a module D_0 and a non-evaluation redex $(\mathbb{D}, R)$ s.t. $(D_0, \{(\mathbb{D}, R)\}) \underset{\cup'}{\Longrightarrow}^* (D, F')$, where $\underset{\cup'}{\Longrightarrow}$ is either a $\Longrightarrow$ step or a $\underset{n-\gamma}{\Longrightarrow}$ step, and $F \subset F'$,
  - or there exists $(D', F') \in \mathrm{dom}(\gamma)$ s.t. $(D', F') \underset{n-\gamma}{\to} (D, F)$.
- If $(D, F) \in \mathrm{dom}(\gamma)$ and $(D, F) \underset{n-\gamma}{\to} (D', F')$, then we say that this reduction is a *γ-development step*, and write $(D, F) \underset{\gamma}{\to} (D', F')$ (and respectively $\underset{n-\gamma}{\Longrightarrow}$ and $\underset{n-\gamma}{\circ\!\!\longrightarrow}$). We also use the notation $\overset{\mathrm{e}}{\underset{\gamma}{\Longrightarrow}}$ for a single erasing γ-development step and $\overset{\mathrm{e}}{\underset{\gamma}{\Longrightarrow}}^*$ for a sequence of such steps.

$\square$

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

Lemma C.10. *If $(\mathbb{D}, R)$ is not a self-referential substitution redex, then $(D, F) \overset{(\mathbb{D},R)}{\underset{\gamma}{\to}} (D', F')$ implies that $F' = F/(\mathbb{D}, R)$, i.e. $(D, F) \xrightarrow{(\mathbb{D},R)} (D', F')$.* $\square$

Proof. By definition C.9 if R is a term redex, then $F' = F/(\mathbb{D}, R)$. If $R = l$ is a substitution redex and it is not self-referential, then the binding label of $(\mathbb{D}, R)$ is not l, so $\tilde{F}_l = \emptyset$, and therefore $F' = F/(\mathbb{D}, R)$. $\square$

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

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

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

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

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

Lemma C.11. *A reduction $(D, F) \underset{\gamma}{\rightarrow} (D', F')$ defined in C.9 satisfies the requirements of axiomatic definition A.11.* □

Proof. By definition $(D, \{(\mathbb{D}, R)\}) \in dom(\gamma)$ for all non-evaluation redexes $(\mathbb{D}, R)$.

If $(D, F) \xRightarrow[\gamma]{\overline{(\mathbb{D},R)}} (D, F')$, then the first part of definition A.11 is satisfied.

If $(D, F) \overset{(\mathbb{D},R)}{\underset{\gamma}{\rightarrow}} (D', F')$, then the 4 requirements in the second part of the definition are satisfied: the first two requirements of A.11 are explicitly stated in C.9. The third one holds because the set of marked redexes in the resulting pair (D', F') is defined as $F/(\mathbb{D}, R)$ for a term redex and as $F/(\mathbb{D}, R)$ with some redexes excluded for a substitution redex. Therefore $F' \subseteq F/(\mathbb{D}, R)$. The forth requirement holds by definition of $\mathrm{dom}(\gamma)$. If $(\mathbb{D}, R)$ is not a self-referential redex, then $(\mathbb{D}, R)/(\mathbb{D}, R) = \emptyset$, and the fifth requirement of the axiomatic definition holds. If $(\mathbb{D}, l)$ is a self-referential redex, then the reduction is

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

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

Note that if $(D, F) \in \mathrm{dom}(\gamma)$ and $(D, F) \overset{(\mathbb{D},R)}{\underset{n-\gamma}{\rightarrow}} (D', F')$, then $(D, F) \overset{(\mathbb{D},R)}{\underset{\gamma}{\rightarrow}} (D', F')$, and therefore if $(D_0, \{(\mathbb{D}, R)\}) \underset{\cup'}{\Rightarrow}^* (D', F')$, then $(D_0, \{(\mathbb{D}, R)\}) \underset{\cup}{\Rightarrow}^* (D', F')$ (see A.12 for definition of $\underset{\cup}{\Rightarrow}$). In the further discussion we use the notation $\underset{\cup}{\Rightarrow}^*$ in this situation.

It easily follows from definition C.9 that $(D, F) \in \mathrm{dom}(\gamma)$ if and only if there exist a module D_0 with a redex $(\mathbb{D}, R)$ s.t. $(D_0, \{(\mathbb{D}, R)\}) \underset{\cup}{\Rightarrow}^* (D', F') \underset{\gamma}{\rightarrow}^* (D, F)$.

Definition C.12. If $(D_0, \{(\mathbb{D}, R)\}) \underset{\cup}{\Rightarrow}^* (D', F') \underset{\gamma}{\rightarrow}^* (D, F)$, then $(\mathbb{D}, R)$ is called a *starting redex* of (D, F). □

A starting redex of a pair (D, F) is not uniquely defined, but this ambiguity does not affect our proofs.

Let us consider three kinds of module redexes: a term redex, a self-referential substitution redex (see definition C.6), and a substitution redex which is not self-referential.

Lemma C.13. *Let $(\mathbb{D}, R)$ be a starting redex of (D, F).*

- *If $(\mathbb{D}, R)$ is a term redex, then every $(\mathbb{D}', R') \in F$ is a term redex,*

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

$\square$

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

The first claim of the lemma easily follows from the observation that if $(\mathbb{D}, R)$ is a term redex, then for every redex $(\mathbb{D}_1, R_1)$ if $(\mathbb{D}', R') \in (\mathbb{D}, R)/(\mathbb{D}_1, R_1)$, then $(\mathbb{D}', R')$ is a term redex (see definition C.7). Therefore all residuals of a term redex w.r.t. any reduction sequence are term redexes.

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

Let $(D_0, \{(\mathbb{D}, l)\}) \underset{\cup}{\Rightarrow}^* (D', F') \underset{\gamma}{\rightarrow}^* (D, F)$, where $(\mathbb{D}, l)$ is a non-self-referential substitution redex. We show that F has only non-self-referential redexes by induction on the total number of steps in the above sequence. The base case of 0 steps satisfies the claim, since the pair $(D_0, \{(\mathbb{D}, l)\})$ has only non-self-referential marked substitution redexes.

Part 1. Suppose $(D_0, \{(\mathbb{D}, l)\}) \underset{\cup}{\Rightarrow}^* (D_1, F_1) \xrightarrow[\cup]{(\mathbb{G},R)} (D_2, F_2)$, and F_1 has only non-self-referential redexes. We want to show that F_2 also has only such redexes. Note that $D_1 \downarrow l = V \in \mathbf{Value}_{\mathcal{T}}$ by definition of a substitution redex. By the inductive hypothesis there is no $(\mathbb{D}_1, l) \in F_1$ s.t. the binding label of $(\mathbb{D}_1, l)$ is l, in other words V does not contain marked occurrences of l. By class preservation property of $\mathcal{T}$ (lemma B.24) $V \neq \mathbb{E}\{R\}$ for any term redex R and $V \neq \mathbb{E}\{l'\}$ for any l'. Therefore the evaluation step occurs in a component of D_1 other than the one bound to l. Let $\tilde{l}$ denote the label of this component.

There are several possibilities for a reduction step $\xrightarrow[\cup]{(\mathbb{G},R)}$: it may be a $\Rightarrow$, which either reduces an unmarked redex (which may be a term or a substitution redex), or a marked redex (i.e. a redex $(\mathbb{D}', l) \in F_1$). Alternatively, $\xrightarrow[\cup]{(\mathbb{G},R)}$ may be a $\underset{\gamma}{\Rightarrow}$ step. However, in all these cases we observe the following: by considering the cases in definition of a residual (definition C.7) we can see that the only way a self-referential marked redex may appear in the component bound to l in (D_2, F_2) is if a marked occurrence of l is copied to this component by a substitution. However, since the reduction step happens in a different component of D_1 (a component with a label $\tilde{l}$), a self-referential redex may not be created.

Part 2. Similarly, suppose that $(D_0, \{(\mathbb{D}, l)\}) \underset{\cup}{\Rightarrow}^* (D', F') \underset{\gamma}{\rightarrow}^* (D_1, F_1) \overset{(\mathbb{D}_1,l)}{\underset{\gamma}{\rightarrow}} (D_2, F_2)$, where F_1 contains only non-self-referential redexes (note that the step $\overset{(\mathbb{D}_1,l)}{\underset{\gamma}{\rightarrow}}$ is a part of a development sequence, therefore it reduces a substitution redex with a label l). By the inductive hypothesis F_1 does not contain self-referential redexes, so the binding label of $(\mathbb{D}_1, l)$ is not l. Therefore reducing this redex can not create a marked occurrence of l in the component bound to l, so no self-referential redexes occur in F_2. This concludes the proof of the second claim of the lemma.

Now suppose $(D_0, \{(\mathbb{D}, l)\}) \underset{\cup}{\Rightarrow}^* (D', F') \underset{\gamma}{\rightarrow}^* (D, F)$, where $(\mathbb{D}, l)$ is a self-referential redex, i.e. $D_0 \downarrow l = \mathbb{C}\{l\}$, where $\mathbb{C}$ is a non-evaluation context, and $\mathbb{D} \downarrow l = \mathbb{C}$. We prove this claim by induction, similarly to the previous claim. Base case: $(D_0, \{(\mathbb{D}, l)\})$ satisfies the claim, since $\{(\mathbb{D}, l)\}$ contains just one self-referential redex.

Part 1. Suppose that $(D_0, \{(\mathbb{D}, l)\}) \underset{\cup}{\Rightarrow}^* (D_1, F_1) \xrightarrow[\cup]{(\mathbb{G},R)} (D_2, F_2)$, and F_1 contains no more than one self-referential redex. We want to show that F_2 contains no more than one self-referential redex. If F_1 does not contain a self-referential redex, then by the same argument as in Part 1 of the proof of the second claim we show that F_2 does not have a self-referential redex. Now suppose that F_1 has a self-referential redex. Since the component bound to l is a value, the evaluation step occurs in a different component (let $\tilde{l}$ denote its label). There are three cases of evaluation step $\Longrightarrow$ reducing an unmarked redex. We show marked redexes by underlining. Note that $\mathbb{C}$ is a non-evaluation context.

$$\begin{array}{llcl}
\text{Term redex:} & [l \mapsto \mathbb{C}\{\underline{l}\}, \tilde{l} \mapsto \mathbb{E}\{R\}, \ldots] & \Longrightarrow & [l \mapsto \mathbb{C}\{\underline{l}\}, \tilde{l} \mapsto \mathbb{E}\{Q\}, \ldots], \\
\text{Substitution redexes:} & [l \mapsto \mathbb{C}\{\underline{l}\}, \tilde{l} \mapsto \mathbb{E}\{l_1\}, l_1 \mapsto V', \ldots] & \Longrightarrow & [l \mapsto \mathbb{C}\{\underline{l}\}, \tilde{l} \mapsto \mathbb{E}\{V'\}, l_1 \mapsto V', \ldots], \\
 & [l \mapsto \mathbb{C}\{\underline{l}\}, \tilde{l} \mapsto \mathbb{E}\{l\}, \ldots] & \Longrightarrow & [l \mapsto \mathbb{C}\{\underline{l}\}, \tilde{l} \mapsto \mathbb{E}\{\mathbb{C}\{\underline{l}\}\}, \ldots].
\end{array}$$

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

If the $\overset{(\mathbb{G},R)}{\underset{\cup}{\Longrightarrow}}$ step is a $\Longrightarrow$ step reducing a marked redex, we have the only possibility (note that a marked redex must be a substitution redex with the label l, and may not occur in a component bound to l since $D_1 \downarrow l \in \mathbf{Value}_{\mathcal{T}}$):

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

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

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

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

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

Part 2. Suppose that $(D_0, \{(\mathbb{D}, l)\}) \underset{\cup}{\Longrightarrow}^* (D', F') \underset{\gamma}{\longrightarrow}^* (D_1, F_1) \overset{(\mathbb{D}_1, l)}{\underset{\gamma}{\rightarrow}} (D_2, F_2)$, where F_1 contains no more than one self-referential redex. If F_1 does not contain any such redexes, then by the same argument as in part 2 of the proof of the second claim we show that F_2 does not contain self-referential redexes as well. Suppose F_1 has one self-referential redex. We have one of the following two cases:

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

In the first case a marked occurrence of l is reduced in a component other than the one bound to l. Then the original self-referential redex is still marked, and therefore F_2 still has one self-referential redex. Note that in the first case $\mathbb{A}$ may or may not be an evaluation context, so the reduction may or may not be an evaluation step. In the second case the self-referential redex is reduced (the reduction step must be a non-evaluation step, since l occurs in a non-evaluation context). Since the step is a γ-development step, l is not marked after the substitution. By the inductive hypothesis (D_1, F_1) has just one self-referential redex, so there are no other labelled occurrences of l in $\mathbb{C}\{\underline{l}\}$. Therefore there are no self-referential redexes in F_2. Combining the two cases, we conclude that F_2 has one or no self-referential redexes. This proves the third claim of the lemma. □

Lemma C.10 suggests that in the case of a term redex or a non-self-referential substitution redex the proofs of properties implying lift and project may be simplified, since γ-developments of these redexes are just extended calculus reductions. Lemma C.13 guarantees that these two cases are indeed separate from the case of a self-referential redex (and from each other), i.e. by starting from a single non-evaluation redex, which is either a term redex, or a non-self-referential substitution redex, in a lift or project diagram one can get only developments of redexes of the same kind. This allows us to consider the first two (simpler) cases of redexes separately from the third (i.e. self-referential redex) in the proofs below.

Properties. In order to prove lift and project (A.17 and A.18 respectively) for the core module calculus $\mathcal{C}$, we need to show properties required by theorems A.32 and A.33. We also prove finiteness of γ-developments and the diamond property of the evaluation relation $\Longrightarrow$, which implies its confluence. Recall that confluence of $\Longrightarrow$ is used in the proof of computational soundness from lift and project (see theorem 3.41).

Lemma C.14 (Finiteness of γ-developments). *Let $(\mathbb{D}, R)$ be a starting redex of (D, F). There is no infinite sequence $(D, F) \underset{\gamma}{\rightarrow} (D_1, F_1) \underset{\gamma}{\rightarrow} (D_2, F_2) \ldots$.* □

Proof. We have two cases:

Case 1. $(\mathbb{D}, R)$ is a term redex. By lemma C.13 all redexes in F are term redexes.

Let $D' = [l_i \overset{m}{\underset{i=1}{\mapsto}} M_i]$, and suppose that for every module component M_i we have defined a weighting I_i according to definition B.6. Then we define a *weighting* of the module to be a tuple $I = (I_1, \ldots, I_m)$. We also extend a module reduction on pairs (D, F) to a module reduction on triples (D, F, I) analogously to definition B.11.

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

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

1. I is decreasing on (D', F'),
2. and $|| (D, I) ||>|| (D', I') ||$.

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

$$(D \downarrow l_k, F \downarrow l_k) \overset{(\tilde{\mathbb{C}}, \tilde{R})}{\underset{dev}{\rightarrow}} (D' \downarrow l_k, F' \downarrow l_k).$$

Taking into account the weighting of D, we can consider the term reduction on triples (recall definition B.11):

$$(D \downarrow l_k, F \downarrow l_k, I_k) \overset{(\tilde{\mathbb{C}}, \tilde{R})}{\underset{dev}{\rightarrow}} (D' \downarrow l_k, F' \downarrow l_k, I'_k),$$

where I_k is the component of the module weighting $(I_1, \ldots, I_k, \ldots, I_m)$ corresponding to the label l_k, and I'_k is the weighting of the result of term reduction (recall definition B.7). Note that I_k is decreasing on $(D \downarrow l_k, F \downarrow l_k)$ by definition of a decreasing module weighting. Then by lemma B.12 I'_k is decreasing and $||$ $(D \downarrow l_k, I_k) ||>|| (D' \downarrow l_k, I'_k) ||$. Since the other components of the module are not changed by the reduction, the new module weighting $I' = (I_1, \ldots, I_{k-1}, I'_k, I_{k+1}, \ldots, I_m)$ is decreasing and $|| (D, I) ||>|| (D', I') ||$.

We also note the following important facts:

1. $|| (D, I) ||> 0$ for any D, I (since the measure of every module component is greater than 0),
2. for every pair (D, F) there exists a decreasing weighting I (such a weighting can be obtained as a combination of decreasing weightings of all components, which exist by lemma B.10)

Combining all the properties of decreasing module weightings that we have shown, we can apply the same argument as in lemma B.13 to show that all module developments are finite when $(\mathbb{D}, R)$ is a term redex.

Note that the erasing γ-development step $\overset{\overline{(\mathbb{D}, R)}}{\underset{\gamma}{\Longrightarrow}}$ also reduces the measure, since it "erases" the marking of a redex.

Case 2. $(\mathbb{D}, R) = (\mathbb{D}, l)$ is a substitution redex. By lemma C.13 all redexes in F are substitution redexes with the label l, and at most one of them is a self-referential redex.

We define a measure[23] $|| (D, F) ||$ to be the number of elements in F. Note that since every $(\mathbb{D}, R)$ is a module redex in D, the measure $|| (D, F) ||$ is finite for every (D, F) (every module has a finite number of redexes). Clearly $|| (D, F) ||\geq 0$ for any D, F.

[23]This measure is independent from the measure defined in case 1 of the proof.

By definition C.7 of a residual and C.9 of γ-development every substitution step reduces the number of marked redexes by one. As in the case of the term redex, an erasing step $\xRightarrow[\gamma]{\overline{(\mathbb{D},l)}}$ also reduces the number of marked redexes, and therefore the measure, by one. Therefore the γ-developments are finite in $\mathcal{C}$. $\square$

Remark C.15. It follows from the proof of lemma C.14 that for a pair (D, F) no γ-development sequence can exceed a maximum length that can be easily determined from the pair. $\square$

Lemma C.16. *The evaluation relation of the calculus $\mathcal{C}$ satisfies the following property: if $D_1 \xRightarrow{(\mathbb{G}_1,R_1)} D_2$ and $D_1 \xRightarrow{(\mathbb{G}_2,R_2)} D_3$, then there exists D_4 s.t. $D_2 \xRightarrow{(\mathbb{G}_2,R_2)/(\mathbb{G}_1,R_1)} D_4$ and $D_3 \xRightarrow{(\mathbb{G}_1,R_1)/(\mathbb{G}_2,R_2)} D_4$, provided $(\mathbb{G}_1, R_1) \neq (\mathbb{G}_2, R_2)$.* $\square$

Proof. The proof is by cases on pairs of redexes $(\mathbb{G}_1, R_1)$ and $(\mathbb{G}_2, R_2)$. Note that the two redexes occur in two different components of D_1, since they are both evaluation redexes, and therefore are independent. $\square$

Note that, as the lemma suggests, each of the sets $(\mathbb{G}_1, R_1)/(\mathbb{G}_2, R_2)$ and $(\mathbb{G}_2, R_2)/(\mathbb{G}_1, R_1)$ consists of just a single redex.

Lemma C.16 implies two important properties of $\Rightarrow_{\mathcal{C}}$ stated in theorem 2.60 and in lemma 2.62 in section 2.4.

Lemma C.17 (Confluence of $\Rightarrow_{\mathcal{C}}$). *The evaluation relation $\Rightarrow_{\mathcal{C}}$ of the calculus $\mathcal{C}$ is confluent.* $\square$

Proof. By lemma C.16. $\square$

Lemma C.18. *If $D \Rightarrow^*_{\mathcal{C}} D' = Eval(D)$, then there is no infinite sequence $D \Rightarrow_{\mathcal{C}} D_1 \Rightarrow_{\mathcal{C}} D_2 \ldots$.* $\square$

Proof. By lemma C.16 if $D \xRightarrow{S_1}{}^*_{\mathcal{C}} D'$ and $D \xRightarrow{S_2}{}^*_{\mathcal{C}} D'$, then the two sequences S_1 and S_2 have the same number of steps. By confluence (lemma C.17) for every D_i s.t. $D \Rightarrow^*_{\mathcal{C}} D_i$ there exists a sequence $D_i \Rightarrow^*_{\mathcal{C}} D'$, therefore there is no infinite evaluation sequence starting from D. $\square$

Lemma C.19. *The evaluation relation of the calculus $\mathcal{C}$ has the following property: if $(D_1, \{(\mathbb{G}, R)\}) \xRightarrow[\gamma]{(\mathbb{G},R)} (D_2, \emptyset)$, $(D_1, \{(\mathbb{G}, R)\}) \xRightarrow{(\mathbb{G}',R')} (D_3, F)$, and $(\mathbb{G}, R) \neq (\mathbb{G}', R')$, then there exists D_4 s.t. $(D_2, \emptyset) \xRightarrow{(\mathbb{G}',R')/(\mathbb{G},R)} (D_4, \emptyset)$ and $(D_3, F) \xRightarrow[\gamma]{(\mathbb{G},R)/(\mathbb{G}',R')} (D_4, \emptyset)$.* $\square$

Proof. The two given reductions imply that $D_1 \xRightarrow{(\mathbb{G},R)} D_2$ by definition A.11, and $D_1 \xRightarrow{(\mathbb{G}',R')} D_3$ by definition A.7. By lemma C.16 there exists D_4 s.t. $D_2 \xRightarrow{(\mathbb{G}',R')/(\mathbb{G},R)} D_4$ and $D_3 \xRightarrow{(\mathbb{G},R)/(\mathbb{G}',R')} D_4$. Note that $F = (\mathbb{G}, R)/(\mathbb{G}', R')$ contains just one redex (by C.16).

Note that $(\mathbb{G}, R)$ can not be a self-referential redex, since the step $(D_1, \{(\mathbb{G}, R)\}) \xRightarrow[\gamma]{(\mathbb{G},R)} (D_2, \emptyset)$ is an evaluation step, but a self-referential redex is a non-evaluation redex. Therefore (lemma C.13) the redex $(\mathbb{G}'', R'') = (\mathbb{G}, R)/(\mathbb{G}', R')$ also can not be self-referential. Since no self-referential redexes are marked, by definition A.7 of extended calculus reduction the reduction step $D_3 \xRightarrow{(\mathbb{G}'',R'')} D_4$ implies that $(D_3, \{(\mathbb{G}'', R'')\}) \xRightarrow{(\mathbb{G}'',R'')} (D_4, \emptyset)$. This, and the fact that by definition C.9 $(D_3, \{(\mathbb{G}'', R'')\}) \in \text{dom}(\gamma)$, imply by lemma C.10 that the step $(D_3, \{(\mathbb{G}'', R'')\}) \xRightarrow{(\mathbb{G}'',R'')} (D_4, \emptyset)$ is a $\underset{\gamma}{\Rightarrow}$ step. So we have shown that $(D_3, F) \underset{\gamma}{\Rightarrow} (D_4, \emptyset)$. $\square$

The condition $(\mathbb{G}, R) \neq (\mathbb{G}', R')$ is necessary. Otherwise the two resulting pairs are the same:

Lemma C.20. *If $(D_1, \{(\mathbb{G}, R)\}) \xRightarrow[\gamma]{(\mathbb{G},R)} (D_2, \emptyset)$, $(D_1, \{(\mathbb{G}, R)\}) \xRightarrow{(\mathbb{G},R)} (D_3, F)$, then $(D_3, F) = (D_2, \emptyset)$.* $\square$

Proof. The only marked redex in this case is not a self-referential one. If $(\mathbb{D}, R)$ is not self-referential, then $(\mathbb{D}, R)/(\mathbb{D}, R) = \emptyset$. $\square$

Lemmas C.19 and C.20 prove γ-confluence of evaluation for a single marked redex. Our goal is to generalize this proof arbitrary pairs $(D_1, F_1) \in dom(\gamma)$, and also to arbitrary γ-development steps (i.e. not only those reducing a redex, but also erasing steps). Below we consider two cases. If the set F_1 does not contain a self-referential redex, it is enough to show the property for one marked redex other than the redex $(\mathbb{G}, R)$ reduced in the given γ-development step. Since in this case a γ-development step is the same as the corresponding extended reduction step, we can use lemma A.40 to combine the results for each marked redex in F_1. The second case is when F_1 contains a self-referential redex. Since in this case an γ-development step is different from an extended reduction step and its result depends on the marking of the self-referential redex, we need to consider two marked redexes (the evaluation redex and the self-referential one). We also need to take into account erasing steps $\stackrel{e}{\Longrightarrow}$. The proofs are given in the following lemmas.

Lemma C.21. *Let $F_1 = \{(\mathbb{G}, R), (\mathbb{D}, \tilde{R})\}$, where $(\mathbb{D}, \tilde{R})$ is not a self-referential redex, and suppose $(D_1, F_1) \xRightarrow[\gamma]{(\mathbb{G},R)} (D_2, F_2)$, $(D_1, F_1) \xRightarrow{(\mathbb{G}',R')} (D_3, F_3)$, where $(\mathbb{G}, R) \neq (\mathbb{G}', R')$. Then there exists (D_4, F_4) s.t. $(D_2, F_2) \xRightarrow{(\mathbb{G}',R')/(\mathbb{G},R)} (D_4, F_4)$ and $(D_3, F_3) \xRightarrow[\gamma]{(\mathbb{G},R)/(\mathbb{G}',R')} (D_4, F_4)$.* $\square$

Proof. By lemma C.13 the redexes $(\mathbb{G}, R)$ and $(\mathbb{D}, \tilde{R})$ are of the same kind. Note also that $(\mathbb{G}, R)$ and $(\mathbb{G}', R')$ can not be in the same module component, since they are both evaluation redexes. Let l, l' denote the labels of the components where $(\mathbb{G}, R)$ and $(\mathbb{G}', R')$ appear, respectively. We also use notation $\tilde{l}$ for the label where $(\mathbb{D}, \tilde{R})$ appears. It may be the case that $\tilde{l}$ is the same as l or l'.

The proof is by cases of the kinds of the three redexes. Note that $(\mathbb{G}, R)$ and $(\mathbb{D}, \tilde{R})$ are of the same kind by lemma C.13.

- *$(\mathbb{G}, R)$, $(\mathbb{D}, \tilde{R})$, and $(\mathbb{G}', R')$ are all term redexes.* If $\tilde{l} \neq l$, $\tilde{l} \neq l'$, then all three redexes occur in different components of the module, so the claim obviously holds. Suppose $\tilde{l} = l'$, i.e. R' and $\tilde{R}$ occur in the same component. If these redexes are independent (i.e. one does not contain the other), then the claim of the lemma clearly holds. It may not be the case that $\tilde{R}$ contains R', since $(\mathbb{G}', R')$ is an evaluation redex, and therefore cannot be contained in another redex. If R' contains $\tilde{R}$, then $R' = (\lambda x.\mathbb{B}\{\tilde{R}\}) \mathbin{@} V$ or $R' = (\lambda x.M) \mathbin{@} \lambda y.\mathbb{B}\{\tilde{R}\}$. In the first case, assuming $\mathbb{B}' = \mathbb{B}[x := V]$ and $\tilde{R}' = \tilde{R}'[x := V]$, we have:

$$
\begin{aligned}
&[l \mapsto \mathbb{G}\{R\}, l' \mapsto \mathbb{G}'\{(\lambda x.\mathbb{B}\{\tilde{R}\}) \mathbin{@} V\}, \ldots] \underset{\gamma}{\Rightarrow} [l \mapsto \mathbb{G}\{Q\}, l' \mapsto \mathbb{G}'\{(\lambda x.\mathbb{B}\{\tilde{R}\}) \mathbin{@} V\}, \ldots] \Rightarrow \\
&[l \mapsto \mathbb{G}\{Q\}, l' \mapsto \mathbb{G}'\{\mathbb{B}'\{\tilde{R}'\}\}, \ldots], \\
&[l \mapsto \mathbb{G}\{R\}, l' \mapsto \mathbb{G}'\{(\lambda x.\mathbb{B}\{\tilde{R}\}) \mathbin{@} V\}, \ldots] \Rightarrow [l \mapsto \mathbb{G}\{R\}, l' \mapsto \mathbb{G}'\{\mathbb{B}'\{\tilde{R}'\}\}, \ldots] \underset{\gamma}{\Rightarrow} \\
&[l \mapsto \mathbb{G}\{Q\}, l' \mapsto \mathbb{G}'\{\mathbb{B}'\{\tilde{R}'\}\}, \ldots].
\end{aligned}
$$

  The other case is similar, with the only difference that the redex $\tilde{R}$ gets duplicated rather than changed. The case when $\tilde{l} = l$ is similar to the case when $\tilde{l} = l'$, since the marked redexes are term redexes, and therefore γ-developments are just extended reductions.

- *$(\mathbb{G}, R)$ and $(\mathbb{D}, \tilde{R})$ are term redexes, $(\mathbb{G}', R')$ is a substitution redex.* In this case R' is an unmarked label (since F_1 contains term redexes, it can not contain a label). Let $R' = l_1$. The label l_1 is bound to a value, so $l_1 \neq l$, $l_1 \neq l'$. If $l_1 \neq \tilde{l}$, then the three redexes are independent, and the claim of the lemma clearly holds. If $l_1 = \tilde{l}$, then reducing the redex $(\mathbb{G}', l_1)$ duplicates $\tilde{R}$. However, it is clear that $(\mathbb{D}, \tilde{R})$ has the same two residuals on both reduction paths.

- *$(\mathbb{G}, R)$ and $(\mathbb{D}, \tilde{R})$ are substitution redexes, $(\mathbb{G}', R')$ is a term redex.* Then $R = \tilde{R} = l_1$. Since F_1 does not contain self-referential redexes, $l_1 \neq l$, $l_1 \neq \tilde{l}$. Since l_1 is bound to a value, $l_1 \neq l'$.

If $\tilde{l} \neq l$, $\tilde{l} \neq l'$, then all 4 labels are distinct, and the claim of the lemma clearly holds. If $\tilde{l} = l'$, i.e. of l_1 of the marked redex $(\mathbb{D}, l_1)$ occurs in the same component as R', and the module can be of one of the following forms (as before, we show marked labels by underlining):

$$\begin{array}{ll} [l \mapsto \mathbb{E}\{\underline{l_1}\}, l' \mapsto \mathbb{A}\{R', \underline{l_1}\}, l_1 \mapsto V, \ldots], & \\ {[l \mapsto \mathbb{E}\{\underline{l_1}\}, l' \mapsto \mathbb{E}'\{(\lambda x.\mathbb{B}\{\underline{l_1}\}) \mathbin{@} V\}, l_1 \mapsto V, \ldots]}, & \text{or} \\ {[l \mapsto \mathbb{E}\{\underline{l_1}\}, l' \mapsto \mathbb{E}'\{(\lambda x.M) \mathbin{@} \lambda y.\mathbb{B}\{\underline{l_1}\}\}, l_1 \mapsto V, \ldots]}, & \end{array}$$

where in the first case $\mathbb{A}\{\square, \underline{l_1}\}$ is an evaluation context[24]. It is easy to see that the claim holds in all three cases (in the last case the redex $(\mathbb{D}, l_1)$ gets duplicated, but its residuals are the same on both reduction paths). The remaining case is when $\tilde{l} = l$, then the module is of the form

$$[l \mapsto \mathbb{A}\{\underline{l_1}, \underline{l_1}\}, l' \mapsto \mathbb{E}\{R'\}, l_1 \mapsto V, \ldots].$$

Again, clearly both reduction paths lead to the same module with the same residual of $(\mathbb{D}, l_1)$.

- $(\mathbb{G}, R)$*,* $(\mathbb{D}, \tilde{R})$*, and* $(\mathbb{G}', R')$ *are all substitution redexes.* Let $R = l_1$ (hence $\tilde{R} = l_1$) and $R' = l_2$. There are 3 possibilities:

  1. $l_1 \neq l_2$,
  2. $l_1 = l_2$, $(\mathbb{G}', l_1) \notin F_1$, i.e. it is not marked,
  3. $l_1 = l_2$, $(\mathbb{G}', l_1) \in F_1$, i.e. it is marked.

  Let us consider all three possibilities:

  1. If $l_1 \neq l_2$, then the 4 labels l, l', l_1, l_2 are all pairwise distinct (in addition to the given conditions, we notice that l_1, l_2 are bound to values, and l, l' contain evaluation redexes, so none of the first two labels equals to any of the other two). We have the following 4 possibilities for $\tilde{l}$:
     If $\tilde{l} = l$, then the module is of the form

     $$[l_1 \mapsto V_1, l_2 \mapsto V_2, l \mapsto \mathbb{A}\{\underline{l_1}, \underline{l_1}\}, l' \mapsto \mathbb{E}\{l_2\}, \ldots],$$

     where, as above, $\mathbb{A}\{\square, \underline{l_1}\}$ is an evaluation context One can check that reducing $(\mathbb{G}, l_1)$ and $(\mathbb{G}', l_2)$ in any order leads to a module with the same residual of $(\mathbb{D}, l_1)$.
     If $\tilde{l} = l'$, then the case is similar to the case when $\tilde{l} = l$.
     If $\tilde{l} = l_2$, then the marked redex $(\mathbb{D}, l_1)$ gets duplicated:

     $$\begin{array}{ll} [l_1 \mapsto V_1, l_2 \mapsto \lambda x.\mathbb{C}\{\underline{l_1}\}, l \mapsto \mathbb{E}\{\underline{l_1}\}, l' \mapsto \mathbb{E}'\{l_2\}, \ldots] & \underset{\gamma}{\Longrightarrow} \\ {[l_1 \mapsto V_1, l_2 \mapsto \lambda x.\mathbb{C}\{\underline{l_1}\}, l \mapsto \mathbb{E}\{V_1\}, l' \mapsto \mathbb{E}'\{l_2\}, \ldots]} & \Longrightarrow \\ {[l_1 \mapsto V_1, l_2 \mapsto \lambda x.\mathbb{C}\{\underline{l_1}\}, l \mapsto \mathbb{E}\{V_1\}, l' \mapsto \mathbb{E}'\{\lambda x.\mathbb{C}\{\underline{l_1}\}\}, \ldots]}, & \\ {[l_1 \mapsto V_1, l_2 \mapsto \lambda x.\mathbb{C}\{\underline{l_1}\}, l \mapsto \mathbb{E}\{\underline{l_1}\}, l' \mapsto \mathbb{E}'\{l_2\}, \ldots]} & \Longrightarrow \\ {[l_1 \mapsto V_1, l_2 \mapsto \lambda x.\mathbb{C}\{\underline{l_1}\}, l \mapsto \mathbb{E}\{\underline{l_1}\}, l' \mapsto \mathbb{E}'\{\lambda x.\mathbb{C}\{\underline{l_1}\}\}, \ldots]} & \underset{\gamma}{\Longrightarrow} \\ {[l_1 \mapsto V_1, l_2 \mapsto \lambda x.\mathbb{C}\{\underline{l_1}\}, l \mapsto \mathbb{E}\{V_1\}, l' \mapsto \mathbb{E}'\{\lambda x.\mathbb{C}\{\underline{l_1}\}\}, \ldots]}. & \end{array}$$

     The remaining possibility is that $\tilde{l} \notin \{l, l', l_2\}$. Then all five labels are distinct, and the claim of the lemma clearly holds. Note that the case $\tilde{l} = l_1$ is impossible because $(\mathbb{D}, l_1) \in F_1$, but F_1 does not contain self-referential redexes.

[24] Here and below we specify only one order in which two (or more) subterms occur in a context if all the cases are analogous. For instance, here we do not consider the case when the evaluation context is $\mathbb{A}\{\underline{l_1}, R'\}$, because it is completely analogous to the case we have considered

2. If $l_1 = l_2$, and $(\mathbb{G}', l_1)$ is not marked, then we have one of the following choices:

$$\begin{aligned}
\tilde{l} \notin \{l, l'\}: &\quad [l_1 \mapsto V_1, l \mapsto \mathbb{E}\{\underline{l_1}\}, l' \mapsto \mathbb{E}'\{l_1\}, \tilde{l} \mapsto \mathbb{C}\{\underline{l_1}\}, \ldots], \\
\tilde{l} = l: &\quad [l_1 \mapsto V_1, l \mapsto \mathbb{A}\{\underline{l_1}, \underline{l_1}\}, l' \mapsto \mathbb{E}'\{l_1\}, \ldots], \\
\tilde{l} = l': &\quad [l_1 \mapsto V_1, l \mapsto \mathbb{E}\{\underline{l_1}\}, l' \mapsto \mathbb{A}\{l_1, \underline{l_1}\}, \ldots].
\end{aligned}$$

In all three cases $(\mathbb{D}, l_1)$ has the same residual on both reduction paths.

3. If $l_1 = l_2$, and $(\mathbb{G}', l_1)$ is marked, then the case is completely analogous to case 2. Even though the occurrence of l_1 in the component bound to l' is marked, this does not affect the non-development reduction, because this reduction does not remove the mark of a label regardless of whether the target of the substitution is marked or unmarked.

□

The remaining case is when the γ-development step erases a redex.

Lemma C.22. *Suppose F_1 does not contain a self-referential redex. If $(D_1, F_1) \overset{\overline{(\mathbb{D},R)}}{\underset{\gamma}{\Longrightarrow}} (D_2, F_2)$, $(D_1, F_1) \overset{(\mathbb{G},\tilde{R})}{\Longrightarrow} (D_3, F_3)$, then there exists (D_4, F_4) s.t. $(D_2, F_2) \overset{(\mathbb{G},\tilde{R})}{\Longrightarrow} (D_4, F_4)$ and $(D_3, F_3) \overset{\mathrm{e}}{\underset{\gamma}{\Rightarrow}}{}^{*} (D_4, F_4)$.* □

Proof. By definition of an erasing γ-development step $(\mathbb{D}, R)$ is marked. Therefore if $(\mathbb{G}, \tilde{R})$ is marked, by lemma C.13 it must be of the same kind as $(\mathbb{D}, R)$. We also observe that the residuals of both redexes do not depend on what other redexes are marked in the modules. We have the following cases:

1. $(\mathbb{D}, R)$ is a term redex. Then:

   - If $(\mathbb{G}, \tilde{R})$ is an unmarked term redex, then either the two redexes are independent, in which case the claim clearly holds, or $(\mathbb{D}, R)$ is contained in $(\mathbb{G}, \tilde{R})$ (note that $(\mathbb{G}, \tilde{R})$ is an evaluation redex, and therefore can not be contained in $(\mathbb{D}, R)$). In the latter case suppose $(\mathbb{D}, R)$ is contained in the operand of $(\mathbb{G}, \tilde{R})$:

   $$\begin{aligned}
   &[l \mapsto \mathbb{E}\{(\lambda x.\mathbb{A}\{x, \ldots, x\}) @ \lambda y.\mathbb{B}\{\underline{R}\}\}, \ldots] \Longrightarrow [l \mapsto \mathbb{E}\{\mathbb{A}\{\lambda y.\mathbb{B}\{\underline{R}\}, \ldots, \lambda y.\mathbb{B}\{\underline{R}\}\}\}, \ldots] \overset{\mathrm{e}}{\underset{\gamma}{\Rightarrow}}{}^{*} \\
   &[l \mapsto \mathbb{E}\{\mathbb{A}\{\lambda y.\mathbb{B}\{R\}, \ldots, \lambda y.\mathbb{B}\{R\}\}\}, \ldots], \\
   &[l \mapsto \mathbb{E}\{(\lambda x.\mathbb{A}\{x, \ldots, x\}) @ \lambda y.\mathbb{B}\{\underline{R}\}\}, \ldots] \overset{\mathrm{e}}{\underset{\gamma}{\Rightarrow}} [l \mapsto \mathbb{E}\{(\lambda x.\mathbb{A}\{x, \ldots, x\}) @ \lambda y.\mathbb{B}\{R\}\}, \ldots] \Longrightarrow \\
   &[l \mapsto \mathbb{E}\{\mathbb{A}\{\lambda y.\mathbb{B}\{R\}, \ldots, \lambda y.\mathbb{B}\{R\}\}\}, \ldots].
   \end{aligned}$$

   The case when $(\mathbb{D}, R)$ is contained in the operator of $(\mathbb{G}, \tilde{R})$ is similar, but no redex duplication happens in this case.

   - If $(\mathbb{G}, \tilde{R})$ is a marked term redex, then the case is analogous to the previous one, but additionally we need to consider the following case: if $(\mathbb{G}, \tilde{R}) = (\mathbb{D}, R)$, then $(D_3, F_3) = (D_4, F_4)$, since the residual of $(\mathbb{G}, \tilde{R})$ is unmarked in D_3.

   - If $(\mathbb{G}, \tilde{R}) = (\mathbb{G}, l)$ is a substitution redex, then it is unmarked. Since $(\mathbb{G}, l)$ is an evaluation redex, the label l can not be contained in $(\mathbb{D}, R)$. Suppose $(\mathbb{D}, R)$ is contained in the value bound to l:

   $$\begin{aligned}
   &[l \mapsto \lambda x.\mathbb{C}\{\underline{R}\}, l' \mapsto \mathbb{E}\{l\}, \ldots] \Longrightarrow [l \mapsto \lambda x.\mathbb{C}\{\underline{R}\}, l' \mapsto \mathbb{E}\{\lambda x.\mathbb{C}\{\underline{R}\}\}, \ldots] \overset{\mathrm{e}}{\underset{\gamma}{\Rightarrow}}{}^{*} \\
   &[l \mapsto \lambda x.\mathbb{C}\{R\}, l' \mapsto \mathbb{E}\{\lambda x.\mathbb{C}\{R\}\}, \ldots], \\
   &[l \mapsto \lambda x.\mathbb{C}\{\underline{R}\}, l' \mapsto \mathbb{E}\{l\}, \ldots] \overset{\mathrm{e}}{\underset{\gamma}{\Rightarrow}} [l \mapsto \lambda x.\mathbb{C}\{R\}, l' \mapsto \mathbb{E}\{l\}, \ldots] \Longrightarrow \\
   &[l \mapsto \lambda x.\mathbb{C}\{R\}, l' \mapsto \mathbb{E}\{\lambda x.\mathbb{C}\{R\}\}, \ldots].
   \end{aligned}$$

   In the remaining case $(\mathbb{D}, R)$ is independent from $(\mathbb{G}, l)$, and the claim of the lemma clearly holds.

2. $(\mathbb{D}, R) = (\mathbb{D}, l)$ is a substitution. We have the subcases:

   - $(\mathbb{G}, \tilde{R})$ is a term redex. It can not be contained in the value bound to l.

     If $(\mathbb{G}, \tilde{R})$ contains the marked occurrence of l of $(\mathbb{D}, l)$, then this occurrence may be duplicated, in which case the resulting erasing γ-development remove all residuals of the marked l from the set of marked redexes. The case similar to the cases above where duplication of the marked redex occurs.

     Otherwise $(\mathbb{D}, l)$ is independent from $(\mathbb{G}, \tilde{R})$, no duplication occurs, and the claim clearly holds.
   - $(\mathbb{G}, \tilde{R}) = (\mathbb{G}, l')$, $l' \neq l$. If $(\mathbb{D}, l)$ is not contained in the value bound to l', then the claim clearly holds. Otherwise the marked occurrence of l gets duplicated by the evaluation redex, and both marked copies of l need to be "erased".
   - $(\mathbb{G}, \tilde{R}) = (\mathbb{G}, l)$. Note that the value bound to l does not contain a marked occurrence of l, since by the condition of the lemma there is no self-referential redex among marked redexes. In this case either $(\mathbb{G}, l)$ is independent of $(\mathbb{D}, l)$ and the claim clearly holds, or $(\mathbb{G}, l) = (\mathbb{D}, l)$:

$$\begin{array}{lll} [l \mapsto V, l' \mapsto \mathbb{E}\{\underline{l}\}, \ldots] & \Rightarrow & [l \mapsto V, l' \mapsto \mathbb{E}\{V\}, \ldots], \\ [l \mapsto V, l' \mapsto \mathbb{E}\{\underline{l}\}, \ldots] & \underset{\gamma}{\overset{\mathrm{e}}{\Rightarrow}} & [l \mapsto V, l' \mapsto \mathbb{E}\{l\}, \ldots] \quad \Rightarrow \\ [l \mapsto V, l' \mapsto \mathbb{E}\{V\}, \ldots]. & & \end{array}$$

$\square$

Lemma C.23. *Suppose F_1 does not contain a self-referential redex. If $(D_1, F_1) \underset{\gamma}{\overset{(\mathbb{G},R)}{\Longrightarrow}} (D_2, F_2)$, $(D_1, F_1) \overset{(\mathbb{G}',R')}{\Longrightarrow} (D_3, F_3)$, then:*

- *If $(\mathbb{G}, R) \neq (\mathbb{G}', R')$, then there exists (D_4, F_4) s.t. $(D_2, F_2) \overset{(\mathbb{G}',R')/(\mathbb{G},R)}{\Longrightarrow} (D_4, F_4)$ and $(D_3, F_3) \underset{\gamma}{\overset{(\mathbb{G},R)/(\mathbb{G}',R')}{\Longrightarrow}} (D_4, F_4)$;*
- *If $(\mathbb{G}, R) = (\mathbb{G}', R')$, then $(D_2, F_2) = (D_3, F_3)$.*

$\square$

Proof. The first part by lemmas C.19 and C.21 and by corollary A.42. The second part by lemma C.20 and by corollary A.42. $\square$

The case when F_1 has a self-referential redex is slightly more complicated: we can not apply corollary A.42, as we did for the case above when F_1 does not have self-referential redexes, because in the case of a self-referential redex γ-development reduction is different from the extended reduction:

$$\begin{array}{lll} [l_1 \mapsto \lambda x.\underline{l_1}, l_2 \mapsto \underline{l_1}] & \Rightarrow & [l_1 \mapsto \lambda x.\underline{l_1}, l_2 \mapsto \lambda x.\underline{l_1}], \quad \text{but} \\ [l_1 \mapsto \lambda x.\underline{l_1}, l_2 \mapsto \underline{l_1}] & \underset{\gamma}{\Rightarrow} & [l_1 \mapsto \lambda x.\underline{l_1}, l_2 \mapsto \lambda x.l_1]. \end{array}$$

The label l_1 in the second component of the result is marked in the first case, but not in the second. However, it turns out that for both reductions a residual set of a redex does not depend on the marking of any redexes in the module other than the self-referential one, as shown in the following two lemmas. Note that the second lemma considers an arbitrary γ-development step, not just an evaluation step, and thus will be used for the proofs of elementary lift and project diagrams later in this section.

Lemma C.24. *Suppose $(D, F) \overset{(\mathbb{G},R)}{\Longrightarrow} (D', F')$, $(\mathbb{D}, l) \in F$ is a self-referential redex, and $(D, F) \in dom(\gamma)$. Let $(\mathbb{D}', l)$ be a non-self-referential redex in D s.t. $(\mathbb{D}', l) \notin F$. Then $(D, F \cup \{(\mathbb{D}', l)\}) \overset{(\mathbb{G},R)}{\Longrightarrow} (D', F' \cup (\mathbb{D}', l)/(\mathbb{G}, R))$.* $\square$

Proof. This follows directly from definition A.7 of extended reduction. Note that the lemma holds for both marked and unmarked $(\mathbb{G}, R)$. ☐

Lemma C.25. *Let* $(D_1, \{(\mathbb{D}, l), (\mathbb{D}', l)\}) \overset{(\mathbb{D},l)}{\underset{\gamma}{\to}} (D_2, F_2)$ *and* $(D_1, \{(\mathbb{D}, l), (\mathbb{D}'', l)\}) \overset{(\mathbb{D},l)}{\underset{\gamma}{\to}} (D_2, F_2')$*, and suppose that no more than one of the three redexes is self-referential.* $(D_1, \{(\mathbb{D}, l), (\mathbb{D}', l), (\mathbb{D}'', l)\}) \in dom(\gamma)$*. Then* $(D_1, \{(\mathbb{D}, l), (\mathbb{D}', l), (\mathbb{D}'', l)\}) \overset{(\mathbb{D},l)}{\underset{\gamma}{\to}} (D_2, F_2 \cup F_2')$*.* ☐

Proof. Since at most one of the three redexes may be a self-referential redex, we consider the following 3 cases:

- None of the three redexes is self-referential. Let $\tilde{\mathbb{D}}$ be a 4-hole context s.t. the first hole contains the value of the component bound to l, then

$$\begin{array}{rccc} \text{Given} & \tilde{\mathbb{D}}\{V, \underline{l}, \underline{l}, l\} & \overset{(\mathbb{D},l)}{\underset{\gamma}{\to}} & \tilde{\mathbb{D}}\{V, V, \underline{l}, l\} \\ \text{and} & \tilde{\mathbb{D}}\{V, \underline{l}, l, \underline{l}\} & \overset{(\mathbb{D},l)}{\underset{\gamma}{\to}} & \tilde{\mathbb{D}}\{V, V, l, \underline{l}\}, \\ \text{we have} & \tilde{\mathbb{D}}\{V, \underline{l}, \underline{l}, \underline{l}\} & \overset{(\mathbb{D},l)}{\underset{\gamma}{\to}} & \tilde{\mathbb{D}}\{V, V, \underline{l}, \underline{l}\}. \end{array}$$

  Here the first reduction corresponds is the reduction when only $(\mathbb{D}', l)$, but not $(\mathbb{D}'', l)$, is marked, and the second reduction is the case when $(\mathbb{D}'', l)$, but not $(\mathbb{D}', l)$, is marked.

- $(\mathbb{D}', l)$ is self-referential (note that this is completely analogous to the case when $(\mathbb{D}'', l)$ is self-referential, so we show only one of these cases). In this case no other marked redex occurs in the component bound to l. The redexes $(\mathbb{D}, l)$ and $(\mathbb{D}'', l)$ may occur in the same or in different components. Let us show the case when they occur in the same component, the other case is analogous:

$$\begin{array}{rccc} \text{Given} & [l \mapsto \lambda x.\mathbb{C}\{\underline{l}\}, l' \mapsto \mathbb{A}\{\underline{l}, l\}, \ldots] & \overset{(\mathbb{D},l)}{\underset{\gamma}{\to}} & [l \mapsto \lambda x.\mathbb{C}\{\underline{l}\}, l' \mapsto \mathbb{A}\{\lambda x.\mathbb{C}\{l\}, l\}, \ldots] \\ \text{and} & [l \mapsto \lambda x.\mathbb{C}\{l\}, l' \mapsto \mathbb{A}\{\underline{l}, \underline{l}\}, \ldots] & \overset{(\mathbb{D},l)}{\underset{\gamma}{\to}} & [l \mapsto \lambda x.\mathbb{C}\{l\}, l' \mapsto \mathbb{A}\{\lambda x.\mathbb{C}\{l\}, \underline{l}\}, \ldots], \\ \text{we have} & [l \mapsto \lambda x.\mathbb{C}\{\underline{l}\}, l' \mapsto \mathbb{A}\{\underline{l}, \underline{l}\}, \ldots] & \overset{(\mathbb{D},l)}{\underset{\gamma}{\to}} & [l \mapsto \lambda x.\mathbb{C}\{\underline{l}\}, l' \mapsto \mathbb{A}\{\lambda x.\mathbb{C}\{l\}, \underline{l}\}, \ldots]. \end{array}$$

- $(\mathbb{D}, l)$ is self-referential (in this case the γ-development step is a non-evaluation step). As in the previous case, the other two redexes may occur in the same component or in two different ones. As before, we show only the former case, the latter one is analogous.

$$\begin{array}{rccc} \text{Given} & [l \mapsto \lambda x.\mathbb{C}\{\underline{l}\}, l' \mapsto \mathbb{A}\{\underline{l}, l\}, \ldots] & \overset{(\mathbb{D},l)}{\underset{\gamma}{\to}} & [l \mapsto \lambda x.\mathbb{C}\{\lambda x.\mathbb{C}\{l\}\}, l' \mapsto \mathbb{A}\{\underline{l}, l\}, \ldots] \\ \text{and} & [l \mapsto \lambda x.\mathbb{C}\{\underline{l}\}, l' \mapsto \mathbb{A}\{l, \underline{l}\}, \ldots] & \overset{(\mathbb{D},l)}{\underset{\gamma}{\to}} & [l \mapsto \lambda x.\mathbb{C}\{\lambda x.\mathbb{C}\{l\}\}, l' \mapsto \mathbb{A}\{l, \underline{l}\}, \ldots], \\ \text{we have} & [l \mapsto \lambda x.\mathbb{C}\{\underline{l}\}, l' \mapsto \mathbb{A}\{\underline{l}, \underline{l}\}, \ldots] & \overset{(\mathbb{D},l)}{\underset{\gamma}{\to}} & [l \mapsto \lambda x.\mathbb{C}\{\lambda x.\mathbb{C}\{l\}\}, l' \mapsto \mathbb{A}\{\underline{l}, \underline{l}\}, \ldots]. \end{array}$$

☐

We can generalize the previous lemma to the case of arbitrary sets of marked redexes in a module:

Corollary C.26. *Let* $(D_1, F_1) \overset{(\mathbb{D},l)}{\underset{\gamma}{\to}} (D_2, F_2)$ *and* $(D_1, \{(\mathbb{D}, l), (\mathbb{D}', l)\}) \overset{(\mathbb{D},l)}{\underset{\gamma}{\to}} (D_2, F_2')$*, where* $(\mathbb{D}', l) \notin F_1$*. If* $(D_1, F_1 \cup \{(\mathbb{D}', l)\}) \in dom(\gamma)$*, then* $(D_1, F_1 \cup \{(\mathbb{D}', l)\}) \overset{(\mathbb{D},l)}{\underset{\gamma}{\to}} (D_2, F_2 \cup F_2')$*.* ☐

Proof. If $F_1 = \emptyset$, the claim trivially holds. If F_1 consists of one redex, the claim holds by lemma C.25.

Suppose F_1 consists of n redexes. By the condition $(D_1, F_1 \cup \{(\mathbb{D}', l)\}) \in dom(\gamma)$ at most one of the marked redexes is self-referential. By lemma C.25 the set of residuals of the marked redex $(\mathbb{D}', l)$ with respect to the redex $(\mathbb{D}, l)$ does not depend on whether any other redexes in the module are marked. Similarly the set of residuals of any redex in F_1 does not depend on whether $(\mathbb{D}', l)$ is marked. Therefore the resulting set of marked redexes is the union $F_2 \cup F_2'$, where $F_2 = F_1/(\mathbb{D}, l)$, and $F_2' = (\mathbb{D}', l)/(\mathbb{D}, l)$. □

The following two lemmas deal with the cases when the self-referential redex is the only marked one (in addition to the redex being reduced) and when there is one more marked redex. Lemmas C.29 and C.30 generalize these two cases to an arbitrary set of marked redexes containing a self-referential redex.

Lemma C.27. *If* $(D_1, \{(\mathbb{D}, l_1), (\mathbb{G}, l_1)\}) \xRightarrow[\gamma]{(\mathbb{G}, l_1)} (D_2, F_2)$*,* $(D_1, \{(\mathbb{D}, l_1), (\mathbb{G}, l_1)\}) \xRightarrow{(\mathbb{G}', R')} (D_3, F_3)$*, where* $(\mathbb{D}, l_1)$ *is a self-referential redex and* $(\mathbb{G}, l_1) \neq (\mathbb{G}', R')$*, then there exists* (D_4, F_4) *s.t.* $(D_2, F_2) \xRightarrow{(\mathbb{G}', R')/(\mathbb{G}, l_1)} (D_4, F_4)$ *and* $(D_3, F_3) \xRightarrow[\gamma]{(\mathbb{G}, l_1)/(\mathbb{G}', R')} (D_4, F_4)$*.* □

Proof. The proof is by cases on $(\mathbb{G}', R')$. Note that both $(\mathbb{G}', R')$ and $(\mathbb{G}, l_1)$ are evaluation redexes, and therefore they occur in two different components, and none of these components is bound to l_1.

- $(\mathbb{G}', R')$ is a term redex:

  $$D_1 \quad = \quad [l_1 \mapsto \lambda x.\mathbb{C}\{\underline{l_1}\}, l \mapsto \mathbb{E}\{\underline{l_1}\}, l' \mapsto \mathbb{E}'\{R'\}, \ldots]$$

  The two redexes reduced are independent, and on both reduction paths we arrive at the module:

  $$D_4 \quad = \quad [l_1 \mapsto \lambda x.\mathbb{C}\{\underline{l_1}\}, l \mapsto \mathbb{E}\{\lambda x.\mathbb{C}\{l_1\}\}, l' \mapsto \mathbb{E}'\{Q'\}, \ldots]$$

  So the self-referential redex has only one residual - itself.

- $(\mathbb{G}', R')$ is a substitution redex, $R' = l_2 \neq l_1$. Analogous to the previous case.

- $(\mathbb{G}', R')$ is a substitution redex, $R' = l_1$. Note that l_1 in $(\mathbb{G}', R')$ is not marked, since, by the conditions of the lemma, only the other two redexes are marked. Then

  $$\begin{array}{rcl} D_1 & = & [l_1 \mapsto \lambda x.\mathbb{C}\{\underline{l_1}\}, l \mapsto \mathbb{E}\{\underline{l_1}\}, l' \mapsto \mathbb{E}'\{l_1\}, \ldots], \\ D_4 & = & [l_1 \mapsto \lambda x.\mathbb{C}\{\underline{l_1}\}, l \mapsto \mathbb{E}\{\lambda x.\mathbb{C}\{l_1\}\}, l' \mapsto \mathbb{E}'\{\lambda x.\mathbb{C}\{\underline{l_1}\}\}, \ldots]. \end{array}$$

□

The case when the two evaluation steps reduce the same redex needs to be treated separately, since the label on one of the redexes gets preserved, and the resulting γ-development step erases this redex:

Lemma C.28. *Suppose* $(D_1, \{(\mathbb{D}, l), (\mathbb{G}, l)\}) \xRightarrow[\gamma]{(\mathbb{G}, l)} (D_2, F_2)$ *and* $(D_1, \{(\mathbb{D}, l), (\mathbb{G}, l)\}) \xRightarrow{(\mathbb{G}, l)} (D_3, F_3)$*, where* $(\mathbb{D}, l)$ *is self-referential. Then* $(D_3, F_3) \xRightarrow[\gamma]{\overline{(\mathbb{G}, l)}} (D_2, F_2)$*.* □

Proof. The following reduction sequences prove the claim. Here $\overset{e}{\underset{\gamma}{\Rightarrow}}$ denotes the erasing γ-development step.

$$\begin{array}{ll} [l \mapsto \lambda x.\mathbb{C}\{\underline{l}\}, l' \mapsto \mathbb{E}\{\underline{l}\}, \ldots] & \underset{\gamma}{\Rightarrow} \\ [l \mapsto \lambda x.\mathbb{C}\{\underline{l}\}, l' \mapsto \mathbb{E}\{\lambda x.\mathbb{C}\{l\}\}, \ldots], & \\ [l \mapsto \lambda x.\mathbb{C}\{\underline{l}\}, l' \mapsto \mathbb{E}\{\underline{l}\}, \ldots] & \Rightarrow \\ [l \mapsto \lambda x.\mathbb{C}\{\underline{l}\}, l' \mapsto \mathbb{E}\{\lambda x.\mathbb{C}\{\underline{l}\}\}, \ldots] & \overset{e}{\underset{\gamma}{\Rightarrow}} \\ [l \mapsto \lambda x.\mathbb{C}\{\underline{l}\}, l' \mapsto \mathbb{E}\{\lambda x.\mathbb{C}\{l\}\}, \ldots]. & \end{array}$$

□

Lemma C.29. *Suppose F_1 has a self-referential redex. If* $(D_1, F_1) \overset{(\mathbb{G},l)}{\underset{\gamma}{\Longrightarrow}} (D_2, F_2)$, $(D_1, F_1) \overset{(\mathbb{G}',R')}{\Longrightarrow} (D_3, F_3)$, *then there exists* (D_4, F_4) *s.t.* $(D_2, F_2) \overset{(\mathbb{G}',R')/(\mathbb{G},l)}{\Longrightarrow} (D_4, F_4)$ *and* $(D_3, F_3) \overset{(\mathbb{G},l)/(\mathbb{G}',R')}{\underset{\gamma}{\Longrightarrow}} (D_4, F_4)$. □

Proof. The proof is by induction on the number of redexes in F_1. If F_1 consists of just the self-referential redex in addition to the redex $(\mathbb{G}, l)$, then by lemma C.27 the claim of the lemma holds.

Suppose the lemma holds for an n-redex subset of F_1, let F'_1 denote this subset. Let $(\mathbb{D}', l) \in F_1$, $(\mathbb{D}', l) \notin F'_1$. Let $(\mathbb{D}, l)$ be the self-referential redex of F'. Note that $(\mathbb{D}', l)$ is not self-referential, since $(D_1, F_1) \in dom(\gamma)$, and therefore by lemma C.13 F_1 has no more than one self-referential redex.

By the inductive hypothesis $(D_1, F'_1) \overset{(\mathbb{G},l)}{\underset{\gamma}{\Longrightarrow}} (D_2, F'_2)$, $(D_1, F'_1) \overset{(\mathbb{G}',R')}{\Longrightarrow} (D_3, F'_3)$ implies that there exists (D_4, F'_4) s.t. $(D_2, F'_2) \overset{(\mathbb{G}',R')/(\mathbb{G},l)}{\Longrightarrow} (D_4, F'_4)$, $(D_3, F'_3) \overset{(\mathbb{G},l)/(\mathbb{G}',R')}{\underset{\gamma}{\Longrightarrow}} (D_4, F'_4)$.

By lemma C.21 $(D_1, \{(\mathbb{D}, l), (\mathbb{G}, l)\}) \overset{(\mathbb{G},l)}{\underset{\gamma}{\Longrightarrow}} (D_2, \tilde{F}_2)$ and $(D_1, \{(\mathbb{D}, l), (\mathbb{G}, l)\}) \overset{(\mathbb{G}',R')}{\Longrightarrow} (D_3, \tilde{F}_3)$ implies that there exists $(D_4, \tilde{F}_4)$ s.t. $(D_2, \tilde{F}_2) \overset{(\mathbb{G}',R')/(\mathbb{G},l)}{\Longrightarrow} (D_4, \tilde{F}_4)$, $(D_3, \tilde{F}_3) \overset{(\mathbb{G},l)/(\mathbb{G}',R')}{\underset{\gamma}{\Longrightarrow}} (D_4, \tilde{F}_4)$.

We combine the two diagrams by lemma C.24 for the $\Longrightarrow$ steps and by lemma C.25 for the $\underset{\gamma}{\Longrightarrow}$ steps: $(D_1, F'_1 \cup \{(\mathbb{D}, l)\}) \overset{(\mathbb{G},l)}{\underset{\gamma}{\Longrightarrow}} (D_2, F'_2 \cup \tilde{F}_2)$ and $(D_1, F'_1 \cup \{(\mathbb{D}, l)\}) \overset{(\mathbb{G}',R')}{\Longrightarrow} (D_3, F'_3 \cup \tilde{F}_3)$ imply that $(D_2, F'_2 \cup \tilde{F}_2) \overset{(\mathbb{G}',R')/(\mathbb{G},l)}{\Longrightarrow} (D_4, F'_4 \cup \tilde{F}_4)$, $(D_3, F'_3 \cup \tilde{F}_3) \overset{(\mathbb{G},l)/(\mathbb{G}',R')}{\underset{\gamma}{\Longrightarrow}} (D_4, F'_4 \cup \tilde{F}_4)$. This shows that the claim of the lemma holds for a set of $n + 1$ redexes of D_1. □

Lemma C.30. *Suppose F_1 has a self-referential redex. If* $(D_1, F_1) \overset{\overline{(\mathbb{D},l)}}{\underset{\gamma}{\Longrightarrow}} (D_2, F_2)$ *and* $(D_1, F_1) \overset{(\mathbb{G},\tilde{R})}{\Longrightarrow} (D_3, F_3)$, *then there exists* (D_4, F_4) *s.t.* $(D_2, F_2) \overset{(\mathbb{G},\tilde{R})}{\Longrightarrow} (D_4, F_4)$ *and* $(D_3, F_3) \overset{e}{\underset{\gamma}{\Longrightarrow}}{}^{*} (D_4, F_4)$. □

Proof. We observe that the only redex in F_1 that affects the reductions is the self-referential redex. The marking of all other redexes (i.e. those not equal to $(\mathbb{D}, l)$ and $(\mathbb{G}, \tilde{R})$) is preserved by the reductions, so we are not considering any other marked redexes.

The subcases are as follows:

1. If $(\mathbb{G}, \tilde{R})$ is a term redex, then the marked occurrence of l corresponding to $(\mathbb{D}, l)$ is either independent of $(\mathbb{G}, \tilde{R})$, or is contained in $(\mathbb{G}, \tilde{R})$. In the former case the claim of the lemma clearly holds. In the latter case the marked occurrence of l may be duplicated by the reduction of $(\mathbb{G}, \tilde{R})$, in which case all the copies of l need to be "erased" by the $\overset{e}{\underset{\gamma}{\Longrightarrow}}{}^{*}$ sequence.

2. If $(\mathbb{G}, \tilde{R}) = (\mathbb{G}, l')$ is a substitution redex and $l' \neq l$, then the only dependency between the two redexes is when $(\mathbb{D}, l)$ occurs in the value bound to l'. In this case:

$$\begin{array}{ll}
[l \mapsto \lambda x.\mathbb{C}\{\underline{l}\}, l' \mapsto \lambda y.\mathbb{B}\{\underline{l}\}, l'' \mapsto \mathbb{E}\{l'\}, \ldots] & \Longrightarrow \\
[l \mapsto \lambda x.\mathbb{C}\{\underline{l}\}, l' \mapsto \lambda y.\mathbb{B}\{\underline{l}\}, l'' \mapsto \mathbb{E}\{\lambda y.\mathbb{B}\{\underline{l}\}\}, \ldots] & \overset{e}{\underset{\gamma}{\Longrightarrow}}{}^{*} \\
[l \mapsto \lambda x.\mathbb{C}\{\underline{l}\}, l' \mapsto \lambda y.\mathbb{B}\{l\}, l'' \mapsto \mathbb{E}\{\lambda y.\mathbb{B}\{l\}\}, \ldots], & \\
[l \mapsto \lambda x.\mathbb{C}\{\underline{l}\}, l' \mapsto \lambda y.\mathbb{B}\{\underline{l}\}, l'' \mapsto \mathbb{E}\{l'\}, \ldots] & \overset{e}{\underset{\gamma}{\Longrightarrow}} \\
[l \mapsto \lambda x.\mathbb{C}\{\underline{l}\}, l' \mapsto \lambda y.\mathbb{B}\{l\}, l'' \mapsto \mathbb{E}\{l'\}, \ldots] & \Longrightarrow \\
[l \mapsto \lambda x.\mathbb{C}\{\underline{l}\}, l' \mapsto \lambda y.\mathbb{B}\{l\}, l'' \mapsto \mathbb{E}\{\lambda y.\mathbb{B}\{l\}\}, \ldots]. &
\end{array}$$

   The other cases are trivial.

3. If $(\mathbb{G}, \tilde{R}) = (\mathbb{G}, l)$, then we have the following possibilities:

- $(\mathbb{G}, l)$ is not marked. In this case the two redexes are independent (since $(\mathbb{D}, l)$ is not self-referential, and therefore can not occur in the component bound to l), and the claim of the lemma holds.
- $(\mathbb{G}, l)$ is marked, $(\mathbb{G}, l) \neq (\mathbb{D}, l)$. The two redexes also must be independent, and the case is similar to previous one.
- $(\mathbb{G}, l) = (\mathbb{D}, l)$. This case is considered in lemma C.28.

□

Lemma C.31 (γ-confluence of Evaluation of $\mathcal{C}$). *The calculus $\mathcal{C}$ has strong γ-confluence of evaluation (property A.22), i.e. if $(D_1, F_1) \underset{\gamma}{\Longrightarrow} (D_2, F_2)$, $(D_1, F_1) \Longrightarrow (D_3, F_3)$, then there exists (D_4, F_4) s.t. $(D_2, F_2) \Longrightarrow (D_4, F_4)$ and $(D_3, F_3) \underset{\gamma}{\Longrightarrow} (D_4, F_4)$.* □

Proof. If F_1 does not contain a self-referential redex, then the lemma follows by lemma C.23, otherwise by lemmas C.29 and C.30. □

Elementary Lift and Project Diagrams. Properties A.25 and A.26 are another two properties required for the proof of lift and project. To prove these properties, we use an approach similar to that of the proof of γ-confluence of evaluation above.

The following property of terms is used in the further proofs.

Lemma C.32. *If $M = \mathbb{E}\{l\}$, then there is no such context $\mathbb{C}$ and redex R that $M = \mathbb{C}\{R\}$ and $R = \mathbb{A}\{l\}$, i.e. such that R contains the occurrence of l.* □

Proof. Suppose such $\mathbb{C}$ and R exist. If $M = \mathbb{E}\{l\}$ and l is contained in a redex R, then $M = \mathbb{C}\{\mathbb{A}\{l\}\}$, and $\mathbb{E} = \mathbb{C}\{\mathbb{A}\}$. Therefore $\mathbb{C} = \mathbb{E}' \in \textbf{EvalContext}_{\mathcal{T}}$, and $M = \mathbb{E}'\{R\}$, which contradicts the class preservation lemma B.24. □

The following 4 lemmas show all cases for the proof of property A.25 (elementary project diagram).

Lemma C.33. *Suppose $(D_1, \{(\mathbb{D}, R), (\mathbb{D}', R')\}) \underset{\gamma}{\circ\!\xrightarrow{(\mathbb{D},R)}} (D_2, F_2)$ and $(D_1, \{(\mathbb{D}, R), (\mathbb{D}', R')\}) \xRightarrow{(\mathbb{G},\tilde{R})} (D_3, F_3)$, where $(\mathbb{D}, R)$ and $(\mathbb{D}', R')$ are not self-referential. Then there exists (D_4, F_4) s.t. $(D_2, F_2) \xRightarrow{(\mathbb{G},\tilde{R})/(\mathbb{D},R)} (D_4, F_4)$ and $(D_3, F_3) \underset{\gamma}{\overset{(\mathbb{D},R)/(\mathbb{G},\tilde{R})}{\longrightarrow^*}} (D_4, F_4)$. Moreover, if the sequence $\underset{\gamma}{\overset{(\mathbb{D},R)/(\mathbb{G},\tilde{R})}{\longrightarrow^*}}$ consists of more than one step, then for any such reduction $(D_3, F_3) \underset{\gamma}{\overset{(\mathbb{D},R)/(\mathbb{G},\tilde{R})}{\longrightarrow^*}} (D_4, F_4)$.* □

Proof. Let l, l', $\tilde{l}$ be the labels of the components containing $(\mathbb{D}, R), (\mathbb{D}', R')$, and $(\mathbb{G}, \tilde{R})$ respectively. We have the following cases:

1. All three redexes are term redexes. When all three redexes are independent, the claim obviously holds. It also clearly holds in the case when $(\mathbb{D}, R)$ and $(\mathbb{G}, \tilde{R})$ are in different terms, even if $(\mathbb{D}', R')$ is in the same term with one of the former redexes. If $(\mathbb{G}, \tilde{R})$ contains $(\mathbb{D}, R)$, then we apply lemma B.18 to the term where both of these redexes occur. This is the only case when $(\mathbb{D}, R)$ may be duplicated, i.e. the sequence $\underset{\gamma}{\overset{(\mathbb{D},R)/(\mathbb{G},\tilde{R})}{\longrightarrow^*}}$ may consist of more than one step. By lemma B.18 the order of steps in this sequence does not matter.

   There are no other cases, since $(\mathbb{D}, R)$ can not contain $(\mathbb{G}, \tilde{R})$ because the latter is an evaluation redex.

2. The marked redexes are term redexes, and the evaluation redex is a substitution redex. Let $\tilde{R} = l_1$. Note that $l_1 \neq \tilde{l}$, since $(\mathbb{G}, l_1)$ is an evaluation redex, and therefore can not be self-referential. We have the following subcases:

(a) Among the 4 labels l, l', l_1, and $\tilde{l}$, no two are equal. In this case the claim of the lemma clearly holds.

(b) $l_1 \notin \{l, l'\}$. It may be the case that $l = \tilde{l} \neq l'$, $l' = \tilde{l} \neq l$, or $l = l' = \tilde{l}$. Assume the third possibility. By lemma C.32 the redexes R and R' can not contain the redex occurrence of l_1. We have the following options:

$$\begin{array}{lll} \text{I.} & [\tilde{l} \mapsto \mathbb{A}\{l_1, R, R'\}, l_1 \mapsto V, \ldots] & R,\ R' \text{ independent} \\ \text{II.} & [\tilde{l} \mapsto \mathbb{A}\{l_1, \mathbb{B}\{R'\}\}, l_1 \mapsto V, \ldots] & R = \mathbb{B}\{R'\} \\ \text{III.} & [\tilde{l} \mapsto \mathbb{A}\{l_1, \mathbb{B}\{R\}\}, l_1 \mapsto V, \ldots] & R' = \mathbb{B}\{R\} \end{array}$$

By lemma B.17 in all three cases the context containing l is still an evaluation context after the reduction of R. The substitution does not affect the reduction of R, and from the properties of the term calculus it follows that in all three cases the residual(s) of R' will be the same regardless of which of the redexes has been reduced first: l_1 or R. No duplication happens in this case, so the resulting γ-development sequence consists of a single step.

The other two cases ($l = \tilde{l} \neq l'$ and $l' = \tilde{l} \neq l$) are analogous to the case we have considered, but simpler.

(c) $\tilde{l} \notin \{l, l'\}$. As in the previous case, let us consider the subcase $l = l' = l_1$ in detail, the other two subcases ($l = l_1 \neq l'$ and $l' = l_1 \neq l$) are similar, but easier.

If $l = l' = l_1$, then both R and R' occur in the value being substituted by the evaluation redex. Since this case is more complex than the previous one, we show the actual reductions rather than just the initial modules. As above, we have the following possibilities for mutual positions of R and R':

I. R and R' are independent:

$$\begin{array}{ll} [\tilde{l} \mapsto \mathbb{E}\{l_1\}, l_1 \mapsto \lambda x.\mathbb{C}\{R, R'\}, \ldots] & \underset{\gamma}{\circ\!\!\longrightarrow} \\ [\tilde{l} \mapsto \mathbb{E}\{l_1\}, l_1 \mapsto \lambda x.\mathbb{C}\{Q, R'\}, \ldots] & \Longrightarrow \\ [\tilde{l} \mapsto \mathbb{E}\{\lambda x.\mathbb{C}\{Q, R'\}\}, l_1 \mapsto \lambda x.\mathbb{C}\{Q, R'\}, \ldots], & \\ [\tilde{l} \mapsto \mathbb{E}\{l_1\}, l_1 \mapsto \lambda x.\mathbb{C}\{R, R'\}, \ldots] & \Longrightarrow \\ [\tilde{l} \mapsto \mathbb{E}\{\lambda x.\mathbb{C}\{R, R'\}\}, l_1 \mapsto \lambda x.\mathbb{C}\{R, R'\}, \ldots] & \underset{\gamma}{\longrightarrow^*} \\ [\tilde{l} \mapsto \mathbb{E}\{\lambda x.\mathbb{C}\{Q, R'\}\}, l_1 \mapsto \lambda x.\mathbb{C}\{Q, R'\}, \ldots]. & \end{array}$$

The order of reduction of the copies of R in the sequence $\underset{\gamma}{\longrightarrow^*}$ does not matter. II. R contains R'. If R' is contained in the operand of R, we have the following:

$$\begin{array}{ll} [\tilde{l} \mapsto \mathbb{E}\{l_1\}, l_1 \mapsto \lambda x.\mathbb{C}\{\lambda y.\mathbb{A}\{y, \ldots, y\} \mathbin{@} \lambda z.\mathbb{B}\{R'\}\}, \ldots] & \underset{\gamma}{\circ\!\!\longrightarrow} \\ [\tilde{l} \mapsto \mathbb{E}\{l_1\}, l_1 \mapsto \lambda x.\mathbb{C}\{\mathbb{A}\{\lambda z.\mathbb{B}\{R'\}, \ldots, \lambda z.\mathbb{B}\{R'\}\}\}, \ldots] & \Longrightarrow \\ [\tilde{l} \mapsto \mathbb{E}\{\lambda x.\mathbb{C}\{\mathbb{A}\{\lambda z.\mathbb{B}\{R'\}, \ldots, \lambda z.\mathbb{B}\{R'\}\}\}\}, l_1 \mapsto \lambda x.\mathbb{C}\{\mathbb{A}\{\lambda z.\mathbb{B}\{R'\}, \ldots, \lambda z.\mathbb{B}\{R'\}\}\}, \ldots], & \\ [\tilde{l} \mapsto \mathbb{E}\{l_1\}, l_1 \mapsto \lambda x.\mathbb{C}\{\lambda y.\mathbb{A}\{y, \ldots, y\} \mathbin{@} \lambda z.\mathbb{B}\{R'\}\}, \ldots] & \Longrightarrow \\ [\tilde{l} \mapsto \mathbb{E}\{\lambda x.\mathbb{C}\{\lambda y.\mathbb{A}\{y, \ldots, y\} \mathbin{@} \lambda z.\mathbb{B}\{R'\}\}\}, l_1 \mapsto \lambda x.\mathbb{C}\{\lambda y.\mathbb{A}\{y, \ldots, y\} \mathbin{@} \lambda z.\mathbb{B}\{R'\}\}, \ldots] & \underset{\gamma}{\longrightarrow^*} \\ [\tilde{l} \mapsto \mathbb{E}\{\lambda x.\mathbb{C}\{\mathbb{A}\{\lambda z.\mathbb{B}\{R'\}, \ldots, \lambda z.\mathbb{B}\{R'\}\}\}\}, l_1 \mapsto \lambda x.\mathbb{C}\{\mathbb{A}\{\lambda z.\mathbb{B}\{R'\}, \ldots, \lambda z.\mathbb{B}\{R'\}\}\}, \ldots]. & \end{array}$$

In this case R' gets duplicated by both redexes, but its residuals are the same on both reduction paths. The order of reduction of the two copies of the redex $\lambda y.\mathbb{A}\{y, \ldots, y\} \mathbin{@} \lambda z.\mathbb{B}\{R'\}$ does not matter.

If R' is contained in the operator of R, then it gets changed, rather than duplicated, by reduction of R. The redex R gets duplicated by the substitution. It is clear that R' has the same residuals on both reduction paths and that the order of reduction of the two copies of R does not matter.

III. R' contains R. Let $R' = \mathbb{A}\{R\}$. It does not matter if R is contained in the operand or in the operator of R', because R' is not reduced in these reductions. We have:

$$\begin{array}{ll}
[\tilde{l} \mapsto \mathbb{E}\{l_1\}, l_1 \mapsto \lambda y.\mathbb{C}\{\mathbb{A}\{R\}\}, \ldots] & \underset{\gamma}{\circ\!\!\longrightarrow} \\
[\tilde{l} \mapsto \mathbb{E}\{l_1\}, l_1 \mapsto \lambda y.\mathbb{C}\{\mathbb{A}\{Q\}\}, \ldots] & \Longrightarrow \\
[\tilde{l} \mapsto \mathbb{E}\{\lambda y.\mathbb{C}\{\mathbb{A}\{Q\}\}\}, l_1 \mapsto \lambda y.\mathbb{C}\{\mathbb{A}\{Q\}\}, \ldots]. & \\
[\tilde{l} \mapsto \mathbb{E}\{l_1\}, l_1 \mapsto \lambda y.\mathbb{C}\{\mathbb{A}\{R\}\}, \ldots] & \Longrightarrow \\
[\tilde{l} \mapsto \mathbb{E}\{\lambda y.\mathbb{C}\{\mathbb{A}\{R\}\}\}, l_1 \mapsto \lambda y.\mathbb{C}\{\mathbb{A}\{R\}\}, \ldots] & \underset{\gamma}{\longrightarrow^*} \\
[\tilde{l} \mapsto \mathbb{E}\{\lambda y.\mathbb{C}\{\mathbb{A}\{Q\}\}\}, l_1 \mapsto \lambda y.\mathbb{C}\{\mathbb{A}\{Q\}\}, \ldots] &
\end{array}$$

The redex R' has two residuals of the form $\mathbb{A}\{Q\}$ on both reduction paths. The copies of the redex R can be reduced in any order.

(d) $l = \tilde{l}$, $l' = l_1$. As in case (b), by lemma C.32 the redex occurrence of l_1 in the component bound to $\tilde{l}$ is independent from the redex R. The initial module is

$$[\tilde{l} \mapsto \mathbb{A}\{l_1, R\}, l_1 \mapsto \lambda x.\mathbb{C}\{R'\}, \ldots].$$

Here R' gets duplicated by the substitution redex, but clearly has the same residuals on both reduction paths. By lemma B.17 $\mathbb{A}\{l_1, Q\} \in \mathbf{EvalContext}_{\mathcal{T}}$.

(e) $l = l_1$, $l' = \tilde{l}$. The initial module is

$$[\tilde{l} \mapsto \mathbb{A}\{l_1, R'\}, l_1 \mapsto \lambda x.\mathbb{C}\{R\}, \ldots].$$

Here R, not R', gets duplicated by the substitution. The two residuals of R can be reduced in any order. R' has only one residual in this case.

3. The marked redexes are substitution redexes, and the evaluation redex is a term redex. Let $R = l_1$, then $R' = l_1$ by lemma C.13. By the condition of this lemma the marked redexes are not self-referential, so $l_1 \neq l$, $l_1 \neq l'$. Also $l_1 \neq \tilde{l}$, since l_1 is bound to a value, and $\tilde{l}$ is bound to an evaluatable term. As in the previous case, we have several subcases of mutual positions of the labels. Let us consider the case when $l = l' = \tilde{l}$, i.e. all three redexes occur in the same term. There may be several possibilities:

- $\tilde{R}$ and the two occurrences of l_1 in the component bound to $\tilde{l}$ are independent, i.e. the initial module is:

$$[\tilde{l} \mapsto \mathbb{A}\{\tilde{R}, \underline{l_1}, \underline{l_1}\}, l_1 \mapsto V, \ldots]$$

Without loss of generality, suppose that the first occurrence of $\underline{l_1}$ is the redex R. By lemma B.16 $\mathbb{A}\{\square, \underline{l_1}, \underline{l_1}\} \in \mathbf{EvalContext}_{\mathcal{T}}$ implies that $\mathbb{A}\{\square, V, \underline{l_1}\} \in \mathbf{EvalContext}_{\mathcal{T}}$, therefore the reduction of $(\mathbb{G}, \tilde{R})/(\mathbb{D}, l_1)$ is indeed an evaluation step. It is clear that both reduction paths result in the same module with the same (single) residual of R'.

- Both R and R' occur in the operand part of $\tilde{R}$:

$$\begin{array}{ll}
[\tilde{l} \mapsto \mathbb{E}\{(\lambda x.\mathbb{A}\{x, \ldots, x\}) \mathbin{@} \lambda y.\mathbb{B}\{\underline{l_1}, \underline{l_1}\}\}, l_1 \mapsto V, \ldots] & \underset{\gamma}{\circ\!\!\longrightarrow} \\
[\tilde{l} \mapsto \mathbb{E}\{(\lambda x.\mathbb{A}\{x, \ldots, x\}) \mathbin{@} \lambda y.\mathbb{B}\{V, \underline{l_1}\}\}, l_1 \mapsto V, \ldots] & \Longrightarrow \\
[\tilde{l} \mapsto \mathbb{E}\{\mathbb{A}\{\lambda y.\mathbb{B}\{V, \underline{l_1}\}, \ldots, \lambda y.\mathbb{B}\{V, \underline{l_1}\}\}\}, l_1 \mapsto V, \ldots], & \\
[\tilde{l} \mapsto \mathbb{E}\{(\lambda x.\mathbb{A}\{x, \ldots, x\}) \mathbin{@} \lambda y.\mathbb{B}\{\underline{l_1}, \underline{l_1}\}\}, l_1 \mapsto V, \ldots] & \Longrightarrow \\
[\tilde{l} \mapsto \mathbb{E}\{\mathbb{A}\{\lambda y.\mathbb{B}\{\underline{l_1}, \underline{l_1}\}, \ldots, \lambda y.\mathbb{B}\{\underline{l_1}, \underline{l_1}\}\}\}, l_1 \mapsto V, \ldots] & \underset{\gamma}{\longrightarrow^*} \\
[\tilde{l} \mapsto \mathbb{E}\{\mathbb{A}\{\lambda y.\mathbb{B}\{V, \underline{l_1}\}, \ldots, \lambda y.\mathbb{B}\{V, \underline{l_1}\}\}\}, l_1 \mapsto V, \ldots]. &
\end{array}$$

The copies of $R = \underline{l}$ can be reduced in any order in the sequence $\underset{\gamma}{\longrightarrow^*}$.

- The other cases (when both R and R' occur in the operator part of $\tilde{R}$, or when one of them occurs in the operator and the other in the operand) are similar.

The other cases ($l = l' \neq \tilde{l}$, $l = \tilde{l} \neq l'$, $l' = \tilde{l} \neq l$, and all the three labels distinct) are similar to the one we have considered, but easier.

4. All three redexes are substitution redexes, where $\tilde{R} = l_2 \neq l_1$. In this case $\tilde{l} \neq l_2$, since $\tilde{R}$ is an evaluation redex. As in case 3, $l_1 \notin \{l, l', \tilde{l}\}$. We have the following cases:

   (a) All 5 labels are different. In this case the lemma trivially holds.

   (b) $l = l' = \tilde{l}$. The initial module is

   $$[\tilde{l} \mapsto \mathbb{A}\{l_2, \underline{l_1}, \underline{l_1}\}, l_1 \mapsto V_1, l_2 \mapsto V_2, \ldots].$$

   By lemma B.16 $\mathbb{A}\{\square, V_1, \underline{l_1}\} \in \mathbf{EvalContext}_{\mathcal{T}}$, so the residual of the evaluation redex is indeed an evaluation redex. In this case no redexes get duplicated, and the claim clearly holds.

   (c) $l = l' = l_2$. Assuming (without loss of generality) that the first marked occurrence of l_1 corresponds to the redex R, the initial and the final modules in this case are:

   $$[\tilde{l} \mapsto \mathbb{E}\{l_2\}, l_1 \mapsto V_1, l_2 \mapsto \lambda x.\mathbb{C}\{\underline{l_1}, \underline{l_1}\}, \ldots],$$
   $$[\tilde{l} \mapsto \mathbb{E}\{\lambda x.\mathbb{C}\{V_1, \underline{l_1}\}\}, l_1 \mapsto V_1, l_2 \mapsto \lambda x.\mathbb{C}\{V_1, \underline{l_1}\}, \ldots].$$

   The two copies of R can be reduced in any order.

   (d) $l = l'$, $l \neq \tilde{l}$, $l \neq l_2$. The claim clearly holds.

   (e) $l = \tilde{l}$, $l' = l_2$. The initial and the final modules are:

   $$[\tilde{l} \mapsto \mathbb{A}\{l_2, \underline{l_1}\}, l_1 \mapsto V_1, l_2 \mapsto \lambda x.\mathbb{B}\{\underline{l_1}\}, \ldots],$$
   $$[\tilde{l} \mapsto \mathbb{A}\{\lambda x.\mathbb{B}\{\underline{l_1}\}, V_1\}, l_1 \mapsto V_1, l_2 \mapsto \lambda x.\mathbb{B}\{\underline{l_1}\}, \ldots].$$

   Again we use lemma B.16 to show that the residual of the evaluation redex is an evaluation redex. Redex R' gets duplicated.

   (f) $l = \tilde{l}$, $l' \neq l_2$, $l' \neq \tilde{l}$. Similar to the previous case.

   (g) $l = l_2$, $l' = \tilde{l}$. The initial and the final modules are:

   $$[\tilde{l} \mapsto \mathbb{A}\{l_2, \underline{l_1}\}, l_1 \mapsto V_1, l_2 \mapsto \lambda x.\mathbb{B}\{\underline{l_1}\}, \ldots],$$
   $$[\tilde{l} \mapsto \mathbb{A}\{\lambda x.\mathbb{B}\{V_1\}, \underline{l_1}\}, l_1 \mapsto V_1, l_2 \mapsto \lambda x.\mathbb{B}\{V_1\}, \ldots].$$

   Here the redex R gets duplicated. Its two residuals can be reduced in any order. The redex R' has just a single residual on both reduction paths.

   (h) $l = l_2$, $l' \neq l_2$, $l' \neq \tilde{l}$. Similar to the previous case.

5. All three redexes are substitution redexes, $\tilde{R} = l_1$. Note that $\tilde{R}$ is not marked since by the condition of the lemma the only two marked redexes are R and R'. Similarly to the previous case, $l_1 \notin \{l, l', \tilde{l}\}$. Since none of the redexes R and R' occur in the component bound to l_1 (i.e. in the value being substituted), no redex duplication is possible, so the claim of the lemma clearly holds in this case.

$\square$

Lemma C.34. *Suppose* $(D_1, \{(\mathbb{D}, l), (\mathbb{D}', l)\}) \overset{(\mathbb{D},l)}{\underset{\gamma}{\circ\!\!\longrightarrow}} (D_2, F_2)$ *and* $(D_1, \{(\mathbb{D}, l), (\mathbb{D}', l)\}) \xRightarrow{(\mathbb{G},\tilde{R})} (D_3, F_3)$*, where* $(\mathbb{D}', l)$ *is self-referential. Then there exists* (D_4, F_4) *s.t.* $(D_2, F_2) \xRightarrow{(\mathbb{G},\tilde{R})/(\mathbb{D},l)} (D_4, F_4)$ *and* $(D_3, F_3) \overset{(\mathbb{D},l)/(\mathbb{G},\tilde{R})}{\underset{\gamma}{\longrightarrow^*}}$

(D_4, F_4). *If the sequence* $\overset{(\mathbb{D},l)/(\mathbb{G},\tilde{R})}{\underset{\gamma}{\longrightarrow^*}}$ *consists of more than one step, then for any order of steps in the reduction sequence we have* $(D_3, F_3) \overset{(\mathbb{D},l)/(\mathbb{G},\tilde{R})}{\underset{\gamma}{\longrightarrow^*}} (D_4, F_4)$. □

Proof. Let $l, l_1, \tilde{l}$ be the labels of components where $(\mathbb{D}', l)$, $(\mathbb{D}, l)$, and $(\mathbb{G}, \tilde{R})$ occur, respectively. Note that $l \neq \tilde{l}$ by class preservation, since $(\mathbb{D}', l)$ is a self-referential redex, and therefore the component bound to l is a value. Also $l_1 \neq l$ by lemma C.13, since $(M_1, F_1) \in dom(\gamma)$, and therefore F_1 contains at most one self-referential redex. We have the following cases:

1. $\tilde{R}$ is a term redex, suppose $\tilde{R} \rightsquigarrow_{\mathcal{T}} \tilde{Q}$. If $l_1 \neq \tilde{l}$, we have the following initial and final modules:

$$\begin{array}{l}[l \mapsto \lambda x.\mathbb{C}\{\underline{l}\}, l_1 \mapsto \mathbb{A}\{\underline{l}\}, \tilde{l} \mapsto \mathbb{E}\{\tilde{R}\}, \ldots],\\ [l \mapsto \lambda x.\mathbb{C}\{\underline{l}\}, l_1 \mapsto \mathbb{A}\{\lambda x.\mathbb{C}\{l\}\}, \tilde{l} \mapsto \mathbb{E}\{\tilde{Q}\}, \ldots].\end{array}$$

   The self-referential redex has just one marked residual – itself.

   If $l_1 = \tilde{l}$, i.e. $\tilde{R}$ and the non-self-referential marked occurrence of l are in the same component, we have one of the following:

   - $\tilde{R}$ and the marked occurrence of l are independent. This case is similar to the case when $l_1 \neq \tilde{l}$. By lemma B.16 the residual of the evaluation redex is itself evaluation redex.
   - The marked l occurs in the operator of $\tilde{R}$ ($\tilde{R}$ is an application since it contains l). Without loss of generality assume that $\underline{l}$ occurs in the first hole of the multi-hole context $\mathbb{A}$ below. The initial and the final modules are:

   $$\begin{array}{l}[l \mapsto \lambda x.\mathbb{C}\{\underline{l}\}, \tilde{l} \mapsto \mathbb{E}\{(\lambda y.\mathbb{A}\{\underline{l}, y, \ldots, y\}) \mathbin{@} V\}, \ldots],\\ [l \mapsto \lambda x.\mathbb{C}\{\underline{l}\}, \tilde{l} \mapsto \mathbb{E}\{\mathbb{A}\{\lambda x.\mathbb{C}\{l\}, V, \ldots, V\}\}, \ldots].\end{array}$$

     As before, the self-referential redex has a single marked residual – itself.
   - The marked l occurs in the operand of $\tilde{R}$. The initial and the final modules are:

   $$\begin{array}{l}[l \mapsto \lambda x.\mathbb{C}\{\underline{l}\}, \tilde{l} \mapsto \mathbb{E}\{(\lambda y.\mathbb{A}\{y, \ldots, y\}) \mathbin{@} \lambda z.\mathbb{B}\{\underline{l}\}\}, \ldots],\\ [l \mapsto \lambda x.\mathbb{C}\{\underline{l}\}, \tilde{l} \mapsto \mathbb{E}\{\mathbb{A}\{\lambda z.\mathbb{B}\{\lambda x.\mathbb{C}\{l\}\}, \ldots, \lambda z.\mathbb{B}\{\lambda x.\mathbb{C}\{l\}\}\}\}, \ldots].\end{array}$$

     Here the non-self-referential marked occurrence of l gets duplicated by the term redex. The resulting module is the same, regardless of which redex (the marked substitution or the application) is performed first and regardless of the order in which the residuals of $(\mathbb{D}, l)$ w.r.t. $(\mathbb{G}, \tilde{R})$ are reduced. The self-referential redex does not have any marked residuals besides itself.

2. $\tilde{R} = l_2 \neq l$ is a substitution redex. Note that $\tilde{l} \neq l_2$, since $\tilde{R}$ is an evaluation redex. We have the following possibilities:

   - $l_1 \neq l_2$, $l_1 \neq \tilde{l}$. It is easy to check that the claim of the lemma holds in this case.
   - $l_1 = l_2$. The reductions on both paths lead to the same module with the same residuals of the self-referential redex. The two residuals of $(\mathbb{D}, l)$ in the γ-development reduction can be reduced in any order.

   $$\begin{array}{lr}[l \mapsto \lambda x.\mathbb{C}\{\underline{l}\}, \tilde{l} \mapsto \mathbb{E}\{l_1\}, l_1 \mapsto \lambda y.\mathbb{A}\{\underline{l}\}, \ldots] & \underset{\gamma}{\circ\!\!\longrightarrow}\\ [l \mapsto \lambda x.\mathbb{C}\{\underline{l}\}, \tilde{l} \mapsto \mathbb{E}\{l_1\}, l_1 \mapsto \lambda y.\mathbb{A}\{\lambda x.\mathbb{C}\{l\}\}, \ldots] & \Longrightarrow\\ [l \mapsto \lambda x.\mathbb{C}\{\underline{l}\}, \tilde{l} \mapsto \mathbb{E}\{\lambda y.\mathbb{A}\{\lambda x.\mathbb{C}\{l\}\}\}, l_1 \mapsto \lambda y.\mathbb{A}\{\lambda x.\mathbb{C}\{l\}\}, \ldots], & \\ [l \mapsto \lambda x.\mathbb{C}\{\underline{l}\}, \tilde{l} \mapsto \mathbb{E}\{l_1\}, l_1 \mapsto \lambda y.\mathbb{A}\{\underline{l}\}, \ldots] & \Longrightarrow\\ [l \mapsto \lambda x.\mathbb{C}\{\underline{l}\}, \tilde{l} \mapsto \mathbb{E}\{\lambda y.\mathbb{A}\{\underline{l}\}\}, l_1 \mapsto \lambda y.\mathbb{A}\{\underline{l}\}, \ldots] & \underset{\gamma}{\longrightarrow^*}\\ [l \mapsto \lambda x.\mathbb{C}\{\underline{l}\}, \tilde{l} \mapsto \mathbb{E}\{\lambda y.\mathbb{A}\{\lambda x.\mathbb{C}\{l\}\}\}, l_1 \mapsto \lambda y.\mathbb{A}\{\lambda x.\mathbb{C}\{l\}\}, \ldots]. & \end{array}$$

- $l_1 = \tilde{l}$. In this case the initial and the final modules are as shown below. By lemma B.16 $\mathbb{A}\{\Box, \lambda x.\mathbb{C}\{l\}\} \in \mathbf{EvalContext}_{\mathcal{T}}$.

$$[l \mapsto \lambda x.\mathbb{C}\{\underline{l}\}, \tilde{l} \mapsto \mathbb{A}\{l_2, \underline{l}\}, l_2 \mapsto V, \ldots],$$
$$[l \mapsto \lambda x.\mathbb{C}\{\underline{l}\}, \tilde{l} \mapsto \mathbb{A}\{V, \lambda x.\mathbb{C}\{l\}\}, l_2 \mapsto V, \ldots].$$

3. $\tilde{R} = l$ is a substitution redex. This occurrence of l is not marked, since by the condition of the lemma only the other two redexes are marked.

   If $l_1 = \tilde{l}$, the initial and the final modules are as follows:

$$[l \mapsto \lambda x.\mathbb{C}\{\underline{l}\}, \tilde{l} \mapsto \mathbb{A}\{l, \underline{l}\}, \ldots],$$
$$[l \mapsto \lambda x.\mathbb{C}\{\underline{l}\}, \tilde{l} \mapsto \mathbb{A}\{\lambda x.\mathbb{C}\{\underline{l}\}, \lambda x.\mathbb{C}\{l\}\}, \ldots].$$

   The residual of the self-referential redex obtained by the $\Longrightarrow$ step is marked, since the marking is removed only by a development step that substitutes into a marked occurrence of the same label. Thus in this case the self-referential redex has two marked residuals in the resulting module. It is easy to see that these residuals are the same on both reduction paths. As in the previous case, we have used lemma B.16 to show that the residual of the evaluation redex is evaluation redex.

   The case when $l_1 \neq \tilde{l}$ is similar to case when $l_1 = \tilde{l}$. In this case the self-referential redex also has two marked residuals.

$\Box$

The next case is when a self-referential redex gets reduced by the $\underset{\gamma}{\circ\!\!\longrightarrow}$ step:

Lemma C.35. *Suppose* $(D_1, \{(\mathbb{D}, l), (\mathbb{D}', l)\}) \underset{\gamma}{\overset{(\mathbb{D},l)}{\circ\!\!\longrightarrow}} (D_2, F_2)$ *and* $(D_1, \{(\mathbb{D}, l), (\mathbb{D}', l)\}) \overset{(\mathbb{G},\tilde{R})}{\Longrightarrow} (D_3, F_3)$*, where* $(\mathbb{D}, l)$ *is self-referential. Then there exists* (D_4, F_4) *s.t.* $(D_2, F_2) \overset{(\mathbb{G},\tilde{R})/(\mathbb{D},l)}{\Longrightarrow} (D_4, F_4)$ *and* $(D_3, F_3) \underset{\gamma}{\overset{(\mathbb{D},l)/(\mathbb{G},\tilde{R})}{\longrightarrow^*}} (D_4, F_4)$*. The* γ*-development sequence* $\underset{\gamma}{\overset{(\mathbb{D},l)/(\mathbb{G},\tilde{R})}{\longrightarrow^*}}$ *either consists of one step, or of two steps, where the self-referential redex is reduced the second.* $\Box$

Proof. Let l, l', and $\tilde{l}$ denotes labels of the components where $(\mathbb{D}, l)$, $(\mathbb{D}', l)$, and $(\mathbb{G}, \tilde{R})$ occur, respectively. Similarly to the previous lemma C.34, $l \neq l'$, $l \neq \tilde{l}$. As in the proof of the previous lemma, we have 3 cases:

1. $\tilde{R}$ is a term redex. Let $\tilde{R} \leadsto_{\mathcal{T}} \tilde{Q}$.

   Suppose $l' = \tilde{l}$. We have the following possibilities:

   - $\tilde{R}$ and the marked occurrence of l are independent in the component bound to $\tilde{l}$. The initial and the final modules are:

$$[l \mapsto \lambda x.\mathbb{C}\{\underline{l}\}, \tilde{l} \mapsto \mathbb{A}\{\tilde{R}, \underline{l}\}, \ldots],$$
$$[l \mapsto \lambda x.\mathbb{C}\{\lambda x.\mathbb{C}\{l\}\}, \tilde{l} \mapsto \mathbb{A}\{\tilde{Q}, \underline{l}\}, \ldots].$$

     Note that since the self-referential redex is reduced by a γ-development step, it does not have a residual, i.e. the occurrence of l after the substitution is unmarked.

   - The marked l occurs in the operator of $\tilde{R}$. Below are the initial and the final modules:

$$[l \mapsto \lambda x.\mathbb{C}\{\underline{l}\}, \tilde{l} \mapsto \mathbb{E}\{(\lambda y.\mathbb{A}\{\underline{l}, y, \ldots, y\}) \mathbin{@} V\}, \ldots],$$
$$[l \mapsto \lambda x.\mathbb{C}\{\lambda x.\mathbb{C}\{l\}\}, \tilde{l} \mapsto \mathbb{E}\{\mathbb{A}\{\underline{l}, V, \ldots, V\}\}, \ldots].$$

- The marked l occurs in the operand of $\tilde{R}$. In this case the marked non-self-referential redex gets duplicated. As in the previous cases, we show the initial and the final modules:

$$\begin{array}{l}[l \mapsto \lambda x.\mathbb{C}\{\underline{l}\}, \tilde{l} \mapsto \mathbb{E}\{(\lambda y.\mathbb{A}\{y, \ldots, y\}) \mathbin{@} \lambda z.\mathbb{B}\{\underline{l}\}\}, \ldots],\\ [l \mapsto \lambda x.\mathbb{C}\{\lambda x.\mathbb{C}\{l\}\}, \tilde{l} \mapsto \mathbb{E}\{\mathbb{A}\{\lambda z.\mathbb{B}\{\underline{l}\}, \ldots, \lambda z.\mathbb{B}\{\underline{l}\}\}\}, \ldots].\end{array}$$

The case when $l' \neq \tilde{l}$ is analogous to the first of the subcases above.

Note that in all cases the resulting γ-development reduction is one-step.

2. $\tilde{R} = l_1 \neq l$ is a substitution redex. In this case $l_1 \neq \tilde{l}$, since an evaluation redex can not be self-referential. We have the following subcases:

   - $l_1 = l'$. The marked non-self-referential redex gets duplicated. The initial and the final modules are:

   $$\begin{array}{l}[l \mapsto \lambda x.\mathbb{C}\{\underline{l}\}, \tilde{l} \mapsto \mathbb{E}\{l_1\}, l_1 \mapsto \lambda y.\mathbb{A}\{\underline{l}\}, \ldots],\\ [l \mapsto \lambda x.\mathbb{C}\{\lambda x.\mathbb{C}\{l\}\}, \tilde{l} \mapsto \mathbb{E}\{\lambda y.\mathbb{A}\{\underline{l}\}\}, l_1 \mapsto \lambda y.\mathbb{A}\{\underline{l}\}, \ldots].\end{array}$$

   - $l' = \tilde{l}$. The initial module in this case is

   $$[l \mapsto \lambda x.\mathbb{C}\{\underline{l}\}, \tilde{l} \mapsto \mathbb{A}\{l_1, \underline{l}\}, l_1 \mapsto V, \ldots],$$

   and the claim clearly holds.

   - $l' \neq l_1$, $l' \neq \tilde{l}$. All three redexes are independent, and the claim clearly holds.

   In all three subcases the resulting γ-development sequence is one-step.

3. $\tilde{R} = l$ is a substitution redex. Note that this occurrence of l is unmarked, since by the condition of the lemma only the other two redexes are marked. In this case the self-referential redex gets duplicated.

   Suppose $l' \neq \tilde{l}$. The order in which its two residuals are reduced is important. It is easy to see that reducing them in the other order leads to a different module, i.e. to a module that does not satisfy the lemma. In this case we show both reduction sequences step-by-step:

$$\begin{array}{ll}[l \mapsto \lambda x.\mathbb{C}\{\underline{l}\}, \tilde{l} \mapsto \mathbb{E}\{l\}, l' \mapsto \mathbb{A}\{\underline{l}\}, \ldots] & \underset{\gamma}{\circ\!\!\longrightarrow}\\ [l \mapsto \lambda x.\mathbb{C}\{\lambda x.\mathbb{C}\{l\}\}, \tilde{l} \mapsto \mathbb{E}\{l\}, l' \mapsto \mathbb{A}\{\underline{l}\}, \ldots] & \Longrightarrow\\ [l \mapsto \lambda x.\mathbb{C}\{\lambda x.\mathbb{C}\{l\}\}, \tilde{l} \mapsto \mathbb{E}\{\lambda x.\mathbb{C}\{\lambda x.\mathbb{C}\{l\}\}\}, l' \mapsto \mathbb{A}\{\underline{l}\}, \ldots]. & \\ [l \mapsto \lambda x.\mathbb{C}\{\underline{l}\}, \tilde{l} \mapsto \mathbb{E}\{l\}, l' \mapsto \mathbb{A}\{\underline{l}\}, \ldots] & \Longrightarrow\\ [l \mapsto \lambda x.\mathbb{C}\{\underline{l}\}, \tilde{l} \mapsto \mathbb{E}\{\lambda x.\mathbb{C}\{\underline{l}\}\}, l' \mapsto \mathbb{A}\{\underline{l}\}, \ldots] & \underset{\gamma}{\circ\!\!\longrightarrow}\\ [l \mapsto \lambda x.\mathbb{C}\{\underline{l}\}, \tilde{l} \mapsto \mathbb{E}\{\lambda x.\mathbb{C}\{\lambda x.\mathbb{C}\{l\}\}\}, l' \mapsto \mathbb{A}\{\underline{l}\}, \ldots] & \underset{\gamma}{\circ\!\!\longrightarrow}\\ [l \mapsto \lambda x.\mathbb{C}\{\lambda x.\mathbb{C}\{l\}\}, \tilde{l} \mapsto \mathbb{E}\{\lambda x.\mathbb{C}\{\lambda x.\mathbb{C}\{l\}\}\}, l' \mapsto \mathbb{A}\{\underline{l}\}, \ldots]. & \end{array}$$

   The case when $l' = \tilde{l}$, i.e. when the marked non-self-referential redex occurs in the component bound to $\tilde{l}$, is completely analogous to the case when $l' \neq \tilde{l}$. The resulting $\underset{\gamma}{\longrightarrow}^*$ also consists of two steps s.t. the second one reduces the self-referential redex. The initial module in this case is:

$$[l \mapsto \lambda x.\mathbb{C}\{\underline{l}\}, \tilde{l} \mapsto \mathbb{A}\{l, \underline{l}\}, \ldots].$$

□

Lemma C.36. *Suppose* $(D_1, \{(\mathbb{D}, R), (\mathbb{D}', R'), (\mathbb{G}, \tilde{R})\}) \underset{\gamma}{\overset{(\mathbb{D},R)}{\circ\!\!\longrightarrow}} (D_2, F_2)$ *and* $(D_1, \{(\mathbb{D}, R), (\mathbb{D}', R')(\mathbb{G}, \tilde{R})\}) \overset{(\mathbb{G},\tilde{R})}{\Longrightarrow} (D_3, F_3)$*. Then there exists* (D_4, F_4) *s.t.* $(D_2, F_2) \overset{(\mathbb{G},\tilde{R})/(\mathbb{D},R)}{\Longrightarrow} (D_4, F_4)$ *and* $(D_3, F_3) \underset{\gamma}{\overset{(\mathbb{D},R)/(\mathbb{G},\tilde{R})}{\longrightarrow^*}} (D_4, F_4)$*. If* $(\mathbb{D}, R)$ *is a self-referential substitution redex, then the sequence* $\underset{\gamma}{\overset{(\mathbb{D},R)/(\mathbb{G},\tilde{R})}{\longrightarrow^*}}$ *consists of two steps s.t. the second step reduces the self-referential redex. Otherwise the* γ*-development sequence is either one-step, or the order of reduction in it does not matter.* □

Proof. By lemma C.13 we have two cases:

1. All three marked redexes are term redexes. The reduction of a term redex does not depend on the marking of any other redex in the module. It is also the case that for a term redex $(\mathbb{D}, R)/(\mathbb{D}, R) = \emptyset$. Therefore the fact that $(\mathbb{G}, \tilde{R})$ is marked does not affect the reduction, and the case is analogous to case 1 of lemma C.33. If the resulting γ-development sequence is multi-step, then the order of reduction of the redexes does not matter.

2. $R = R' = \tilde{R} = l$. Let l_1, l', and $\tilde{l}$ be the labels of components where the redexes R, R', and $\tilde{R}$ occur, respectively. $l \neq \tilde{l}$, since $(\mathbb{G}, \tilde{R})$ is an evaluation redex. We have the following subcases:

   (a) $l_1 \neq l$, $l' \neq l$, i.e. none of the redexes is self-referential. Suppose that $l' = l_1 = \tilde{l}$, i.e. all the redexes occur in the same component. Assuming, without loss of generality, that context $\mathbb{A}$ contains them in the order $\tilde{R}, R, R'$, we have the following initial and final modules:

   $$[\tilde{l} \mapsto \mathbb{A}\{\underline{l}, \underline{l}, \underline{l}\}, l \mapsto V, \ldots],$$
   $$[\tilde{l} \mapsto \mathbb{A}\{V, V, \underline{l}\}, l \mapsto V, \ldots].$$

   We use lemma B.16 to show that after the $\underset{\gamma}{\circ\!\!\longrightarrow}$ step the residual of $(\mathbb{G}, \tilde{R})$ is an evaluation redex. The cases when the three redexes occur in different components are similar.

   (b) $l' = l$, i.e. the redex $(\mathbb{D}', R')$ is self-referential. Assuming that the other two redexes occur in the same component, we have the following reductions:

   $$\begin{array}{ll} [l \mapsto \lambda x.\mathbb{C}\{\underline{l}\}, \tilde{l} \mapsto \mathbb{A}\{\underline{l}, \underline{l}\}, \ldots] & \underset{\gamma}{\circ\!\!\longrightarrow} \\ [l \mapsto \lambda x.\mathbb{C}\{\underline{l}\}, \tilde{l} \mapsto \mathbb{A}\{\underline{l}, \lambda x.\mathbb{C}\{l\}\}, \ldots] & \Longrightarrow \\ [l \mapsto \lambda x.\mathbb{C}\{\underline{l}\}, \tilde{l} \mapsto \mathbb{A}\{\lambda x.\mathbb{C}\{\underline{l}\}, \lambda x.\mathbb{C}\{l\}\}, \ldots], & \\ [l \mapsto \lambda x.\mathbb{C}\{\underline{l}\}, \tilde{l} \mapsto \mathbb{A}\{\underline{l}, \underline{l}\}, \ldots] & \Longrightarrow \\ [l \mapsto \lambda x.\mathbb{C}\{\underline{l}\}, \tilde{l} \mapsto \mathbb{A}\{\lambda x.\mathbb{C}\{\underline{l}\}, \underline{l}\}, \ldots] & \underset{\gamma}{\longrightarrow} \\ [l \mapsto \lambda x.\mathbb{C}\{\underline{l}\}, \tilde{l} \mapsto \mathbb{A}\{\lambda x.\mathbb{C}\{\underline{l}\}, \lambda x.\mathbb{C}\{l\}\}, \ldots]. & \end{array}$$

   Here the self-referential redex has two marked residuals. The case when the two non-self-referential redexes occur in different components is analogous.

   (c) $l_1 = l$, i.e. $(\mathbb{D}, R)$ is self-referential. Again we show the case when the other two redexes occur in the same component. The following reductions prove the claim of the lemma.

   $$\begin{array}{ll} [l \mapsto \lambda x.\mathbb{C}\{\underline{l}\}, \tilde{l} \mapsto \mathbb{A}\{\underline{l}, \underline{l}\}, \ldots] & \underset{\gamma}{\circ\!\!\longrightarrow} \\ [l \mapsto \lambda x.\mathbb{C}\{\lambda x.\mathbb{C}\{l\}\}, \tilde{l} \mapsto \mathbb{A}\{\underline{l}, \underline{l}\}, \ldots] & \Longrightarrow \\ [l \mapsto \lambda x.\mathbb{C}\{\lambda x.\mathbb{C}\{l\}\}, \tilde{l} \mapsto \mathbb{A}\{\lambda x.\mathbb{C}\{\lambda x.\mathbb{C}\{l\}\}, \underline{l}\}, \ldots], & \\ [l \mapsto \lambda x.\mathbb{C}\{\underline{l}\}, \tilde{l} \mapsto \mathbb{A}\{\underline{l}, \underline{l}\}, \ldots] & \Longrightarrow \\ [l \mapsto \lambda x.\mathbb{C}\{\underline{l}\}, \tilde{l} \mapsto \mathbb{A}\{\lambda x.\mathbb{C}\{\underline{l}\}, \underline{l}\}, \ldots] & \underset{\gamma}{\circ\!\!\longrightarrow} \\ [l \mapsto \lambda x.\mathbb{C}\{\underline{l}\}, \tilde{l} \mapsto \mathbb{A}\{\lambda x.\mathbb{C}\{\lambda x.\mathbb{C}\{l\}\}, \underline{l}\}, \ldots] & \underset{\gamma}{\circ\!\!\longrightarrow} \\ [l \mapsto \lambda x.\mathbb{C}\{\lambda x.\mathbb{C}\{l\}\}, \tilde{l} \mapsto \mathbb{A}\{\lambda x.\mathbb{C}\{\lambda x.\mathbb{C}\{l\}\}, \underline{l}\}, \ldots]. & \end{array}$$

Here, as in case 3 of lemma C.35, the order in which the two resulting γ-development steps are performed matters: switching the order will result in a reduction that does not satisfy the lemma.

Note that in cases (a) and (b) the γ-development reduction is one-step, and in (c) the self-referential redex is reduced last.

□

Lemma C.37 (Elementary Project Diagram for $\mathcal{C}$). *If* $(D_1, F_1) \circ\!\xrightarrow[\gamma]{(\mathbb{D},R)} (D_2, F_2)$ *and* $(D_1, F_1) \xRightarrow{(\mathbb{G},\tilde{R})} (D_3, F_3)$*, then there exists* (D_4, F_4) *s.t.* $(D_2, F_2) \xRightarrow{(\mathbb{G},\tilde{R})/(\mathbb{D},R)} (D_4, F_4)$ *and* $(D_3, F_3) \overset{(\mathbb{D},R)/(\mathbb{G},\tilde{R})}{\underset{\gamma}{\longrightarrow^*}} (D_4, F_4)$. □

Proof. If F_1 does not have a self-referential redex, then the lemma follows from lemma C.33 if $\tilde{R}$ is not marked and from the cases 1 or 2(a) of lemma C.36 if $\tilde{R}$ is marked by induction on the number of redexes in F_1. The proof is analogous to the proof of lemma C.23.

Suppose F_1 has a self-referential redex, but $(\mathbb{D}, R)$ is not self-referential. Let $(\mathbb{D}', R')$ be the self-referential redex in F_1. By lemma C.34 or part 2(b) of lemma C.36 the claim of the lemma holds for the initial pair $(D_1, \{(\mathbb{D}, R), (\mathbb{D}', R')\})$ or $(D_1, \{(\mathbb{D}, R), (\mathbb{D}', R'), (\mathbb{G}, \tilde{R})\})$ in the case when $(\mathbb{G}, \tilde{R})$ is marked. The rest of the redexes in F_1 are not self-referential, since $(D_1, F_1) \in dom(\gamma)$. Let $(\mathbb{D}'', R'') \neq (\mathbb{D}, R)$ be another redex in F_1. Note that it is a substitution redex by lemma C.13. Then by lemma C.33 or by case 2(a) of lemma C.36 the claim of the lemma holds for the initial pair $(D_1, \{(\mathbb{D}, R), (\mathbb{D}'', R'')\})$ (or $(D_1, \{(\mathbb{D}, R), (\mathbb{D}'', R''), (\mathbb{G}, \tilde{R})\})$ for a marked $(\mathbb{G}, \tilde{R})$). Then by corollary C.26 for the $\underset{\gamma}{\rightarrow}$ steps and by lemma C.24 for the $\Longrightarrow$ steps we can combine the diagrams for individual redexes in F_1 into a diagram for the initial pair (D_1, F_1). We use the fact that, since a self-referential redex is not reduced in any of the reductions, the redexes in the sequence $\overset{(\mathbb{D},R)/(\mathbb{G},\tilde{R})}{\underset{\gamma}{\longrightarrow^*}}$ can be reduced in any order, so we can assume that for all individual diagrams they are reduced in the same fixed order.

Suppose now that $(\mathbb{D}, R)$ is a self-referential redex. Then by lemma C.35 or by case 2(c) of lemma C.36 the lemma holds for an initial pair $(D_1, \{(\mathbb{D}, R), (\mathbb{D}'', R'')\})$ for each $(\mathbb{D}'', R'') \in F_1$ (or for an initial pair $(D_1, \{(\mathbb{D}, R), (\mathbb{D}'', R''), (\mathbb{G}, \tilde{R})\})$ in the case when $(\mathbb{G}, \tilde{R})$ is marked). Then, as in the previous case, we combine the individual diagrams into the diagram for the initial pair (D_1, F_1). Note that in this case the order of reduction of the sequence $\overset{(\mathbb{D},R)/(\mathbb{G},\tilde{R})}{\underset{\gamma}{\longrightarrow^*}}$ is fixed (the self-referential redex in this sequence is reduced last). □

Now we show the dual property A.26.

Before we can prove it, let us show some properties of substitution which we are going to use in the proofs. The essence of these properties is that a non-evaluation substitution step can not create an evaluation redex (either a term, or a substitution redex) that has not existed in the original term. The property is the converse of the one stated in lemma B.16: the lemma below explicitly restricts the substitution to be a non-evaluation step, however one may notice that the only substitution possible into a term of the form $\mathbb{E}\{R\}$ containing a label is a non-evaluation substitution, since such a label may appear only in a non-evaluation context.

Lemma C.38. *1. If* $\mathbb{C} \notin \mathbf{EvalContext}_{\mathcal{T}}$ *and* $\mathbb{C}\{V\}$ *is a term redex, then* $\mathbb{C}\{l\}$ *is a term redex.*

2. *If* $\mathbb{A}\{V, \square\} \in \mathbf{EvalContext}_{\mathcal{T}}$ *and* $\mathbb{A}\{\square, R\} \notin \mathbf{EvalContext}_{\mathcal{T}}$ *(or, respectively,* $\mathbb{A}\{\square, V\} \in \mathbf{EvalContext}_{\mathcal{T}}$ *and* $\mathbb{A}\{R, \square\} \notin \mathbf{EvalContext}_{\mathcal{T}}$*), then* $\mathbb{A}\{l, \square\} \in \mathbf{EvalContext}_{\mathcal{T}}$ *(respectively,* $\mathbb{A}\{\square, l\} \in \mathbf{EvalContext}_{\mathcal{T}}$*).*

3. *If* $\mathbb{A}\{V, \square\} \in \mathbf{EvalContext}_{\mathcal{T}}$ *and* $\mathbb{A}\{\square, l_1\} \notin \mathbf{EvalContext}_{\mathcal{T}}$ *(or, respectively,* $\mathbb{A}\{\square, V\} \in \mathbf{EvalContext}_{\mathcal{T}}$ *and* $\mathbb{A}\{l_1, \square\} \notin \mathbf{EvalContext}_{\mathcal{T}}$*), then* $\mathbb{A}\{l, \square\} \in \mathbf{EvalContext}_{\mathcal{T}}$ *(respectively,* $\mathbb{A}\{\square, l\} \in \mathbf{EvalContext}_{\mathcal{T}}$*), where* l_1 *may or may not be the same as* l.

□

Proof. 1. A term redex R is either c_1 *op* c_2, where both constants occur in evaluation context, so $R \neq \mathbb{C}\{V\}$, where $\mathbb{C} \notin \mathbf{EvalContext}_{\mathcal{T}}$, or $\lambda x.M$ @ V, where the only non-evaluation contexts occur inside V or inside M, so the claim clearly holds.

2. By induction on the structure of an evaluation context.

3. Same as the previous case.

□

Similarly to the proof of the property A.25 (lemma C.37 above), we first show the property for 4 different cases of kinds, positions, and markings of the redexes. The following 4 lemmas correspond to these cases.

Remark C.39. In the proofs of the auxiliary lemmas for elementary project diagrams (lemmas C.33–C.36) we have used the notion of independent redexes (definition C.5), in particular some of the cases considered in the proofs where the cases when the evaluation redex $(\mathbb{G}, \tilde{R})$ was independent from the non-evaluation redex $(\mathbb{D}, R)$ (see lemma C.37). This definition is not applicable in the case of lemma C.44 (elementary lift diagram), since the non-evaluation redex occurs in the initial module D_1, but the evaluation redex occurs in the module D_2 obtained from D_1 by reducing the non-evaluation redex. We extend the notion of independent redexes to this case as follows: we say that $(\mathbb{G}, \tilde{R})$ and $(\mathbb{D}, R)$ are *independent* if $(\mathbb{G}, \tilde{R})$ is independent from $(\mathbb{D}, Q)$ in D_2 (as a subterm, see definition C.5), where $R \rightsquigarrow Q$ if R is a term redex, and $Q = V$ if $R = l$ is a substitution redex. The notion is well-defined, since $(\mathbb{G}, \tilde{R})$ is an evaluation redex, and therefore can not be contained in the value being substituted.

We also say that $(\mathbb{G}, \tilde{R})$ is independent from $(\mathbb{D}', R')$ if it is independent from its residual(s) in D_2. □

Lemma C.40. *Suppose $(\mathbb{D}, R)$ and $(\mathbb{D}', R')$ are not self-referential, and $(D_1, \{(\mathbb{D}, R), (\mathbb{D}', R')\}) \overset{(\mathbb{D},R)}{\underset{\gamma}{\circ\!\!\longrightarrow}} (D_2, F_2)$ $\overset{(\mathbb{G},\tilde{R})}{\Longrightarrow} (D_4, F_4)$, then there exists (D_3, F_3) s.t. $(D_1, \{(\mathbb{D}, R), (\mathbb{D}', R')\}) \overset{(\mathbb{G}',\tilde{R}')}{\Longrightarrow} (D_3, F_3) \overset{(\mathbb{D},R)/(\mathbb{G}',\tilde{R}')}{\underset{\gamma}{\longrightarrow^*}} (D_4, F_4)$ s.t. $(\mathbb{G}, \tilde{R}) = (\mathbb{G}', \tilde{R}')/(\mathbb{D}, R)$. If the sequence $\overset{(\mathbb{D},R)/(\mathbb{G}',\tilde{R}')}{\underset{\gamma}{\longrightarrow^*}}$ consists of more than one step, then for any such reduction $(D_3, F_3) \overset{(\mathbb{D},R)/(\mathbb{G}',\tilde{R}')}{\underset{\gamma}{\longrightarrow^*}} (D_4, F_4)$.* □

Proof. Since $(D_1, \{(\mathbb{D}, R), (\mathbb{D}', R')\}) \in dom(\gamma)$, the two marked redexes are of the same kind, and if they are substitution redexes, then $R = R' = l_1$. Let $l, l', \tilde{l}$ be the labels of the components containing $(\mathbb{D}, R)$, $(\mathbb{D}', R')$, and $(\mathbb{G}, \tilde{R})$, respectively. We have the following cases:

1. All three redexes are term redexes. If all three redexes occur in different terms, the claim of the lemma clearly holds. If the three redexes all occur in the same term, then the claim holds by lemma B.28. If $l = \tilde{l} \neq l'$, then the claim holds by lemma B.25 (clearly $(\mathbb{D}', R')$, which occurs not in the same component where the other two redexes occur, has a single residual on both reduction paths). If $l = l' \neq \tilde{l}$ or $\tilde{l} = l' \neq l$, then $(\mathbb{D}, R)$ and $(\mathbb{G}, \tilde{R})$ are independent (see remark C.39), and clearly the claim of the lemma holds.

2. The marked redexes are term redexes, and the evaluation redex is a substitution redex. Let $\tilde{R} = l_1$. Since $(\mathbb{G}, l_1)$ is an evaluation redex, $l_1 \neq \tilde{l}$. We have the following subcases:

   (a) Among the 4 labels l, l', l_1, and $\tilde{l}$, no two are equal. Then the claim of the lemma clearly holds.

   (b) $l_1 \notin \{l, l'\}$. Then

   $$D_2 = [\tilde{l} \mapsto \mathbb{E}\{l_1\}, l_1 \mapsto V, \ldots].$$

   By lemma B.24

   $$D_1 = [\tilde{l} \mapsto \mathbb{E}'\{l_1\}, l_1 \mapsto V, \ldots].$$

Suppose $l = l' = \tilde{l}$, i.e. both marked redexes occur in the same component as $(\mathbb{G}, l_1)$ and the marked redexes are independent from each other. By lemma C.32 the substitution occurrence of l_1 is not contained in R or R', i.e. $\mathbb{E}'\{l_1\} = \mathbb{A}\{l_1, R, R'\}$. Recall that in the proofs in this section we write $\mathbb{A}\{l_1, R, R'\}$ to mean that the context $\mathbb{A}$ contains the three terms in some, not necessarily the specified, order. Then the following reductions prove the claim of the lemma:

$$\begin{array}{lclcl} [\tilde{l} \mapsto \mathbb{A}\{l_1, R, R'\}, l_1 \mapsto V, \ldots] & \Rightarrow & [\tilde{l} \mapsto \mathbb{A}\{V, R, R'\}, l_1 \mapsto V, \ldots] & \underset{\gamma}{\rightarrow} & [\tilde{l} \mapsto \mathbb{A}\{V, Q, R'\}, l_1 \mapsto V, \ldots], \\ [\tilde{l} \mapsto \mathbb{A}\{l_1, R, R'\}, l_1 \mapsto V, \ldots] & \underset{\gamma}{\circ\!\!\longrightarrow} & [\tilde{l} \mapsto \mathbb{A}\{l_1, Q, R'\}, l_1 \mapsto V, \ldots] & \Rightarrow & [\tilde{l} \mapsto \mathbb{A}\{V, Q, R'\}, l_1 \mapsto V, \ldots]. \end{array}$$

The cases when $\tilde{l} = l = l'$ and R contains R' or R' contains R are similar. The cases when $l = \tilde{l}, l' \neq \tilde{l}$, or $l' = \tilde{l}, l \neq \tilde{l}$, or $l = l' \neq \tilde{l}$ are also similar, but simpler.

(c) $\tilde{l} \notin \{l, l'\}$. Note that the case when $l = l' \neq l_1$ has been considered above. The remaining subcases are $l = l' = l_1$, $l = l_1 \neq l'$, and $l' = l_1 \neq l$. As above, we consider the case $l = l' = l_1$ in detail, the other two cases are similar, but simpler. We have the following possibilities:

I. R, R' are independent in D_1. In this case:

$$D_2 = [\tilde{l} \mapsto \mathbb{E}\{l_1\}, l_1 \mapsto \lambda y.\mathbb{A}\{Q, R'\},$$

where $R \rightsquigarrow_{\mathcal{T}} Q$. Then

$$D_1 = [\tilde{l} \mapsto \mathbb{E}\{l_1\}, l_1 \mapsto \lambda y.\mathbb{A}\{R, R'\},$$

and the reduction in case 2(c)I of the proof of lemma C.33 proves the claim. Note that the component bound to l_1 in D_1 is a value, therefore the substitution is possible.

II. R contains R'. Then

$$D_2 = [\tilde{l} \mapsto \mathbb{E}\{l_1\}, l_1 \mapsto \lambda y.\mathbb{A}\{Q\},$$

where Q contains R', possibly several copies if R' is contained in the operand of R, and therefore is duplicated. Again the reduction in the respective case (case 2(c)II) of lemma C.33 proves the claim.

III. R' contains R. The case is similar to the previous case and to case 2(c)III of lemma C.33.

(d) $l = \tilde{l}$, $l' = l_1$. As in case (a) of this proof, the substitution redex l_1 in D_2 is independent from the result of reduction of R by lemma C.32. Therefore

$$D_1 = [\tilde{l} \mapsto \mathbb{A}\{l_1, R\}, l_1 \mapsto \lambda x.\mathbb{B}\{R'\}],$$

and the claim follows by the reduction.

(e) $l' = \tilde{l}$, $l = l_1$. Then

$$D_1 = [\tilde{l} \mapsto \mathbb{A}\{l_1, R'\}, l_1 \mapsto \lambda x.\mathbb{B}\{R\}].$$

Note that since l_1 is bound to a value containing the result of reduction of a non-evaluation redex, this component is a λ-abstraction in D_2, and therefore in D_1.

3. The marked redexes are substitution redexes, and the evaluation redex is a term redex. Let $R = R' = l_1$ (since the initial pair is in $dom(\gamma)$, it must be the case that R and R' are the same label). By the condition of the lemma none of the redexes is self-referential, therefore $l \neq l_1$, $l' \neq l_1$. It also must be the case that $l_1 \neq \tilde{l}$, since l_1 is bound to a value in D_1, and therefore in D_2, and $\tilde{l}$ is bound to an evaluatable term in D_2. As in the previous case, let us assume that $l = l' = \tilde{l}$. There are the following possibilities:

- The evaluation redex is independent from both substitution redexes. Then

$$D_2 = [\tilde{l} \mapsto \mathbb{A}\{\tilde{R}, V, \underline{l_1}\}, l_1 \mapsto V, \dots].$$

By lemma C.38 $\mathbb{A}\{\Box, \underline{l_1}, \underline{l_1}\} \in \textbf{EvalContext}_{\mathcal{T}}$, and it is easy to check that the claim of the lemma holds. No duplication of redexes occurs in this case.

- Both the marked occurrence of l_1 of the redex R' and the value V which is the result of the non-evaluation substitution redex occur in the operand of $\tilde{R}$. Then

$$D_2 = [\tilde{l} \mapsto \mathbb{E}\{(\lambda x.\mathbb{A}\{x, \dots, x\}) \mathbin{@} \lambda y.\mathbb{B}\{V, \underline{l_1}\}\}, l_1 \mapsto V, \dots].$$

The occurrence of V here is under a λ, i.e. in a non-evaluation context, and therefore by $(\lambda x.\mathbb{A}\{x, \dots, x\}) \mathbin{@} \lambda y.\mathbb{B}\{\underline{l_1}, \underline{l_1}\}$ is a redex by part 1 of lemma C.38. The rest of the claim follows by the reduction in the proof of the analogous case of lemma C.33.

- The remaining cases (both R and R' occur in the operator part of $\tilde{R}$, or when one of the two marked redexes occurs in the operand, and the other in the operator) are similar.

The other cases ($l = l' \neq \tilde{l}$, $l = \tilde{l} \neq l'$, $l' = \tilde{l} \neq l$, and the case when all the three labels are distinct) are similar.

4. All three redexes are substitution redexes, where $\tilde{R} = l_2 \neq l_1$. In this case $\tilde{l} \neq l_2$, since $\tilde{R}$ is an evaluation redex. As in case 3, $l_1 \notin \{l, l', \tilde{l}\}$. As in the proof of lemma C.33, we have the following subcases:

   (a) All 5 labels are distinct, then the lemma trivially holds.

   (b) $l = l' = \tilde{l}$. Then

   $$D_2 = [\tilde{l} \mapsto \mathbb{A}\{l_2, V_1, \underline{l_1}\}, l_1 \mapsto V_1, l_2 \mapsto V_2, \dots].$$

   The label l_2 occurs independently from V_1 in the component bound to $\tilde{l}$, since l_2 must occur in an evaluation context, and therefore can not occur inside a value[25]. By lemma C.38 $\mathbb{A}\{l_2, \underline{l_1}, \underline{l_1}\} \in \textbf{EvalContext}_{\mathcal{T}}$, and the claim of the lemma holds.

   (c) $l = l' = l_2$. Then

   $$D_2 = [\tilde{l} \mapsto \mathbb{E}\{l_2\}, l_2 \mapsto \lambda x.\mathbb{C}\{V_1, \underline{l_1}\}, l_1 \mapsto V_1, \dots].$$

   In this case

   $$D_1 = [\tilde{l} \mapsto \mathbb{E}\{l_2\}, l_2 \mapsto \lambda x.\mathbb{C}\{\underline{l_1}, \underline{l_1}\}, l_1 \mapsto V_1, \dots],$$

   and the claim clearly holds.

   (d) $l = l'$, $l \neq \tilde{l}$, $l \neq l_2$. The two marked redexes are independent from both the evaluation redex and the value being substituted into it. The claim of the lemma clearly holds.

   (e) $l = \tilde{l}$, $l' = l_2$. Then

   $$D_2 = [\tilde{l} \mapsto \mathbb{A}\{l_2, V_1\}, l_2 \mapsto \lambda x.\mathbb{B}\{\underline{l_1}\}, l_1 \mapsto V_1, \dots].$$

   Similarly to the case (b), l_2 is independent from V_1 in the component bound to $\tilde{l}$. By lemma C.38 $\mathbb{A}\{\Box, \underline{l_1}\} \in \textbf{EvalContext}_{\mathcal{T}}$, and again it is easy to check that the claim holds.

   (f) $l = \tilde{l}$, $l' \neq l_2$, $l' \neq \tilde{l}$. Similar to the previous case.

[25] Note that even if labels were values, V_1 still could not have been equal to l_2, since the substitution $D_1 \multimapdotbothA D_2$ is a non-evaluation step, but l_2 must occur in an evaluation context in D_2.

(g) $l = l_2$, $l' = \tilde{l}$. In this case

$$D_2 = [\tilde{l} \mapsto \mathbb{A}\{l_2, \underline{l_1}\}, l_2 \mapsto \lambda x.\mathbb{B}\{V_1\}, l_1 \mapsto V_1, \ldots].$$

Again, the claim of the lemma clearly holds.

(h) $l = l_2$, $l' \neq l_2$, $l' \neq \tilde{l}$. Similar to the previous case.

5. All the three redexes are substitution redexes, $R = R' = \tilde{R} = l_1$. $\tilde{R}$ can not be marked by the condition of the lemma (only the other two redexes are marked). Similarly to the previous case, $l_1 \notin \{l, l', \tilde{l}\}$. Therefore no redex duplication is possible, and the claim of the lemma holds in this case.

□

Lemma C.41. *Suppose* $(\mathbb{D}', R')$ *is a self-referential redex. If* $(D_1, \{(\mathbb{D}, R), (\mathbb{D}', R')\}) \underset{\gamma}{\circ\!\xrightarrow{(\mathbb{D},R)}} (D_2, F_2) \xRightarrow{(\mathbb{G},\tilde{R})} (D_4, F_4)$*, then there exists* (D_3, F_3) *s.t.* $(D_1, \{(\mathbb{D}, R), (\mathbb{D}', R')\}) \xRightarrow{(\mathbb{G}',\tilde{R}')} (D_3, F_3) \overset{(\mathbb{D},R)/(\mathbb{G}',\tilde{R}')}{\underset{\gamma}{\longrightarrow^*}} (D_4, F_4)$ *s.t.* $(\mathbb{G}, \tilde{R}) = (\mathbb{G}', \tilde{R}')/(\mathbb{D}, R)$*. If the sequence* $\overset{(\mathbb{D},R)/(\mathbb{G}',\tilde{R}')}{\underset{\gamma}{\longrightarrow^*}}$ *consists of more than one step, then for any such reduction* $(D_3, F_3) \overset{(\mathbb{D},R)/(\mathbb{G}',\tilde{R}')}{\underset{\gamma}{\longrightarrow^*}} (D_4, F_4)$. □

Proof. Let $l, l_1, \tilde{l}$ be the labels of the components where $(\mathbb{D}', l)$, $(\mathbb{D}, l)$, and $(\mathbb{G}', \tilde{R}')$ occur, respectively. Note that $\tilde{l} \neq l$, since $(\mathbb{G}', \tilde{R}')$ is an evaluation redex, and $l_1 \neq l$, since there may be at most one marked self-referential redex. We have the following subcases:

1. $\tilde{R}$ is a term redex. If $l_1 \neq \tilde{l}$, then the evaluation redex is independent from both substitution redexes, and the claim of the lemma clearly holds. If $l_1 = \tilde{l}$, then we have one of the following:

   - $\tilde{R}$ is independent from the marked occurrence of l. The case is similar to the case when $l_1 \neq \tilde{l}$. By lemma C.38 $\tilde{R}$ occurs in an evaluation context in D_1.
   - The result of the substitution of l occurs in $\tilde{R}$.
     The case when $\tilde{R}$ is of the form c_1 *op* c_2 is impossible, since then either c_1 or c_2 must be the result of the substitution, but the component bound to l is a λ-abstraction, not a constant, since it contains an occurrence of l.
     If $\tilde{R}$ is an application and the result of the substitution occurs in its operator, then we have

     $$\begin{aligned} D_2 &= [l \mapsto \lambda x.\mathbb{C}\{\underline{l}\}, \tilde{l} \mapsto \mathbb{E}\{(\lambda y.\mathbb{A}\{\lambda x.\mathbb{C}\{l\}, y, \ldots, y\}) \mathbin{@} V\}, \ldots], \\ D_1 &= [l \mapsto \lambda x.\mathbb{C}\{\underline{l}\}, \tilde{l} \mapsto \mathbb{E}\{(\lambda y.\mathbb{A}\{\underline{l}, y, \ldots, y\}) \mathbin{@} V\}, \ldots]. \end{aligned}$$

     Note that in l occurs unmarked in the component bound to $\tilde{l}$ in D_2, since the non-evaluation step is a γ-development . The claim easily follows, similarly to the respective case of lemma C.34.
     If $\tilde{R}$ is an application and the result of the substitution occurs in its operand, then

     $$\begin{aligned} D_2 &= [l \mapsto \lambda x.\mathbb{C}\{\underline{l}\}, \tilde{l} \mapsto \mathbb{E}\{(\lambda y.\mathbb{A}\{y, \ldots, y\}) \mathbin{@} \lambda z.\mathbb{B}\{\lambda x.\mathbb{C}\{l\}\}\}, \ldots], \\ D_1 &= [l \mapsto \lambda x.\mathbb{C}\{\underline{l}\}, \tilde{l} \mapsto \mathbb{E}\{(\lambda y.\mathbb{A}\{y, \ldots, y\}) \mathbin{@} \lambda z.\mathbb{B}\{\underline{l}\}\}, \ldots]. \end{aligned}$$

     The claim easily follows.

2. $\tilde{R} = l_2 \neq l$ is a substitution redex. $\tilde{R}$ is an evaluation redex, therefore $\tilde{l} \neq l_2$. We have the following possibilities:

   - $l_1 \neq l_2$, $l_1 \neq \tilde{l}$. In this case the three redexes occur in different components, and the claim of the lemma clearly holds.

- $l_1 = l_2$, i.e. the non-evaluation redex occurs in the value being substituted by the evaluation step. In this case

$$D_2 = [l \mapsto \lambda x.\mathbb{C}\{\underline{l}\}, \tilde{l} \mapsto \mathbb{E}\{l_1\}, l_1 \mapsto \mathbb{A}\{\lambda x.\mathbb{C}\{l\}\}, \dots].$$

Then $\mathbb{A} \neq \Box$, since the γ-development step is a non-evaluation step. Note that l_1 must be bound to a value for $\tilde{R}$ to be a redex in D_2, so $\mathbb{A} = \lambda y.\mathbb{B}$, and

$$D_1 = [l \mapsto \lambda x.\mathbb{C}\{\underline{l}\}, \tilde{l} \mapsto \mathbb{E}\{l_1\}, l_1 \mapsto \lambda y.\mathbb{B}\{\underline{l}\}, \dots].$$

The rest of the claim follows from the reduction in the respective case of lemma C.34.

- $l_1 = \tilde{l}$. Then

$$D_2 = [l \mapsto \lambda x.\mathbb{C}\{\underline{l}\}, \tilde{l} \mapsto \mathbb{A}\{l_2, \lambda x.\mathbb{C}\{l\}\}, \dots].$$

Similarly to the case 4(b) of lemma C.40, the label l_2 occurs independently from $\lambda x.\mathbb{C}\{l\}$ in the component bound to $\tilde{l}$, since l_2 must occur in an evaluation context, and therefore can not occur under a λ. By lemma C.38 $\mathbb{A}\{\Box, \underline{l}\} \in \mathbf{EvalContext}_{\mathcal{T}}$. The rest of the claim easily follows.

3. $\tilde{R} = l$ is a substitution redex. This occurrence of l is not marked, since by the condition of the lemma only the other two redexes are marked.

   If $l_1 = \tilde{l}$, then

$$D_2 = [l \mapsto \lambda x.\mathbb{C}\{\underline{l}\}, \tilde{l} \mapsto \mathbb{A}\{l, \lambda x.\mathbb{C}\{l\}\}, \dots].$$

   Similarly to the previous case, l is independent from $\lambda x.\mathbb{C}\{\underline{l}\}$ in the component bound to $\tilde{l}$. The rest of the proof is similar to the respective case of lemma C.34.

   The case when $l_1 \neq \tilde{l}$ is similar.

□

Lemma C.42. *Suppose* $(\mathbb{D}, R)$ *is a self-referential redex. If* $(D_1, \{(\mathbb{D}, R), (\mathbb{D}', R')\}) \circ\!\!\xrightarrow[\gamma]{(\mathbb{D},R)} (D_2, F_2) \xRightarrow{(\mathbb{G},\tilde{R})} (D_4, F_4)$*, then there exists* (D_3, F_3) *s.t.* $(D_1, \{(\mathbb{D}, R), (\mathbb{D}', R')\}) \xRightarrow{(\mathbb{G}',\tilde{R}')} (D_3, F_3) \overset{(\mathbb{D},R)/(\mathbb{G}',\tilde{R}')}{\underset{\gamma}{\longrightarrow^*}} (D_4, F_4)$ *s.t.* $(\mathbb{G}, \tilde{R}) = (\mathbb{G}', \tilde{R}')/(\mathbb{D}, R)$*. The sequence* $\overset{(\mathbb{D},R)/(\mathbb{G}',\tilde{R}')}{\underset{\gamma}{\longrightarrow^*}}$ *consists either of a single step, or of two steps, where the self-referential residual of* $(\mathbb{D}, R)$ *is reduced second.* □

Proof. Let l, l', and $\tilde{l}$ denotes labels of the components where $(\mathbb{D}, l), (\mathbb{D}', l)$, and $(\mathbb{G}, \tilde{R})$ occur, respectively. Similarly to the previous lemma C.41, $l \neq l'$, $l \neq \tilde{l}$. We have 3 cases:

1. $\tilde{R}$ is a term redex. Suppose $l' = \tilde{l}$. We have the following possibilities:

   (a) $\tilde{R}$ and the marked occurrence of l are independent in the component bound to $\tilde{l}$. Then

$$D_2 = [l \mapsto \lambda x.\mathbb{C}\{\lambda x.\mathbb{C}\{l\}\}, \tilde{l} \mapsto \mathbb{A}\{\tilde{R}, \underline{l}\}, \dots].$$

   Then

$$D_1 = [l \mapsto \lambda x.\mathbb{C}\{\underline{l}\}, \tilde{l} \mapsto \mathbb{A}\{\tilde{R}, \underline{l}\}, \dots],$$

   and the rest of the proof is the same as in the respective case of lemma C.35.

   (b) The marked l occurs in the operator of $\tilde{R}$. The case is similar to the previous one.

(c) The marked l occurs in the operand of $\tilde{R}$. The case is similar to the previous one.

If $l' \neq \tilde{l}$, the case is also similar to the previous ones.

2. $\tilde{R} = l_1 \neq l$ is a substitution redex. If $l' = l_1$, then

$$D_2 = [l \mapsto \lambda x.\mathbb{C}\{\lambda x.\mathbb{C}\{l\}\}, \tilde{l} \mapsto \mathbb{E}\{l_1\}, l_1 \mapsto \lambda y.\mathbb{A}\{\underline{l}\}, \ldots]$$
$$D_1 = [l \mapsto \lambda x.\mathbb{C}\{\underline{l}\}, \tilde{l} \mapsto \mathbb{E}\{l_1\}, l_1 \mapsto \lambda y.\mathbb{A}\{\underline{l}\}, \ldots].$$

The claim easily follows.

If $l' = \tilde{l}$, then

$$D_2 = [l \mapsto \lambda x.\mathbb{C}\{\lambda x.\mathbb{C}\{l\}\}, \tilde{l} \mapsto \mathbb{A}\{l_1, \underline{l}\}, l_1 \mapsto V, \ldots].$$

The case is similar to the case when $l' = l_1$, and so is the case when $l' \notin \{l_1, \tilde{l}\}$.

3. $\tilde{R} = l$. By the condition of the lemma this occurrence of l is unmarked, since only the other two redexes are marked.

Suppose $l' \neq \tilde{l}$. Then

$$D_2 = [l \mapsto \lambda x.\mathbb{C}\{\lambda x.\mathbb{C}\{l\}\}, \tilde{l} \mapsto \mathbb{E}\{l\}, l' \mapsto \mathbb{A}\{\underline{l}\}, \ldots],$$

and the reduction given in the case 3 of lemma C.35 proves the claim. Note that the self-referential redex gets duplicated by the evaluation step $D_1 \Longrightarrow D_3$, and the reduction $D_3 \underset{\gamma}{\longrightarrow^*} D_4$ first reduces the copy of the redex in the component bound to $\tilde{l}$, and then the original self-referential redex (in the component bound to l).

The case when $l' = \tilde{l}$ is analogous.

□

Lemma C.43. *Suppose* $\{(\mathbb{D}, R), (\mathbb{D}', R')\} \cap \tilde{F} = \emptyset$. *Let* $(D_1, \{(\mathbb{D}, R), (\mathbb{D}', R')\} \cup \tilde{F}) \overset{(\mathbb{D},R)}{\underset{\gamma}{\circ\!\!\longrightarrow}} (D_2, F \cup F' \cup \{(\mathbb{G}, \tilde{R})\}) \overset{(\mathbb{G},\tilde{R})}{\Longrightarrow} (D_4, F_4)$, *where* F, F' *are the sets of marked residuals of* $(\mathbb{D}, R), (\mathbb{D}', R')$, *respectively. Then* $\tilde{F} = \{(\mathbb{G}', \tilde{R}')\}$ *s.t.* $(\mathbb{G}, \tilde{R}) = (\mathbb{G}', \tilde{R}')/(\mathbb{D}, R)$, *and there exists* (D_3, F_3) *s.t.* $(D_1, \{(\mathbb{D}, R), (\mathbb{D}', R')\}) \overset{(\mathbb{G}',\tilde{R}')}{\Longrightarrow} (D_3, F_3) \overset{(\mathbb{D},R)/(\mathbb{G}',\tilde{R}')}{\underset{\gamma}{\longrightarrow^*}} (D_4, F_4)$. *If* $(\mathbb{D}, R)$ *is a self-referential redex, then its self-referential residual is reduced last in the two-step sequence* $\overset{(\mathbb{D},R)/(\mathbb{G}',\tilde{R}')}{\underset{\gamma}{\longrightarrow^*}}$. *Otherwise the order of reductions in the sequence* $\overset{(\mathbb{D},R)/(\mathbb{G}',\tilde{R}')}{\underset{\gamma}{\longrightarrow^*}}$ *does not matter.* □

Proof. A marked redex can not be a residual of a non-marked one, therefore $\tilde{F} \neq \emptyset$. By lemma C.13 we have the following two cases:

1. All the marked redexes are term redexes. Let $l_1, l', \tilde{l}$ be the labels of the components where the redexes $(\mathbb{D}, R), (\mathbb{D}', R')$, and $(\mathbb{G}, \tilde{R})$ occur, respectively.

If $\tilde{l} \neq l_1$, then the reduction of the non-evaluation redex $(\mathbb{D}, R)$ does not affect the evaluation redex $(\mathbb{G}, \tilde{R})$. The component bound to $\tilde{l}$ does not change, so $\tilde{F} = \{(\mathbb{G}', \tilde{R})\}$, and $(\mathbb{G}, \tilde{R}) = (\mathbb{G}', \tilde{R})/(\mathbb{D}, R)$. The evaluation redex is marked in D_1 since it is marked in D_2. The rest of the claim follows similarly to case 1 of the proof of lemma C.36.

If $\tilde{l} = l_1$, then the claim follows from lemma B.28. In this case when $l' \neq \tilde{l}$ $(\mathbb{D}', R')$ has one residual (itself), and when we apply lemma B.28, we don't consider $(\mathbb{D}', R')$ as a marked redex, since it occurs in a different component. If $l' = \tilde{l}$, then we consider all three marked redexes in the application of lemma B.28.

2. All the marked redexes are substitution redexes with the same label l. Let $l_1, l', \tilde{l}$ be as in case 1. Note that $l \neq \tilde{l}$, since $(\mathbb{G}, \tilde{R})$ is an evaluation redex. We have the following subcases:

   (a) $l_1 \neq l$, $l' \neq l$, i.e. none of the redexes is self-referential. Suppose that $l_1 = l' = \tilde{l}$, i.e. all redexes occur in the same component. Then

   $$D_2 = [\tilde{l} \mapsto \mathbb{A}\{\underline{l}, V, \underline{l}\}, l \mapsto V, \dots],$$

   where the first marked occurrence of l corresponds to the redex $(\mathbb{G}, \tilde{R})$, and the second to the redex $(\mathbb{D}', R')$. Note that the occurrence of $\underline{l}$ corresponding to the evaluation redex does not occur in V because it occurs in an evaluation context, and the one corresponding to $(\mathbb{D}', R')$ does not occur in V by the assumption that none of the redexes is self-referential. Then

   $$D_1 = [\tilde{l} \mapsto \mathbb{A}\{\underline{l}, \underline{l}, \underline{l}\}, l \mapsto V, \dots],$$

   by lemma C.38 $\mathbb{A}\{\square, \underline{l}, \underline{l}\} \in \mathbf{EvalContext}_{\mathcal{T}}$, and the claim of the lemma easily follows. The cases when the three redexes occur in at least two different components are similar.

   (b) $l' = l$, i.e. the redex $(\mathbb{D}', R')$ is self-referential. Suppose $\tilde{l} = l_1$, then

   $$\begin{aligned} D_2 &= [l \mapsto \lambda x.\mathbb{C}\{\underline{l}\}, \tilde{l} \mapsto \mathbb{A}\{\underline{l}, \lambda x.\mathbb{C}\{l\}\}, \dots], \\ D_1 &= [l \mapsto \lambda x.\mathbb{C}\{\underline{l}\}, \tilde{l} \mapsto \mathbb{A}\{\underline{l}, \underline{l}\}, \dots]. \end{aligned}$$

   As in the case 4(b) of the proof of lemma C.40, the marked occurrence of l in the component bound to $\tilde{l}$ in D_2 is independent from $\lambda x.\mathbb{C}\{l\}$, since it occurs in an evaluation context. The rest of the proof is by the reduction in the case 2(b) of the proof of lemma C.36. The case when the two non-self-referential redexes occur in different components is analogous.

   (c) $l_1 = l$, i.e. $(\mathbb{D}, R)$ is self-referential. If $\tilde{l} = l'$, we have:

   $$\begin{aligned} D_2 &= [l \mapsto \lambda x.\mathbb{C}\{\lambda x.\mathbb{C}\{l\}\}, \tilde{l} \mapsto \mathbb{A}\{\underline{l}, \underline{l}\}, \dots], \\ D_1 &= [l \mapsto \lambda x.\mathbb{C}\{\underline{l}\}, \tilde{l} \mapsto \mathbb{A}\{\underline{l}, \underline{l}\}, \dots]. \end{aligned}$$

   The reduction in the case 2(c) of the proof of lemma C.36. The case when the redex $(\mathbb{D}', R')$ occurs not in the same component as the evaluation redex is analogous.

□

Lemma C.44 (Elementary Lift Diagram for $\mathcal{C}$). *If* $(D_1, F_1) \underset{\gamma}{\overset{(\mathbb{D},R)}{\circ\!\!\longrightarrow}} (D_2, F_2) \overset{(\mathbb{G},\tilde{R})}{\Longrightarrow} (D_4, F_4)$*, then there exists* (D_3, F_3) *s.t.* $(D_1, F_1) \overset{(\mathbb{G}',\tilde{R}')}{\Longrightarrow} (D_3, F_3) \underset{\gamma}{\overset{(\mathbb{D},R)/(\mathbb{G}',\tilde{R}')}{\longrightarrow^*}} (D_4, F_4)$ *s.t.* $(\mathbb{G}, \tilde{R}) = (\mathbb{G}', \tilde{R}')/(\mathbb{D}, R)$. □

Proof. The proof is by lemmaC.40, C.41, C.42, and C.43 analogous to the proof of lemma C.37. □

It remains to show standardization of complete γ-developments . It follows from the lemmas C.35 and C.36 for the elementary project diagram and from lemmas C.42 and C.43 for the elementary lift diagram that when the self-referential redex is duplicated, the elementary diagrams commute if the self-referential copy of the redex is reduced after the non-self-referential one. For instance, if the two residuals of the self-referential redex in case 3 of lemma C.42 are reduced in the other order, the diagram does not commute:

$$\begin{aligned} &[l \mapsto \lambda x.\mathbb{C}\{\underline{l}\}, \tilde{l} \mapsto \mathbb{E}\{l\}, l' \mapsto \mathbb{A}\{\underline{l}\}, \dots] && \underset{\gamma}{\circ\!\!\longrightarrow} \\ &[l \mapsto \lambda x.\mathbb{C}\{\lambda x.\mathbb{C}\{l\}\}, \tilde{l} \mapsto \mathbb{E}\{l\}, l' \mapsto \mathbb{A}\{\underline{l}\}, \dots] && \Longrightarrow \\ &[l \mapsto \lambda x.\mathbb{C}\{\lambda x.\mathbb{C}\{l\}\}, \tilde{l} \mapsto \mathbb{E}\{\lambda x.\mathbb{C}\{\lambda x.\mathbb{C}\{l\}\}\}, l' \mapsto \mathbb{A}\{\underline{l}\}, \dots], \end{aligned}$$

but the sequence below leads to a module different from the one above:

$$\begin{array}{ll}
[l \mapsto \lambda x.\mathbb{C}\{\underline{l}\}, \tilde{l} \mapsto \mathbb{E}\{l\}, l' \mapsto \mathbb{A}\{\underline{l}\}, \ldots] & \Longrightarrow \\
[l \mapsto \lambda x.\mathbb{C}\{\underline{l}\}, \tilde{l} \mapsto \mathbb{E}\{\lambda x.\mathbb{C}\{\underline{l}\}\}, l' \mapsto \mathbb{A}\{\underline{l}\}, \ldots] & \underset{\gamma}{\circ\!\!\longrightarrow} \\
[l \mapsto \lambda x.\mathbb{C}\{\lambda x.\mathbb{C}\{l\}\}, \tilde{l} \mapsto \mathbb{E}\{\lambda x.\mathbb{C}\{\underline{l}\}\}, l' \mapsto \mathbb{A}\{\underline{l}\}, \ldots] & \underset{\gamma}{\circ\!\!\longrightarrow} \\
[l \mapsto \lambda x.\mathbb{C}\{\lambda x.\mathbb{C}\{l\}\}, \tilde{l} \mapsto \mathbb{E}\{\lambda x.\mathbb{C}\{\lambda x.\mathbb{C}\{\lambda x.\mathbb{C}\{l\}\}\}\}, l' \mapsto \mathbb{A}\{\underline{l}\}, \ldots]. &
\end{array}$$

Note that it is not only the sets of marked redexes, but the modules themselves are different.

Lemma C.45. *If* $(D_1, F_1) \underset{\gamma}{\overset{(\mathbb{D},R)}{\circ\!\!\longrightarrow}} (D_2, F_2) \underset{\gamma}{\overset{(\mathbb{G},\tilde{R})}{\Longrightarrow}} (D_4, F_4)$*, where* $(\mathbb{D}, R)$ *is not self-referential, then there exists* (D_3, F_3) *s.t.* $(D_1, F_1) \underset{\gamma}{\Rightarrow} (D_3, F_3) \underset{\gamma}{\longrightarrow^*} (D_4, F_4)$*.* $\square$

Note that when $(\mathbb{D}, R)$ is a self-referential redex, then the claim of the above lemma does not hold. For instance, consider:

$$\begin{array}{ll}
[l \mapsto \lambda x.\underline{l}, l' \mapsto \underline{l}] & \underset{\gamma}{\circ\!\!\longrightarrow} \\
[l \mapsto \lambda x.\lambda x.l, l' \mapsto \underline{l}] & \underset{\gamma}{\Longrightarrow} \\
[l \mapsto \lambda x.\lambda x.l, l' \mapsto \lambda x.\lambda x.l], & \text{but} \\
[l \mapsto \lambda x.\underline{l}, l' \mapsto \underline{l}] & \underset{\gamma}{\circ\!\!\longrightarrow} \\
[l \mapsto \lambda x.\underline{l}, l' \mapsto \lambda x.l] & \underset{\gamma}{\Longrightarrow} \\
[l \mapsto \lambda x.\lambda x.l, l' \mapsto \lambda x.l] &
\end{array}$$

Lemma C.46 (Weak Standardization of γ-developments in $\mathcal{C}$). *$\mathcal{C}$ has property A.31, i.e. if* $(D_1, \{(\mathbb{D}, R)\}) \underset{\gamma}{\overset{(\mathbb{D},R)}{\circ\!\!\longrightarrow}} (D_2, \emptyset) \Longrightarrow^* (D_4, \emptyset)$*,* $(D_1, \{(\mathbb{D}, R)\}) \Longrightarrow^* (D_3, F_3)$*, and the two evaluation sequences are related by elementary diagrams (as defined in A.30), then there exists* (D', F') *s.t.* $(D_3, F_3) \underset{\gamma}{\Rightarrow^*} (D', F') \underset{\gamma}{\circ\!\!\longrightarrow^*} (D_4, \emptyset)$*.* $\square$

Proof. Case 1. If $(\mathbb{D}, R)$ is not self-referential, then lemma C.45 is applicable to any two marked redexes (since by lemma C.13 all marked residuals of $(\mathbb{D}, R)$ are also non-self-referential). Since the γ-developments are finite (by lemma C.14), by lemma A.39 we can show strong standardization of γ-developments (property A.29).

Case 2. If $(\mathbb{D}, R)$ is self-referential, then the proof is by induction on the number of steps in the sequence $(D_1, \{(\mathbb{D}, R)\}) \Longrightarrow^* (D_3, F_3)$. Let n denote this number. We want to prove that for every such (D_3, F_3) and the corresponding $(D_4, \emptyset)$ there exists a standard complete development $(D_3, F_3) \underset{\gamma}{\Rightarrow^*} (D', F') \underset{\gamma}{\circ\!\!\longrightarrow^*} (D_4, \emptyset)$ s.t. the self-referential redex is reduced last.

Base case ($n = 1$). By lemma C.37 there exists a γ-development $(D_3, F_3) \underset{\gamma}{\longrightarrow^*} (D_4, \emptyset)$. By lemma C.13 F_3 contains a single self-referential redex, and by lemmas C.35, C.36, C.42, and C.43 the self-referential residual of the redex is reduced last, i.e. $(D_3, F_3) \underset{\gamma}{\longrightarrow^*} (D', F') \underset{\gamma}{\circ\!\!\longrightarrow} (D_4, \emptyset)$, where the sequence $(D_3, F_3) \underset{\gamma}{\longrightarrow^*} (D', F')$ does not reduce self-referential redexes. Then by lemma C.45 there exists (D'', F'') s.t. $(D_3, F_3) \underset{\gamma}{\Rightarrow^*} (D'', F'') \underset{\gamma}{\circ\!\!\longrightarrow^*} (D', F')$, and the claim of the lemma holds.

Induction Step. Suppose $(D_1, \{(\mathbb{D}, R)\}) \Longrightarrow^* (D_3', F_3') \Longrightarrow (D_3, F_3)$, $(D_2, \emptyset) \Longrightarrow^* (D_4', \emptyset)$, the two evaluation sequences are related by the elementary diagrams, and for (D_3', F_3') and $(D_4', \emptyset)$ the claim of the lemma holds. By the condition of the lemma there exists an evaluation sequence related by elementary diagrams to the sequence $(D_1, \{(\mathbb{D}, R)\}) \Longrightarrow^* (D_3', F_3') \Longrightarrow (D_3, F_3)$. This implies that there exist $(\tilde{D}_3, \tilde{F}_3)$ and $(\tilde{D}_3', \tilde{F}_3')$ s.t. $(D_3', F_3') \underset{\gamma}{\Rightarrow^*} (\tilde{D}_3', \tilde{F}_3') \overset{0/1}{\Longrightarrow} (\tilde{D}_3, \tilde{F}_3)$ and $(D_3, F_3) \underset{\gamma}{\Rightarrow^*} (\tilde{D}_3, \tilde{F}_3)$, and we have one of the two possibilities:

- $(\tilde{D}'_3, \tilde{F}'_3) \overset{0/1}{\Longrightarrow} (\tilde{D}_3, \tilde{F}_3)$ is a 0-step reduction. Since the two evaluation sequences are related by elementary diagrams, $(\tilde{D}'_3, \tilde{F}'_3) \underset{\gamma}{\circ\!\!\longrightarrow^*} (D'_4, \emptyset)$. By the induction hypothesis the last redex reduced in the non-evaluation γ-development sequence is the self-referential residual of $(\mathbb{D}, R)$, therefore the sequence $(D_3, F_3) \underset{\gamma}{\Longrightarrow^*} (\tilde{D}_3, \tilde{F}_3) \underset{\gamma}{\circ\!\!\longrightarrow^*} (D'_4, \emptyset)$ satisfies the claim of the lemma.

- $(\tilde{D}'_3, \tilde{F}'_3) \overset{0/1}{\Longrightarrow} (\tilde{D}_3, \tilde{F}_3)$ is a 1-step reduction. The two evaluation sequences are related by elementary diagrams, therefore there exists $(D_4, \emptyset)$ s.t. $(\tilde{D}'_3, \tilde{F}_3) \underset{\gamma}{\circ\!\!\longrightarrow^*} (D'_4, \emptyset) \Longrightarrow (D_4, \emptyset)$, $(\tilde{D}_3, \tilde{F}_3) \underset{\gamma}{\longrightarrow^*} (D_4, \emptyset)$, and the reductions are constructed from the elementary diagrams. By the inductive hypothesis the last redex in the sequence $(\tilde{D}'_3, \tilde{F}_3) \underset{\gamma}{\circ\!\!\longrightarrow^*} (D'_4, \emptyset)$ is the self-referential residual of $(\mathbb{D}, R)$. Let $(\mathbb{D}', R')$ denote this redex. By lemma C.13 the other redexes reduced in this sequence are not self-referential. Since the sequence $(\tilde{D}_3, \tilde{F}_3) \underset{\gamma}{\longrightarrow^*} (D_4, \emptyset)$ is constructed from $(\tilde{D}'_3, \tilde{F}'_3) \underset{\gamma}{\circ\!\!\longrightarrow^*} (D'_4, \emptyset)$ via the elementary diagrams, there exist (D'', F'') and (D''', F''') s.t. $(\tilde{D}'_3, \tilde{F}'_3) \underset{\gamma}{\circ\!\!\longrightarrow^*} (D'', F'') \underset{\gamma}{\circ\!\!\xrightarrow{(\mathbb{D}', R')}} (D'_4, \emptyset)$, $(\tilde{D}_3, \tilde{F}_3) \underset{\gamma}{\longrightarrow^*} (D''', F''') \underset{\gamma}{\longrightarrow^*} (D_4, \emptyset)$, and $(D'', F'') \Longrightarrow (D''', F''')$. Let $(\mathbb{D}'', R'')$ be the self-referential residual of $(\mathbb{D}', R')$. By lemma C.35 $(\mathbb{D}'', R'')$ is reduced last in the sequence $(D''', F''') \underset{\gamma}{\longrightarrow^*} (\hat{D}, \hat{F}) \underset{\gamma}{\circ\!\!\xrightarrow{(\mathbb{D}'', R'')}} (D_4, \emptyset)$. A self-referential redex is a non-evaluation redex, therefore the step $\underset{\gamma}{\circ\!\!\xrightarrow{(\mathbb{D}'', R'')}}$ is a non-evaluation step. Since no other redexes in the sequence $(\tilde{D}_3, \tilde{F}_3) \underset{\gamma}{\longrightarrow^*} (D''', F''') \underset{\gamma}{\longrightarrow^*} (\hat{D}, \hat{F})$ are self-referential, by lemma C.45 there exists $(\hat{D}', \hat{F}')$ s.t. $(\tilde{D}_3, \tilde{F}_3) \underset{\gamma}{\Longrightarrow^*} (\hat{D}', \hat{F}') \underset{\gamma}{\circ\!\!\longrightarrow^*} (\hat{D}, \hat{F})$, which proves the claim of the lemma.

□

Recall that classification of modules is defined in section 2.4 as

$$Cl_{\mathcal{C}}(D) = \begin{cases} \textbf{evaluatable}_{\mathcal{C}} & \text{if there exists } D' \text{ s.t. } D \Longrightarrow_{\mathcal{C}} D' \\ [v_i \overset{n}{\underset{i=1}{\mapsto}} Cl_{\mathcal{T}}(V_i)] & \text{if } D = [v_i \overset{n}{\underset{i=1}{\mapsto}} V_i, h_j \overset{m}{\underset{j=1}{\mapsto}} V'_j], \\ \textbf{error}_{\mathcal{C}} & \text{otherwise} \end{cases}$$

We omit an obvious proof of the following lemma:

Lemma C.47 (Class Preservation of $\mathcal{C}$). *If $D_1 \circ\!\!\longrightarrow_{\mathcal{C}} D_2$, then $Cl_{\mathcal{C}}(D_1) = Cl_{\mathcal{C}}(D_2)$.* □

We can also show class preservation for the classification of the core module calculus defined in [MT00]:

$$Cl_{\mathcal{C}}(D) = [l_i \overset{n}{\underset{i=1}{\mapsto}} Cl_{\mathcal{T}}(M_i)], \text{ where } D = [l_i \overset{n}{\underset{i=1}{\mapsto}} M_i]. \quad \text{([MT00] definition)}$$

Theorem C.48 (Computational Soundness of $\mathcal{C}$). *If $D_1 \leftrightarrow_{\mathcal{C}} D_2$, then $Outcome_{\mathcal{C}}(D_1) = Outcome(D_2)$.* □

Proof. By theorems A.32 and A.33 $\mathcal{C}$ has lift and project properties, since it satisfies the elementary diagrams (lemmas C.37 and C.44), has weak γ-confluence of evaluation (lemma C.31), and weak standardization of γ-developments (lemma C.46). These results are shown for pairs (D, F), and by lemma A.20 they hold for modules with no marked redexes. $\mathcal{C}$ also has confluence of evaluation (lemma C.17) and class preservation (lemma C.47), therefore by theorem 3.41 it is computationally sound. □

D Soundness of the Linking Calculus.

This section is under construction.

E Soundness of Calculi Extended with Garbage Collection

This section is under construction.